Neural Architecture Search Survey: A Hardware Perspective

KRISHNA TEJA CHITTY-VENKATA and ARUN K. SOMANI, Iowa State University, USA

We review the problem of automating hardware-aware architectural design process of Deep Neural Networks (DNNs). The field of Convolutional Neural Network (CNN) algorithm design has led to advancements in many fields, such as computer vision, virtual reality, and autonomous driving. The end-to-end design process of a CNN is a challenging and time-consuming task, as it requires expertise in multiple areas such as signal and image processing, neural networks, and optimization. At the same time, several hardware platforms, general- and special-purpose, have equally contributed to the training and deployment of these complex networks in a different setting. **Hardware-Aware Neural Architecture Search** (HW-NAS) automates the architectural design process of DNNs to alleviate human effort and generate efficient models accomplishing acceptable accuracy-performance tradeoffs. The goal of this article is to provide insights and understanding of HW-NAS techniques for various hardware platforms (MCU, CPU, GPU, ASIC, FPGA, ReRAM, DSP, and VPU), followed by the co-search methodologies of neural algorithm and hardware accelerator specifications.

CCS Concepts: • **Computer systems organization** → **Architectures**; • **Computing methodologies** → **Machine learning**; **Learning paradigms;**

Additional Key Words and Phrases: Deep Neural Networks, Convolutional Neural Networks, Neural Architecture Search, Hardware-Aware Neural Architecture Search, general purpose hardware, domain specific accelerators, quantization, accelerator network co-search, literature review, survey, CPU, GPU, ASIC, FPGA

ACM Reference format:
Krishna Teja Chitty-Venkata and Arun K. Somani. 2022. Neural Architecture Search Survey: A Hardware Perspective. *ACM Comput. Surv.* 55, 4, Article 78 (November 2022), 36 pages.
https://doi.org/10.1145/3524500

1 INTRODUCTION

Convolutional Neural Networks (CNNs) are de facto in many computer vision tasks, such as Image Classification [70] and Object Detection [135]. This success of CNN is mainly due to its automatic feature extraction ability from raw data. A CNN design of efficient architecture operations, connections, and hyperparameters usually guarantees high accuracy.

Efficient Network Design: Manually designed CNNs require in-depth domain knowledge and substantial computing resources to evaluate different architectures in the trial-and-error procedure. Also, prior knowledge in the design process will likely restrict building new architectures for various tasks to obtain better accuracy. **Automated Machine Learning (AutoML)** is a method to

The research reported in this article is supported by the Philip and Virginia Sproul Professorship at Iowa State University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s).
Authors' address: K. T. Chitty-Venkata and A. K. Somani, Iowa State University, Ames, Iowa, 50010; emails: {krishnat, arun}@iastate.edu.

This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2022 Copyright held by the owner/author(s).
0360-0300/2022/11-ART78
https://doi.org/10.1145/3524500

automate the entire design pipeline of **Machine Learning (ML)** models [121]. **Neural Architecture Search (NAS)**, a subset of AutoML, is a technique used to automate the neural architecture design process for a given dataset. Several NAS methods have been developed, starting from **Reinforcement Learning (RL)**-based methods [106, 150] to gradient descent algorithms [86]. These methods have shown that automatically designed networks could outperform manually designed CNNs in different aspects such as accuracy and performance. NAS has been very successful in discovering models that achieve **state-of-the-art (SOTA)** accuracy on tasks such as Image Classification [86], Semantic Segmentation [114], Machine Translation [125], and so on.

Hardware for Deep Learning: Orthogonal to Neural Network algorithm lies an efficient hardware design, which has been a driving force to the evolution of modern Deep Learning. A wide range of hardware solutions is available to accelerate the performance of DNNs, ranging from general-purpose hardware to special-purpose devices. It is not straightforward to choose a hardware platform that works in all the scenarios, as it depends on the application and deployment environment. For example, an edge MCU requires a small area and power, whereas a cloud accelerator like Tensor Processing Unit [65] can execute large models without any size and power constraints. **Single-instruction multiple-data (SIMD)** CPU machines are very effective but also require higher area and energy consumption. Beyond the SIMD CPUs, application-specific hardware and **Neural Processing Units (NPUs)** offer better performance. However, **Field Programmable Gate Array (FPGA)** provides great hardware and software design flexibility. Hence, the underlying hardware architecture plays an important role in designing cost-effective DNNs.

Hardware-aware Neural Architecture Search (HW-NAS) has emerged as one of the most promising techniques to automatically generate efficient CNN models accomplishing acceptable accuracy-performance tradeoffs. HW-NAS algorithms explore the search space of a CNN by jointly optimizing the accuracy and hardware execution metrics such as latency, energy, size, and so on. Many works [8, 119, 132] have shown that a model searched/optimized for a given hardware may or may not be efficient on different hardware. For example, FBNet [132] showed that the CNN model searched with respect to iPhoneX takes 19.84 ms to execute, but the latency increases to 23.33 ms on Samsung S8. However, the CNN architecture searched for Samsung S8 executes with a latency of 22.12 ms, but consumes 27.53 ms on iPhoneX.

This survey article provides an in-depth literature review of several Hardware-aware CNN NAS algorithms targeting MCU, CPU, GPU, ASIC, FPGA, ReRAM, DSP, and VPU, followed by an overview of algorithm-accelerator co-search. HW-NAS requires cross-disciplinary knowledge in device-specific compilation, hardware micro-architecture, neural network design, and efficient NAS algorithms. Many AutoML and NAS survey papers [29, 48, 110, 131] have focused on the theoretical concepts of architecture search without extensively discussing hardware-based approaches. A recent review article [6] provides a brief overview of HW-NAS. However, an in-depth review in this article includes CNN primitives, HW-NAS methods on a wide range of devices, and hardware-algorithm co-search. The organization of this article is detailed in Table 1. This survey primarily covers research work reported in literature from 2017 to August 2021.

2 NEURAL NETWORK TERMINOLOGY AND DEFINITIONS

This section provides a brief overview of various primitive Convolution operations, which are the building blocks of CNNs. This set of operations are very important to understand, as most of the NAS methods utilize them in their search space.

Standard Spatial Convolution. A Convolution operation plays a pivotal role in feature extraction [70]. A standard spatial Convolution layer consists of a set of 3-D filters where each 2-D filter is decomposed into distinct k*x*k kernels. Each kernel convolves with a region in one channel of

Table 1. Organization of the Article

Section Name	Summary of Each Section	
Section 2: CNN Review	Standard Convolution Dilation Convolution [144] Depthwise Convolution [51, 117] Pointwise Convolution	Squeeze and Excitation [53] Group Convolution [70] MBConv [42, 50, 112, 132] Shufflenet Choice [11, 95]
Section 3: HW-NAS	**Section 3.1:** Search Space (SS) **Section 3.2:** Construction of SS **Section 3.3:** Search Strategy	**Section 3.4:** Network Accuracy **Section 3.5:** HW Metrics **Section 3.6:** Searched Model
Section 4: MCU-NAS	**Section 4.1:** MCU-specific [5, 13, 33, 80–82, 88, 89, 98, 126]	
Section 5: CPU-NAS	**Section 5.1:** Mobile/Edge CPUs [8, 9, 50, 99, 100, 116, 119, 132, 134] **Section 5.2:** High-end/Desktop CPUs [8, 11, 101, 120, 136]	
Section 6: GPU-NAS	**Section 6.1:** High-end GPUs [8, 32, 35, 41, 52, 54, 85, 118, 123, 143] **Section 6.2:** Edge/Mobile GPUs [18, 78, 91, 93, 94, 114, 124]	
Section 7: Accelerator-NAS	**Section 7.1:** ASIC [2, 14, 42, 73, 74] **Section 7.2:** FPGA [31, 44, 61, 146]	**Section 7.3:** VPU and DSP [24, 147] **Section 7.4:** ReRAM [145]
Section 8: Accelerator & Arch. Co-search	**Section 8.1:** FPGA [1, 44, 63, 64]	**Section 8.2:** ASIC [15, 84, 97, 138, 141, 149] **Section 8.3:** ReRAM [102]
Section 9: Searching for Precision	**Section 9.1:** Mixed Precision Quantization Search [10, 30, 56, 107, 127, 133] **Section 9.2:** Network Architecture and Precision Co-search [39, 137] **Section 9.3:** Accelerator, Arch., Precision Co-search [26, 37, 62, 79, 83]	
Section 10: Other forms of HW-NAS	**Section 10.1:** Pruning [49, 128, 139] **Section 10.2:** NAS Processor [96] **Section 10.3:** Winograd [34] **Section 10.4:** NAS for RL Agent [36]	**Section 10.5:** Fault Tolerant NAS [77] **Section 10.6:** EDA [104] **Section 10.7:** Adversarial NAS [38, 129] **Section 10.8:** Benchmarks [28, 75]

the input feature map to produce a partial sum. All partial sums across the entire input channels "I" are accumulated to produce one output pixel.

Dilation Convolution. A Dilation Convolution expands the kernel's receptive field by inserting zeros between the kernel of a Convolutional layer [144]. A dilation rate of 1 is a normal Convolution, and a k-Dilated Convolution is a Standard Convolution with "k" defined gaps in the kernel.

Depthwise Convolution. Spatial Convolution applies a 3-D filter to all the channels in the input feature map, requiring k*k*I multiplications to generate one output pixel. A Depthwise Convolution [51] produces one output pixel by applying a filter (k*x*k) to only one channel of the input feature map. Every kxkxI filter is first divided into c slices, each of dimension kxkx1. The Convolution operation is performed individually in each channel with the corresponding slice of the filter. Hence, the number of multiplications is reduced by an order of "I" in each layer. **MixConv** [117] partitions channels into groups and applies different kernel sizes to each group.

Pointwise Convolution. A Pointwise Convolution is a Standard Convolution whose kernel size is 1×1. An input feature map of size ($H x W x C_i$) is converted to ($H x W x C_o$) by convolving with ($1\ x\ 1\ x\ C_i\ x\ C_o$) filter. A **Depthwise-separable Convolution** is a sequence of Depthwise Convolution of each channel, followed by a Pointwise Convolution to combine the outputs in the depth dimension.

Squeeze and Excitation (SE). An SE connection [53] is intended to improve the quality of feature maps by modeling interdependencies between them. It can be added to any Standard Convolution as an auxiliary parallel connection. In the first "squeeze" step of the supplemental path, a global average pooling is used to collect the representation of each channel in the input feature map, in the form of a 1*x*1*x*k vector. In the second "excitation" step, a fully connected layer is applied to the pooled vector to produce a collection of 1*x*1*x*k per-channel weights, whose channel

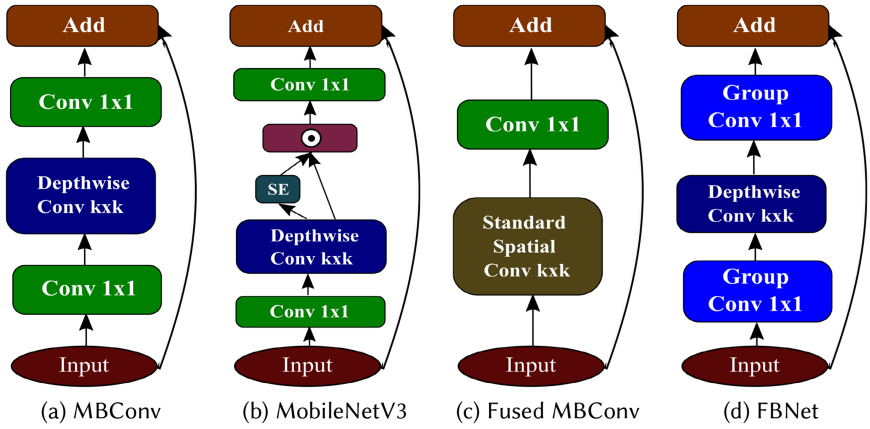


Fig. 1. Types of mobile inverted bottleneck layer (MBConv) types.

size is equal to the number of channels in the input feature map. The per-channel weights and input feature maps are multiplied channel-wise to produce the final SE block.

Group Convolution. The concept of Group Convolution was first used in AlexNet [70], where the filters and input activations were divided into "g" groups. A filter in each group convolves only with the activations in the corresponding group, thereby reducing the number of multiplications to produce the output activation, compared to the Standard Convolution.

Mobile Inverted Bottleneck Layer (MBConv). MBConv layers are the basic units of MobileNetV2 network [112], which is composed of a series of three operations: 1 × 1 Pointwise Convolution on (H x W x c) input feature map to produce (H x W x (ct)) output feature map -> kxk Depthwise Convolution on (H x W x (ct)) with stride "s" to produce (H/s x W/s x ct) -> 1 × 1 Pointwise Convolution on (H/s x W/s x ct) to produce ((H/s) x (W/s) x c'). The expansion ratio "t" controls the widening of feature maps, and "ct" are known as expansion channels.

A **Fused-MBConv** [42] is derived from MBConv blocks by fusing the first 1 × 1 expansion Convolution and the k*x*k Depthwise Convolution to form a k*x*k Standard Convolution. Even though, this fusion has more parameters over the baseline MBConv, they run faster on few devices for some input and output feature map shapes. **FBNet Block** [132] replaces the 1 × 1 point-wise Convolution in the traditional MBConv with the 1 × 1 group Convolution (Figure 1(d)). The **MobileNetV3-MBConv** module [50] is obtained by adding an SE connection to the MBConv block (Figure 1(b)).

ShuffleNet Choice Block. ShuffleNetV2 [95] adopts a series of efficient operations such as group Convolutions, balanced Convolutions, channel shuffling, and channel concatenation to reduce the computation overhead. The MobileNetV1 [51], MobileNetV2 [112], MobileNetV3 [50], and ShuffleNet [95] networks are used backbones in many HW-NAS methods.

3 HARDWARE-AWARE NEURAL ARCHITECTURE SEARCH (HW-NAS)

Neural Architecture Search (NAS) methods can search network architectures that are more accurate and hardware-efficient compared to the handcrafted/manually designed models. The task of NAS is very close to a conventional deep learning task. For a given dataset D with input-output pair (x, y), we need to learn the best network architecture that automatically fits the dataset. Given an architectural space, the goal is to find an optimal network architecture such that it performs best on the validation dataset (D_{val}).

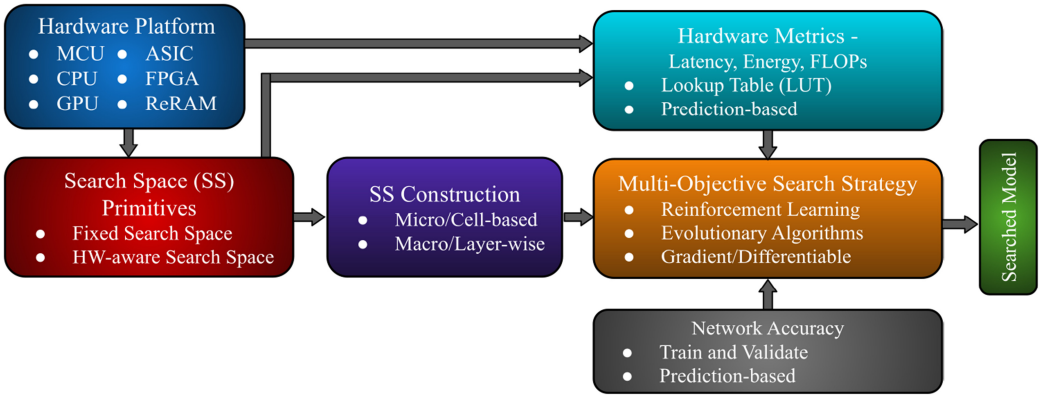


Fig. 2. Pipeline of hardware-aware neural architecture search.

The plethora of hardware devices available for various scenarios and circumstances makes it very difficult to choose one system for all cases. DNNs are typically trained on GPUs but deployed for inference on a variety of hardware, such as MCU, CPU, ASICs, and so on. HW-NAS aims to automate the design process of deep neural networks to find models that have a good tradeoff between accuracy and performance metrics such as latency [119, 132], FLOPS [119, 120], power consumption, energy [39], and memory usage [81]. HW-NAS algorithm takes the network search space, target hardware characteristics, and the task as input to generate hardware adapted architectures as outputs. The typical pipeline of HW-NAS is shown in Figure 2.

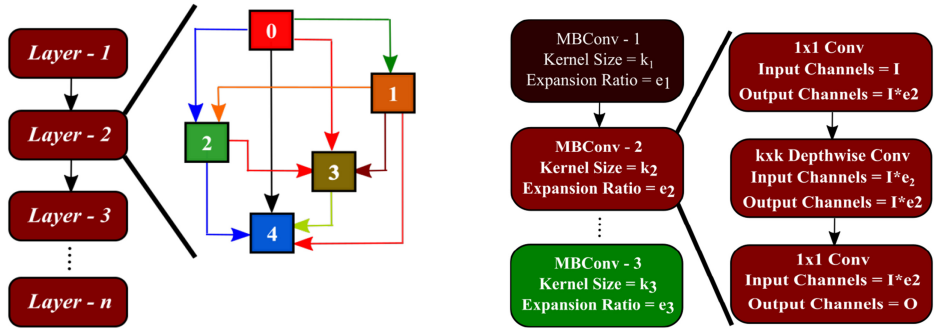
3.1 Search Space Primitives

The design of search space for a CNN is combinatorial, and the search complexity increases with an increase in the number of elements. Consider an example of ResNet18 network [47] that has 16 Convolutional layers. The number of possible child networks with {3, 5, 7} kernel and {32, 64, 128, 256, 512} channel choices are $(3 \times 5)^{16}$. Hence, it is extremely crucial to choose the search elements to find the network in a limited time. The design of search space for HW-NAS can be divided into two categories based on the influence of the hardware, namely, **(i)** Fixed Search Space, **(ii)** Hardware-aware Search Space.

3.1.1 Fixed Search Space. It consists of a manually designed operations and structure of each block, irrespective of the hardware in consideration. For example, ProxylessNAS [8] consists of MobileNetV2 as backbone with MBConv block to be searched from the following elements: {3, 5, 7} kernel sizes and {3, 6} expansion ratios for all the devices (Mobile, CPU, and GPU).

3.1.2 Hardware-aware Search Space (HW-SS). The operations in the search space are adapted depending on the underlying hardware either inter-layer within the network or intra-layer throughout the network due to following reasons:

(1) Few hardware platforms may have a specific requirement or a prerequisite while running a Convolution operation. For instance, a Hexagon v62 DSP favors an operation with filter sizes in multiples of 32, whereas other devices may or may not have any such requirement [20].
(2) The same operation may have very different latency on different devices, and for the same type of hardware, different operations at each level may have different execution time. Zhang et al. [147] provide an in-depth analysis by running different operations at each stage of the



(a) Cell-based Search (b) Layer-wise Search

Fig. 3. Neural architecture search space.

network on Hexagon 685 DSP, Snapdragon 845 CPU, and Movidius **Vision Processing Unit (VPU)**. The runtime of the ShuffleNet Choice_3 block is almost equal to the latency of the ShuffleNet Choice_3_SE on the DSP, but the former is faster than the latter by up to 24× on the VPU. The operation with the smallest latency on DSP is the Depthwise Separable Convolution, while ShuffleNet Choice_3 runs faster on the CPU compared to the other operations.

(3) Few operations or filter sizes may not satisfy the memory requirement of resource constraint devices. TinyNAS [81] optimizes the search space automatically to ensure the operations fit on the tiny MCU. It relies on the insight that an operation with higher FLOPs on the given memory constraint can produce better accuracy.

(4) A large search space with many possible candidate operations may limit the search method to find better accuracy-latency tradeoff models [99]. Zeng et al. proposed a black-box profiling [146] to obtain a small space for the target accelerator.

3.2 Search Space Construction

3.2.1 Cell-based/Micro Search. The Cell-based NAS works on the principle that one can search for a cell/module to form a CNN model instead of searching for the network end-to-end. A **Directed Acyclic Graph (DAG)** or a cell topology represents the building blocks of the searched network. An example of such cell/DAG architecture is illustrated in Figure 3(a). Each edge in the cell is an operation (Convolution, pooling, or a skip connection) present in the pre-defined search space. The task is to search for individual operations on each edge. The searched cell is stacked multiple times to form the final CNN, therefore reusing the same cell architecture throughout the network. There exist two cell types based on the resolution of input and output feature maps: A "Normal Cell" does not change the resolution (Height and Width) of the feature map, whereas a "Reduction Cell" halves the size of the input feature map. The cell-based NASNet network [150] has lower parameters and FLOPs compared to MobileNet [51]. However, the fragmented structure in the cell is not friendly to many hardware platforms, leading to significantly high runtime. Hence, only a few methods, such as "new search space," a modified version of cell-based space suitable for efficient execution on a multi-GPU system [35], EdgeNAS [93] on Jetson Xavier GPU, and **LA-DNAS** [136] on Tesla-P100 GPU employ cell-based architecture.

3.2.2 Layer-wise/Macro Network Search. The single cell structure can have a varying impact on accuracy and latency at each neural network layer. Layer-wise/Macro Search overcomes this limitation by searching different operations/block configurations at each layer of the network to

Table 2. Macro-architecture of MobileNetV2 [112], EfficientNet-B0 [120], and FBNet [132]

Stage	Input	MobileNetV2 [112]			EfficientNet-B0 [120]			FBNet [132]			s
		Operator	C	B	Operator	C	B	Operator	C	B	
1	224^2	3 × 3 Conv2D	32	1	3 × 3 Conv2D	32	1	3 × 3 Conv2D	16	1	2
2	112^2	MBConv_e_k	16	1	MBConv_1_3	16	1	FBNet_e_k_g	16	1	1
3	112^2	MBConv_e_k	24	2	MBConv_6_3	24	2	FBNet_e_k_g	24	4	2
4	56^2	MBConv_e_k	32	3	MBConv_6_5	40	2	FBNet_e_k_g	32	4	2
5	28^2	MBConv_e_k	64	4	MBConv_6_3	80	3	FBNet_e_k_g	64	4	2
6	14^2	MBConv_e_k	96	3	MBConv_6_5	112	3	FBNet_e_k_g	112	4	1
7	14^2	MBConv_e_k	160	3	MBConv_6_5	192	4	FBNet_e_k_g	184	1	2
8	7^2	MBConv_e_k	320	1	MBConv_6_3	320	1	FBNet_e_k_g	352	1	1
9	7^2	Pooling, FC	1,280	1	Pooling, FC	1280	1	Pooling, FC	1,504	1	1

e, k, g in MBConv_e_k, FBNet_e_k_g stands for expansion ratio, kernel size of depthwise, and number of groups in pointwise Conv. **C**: Number of Output Channels of the stage; **B**: Number of Blocks in each stage; **s**: stride size.

obtain more efficient models that are suitable for efficient inference on many devices due to the regularized topology. The most common layer-wise search spaces are:

(1) **MobileNetV1-space** [51] uses a series of Depthwise-Separable Convolutions [16] for efficient inference on mobile phones with search element being number of filters in each block.
(2) **MobileNetV2-space** [112] is built on the MBConv blocks (Figure 1(a)), whose macro-architecture is given in Table 2. The MBConv blocks are grouped to form a stage, and each stage shares the same output channels/filters (C). The search components in each MBConv module are kernel size in the Depthwise Convolution and expansion ratios/channels.
(3) **MobileNetV3-space** [50] has the macro-architecture similar to MobileNetV2 along with the use of a hard swish activation and SE connection in each MBConv block (Figure 1(b)).
(4) **MNASNet-space** [119] is a stage-wise factorized hierarchical search, where a few blocks are grouped to form a stage. The same operation is searched for all the modules in a single stage.
(5) **EfficientNet-B0** [120] is obtained by applying multi-objective RL-NAS on the MNASNet-space. The searched EfficientNet-B0 (Table 2) is followed by compound scaling to obtain robust models for target latency (discussed in Section 5.2)
(6) **FBNet-space** [132] is built on the MobileNetV2 macro-architecture (Table 2), where FBNet blocks (Figure 1(d)) are used in place of MBConv modules. The 1 × 1 Pointwise Convolution in the traditional MBConv is replaced with the 1 × 1 Group Convolution. The search elements are the same as MobileNetV2-space along with the group size in the 1 × 1 Group Convolution.
(7) **ShuffleNetV2-space** [95] is obtained by stacking shufflenet choice blocks, and the search elements are similar to MobileNetV2 due to its similar macro-architecture and block elements.
(8) **Mixed Search Space** incorporates modules belonging to different backbone networks. For example, HURRICANE [147] builds an elaborate search space consisting of all the fundamental units from MobileNetV1 [51], MobileNetV2 [112], MobileNetV3 [50], and ShuffleNetV2 [95].

3.3 Search Strategy

3.3.1 Reinforcement Learning (RL)-based NAS. Zoph et al. [150] restarted the field of NAS using the RL method in 2017. A typical RL algorithm consists of an agent that learns through its interaction with the environment. The agent can learn to alter its decisions based on the actions, outcomes, and rewards received [3]. With every change in the state of the environment, the agent

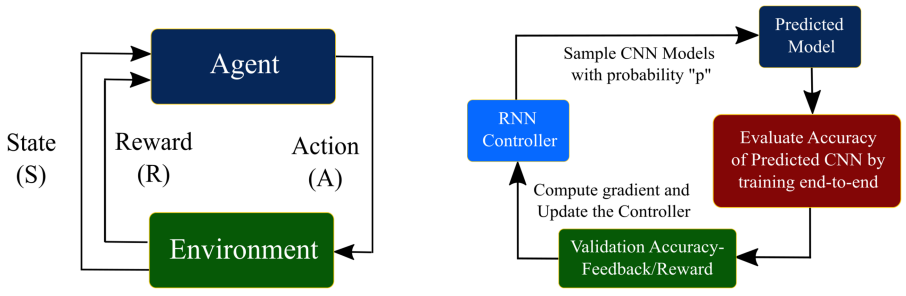


Fig. 4. Reinforcement learning-based neural architecture search [150].

receives a scalar reward, and the best sequence of actions to be taken is determined by the rewards it received. The goal of the agent is to return the optimal steps to be taken that maximize the expected reward.

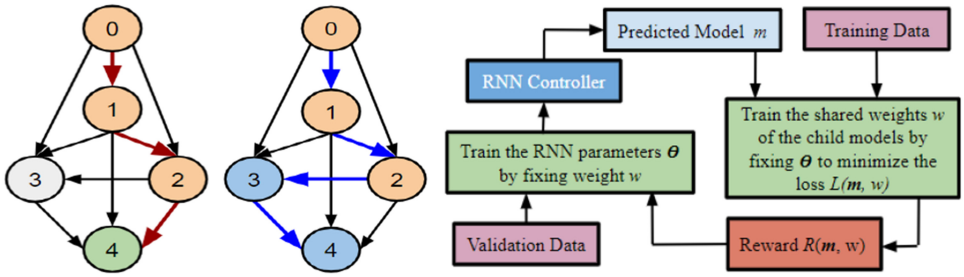
The Reinforcement Learning-based NAS method [150] discovers the best-performing architecture similar to the standard RL method, as shown in Figure 4(b). It uses a **Recurrent Neural Network (RNN)** controller as an agent to interact with the environment consisting of all possible neural network architectures. The predicted CNN models from the RNN controller are trained end-to-end and validated using a validation dataset. The validation accuracy is used as a feedback signal (reward) to update the controller's next candidate model. In RL terminology, CNN model moves from the current state to a state where the validation accuracy (reward) increases. The policy network generates a batch of actions $a_{t,n}$, which produce a series of child networks. The rewards from the child networks are accumulated to train the policy network using REINFORCE [130].

3.3.2 ENAS: Efficient Neural Architecture Search. **ENAS** [106] alleviates the search cost by training a controller to identify a subset of architecture within a **Directed Acyclic Graph (DAG)**. The search space of NAS can be viewed as a superposition of all the child networks in the form of a DAG. ENAS shares all the weight tensors of all the child models within the same iteration and successive predictions. Figures 5(a) and 5(b) illustrate paths (red and blue) of two child models predicted by the controller, where the paths (0→1), (1→2) are common to both the sub-networks. ENAS shares the Convolutional weight parameters of both the paths and runs multiple child networks in parallel on a single GPU. Also, if these paths are common in the predicted network of the successive iteration, then the corresponding weight tensors are retained without training the new architecture.

3.3.3 Reinforcement Learning-based HW-NAS. RL-based HW-NAS methods utilize an RNN controller to sample models with high accuracy and low computation cost. Along with the validation accuracy, these methods also consider hardware metrics as a mixed feedback signal. Given a CNN model M, let Acc(M) denote its validation accuracy, Lat(M) denotes the latency on the target hardware, and let T be the target latency. A usual method is to consider the target latency T as a hard constraint and sample the model with high accuracy under the given constraint, as shown in Equation (1).

$$\max_{M} \; Acc(M); \;\; s.t \;\; Lat(M) \;\leq\; T \tag{1}$$

A customized weighted product term can be used to approximate and find Pareto optimal models, with a tunable parameter γ to enable a tradeoff between accuracy and latency, as given in



(a) DAG-1 (b) DAG-2 (c) ENAS

Fig. 5. ENAS: Efficient neural architecture search [106].

Equation (2).

$$\max_{M} \quad Acc(M) * \left(\frac{Lat(M)}{T}\right)^{\gamma} \tag{2}$$

MONAS [52] defines the mixed reward function by considering power, energy, and MACs as hard constraints along with the validation accuracy. The reward signal incorporates both accuracy and energy, with an α parameter to find models with high accuracy and low energy, as per Equation (3).

$$R = \alpha * Accuracy - (1 - \alpha) * Energy, \quad \alpha \in [0, 1] \tag{3}$$

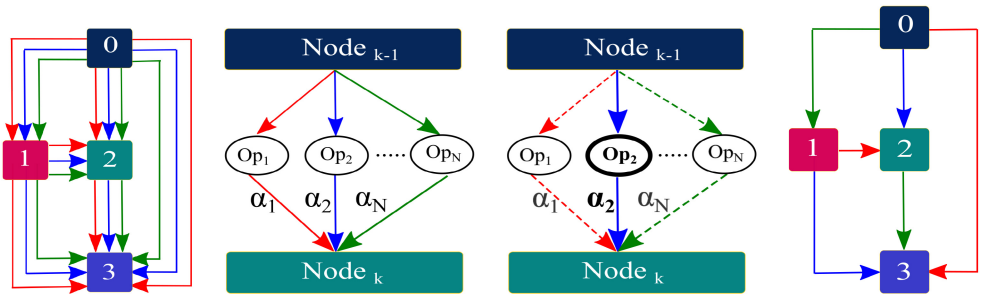
MONAS [52] also finds models with lower power by treating it as a hard constraint. The number of MAC operations is considered as a proxy to measure power. By treating MACs as a hard constraint below a threshold, the controller samples networks with less power, as shown in Equation (4).

$$R = \begin{cases} Accuracy, & if\ MACs\ (power)\ <\ threshold \\ 0, & otherwise \end{cases} \tag{4}$$

To find highly accurate models, accuracy is treated as a hard constraint, thereby forcing the controller to sample only the models with high accuracy, as illustrated in Equation (5).

$$R = \begin{cases} 1 - Energy, & if\ accuracy\ >\ threshold \\ 0, & otherwise \end{cases} \tag{5}$$

3.3.4 Evolutionary Search. Evolutionary architecture search is a population-based computational method to simulate the evolution of species to solve the NAS optimization problem [87]. Initially, a population is initialized where each individual represents a unique DNN solution, and each candidate is evaluated based on the fitness function (typically validation accuracy). In each search iteration, the fittest candidates are selected as parents for reproduction based on the validation accuracy. For example, if the population size is 2,000 and the parent ratio is 0.2, then 400 fittest models are selected as parents. A child is generated by a crossover of two randomly selected parents, i.e., the child's architecture parameter is randomly chosen from one of the two parents. The resultant child undergoes a mutation based on the mutation setting where the child's search parameters, such as an operation, kernel size, and so on, are chosen from the giant search space. The fittest candidates are selected from a pool of parents, children, and mutated children in the succeeding iteration, and the process is repeated until the maximum search iteration number is reached.



(a) Super Cell (b) Mixed Operation (c) Sampled Operation (d) Final Cell

Fig. 6. DARTS: Differentiable architecture search [86].

3.3.5 Hardware-aware Evolutionary Search. Similar to the previous RL-based HW-NAS, the Hardware-aware Evolutionary Search method employs both accuracy and hardware metric (e.g., latency) tuned by a parameter "t" (t ϵ [0, 1]) in the fitness function, given in Equation (6). A higher "t" value favors latency over accuracy and vice versa.

$$fitness\_function = (1 - t) * accuracy + t * latency \tag{6}$$

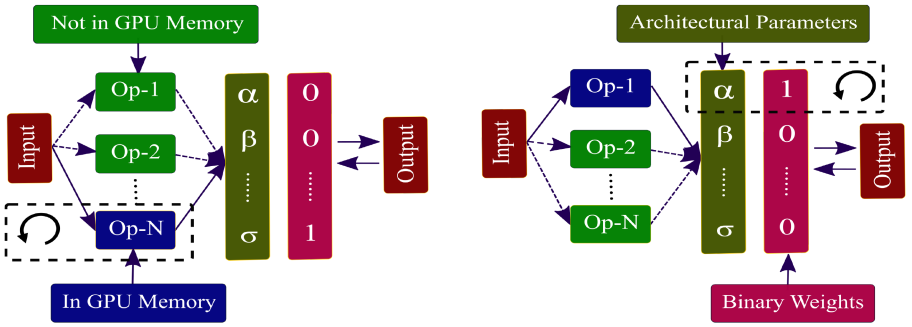
3.3.6 Differentiable Architecture Search (DARTS). A pioneering work, DARTS [86] significantly reduces the search time by formulating the NAS problem in a continuous differentiable manner, instead of a discrete RL-based method. It uses a softmax approach to relax the discrete space, allowing standard gradient descent to search for an architecture without the use of any RNN controllers. The key mechanism in this method is to construct a super-network, where each node is populated with all the operations (Convolution, Maxpool, etc.) in the search space to form a mixed operation set (Figure 6(a)). Architectural parameter (α), auxiliary to the Convolution weight, is defined for each operation in the mixed operation set (Figure 6(b)). The input feature map at the input node is broadcasted to all the operations in the mixed operation set, and the feature map at the output node is a weighted (softmax) sum of all input edges. If ($\alpha_1, \alpha_2, \ldots, \alpha_n$) correspond to ($\mathrm{Op}_1, \mathrm{Op}_2, \ldots, \mathrm{Op}_n$) operations, then the feature map at output node is given as per Equation (7).

$$O^{DARTS}(x) = \sum_{o \in O} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}^{(i,j)})} Op(x) \tag{7}$$

The weights (w^*) and alphas (α) of the supernet are alternatively updated by minimizing the training loss $\mathcal{L}_{train}$ and validation loss $\mathcal{L}_{val}$, respectively. The final neural architecture at the end of training the supernet (search process) is derived by sampling one edge at every node that has the highest α value $o^{(i,j)} = \mathrm{argmax}_{o \in O} \ \alpha_o^{(i,j)}$, as shown in Figures 6(c), 6(d).

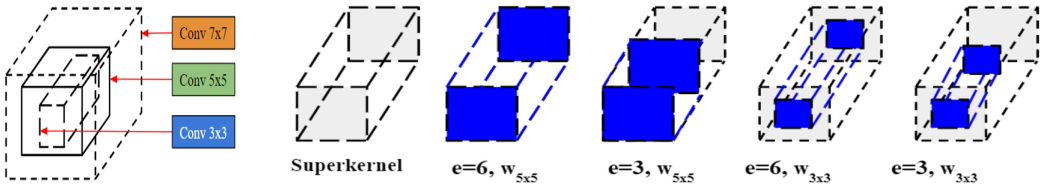
3.3.7 Differentiable Neural Architecture Search (DNAS). Similar to DARTS [86], Wu et al. proposed **DNAS** [132] to replace the softmax function with Gumbel-Softmax [59]. If op_i^l and α_i^l represent ith operation of layer l and the architecture parameter, respectively, then the output feature map at layer $l + 1$, for x^l input feature map, is given in Equation (8).

$$x^{l+1} = \sum_i u_i^l \cdot \mathrm{op}_i^l(x^l), u_i^l = \frac{\exp\left(\left(\alpha_i^l + g_i^l\right)/\tau\right)}{\sum_j \exp\left(\left(\alpha_j^l + g_j^l\right)/\tau\right)}, \tag{8}$$



(a) Weight Update (b) Architecture Parameter Update

Fig. 7. ProxylessNAS [8].



(a) Super Kernel (b) Kernel and Expansion Channel Encoding

Fig. 8. Single Path NAS [116].

where g_i^l is a random variable i.i.d. sampled from $Gumbel(0, 1)$ and τ is the temperature parameter. Gumbel-Softmax converts the non-differentiable sampling into a continuous differentiable sampling.

3.3.8 ProxylessNAS. The gradient-based differential search methods [86, 132] significantly reduce the search time compared to its predecessor (RL-based). However, it requires "N" times GPU memory (N stands for number of choices in the search space) for every edge of the super-network as it trains all the edges in the super-network, thereby bringing a significant computational burden on the GPU. **ProxylessNAS** [8] solves the GPU memory problem by binarizing the architecture parameters and forcing only one path among all the candidates to be active during the search process. Also, it finds an architecture directly on the target task with the actual latency constraints in a reasonable time (200 GPU hours). The DARTS method uses cell-based search to update all the architecture parameters by gradient descent at once, but ProxylessNAS updates only one edge in the layer-wise search by sampling only one path in the super-network. The architecture parameters $\{\alpha_i\}$ are transformed into binary gates. As shown in Figure 7(a), the architecture parameters are frozen, and binary gates are stochastically sampled for each batch of training steps to update the Convolutional weights. The weight parameters of the sampled active paths are updated using gradient descent on the training dataset. The architecture parameters are updated using the validation dataset by freezing the weights and resetting the binary gates, as shown in Figure 7(b). These steps are performed alternatively, and the final network is obtained by pruning the redundant paths.

3.3.9 Single-Path NAS. Even though ProxylessNAS [8] greatly reduces the search time, it still requires a large search space, thereby requiring a significant search time (200 GPU hours) on the ImageNet dataset. **Single-Path NAS (SPNAS)** [116] encodes all the architectural decisions (kernel size and expansion channel of an MBConv) within a shared super-kernel (Figure 8(a)). Hence, it

remarkably reduces the number of trainable parameters by having only a single path in a layer. Figure 8(b) illustrates the single shared weight, which encodes the decision of choosing a kernel between 3 or 5 and an expansion ratio between 3 or 6. SPNAS formulates the condition function where the L_2-norm of the weight is compared to a trained threshold that controls the decision, as per Equation (9).

$$\mathbf{w}_k = \mathbf{w}_{3\times3} + \mathbb{1}(\| \mathbf{w}_{5\times5\setminus3\times3} \|^2 > t_{k=5}) \cdot \mathbf{w}_{5\times5\setminus3\times3} \tag{9}$$

3.3.10 Differentiable HW-NAS. The computational demand of RL-based HW-NAS methods makes it very difficult to search for efficient networks. To find the architecture without any accuracy and performance restriction, each network sampled by the RNN controller needs to be trained to obtain accuracy and run on the target hardware to check the performance metric. However, the model that meets the predefined latency target must also be trained end-to-end to determine the accuracy. Differentiable HW-aware NAS adds the hardware metric directly to the loss function, thereby significantly decreasing the search time. The search problem can be formulated as multi-objective optimization (Equation (10)).

$$\min_{a\epsilon\mathcal{A}} \mathcal{L}_{val}(w^*, \alpha) + \lambda C(Lat(\alpha)), \tag{10}$$

$$s.t\ w^* = \arg\min_{w} \mathcal{L}_{train}(w, \alpha), \tag{11}$$

where w and α are the Convolutional weights and the architecture parameters, respectively. Given a supernet $\mathcal{A}$, the goal is to search for a sub-network $\alpha^* \epsilon\mathcal{A}$ that minimizes the validation loss $\mathcal{L}_{val}$ (w^*, α) and overall latency cost C(Lat(α)). The cost function is a operator latency-weighted sum of the architectural parameters (α), as given in Equation (12). Majority of the gradient HW-NAS methods use one of the following differentiable methods: softmax-based DARTS [86], Gumbel softmax-based DNAS [132], ProxylessNAS [8], Single Path NAS [116].

$$C(Lat(\alpha)) = \sum_{l}^{L} Lat(\alpha^l) = \sum_{l}^{L} \left(\alpha_1^l * Lat\left(\text{op}_1^l\right) + \alpha_2^l * \text{Lat}\left(\text{op}_2^l\right) + \cdots + \alpha_n^l * \text{Lat}\left(\text{op}_n^l\right)\right) \tag{12}$$

3.4 Network Accuracy

Validation accuracy is the primary component of any deep learning model on which several NAS methods try to optimize the network. During the search process, the accuracy can be obtained by training and validating the model or by using accuracy predictors from a set of pre-trained models.

3.4.1 Train and Validate. The most obvious method of obtaining the accuracy is to validate the trained network on the target dataset. It requires several architectures to be trained end-to-end.

3.4.2 Accuracy Prediction. As training and validating a sampled network (in RL methods) or super-networks (in gradient methods) is a time-consuming task, few works propose to predict the final accuracy by using an ML methodology instead of fully training them. Since the accuracy is calculated on the testing dataset, the predictor is independent of the hardware and can be re-used for the target device. Baker et al. [4] utilizes a radial basis to calculate the final validation accuracy of the models by training them only for 25% of their epochs. **PNAS** [85] employs a **Multi Layer Perceptron (MLP)** and an RNN to anticipate the improved accuracy of the new architecture. **Once-For-All** method [9] pre-trains an MLP on the dataset of a large number (16K) of randomly sampled networks from the search space to predict the accuracy. **ChamNet** [20] trains many models, but selects only a few highly accurate models with diverse performance efficiency (latency, energy) to train an ML model for prediction. Although the accuracy estimation through

an ML model could be very helpful in reducing the search time, the predicted accuracy of the network may not be as precise as the trained one. **HAO** [26] estimates the accuracy by stacking the operators in each layer and applying Support Vector Regressor.

3.5 Hardware Metrics

The accuracy evaluation of a network is an algorithmic step, while the hardware metric estimation such as latency or energy is dependent on the architecture and compiler of the target system. The NAS algorithms incorporate the actual hardware performance metric or hardware-agnostic proxy metrics such as FLOPs or number of weights. For example, number of MACs is used as a strong proxy for model latency in **μNAS** [80]. However, in practice, a CNN model with lower proxy metrics such as FLOPs may not be faster. For instance, NasNet [150] has a similar FLOP count as MobileNetV1 [51], but the former is slower than the latter. The performance metrics are either drawn from a pre-built look-up table from the hardware or estimated using a prediction model.

3.5.1 Look-up Table (LUT). An LUT consists of pre-collected cost of each operator in the search space on the target hardware. For example, FBNet [132], ProxylessNAS [8], and SPNAS [116] methods construct an LUT to hold runtime latencies of different operators on the target device.

3.5.2 Hardware Metric Prediction. The performance evaluation process of an architecture through direct measurement on hardware or fetching from LUT is an intensive procedure, as it requires us to execute all the operators on the target system. Hence, to alleviate the computational burden, a few ML-based regression models have been used to predict a model's performance. **ChamNet** [20] incorporates energy, accuracy, and latency predictors in the search process to effectively search for a CNN on the target system. A few NAS benchmarks [25, 115, 140] consisting of accuracy and latency measurements have been released to improve search efficiency. **BRP-NAS** [28] predicts the hardware performance using a **Graph Convolutional Network (GCN)**.

Lu et al. [90] demonstrate latency monotonicity among diverse hardware platforms and utilize only one proxy device's latency predictor for the search process instead of repeating the latency calculation on all the target devices. The models searched for one proxy device can be reused for new target hardware if there exists strong latency monotonicity (based on Spearman's Rank Correlation Coefficient). In the case of weak latency monotonicity, they propose an efficient transfer learning method to adapt the proxy device's latency predictor to the target hardware.

3.6 Searched Model

The final searched network indicates the neural architecture found after the exhaustive search process. Most of the search methods in the literature specialize the search algorithm for the target platform. For example, ProxylessNAS [8] specialized the models individually for CPU, GPU, and mobile devices. It showed that a model optimized for a device does not run better on other platforms and vice versa. However, this may or may not be true for all scenarios. Chu et al. [17] proposed a methodology to search for a single network that can guarantee optimal performance on a CPU, DSP, GPU, and EdgeTPU. They consider the hardware metrics in the reward function of the search process as the average or maximum of all the normalized metrics on the hardware platforms.

4 MICRO-CONTROLLER UNIT AWARE NEURAL ARCHITECTURE SEARCH

Deep Learning on tiny MCUs is attractive due to low cost, less power consumption, computing at the edge, and so on, and extremely challenging at the same time, as they have strict memory constraints. All the static data, such as the model's weight parameters and program code, must fit within the flash memory of MCU. A Convolution layer with 162 or 182 channels does not make

a significant difference on a large memory device like GPU, but the layer with a large channel size may exceed the memory constraint set by the MCU. The tiny CNN model should deliver the desired accuracy while satisfying the hardware constraints. Hence, the task of MCU-Aware NAS can be directly translated into memory-aware NAS with the upper bound set to the flash memory capacity.

4.1 MCU-specific

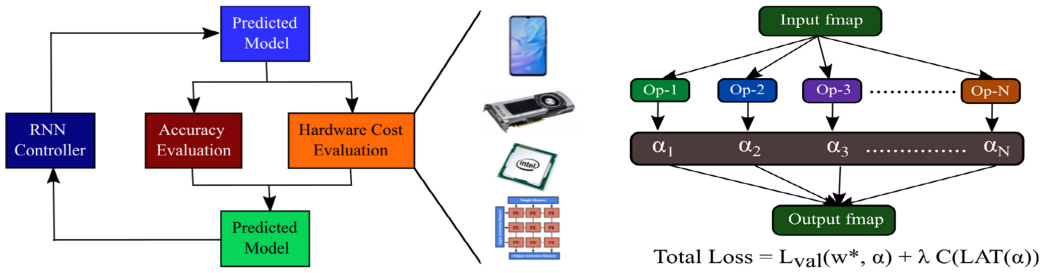
Lin et al. proposed **MCUNet** framework [81] that co-designs **TinyNAS**, a method to fit the network for the given memory constraint, and **TinyEngine**, a lightweight inference engine. The network-library pair enables large-scale deep learning on MCUs with 320 kB of memory and 1 MB of storage. TinyNAS optimizes the search space automatically to satisfy the resource restrictions, followed by an architecture search on the optimized search space. The input resolution and width multiplier of the MnasNet search space [119] are chosen from {48, 64, 80, . . . , 192, 208, 224} and {0.2, 0.3, 0.4, . . . , 1.0}, respectively. Evolution search is used, given the hard memory constraint, to search for the best network within the optimized search space. TinyEngine optimizes the memory scheduling of each sampled model to measure memory usage. MCUNet attains a speedup of 1.7–3.3× compared to CMSIS-NN [71] and TF-Lite Micro [21]. MCUNet achieves 70.7% accuracy on the ImageNet dataset [23], similar to ResNet18 [47] and MobileNetV2 [112] models.

Lin et al. proposed **MCUNetV2** framework [82] to co-optimize CNN architecture and inference scheduling. MCUNetV2 addresses the memory bottleneck by **(i)** a patch-by-patch execution for memory-intensive Convolutions to reduce the peak memory usage of networks by operating only on a sub-region of the featuremap, instead of the whole activation, **(ii)** receptive field redistribution to minimize the computation overhead by shifting the computation to the later stage of the model. MCUNetV2 achieves 71.8% ImageNet accuracy and > 90% on the visual wake words dataset.

MicroNets [5] leverages **Differentiable NAS (DNAS)** to search for models that meet the latency, memory, and energy constraints on an MCU. During the search process, the number of model **operations (ops)** is treated as a proxy for latency due to their linear relationship. The regularization term for latency is defined in the form of a decision vector z: $\sum_{i=1}^{N} z_N c_N$, where c_N is the operation count of the edge N in the supernetwork. Similar to latency, the memory size is regularized by $\sum_{k=1}^{K} z_k|\theta_k|$, where $|\theta_k|$ is the parameter size of operation k. MicroNets uses the MobileNetV2 [112] as a backbone for the Visual Wake Words task, and the width (between 10% and 100% of predefined width) of the first and last Convolution operation is searched in each MB-Conv. A modified MobileNetV1-search space [51] is used as a backbone for Keyword Spotting and Anomaly Detection tasks. The number of channels and the number of layers are searched for this backbone while satisfying the constraints. MicroNets attain SOTA results for all the three benchmark tasks.

Sparse Architecture Search (SpArSe) [33] integrates MCU-aware NAS with pruning to jointly optimize accuracy, model size, and the working memory size. SpArSe considers the network architecture and working memory of models as a **Multi-Objective Bayesian Optimizer (MOBO)** to eliminate the networks with the working memory size exceeding the memory handled by an MCU. It achieves high accuracy results on the standard small-memory vision datasets, such as MNIST, CIFAR10, and CUReT, on a search space consisting of a series of Convolution and max pool.

μNAS [80] targets validation accuracy, peak memory usage, model size, latency, and the processor speed for "mid-tier" MCUs. The number of MACs is used as a strong proxy for the model latency, which is confirmed by plotting the end-to-end model latency with the corresponding MACs of 1,000 random networks in the search space. Random scalarization [105] method is used to merge different objectives to form a single target function for optimization. A scalar term λ_i^t is linked to



(a) Reinforcement Learning-based HW-NAS (b) Gradient/Differentiable based HW-NAS

Fig. 9. Hardware-aware neural architecture search (HW-NAS) methods.

each of the four objectives, which states the relative importance of each objective. The **aging evolution (AE)** algorithm [109] is used in the search process by repeatedly evolving a set of network points. They also incorporate model compression in the search process by gradually pruning the weight groups using the L_2 norm of channels till the network reaches a predefined target sparsity level. μNAS can (i) improve accuracy by 4.8%, (ii) reduce the MACs by 2×, compared to the previous works (MCUNets, MicroNets, and SpArSe).

The intermittent inference is crucial on energy-harvesting lightweight MCUs, which refers to executing DNNs during each power cycle. **Intermittent-Aware NAS (iNAS)** [98] incorporates intermittent execution behavior into existing NAS frameworks to search for architectures with the intermittent execution designs that are feasible on the intermittently powered systems. The iNAS-searched models can reduce the intermittent DNN inference latency by 60% while achieving similar accuracy with a slight increase in search cost.

Apart from directly targeting the memory of an MCU, other works on embedded systems and IOT indirectly utilize tiny processor equivalent of an MCU. The NAS methods specific to embedded systems include **DeepMaker** [88] and **E-DNAS** [89]. The IoT-devices-aware NAS methods [13, 126] find a robust network to be deployed.

5 CPU-AWARE NEURAL ARCHITECTURE SEARCH (CPU-NAS)

5.1 Mobile/Edge CPUs

A highly accurate CNN model often comes with high computational cost and parameters, making it a challenge to deploy them on mobile devices where computing capacity and memory are limited. The early HW-NAS methods, specific to mobile CPUs, proved to be very successful and paved the way for other devices. The search methods can be divided into RL, Gradient, and Evolutionary.

5.1.1 RL-based Methods. **MnasNet** [119] was the first to implement mobile-aware NAS by incorporating network validation accuracy and hardware performance (latency) as a multi-objective reward function (Acc. x (Latency/Target)$^{\gamma}$) in the RL-based search process (Figure 9(a)). MnasNet constructs a latency LUT on the MobileNetV2 [112] backbone network to search directly on the ImageNet dataset. MnasNet [119] outperforms the hardware-agnostic NASNet [150] in terms of parameters (4.8× fewer), MACs (10× fewer), and accuracy (0.5% more). **MobileNetV3** [50] achieves a better accuracy-latency tradeoff on Pixel 1 than MobileNetV2 and MnasNet using the RL multi-objective strategy. It extends the MobileNetV2-search space by adding SE connections in the MB-Conv blocks and utilizes the NetAdapt algorithm [139] for fine-tuning the number of filters.

5.1.2 Gradient-based Methods. **ProxylessNAS-mobile** [8] (Section 3.3.8) incorporates the latency of each layer on Google Pixel 1 in the loss function as a function of architecture parameters

(Equation (10)). A latency prediction model is used to predict the latency of each operation during the search process (RMSE of 0.75 ms), whereas the actual latency on the mobile is used to report the searched model. ProxylessNAS-mobile directly searches on the ImageNet dataset and outperforms MobileNetV2 and MnasNet networks with 200 times less search cost on GPU.

Wu et al. proposed **FBNet** [132] to search for mobile-efficient CNNs on Samsung S8 and iPhoneX by using DNAS (Section 3.3.7). The Gumbel-Softmax [59] is used to optimize the architectural probability distribution better than the softmax on FBnet-search space (Figure 1(d)). The multi-objective loss function is given in Equation (13), where **CE** is the cross-entropy loss and LAT(a) is a function of latency and architectural parameters. An LUT is used to estimate the end-to-end latency by measuring the runtime of individual operators on the devices.

$$\mathcal{L}(a, w_a) = \text{CE}(a, w_a) \cdot \alpha \log(\text{LAT}(a))^{\beta}. \tag{13}$$

Single-Path NAS (SPNAS) [116] encodes all the MBConv search elements (kernel size and expansion channels) in the form of a shared super-kernel (Figure 8(a)). The hardware awareness in the search strategy is achieved by integrating a latency cost function in the final loss function (Equation (10)). SPNAS outperforms ProxylessNAS [8], MnasNet [119], and FBNet [132] in terms of accuracy (74.95%) at a significantly low search cost and latency of 80 ms on Pixel 1 phone.

5.1.3 Evolutionary Search Methods. **Once-For-All (OFA)** [9] trains a large over-parameterized network and samples different sizes from the supernet using the progressive shrinking algorithm to specialize the sub-network on different hardware platforms (Mobile, CPU, GPU, FPGA). The OFA-Large model achieves 80% top-1 accuracy on the ImageNet dataset and outperforms MobileNetV3 [50] on Samsung S7, Samsung Note8, Samsung Note10, Google Pixel 1, Google Pixel 2, and LG G8 mobile phones. The main advantage of OFA is that the supernetwork is trained only once, irrespective of the target hardware on which the sub-network is being deployed. However, the supernet training is computationally very expensive, requiring 1,200 GPU hours on V100.

HNAS [134] initially trains a supernetwork, consisting of all possible candidate operations. The probability of each operation in a layer is estimated using accuracy and latency predictor (measured on Pixel 3 using Tensorflow-Lite [21]). The operations whose probabilities fall below a predefined threshold are pruned layer-by-layer. The supernetwork is fine-tuned, and the cycle is repeated iteratively until the architecture satisfies a stopping criterion. For the same top-1 accuracy on ImageNet, HNAS outperforms FBNet and ProxylessNAS on the Pixel 3 mobile in terms of latency.

DONNA [100] divides the model into blocks and employs an accuracy predictor that is built using a **Blockwise Knowledge Distillation (BKD)** from a reference model to mimic the student model. The predictor enables the algorithm to search across diverse backbones and varying macro-architectural features. DONNA first builds a library of networks sampled from the search space and trains a linear model to predict the accuracy of a set of blocks in the evolutionary search process. DONNA searched model is 10% faster and 0.5% more accurate than MobileNetV2—1.4× on a Samsung S20.

Mills et al. [99] initially perform an extensive latency profiling of neural blocks/units in the Once-for-All (MobileNetV3), ProxylessNAS, and ResNet50 design spaces across multiple devices, such as Huawei Kirin 9000 NPU, Samsung Note10, and so on. They provide a few insights and methodologies to find Pareto-optimal models frontiers in terms of accuracy and latency by pruning the original search space. For example, MBConv3–3, MBConv3–7, and MBConv4–3 should be removed from search space to increase accuracy in the ProxylessNAS-GPU space. The pruned search space on the three design spaces provides better latency and accuracy values.

5.2 Desktop/High-end CPUs

CPU is a powerful and imperative hardware platform for running DNNs in various settings, such as laptops, desktops, mobile, and high-performance computing servers. CPUs are ideal for training DNN models with small datasets and deploying pre-trained models for real-time applications. **ProxylessNAS-CPU** [8] model is searched specific to the 2.40 GHz Intel Xeon CPU E5-2640 v4 by collecting data in the LUT. The searched MobileNetV2-like network is deep and prefers small and narrow MBConvs, as CPUs do not have high parallelism. ProxylessNAS-REINFORCE model for CPU consists of many zero operators in the initial layers.

EfficientNet [120] proposed three model scaling factors: (i) **width** represents the number of filters, (ii) **depth** indicates the total number of layers in the network, and (iii) **resolution** corresponds to the dimensions (height and width) of input image. The depth and width of a backbone network (MnasNet) is scaled up uniformly along with the input image size until the latency constraint is met (compound scaling), instead of directly searching a large model for the target latency. Multi-objective RL-NAS (Acc. x $(\text{FLOPS/Target})^{\gamma}$) is initially used to search for operators at each stage of the backbone network. Compound scaling is applied on the searched network (EfficientNet-B0) from previous step to produce series of models (B1, B2, B3, B4, B5, B6, B7). EfficientNet-B1 is more accurate and faster than ResNet-152 [47] on Intel Xeon CPU E5-2690.

LA-DNAS [136] trains a multi-layer perceptron-based **Latency Prediction Module (LPM)** to predict the latency of a network. The latency loss function is added to the final loss function in the gradient method on the DARTS-search space [86]. LA-DNAS randomly sampled 100K architectures (80K for training and 20K for inference) to measure latency on the Intel E5-1620 CPU.

HardCoRe-NAS [101] constructs a flexible search space consists of micro and macro space in a MobileNetV2-like network. The micro search controls the structure inside each block of a stage, and macro space specifies how to connect these blocks within each stage. It is a differentiable method on the EfficientNet search space [120] with strict latency constraints. For various latency levels on the Intel Xeon CPU, the HardCoRe-NAS-searched model outperforms OFA [9], EfficientNetB0 [120], and mobile models [50, 51, 112, 119] in terms of accuracy and latency.

DcaNAS [11] first presents practical guidelines for CNN execution on a Desktop CPU (x86), followed by designing SG blocks, a CPU-friendly module as the fundamental units. DcaNAS consists of three search elements: **(i) width**: the number of channels in each SG module; **(ii) depth**: the number of SG modules. A skip connection (1×1 DWConv) replaces the entire SG block if the skip connection is chosen in the search process; **(iii) Attention**: the decision to be made whether to include the SE connection in an SG block. GreedyNAS [142], a method that samples and trains only high-quality paths to improve the search efficiency, is used to shrink many redundant paths in the DcaNAS supernetwork. DcaNAS can search for a series of different SG modules that meet the latency constraints of a Desktop CPU, based on the sampled search space and the search elements.

6 GPU-AWARE NEURAL ARCHITECTURE SEARCH (GPU-NAS)

GPU is an important computing device utilized in many scientific fields due to its high performance and user-friendly programming environment. A typical GPU consists of thousands of cores to enable parallel computing. This section presents some of the GPU-aware NAS methods for efficient inference.

6.1 High-end/High-performance GPUs

The **ProxylessNAS-GPU** [8] model prefers a shallow and wide network, particularly in the initial layers where activation matrices have high resolution. Due to the high parallelism available on the GPU cards, the searched model retains large MBConv operations such as 7×7 MBConv6.

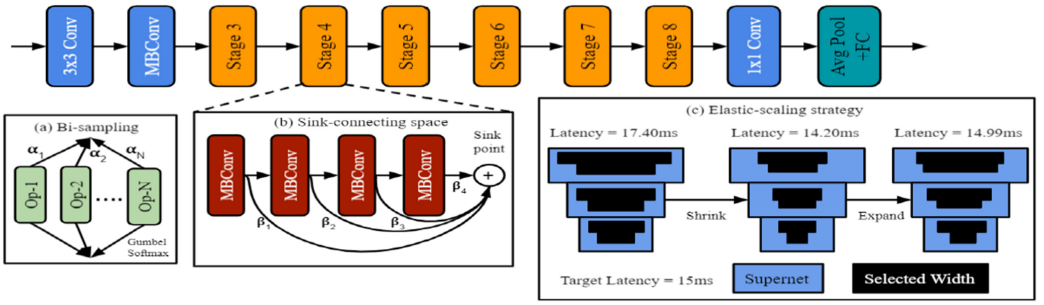


Fig. 10. The framework of TF-NAS [54].

The ProxylessNAS-GPU model attains 75.1% accuracy on ImageNet dataset (3.1% better than MobileNetV2) and 5.1 ms latency on V100 GPU (1.2× faster than MobileNetV2).

Tan et al. proposed **EfficientNetV2** [118] to enhance the training time and parameter efficiency compared to the previous works. They introduce "Training-aware NAS" to search for the best combination of the traditional MBConv (Figure 1(a)) and fused-MBConv (Figure 1(c)) layers using the RL-based multi-objective search method. It can be inferred from the searched model that fused-MBConvs offer better performance in the early stages than MBConvs, and small kernel size are efficient with more layers. EfficientNetV2 runs 3× faster than EfficientNet [120] on V100 GPU.

The **Single Path One-Shot (SPOS)** method [41] alleviates the training burden by constructing a simplified supernetwork in which all different architectures are single paths. SPOS can be applied to any differential search method and it outperforms the baseline search methods (FBNet, ProxylessNAS) on Titan XP GPU in terms of FLOPS and latency.

Fang et al. proposed **DenseNAS** [32], a differentiable method [86] to search for width (number of channels), depth (number of blocks), and spatial resolution on a densely connected search space [55]. A block denotes a series of operations where the output feature map has the same activation resolution and channels. The routing blocks in the network consist of accumulation (element-wise addition) of activations from the preceding blocks followed by basic/fundamental units in the search space. The proposed supernet in DenseNAS is a concatenation of densely connected routing blocks of different widths. They constructed an LUT to store the cost of each operation and also adds the latency cost (Equation (12)) to the final loss cost function (Equation (10)). DenseNAS achieves an accuracy of 75.3% on the ImageNet dataset with 17.9 ms latency on TITAN-XP GPU on MobileNetV2 search space, better than ProxylessNAS-GPU [8] and FBNet-C [132].

Three-Freedom NAS (TF-NAS) [54] operates on three degrees of freedom, namely, operation-level, depth-level, and width-level on EfficientNet search space [120]. At **Operation Level**, a bi-sampling search algorithm (Figure 10(a)) is used to moderate the operation collapse. At **Depth Level**, a sink-connecting search space ensures the mutual exclusion between skip connection and other operations in the predefined space (Figure 10(b)). At **Width Level**, a progressive fine-grained elastic scaling method obtains a network with precise latency constraint (Figure 10(c)). TF-NAS achieves 76.9% top-1 accuracy on ImageNet dataset with 18.03 ms latency on Titan RTX GPU, better than EfficientNet-B0 [120] and ProxylessNAS-GPU [8].

UNAS [123] is a unified framework for NAS that combines the RL and gradient-based methods under a single framework. UNAS brings the best of both worlds by enabling NAS to search for differentiable and non-differentiable elements. Also, it introduces an objective function on the generalization gap to stop selecting the networks that are often susceptible to overfitting. The core UNAS method is applied on the DARTS (cell-based) and ProxylessNAS (MobileNetV2 layer-wise)

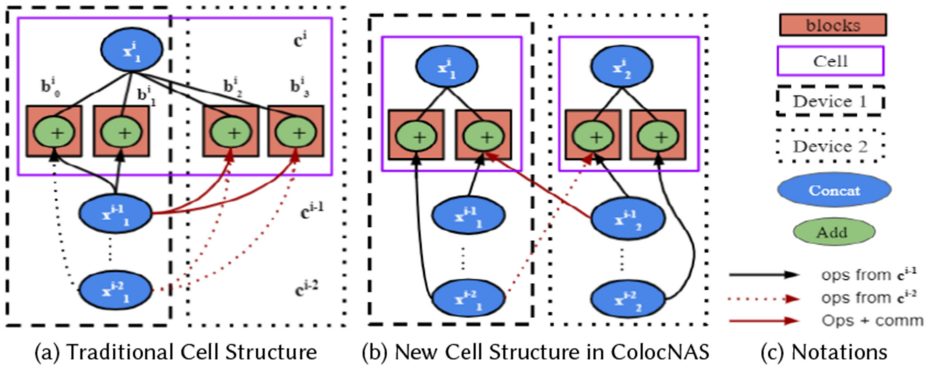


Fig. 11. Comparison of traditional cell structure with the new cell in ColocNAS [35].

search space. The searched network of UNAS is more accurate and faster on Nvidia V100 than the ProxylessNAS-GPU [8] model even though the former has three more MBConv blocks compared to the latter (ProxylessNAS-GPU) model. **Device-aware Progressive Search (DPPNet)** utilizes the Progressive NAS method [85] on mobile network search space (Depthwise Convolution) to search for efficient models on a workstation with Titan X GPU and Nvidia Jetson embedded system.

6.1.1 Energy-efficient Models on GPU. **MONAS** [52] uses a controller-based RL methodology to design energy-efficient CNNs on GPUs. The accuracy and energy consumption are included in the multi-objective reward function, as given in Equations (3)–(5). Nvprof, an Nvidia profiling tool, is used to measure the peak and average power. Young et al. [143] utilized an evolutionary search approach to optimize the hyperparameters and depth of a CNN. Experiments on CIFAR-10, CIFAR-100 datasets have shown that the searched networks produce good accuracy and energy per inference tradeoff.

6.1.2 Multi-GPU Platform. **Multi-GPUs** are widely used in datacenters to speed up the DNN training process. The two kinds of parallelism while training/inference of a neural network on a multiple GPU platform are model and data parallelism. Fu et al. [35] identified the following problems while implementing the layer-based and the traditional cell-based backbone network on a multi-GPU platform: The **Layer-wise Network** has a chain-like structure where computation in a layer is initiated only after receiving outputs from the previous layer. Hence, this data dependency causes difficulty in the parallel execution of the network across multiple devices. The traditional **Cell-based** structure has inherent parallelism due to the presence of various independent paths inside the cell, as shown in Figure 11(a). The concatenation operation at the end of the cell acts as a synchronization unit that gathers outputs from all the intermediate nodes within the cell. Hence, this unit hinders further parallelism, as it incurs severe communication overhead between the devices (red arrow in Figure 11(a)). **ColocNAS** [35] re-designs the traditional cell-based layout to form a "new search space" that incurs less communication overhead among the available GPU devices by exploring the connectivity patterns (Figure 11(b)). ColocNAS is 1.16× faster than ProxylessNAS-GPU on an Nvidia Tesla K80 GPU for the same accuracy of 74.6%.

6.2 Edge/Mobile GPUs

Luo et al. proposed **EdgeNAS** [93] to search efficient networks for edge GPU devices using an MLP-based latency predictor. The data for the predictor is obtained by measuring latency and FLOPs on the target edge device of randomly sampled 100,000 models from the DARTS-search space. The

latency estimator is directly incorporated into the framework to guide the search process. EdgeNAS generates accuracy-latency tradeoff networks for CIFAR datasets on Nvidia Jetson Xavier.

Mobile GPU-Aware (MoGA) NAS [18] considers accuracy, latency, and the number of parameters to design efficient models for mobile GPUs. NSGA-II [22] solves the weighted fitness function in the multi-objective optimization space. MoGA is built on the MobileNetV3 [50] backbone network, with three dimensions in the MBConv search space: **(i)** (3, 5, 7) kernel size, **(ii)** (3, 6) expansion ratio, and **(iii)** whether to enable SE attention mechanism. The NAS pipeline consists of a supernet, trained by the FairNAS method [19] for fast accuracy evaluation, a GPU latency LUT. MoGA attains 75.3% accuracy on ImageNet, which is 0.3% more accurate than MobileNetV3 with the similar number of parameters and FLOPs on the mobile GPU setting (Qualcomm and Xiaomi).

Partial Order Pruning (POP) [78] filters out the networks that are unlikely to provide a good speed-accuracy tradeoff on the target dataset and device based on a partial order assumption. It assumes that the narrower model is more efficient and produces less accuracy than a wider network. POP considers accuracy and inference latency as a multi-objective function and uses a cutting plane algorithm to solve the search optimization problem, instead of RL, AE, and gradient methods. It iteratively searches for the models with the best accuracy/latency tradeoff in the enlarged search space and the pruned search space is updated every time POP trains a new network. POP also builds an LUT using TensorRT, which is used to guide the search space pruning. POP outperforms ProxylessNAS-GPU in terms of accuracy on ImageNet and latency on Nvidia Jetson TX2.

Lyu et al. [94] proposed a ProxylessNAS-based search methodology on an improved MobileNetV2 space to search for efficient Pareto-optimal models on edge GPUs. The model searched with respect to Nvidia Jetson NANO GPU is 1.2× faster than ProxylessNAS, with accuracy being slightly lower. Ipenburg et al. [124] search for a cell architecture to accelerate the searched model on NVIDIA Jetson Nano GPU. They use **Stochastic Neural Architecture Search (SNAS)** to optimize various metrics, namely, accuracy, number of parameters, latency, and power. The searched networks on the CIFAR-10 dataset reduce power and latency on the target device while maintaining accuracy.

Lu et al. [91] utilize RL controller-based multi-objective search algorithm to search on variety of Edge-cloud hardware platforms that includes **(i) Edge Device:** An Nvidia Jetson Nano with a Quad-Core ARM Cortex-A57 CPU, **(ii) Edge Server:** A MacBook Pro with Intel 6 Core i7@2.6 GHz CPU, **(iii) Cloud Server:** 8 2.5 GHz Intel Xeon Platinum CPUs and NVIDIA V100 GPU with 32 GB graphic memory. Experiments on real-world time series forecasting for sensor dataset on **Temporal Convolutional Network (TCN)** search space show an improved parameter efficiency of the searched model compared to an LSTM.

7 ACCELERATOR-AWARE NEURAL ARCHITECTURE SEARCH (ACCELERATOR-NAS)

7.1 Application-Specific Integrated Circuit (ASIC)

Due to the slowing down on the progress of Moore's law, domain-specific hardware have been developed to keep up with the ever-increasing size of modern DNNs. A typical DNN accelerator such as a **Tensor Processing Unit (TPU)** [65] consists of the necessary logic (Matrix Multiplication, pooling, ReLU unit) and control (dataflow of weights and activations) on the chip. The architecture comprises an array of **Processing Elements (PEs)** to perform matrix multiplication, global buffers to store the input/output weights and activations and a **Network-on-Chip (NoC)** for data movement.

APNAS [2] is an ENAS-based [106] search method (Figure 5, Figure 9(a)) to design efficient models for ASICs with arrays sizes of 8×8 and 16×16 on CIFAR dataset [69]. APNAS employs an analytical model to estimate the processing time on the array by considering varying sized

Convolutional layers and array sizes. The multi-objective reward function (R_{AP}) of model m, consisting of both accuracy (A_m) and normalized cycle count (C_{m_norm}) is given as $R_{AP}(m; \omega) = A_m - \alpha * C_{m\_norm}$.

Array Aware NAS (AANAS) [14] co-designs the search space with respect to the underlying systolic array size of the accelerator. The number of expansion channels of each MBConv block is chosen as a perfect multiple of the array size such that the Convolution layer can fully utilize the array. Experiments on varying array sizes on TPU and Eyeriss show that the searched networks have better accuracy-latency tradeoff compared to MobileNetV2 on the CIFAR-10 dataset.

Gupta et al. proposed **Accelerator Aware NAS** [42] targeting EdgeTPUs to design CNN models for the ImageNet dataset. They showed that a Fused MBConv (Figure 1(c)) runs faster than MBConv block, depending on the input, output, and expansion channels on an EdgeTPU, although the former block (Fused MBConv) has more parameters and MAC operations. For example, a fused MBConv block with 3×3 standard Convolution runs 1.4× faster than the MBConv layer on an input activation of (28, 28, 32) to produce (28, 28, 256). This is because the Depthwise Convolution operation does not fully utilize the systolic array. The Edge TPU accelerator-aware NAS on EfficientNet backbone outperforms Resnet50 [47] in Coral devices, and the MobileNet backbone outperforms MobileNetV3 in Pixel 4 in terms of accuracy and latency by using an RL-based search method. The principles of Accelerator-aware NAS and the searched model for Google TPU has been integrated into the recently released Pixel 6 phone [43].

EfficientNet-X [74] method designs CNN for datacenter-accelerators such as TPU and tensorcore enabled-GPUs by using **latency-aware compound scaling (LACS)**, a method that combines latency and compound scaling. The authors define three accelerator-friendly improvements on the EfficientNet-search space [120] to suit the accelerator requirements: **(i) Space-to-depth:** Customize nxn Convolution with stride "n" to reshape the HxWxC tensor to H/n x W/n x $C*n^2$, **(ii) Fused Convolution:** a block-wise search for MBConv [112] and Fused-MBConv [42], **(iii) Activation Function:** a block-wise search between ReLU and swish. For the same accuracy, the EfficientNet-X series networks are twice faster than EfficientNet series on TPUV3 and V100 GPU accelerators.

S^3NAS [73] is an NPU-based search method consisting of three steps: **(i)** a MobileNetV2-based super-network design by assigning different number of blocks in each stage (specifically, more blocks to the stages with large width to decrease latency) and using MixConv approach [117]; **(ii)** a modified Single-Path Search [116] (Section 3.3.9); **(iii)** Compound Scaling [120], to scale up the model, followed by adding an SE connection and an h-swish activation function. The latency loss function is designed in such a way that it is activated only if the target latency is smaller than the estimated latency to ease the search process. The experiments on MIDAP NPU [67] show that S^3NAS-searched model is twice faster and 2.1% more accurate than EfficientNet-EdgeTPU [42].

EH-DNAS [60] is an end-to-end HW-aware Differentiable NAS algorithm on customized accelerators where the hardware cost function is approximate from a learned DNN instead of linear approximations of metrics. A dataset consisting of architectures sampled from the search space and corresponding latencies is gathered by measuring on the target hardware. Experiments on DARTS and FBNet search space combined with the learned loss show improved latency values on CIFAR-10 and ImageNet dataset for a wide range of platforms.

FLASH [76] makes remarkable progress in terms of the search algorithm to reduce the search time significantly and targets **In-memory Computing (IMC)**-based hardware accelerator. The authors initially propose NN-Degree, an analytical metric to quantify the architectural characteristics of networks with skip connections. The accuracy predictor is built with as low as 25 samples from the search space, which consists of 63 billion configurations. NN-Degree also allows to do NAS without any training and can search for a network 0.11 seconds on the CIFAR-10 dataset.

7.2 Field Programmable Gate Array (FPGA)

FPGA is a reconfigurable hardware that enables system designers to incorporate the necessary logic in accordance with the target DNN algorithm [40]. The specialized hardware is easy to program and can achieve high parallelism, speed, performance, and power consumption.

FNAS [61] is an RL-based FPGA-aware NAS method (Figure 9(a)) to search an architecture for the given inference latency. It uses a performance abstraction model to estimate the latency of neural networks without training. FNAS prunes networks that do not satisfy the constraints in the search space, thereby increasing the search efficiency by a factor of 11.13×.

Zeng et al. [146] initially considers elements from Vgg, ResNet and MobileNet networks in an enlarged search space. They apply **Black-box Profiling** tuning method to obtain a small and dynamic elements from the large space based on the accuracy of the network and latency specifications of the target accelerator (FPGA). The RL-NAS or DNAS can be applied on the reduced search space to design efficient models for ISA-based FPGA accelerator.

Fan et al. [31] first propose a novel FPGA architecture and employ DNAS method (Section 3.3.7) to search for efficient CNN using AnyNet block (a Convolution followed by a Group Convolution) as backbone [108] on the optimized FPGA hardware. The search dimensions are {1, 3} kernels sizes in both the Convolutions and {1, 2, 4} group number in Group Convolution. The searched is 9.65% accurate and 12.4 times faster than the searched network in the previous work [63].

7.3 Vision Processing Unit (VPU) and Digital Signal Processor (DSP)

Intel Myriad VPU [58] is a parallel vector processor with a power budget of 1W [111]. The term vision is used in VPU as the device specifically targets the acceleration of computer vision applications. CNN computation can also be performed on other types of accelerators such as a DSP.

Donegan et al. proposed MyriadX VPU-aware NAS [24] by utilizing the ProxylessNAS [8] method on MobileNetV2 search space. A latency LUT is constructed for every operation in the search space, and a latency-aware cost function is added to guide the search process. The VPU-aware searched CNN model is 1% more accurate and 2.6 ms faster than the MobileNetV2.

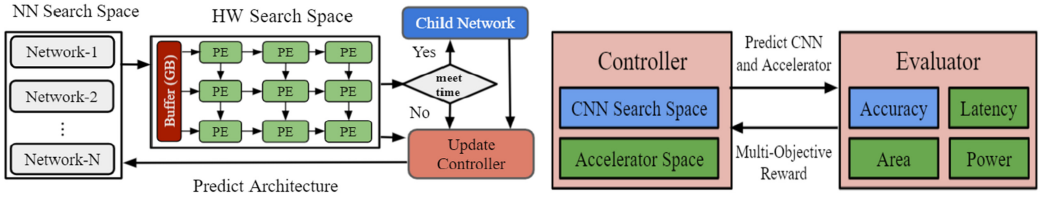
HURRICANE [147] first constructs an enlarged search space of 32 operators, consisting of the fundamental units from MobileNetV1 [51], MobileNetV2 [112], MobileNetV3 [50], and ShuffleNetV2 [95] networks. A subset of blocks (4–5) is sampled layer-by-layer from the whole space, depending on the hardware performance and FLOP count. HURRICANE applies One-shot NAS [41] on the reduced HW-aware search space, and the VPU- and DSP-searched network is 1.83% and 1.19% accurate than ProxylessNAS-R [8] for almost (or better) the same latency.

7.4 Resistive Random Access Memory (ReRAM) crossbars

ReRAM accelerators can execute high-performance DNN inference, as the crossbars can perform analog matrix multiplication at low latency and energy cost. Yuan et al. proposed **NAS4RRAM** [145] to search for optimal models for the given ReRAM accelerator. ResNet [47] is used as the backbone network to search for the following dimensions: **(i)** Number of layers and **(ii)** Number of channels varying from 16 to 64 in the steps of 4. NAS4RRAM outperforms the accuracy of ResNet using evolutionary algorithm on the CIFAR dataset.

8 CO-DESIGN OF HARDWARE ACCELERATOR AND CNN ALGORITHM

The hardware optimization is coupled with the neural architecture, and the network design depends on the underlying hardware (HW-NAS). A simple heuristic to search for a model with given latency and energy constraints on the target hardware would be sub-optimal and vice versa. Therefore, jointly exploring both the spaces achieves hardware efficiency and accuracy for a better



(a) RL-based Co-design [63] (b) Codesign-NAS [1]

Fig. 12. Reinforcement learning-based network and accelerator architecture co-search methods.

design tradeoff. Neural Network architecture and hardware accelerator co-search is a great way to achieve specialization and acceleration for DNNs to search for the optimal solution. This section focuses on such CNN model and accelerator (hardware architecture or compiler mapping) co-design methods to achieve the best accuracy in algorithm domain and performance in hardware space.

A typical CNN search space consists of a single dimension, i.e., the algorithmic specifications such as the number of filters, depth, kernel size, and so on. However, the search space for hardware accelerator can be divided into the following two types: **(1) Accelerator Search Space:** It includes the physical properties or micro-architecture of an ASIC, FPGA, and ReRAM such as computing array size, buffer size, PE inter-connection, PE register size, and so on. **(2) Compiler Mapping Search Space:** It consists of implementation routines for a DNN accelerator such as loop order, mapping, parallelization, tiling sizes, and so on. For general-purpose devices, such as CPUs and GPUs, compiler search space include kernel fusion, memory access optimization, and so on. Co-searching DNN algorithm and CPU/GPU physical hardware specifications like cache/RAM size is not viable, because these devices are general-purpose devices and are used for a variety of tasks. MCUNet [81] and MCUNetV2 [82] can be characterized as an algorithm-compiler co-search method on a fixed MCU.

8.1 Field Programmable Gate Array (FPGA)

A large number of initial works targeted neural and FPGA architecture co-search due to the high flexibility offered by the hardware. In a typical RL co-search method, the controller predicts a network-hardware pair from the joint search space. The generated network is trained to capture the accuracy and performance cost on the predicted accelerator design. The controller is updated based on the accuracy and hardware cost as inputs, and the next set of candidates is generated.

Jiang et al. [63] employed a two-level RL-based co-exploration framework on the MobileNetV2 backbone. The first fast phase tweaks the hyperparameters and removes the network that does not meet the hardware specifications (Figure 12(a)). The second slow phase trains the predicted network to obtain accuracy and optimizes the RNN controller to predict a better neural and hardware architecture. The results from the experiments on CIFAR-10 and ImageNet show that the co-searched network and hardware outperforms HW-NAS and manual FPGA-optimized hardware.

HotNAS [64] co-explores the joint space from a hot state (set of pre-trained networks: VGGNet, ResNet, Mobilenet, MNASNet) instead of a cold state (from scratch). The pre-trained models that violate the latency constraints are customized to the underlying hardware. This framework integrates model compression techniques into the search process. Also, the quantization of weight parameters reduces the weight loading time. HotNAS reduces the search time from 200 to 3 GPU Hours and produces a 5.79% gain in accuracy on ImageNet compared to the existing methods.

Codesign-NAS [1] combines network latency and accelerator area into a single constraint of **performance-per-area (perf/area)** and treats it as a threshold value. An RL-controller

(Figure 12(b)) attempts to search for a CNN on the NASBench [140] search space and CHaiDNN [57] accelerator space that maximizes the validation accuracy under the combined constraint.

Hao et al. [44] proposed a co-search methodology consisting of **(i) Bundle-Arch**, a fundamental unit/bundle composed of a Convolution (1×1 or 3×3 or 5×5), followed by Depthwise (3×3 or 5×5 or 7×7), **(ii) Auto-DNN** for HW-NAS on the bundle-based backbone network, **(iii) Tile-Arch**, an FPGA template for DNN implementation, and **(iv) Auto-HLS** engine to generate the accelerator (FPGA) code for the searched networks. The searched CNN-accelerator pair using stochastic coordinate descent on PYNQ-Z1 embedded FPGA outperforms the previous works in terms of IoU, FPS, power, and energy efficiency on the object detection task.

8.2 Application-Specific Integrated Circuit (ASIC)

Yang et al. proposed **ASICNAS** [138], an RL-based method to co-search for neural architectures and the associated heterogeneous ASIC accelerator design. The accelerator search includes resource allocation to the sub-accelerators and dataflow for each of the sub-accelerators from the following set of dataflows: Shidiannao [27], NVDLA [103], and row-stationary [12]. ASICNAS outperforms NAS→ASIC (successive NAS and ASIC design) by 17.77%, 2.49×, and 2.32× reduction on latency, energy, and area, respectively, on CIFAR-10.

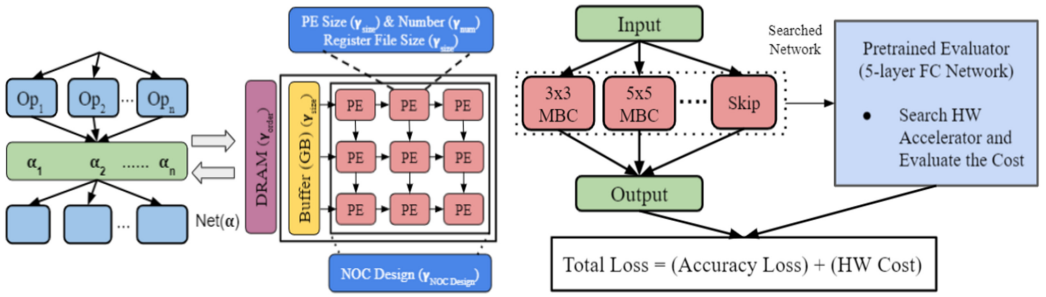
NAHAS [149] targets co-design of EdgeTPUs and CNNs on the EfficientNet [120] search space with compound scaling and the search elements being kernel size and expansion ratios of MBConv. The evolved search space for EfficientNet-search space [120] consists of a Fused-MBConv for initial layers and MBConv for all the other layers, as the Fused-MBConv becomes less efficient in later stages of the network [42]. The edge accelerator is an industry standard device with search components of a vanilla TPU accelerator such as array dimensions, SIMD units, local memory, and register file size. The weighted reward function is a product of network accuracy, model latency, and total area of the accelerator, as given in Equation (14).

$$\max_{\alpha,h} Accuracy(\alpha, h) \left(\frac{Latency(\alpha, h)}{T_{latency}}\right)^{w_0} \left(\frac{Area(h)}{T_{area}}\right)^{w_1} \tag{14}$$

The RL-based methods can be replaced by the differentiable co-search techniques, as the former require large search time. **DNA** [141] allows differentiable co-search of the neural architecture and discrete accelerator specifications in a single iteration. The supernetwork is built using the Gumbel Softmax-based DNAS (Section 3.3.7) on the FBNet [132] search space elements (Figure 13(a)). The main challenge is to obtain the HW-cost (L_{hw}) in the multi-objective function (Equation (15)), as the network loss L_{val} can be obtained from the supernetwork. At each search iteration, M architectures are sampled from the supernetwork NET(α). The optimal hardware for each network is obtained by creating candidate accelerator designs beforehand and selecting them using Gumbel softmax. The HW-loss (L_{hw}) is approximately obtained using the weighted sum of cost of layer-wise operators. DNA outperforms all the previous RL-search methods in terms of search time and accuracy.

$$\min_{\alpha} \; L_{val}(\omega^*, NET(\alpha)) + \lambda L_{hw}(NET(\alpha), HW(\gamma^*)) \tag{15}$$

DANCE [15] consists of **(i)** a ProxylessNAS-based supernetwork, **(ii)** an MLP-based hardware-generator and performance-evaluator (Figure 13(b)). The MLP is pretrained on the dataset generated by randomly sampling 50K models in the neural search space and measuring the desired cost metric (latency, area, energy) on the accelerator search space. During the search process, the predicted network from the supernetwork is given as an input to the pretrained generator to predict accelerator design and estimate the corresponding cost. The losses from the supernetwork and evaluator are combined to guide the joint search process, as the supernetwork and MLP are both differentiable with respect to gradient descent. The accelerator search space includes systolic



(a) DNA [141] (b) DANCE [15]

Fig. 13. Gradient (Differentiable) network and accelerator architecture co-search methods.

array dimensions and size of the register file, and the MobileNetV2 elements in the neural search space. The results from the experiments show that DANCE can outperform previous RL methods [1, 92, 138] in terms of accuracy on CIFAR-10 with significant improvement in search time.

NAAS [84] co-explores micro-architecture specifications, compiler mapping routines and neural architecture components (on ResNet50 [47] search space) in a single optimization loop using evolutionary search strategy. The searched accelerator-CNN pair reduces the Energy-Delay Product (by a factor of 4.4×) and improves accuracy by 2.7% compared to Eyeriss-ResNet50 implementation.

8.2.1 Graph Neural Networks (GNNs). GNNs are a class of networks that achieved great success solving graph-based problems such as node classification, recommendation systems, and so on. **GN-NAS** [97] is a technique to co-search GNN Architecture and **Network-on-Chip (NoC)** Accelerator using an RL-based controller on a GNN-specific search space. **G-CoS** [148] is a GNN co-search framework for network structure and accelerator architecture. G-CoS can search for the matched GNN structures and hardware accelerators by **(i)** defining a generic GNN accelerator search space that is suitable to numerous GNN structures and **(ii)** using a one-shot GNN and accelerator co-search method to efficiently search for the GNN structures and their accelerators.

8.3 Resistive Random Access Memory (ReRAM) crossbars

NAX [102] is a ProxylessNAS-based search method to co-design the crossbar size and neural hyperparamters under non-ideal circumstances. The co-design of array and kernel size is essential in each layer, as the accuracy of a CNN model decreases with an increase in the crossbar size due to the errors induced by the analog computation. The neural-crossbar joint search space for Tiny ImageNet consists of $\{3 \times 3, 5 \times 5\}$ kernel and $\{128 \times 128, 64 \times 64\}$ crossbar sizes on ResNet50 backbone. The branches in the supernet are composed of all the combinations of kernel and array size in the search space. The searched network 4% lower energy-delay-area product compared to ResNet-18.

9 SEARCHING FOR PRECISION

Network Quantization, a method to reduce the precision (bit-width) of CNN parameters, has shown tremendous success in reducing the memory cost and increasing inference efficiency on widely available hardware platforms. Many studies have shown that the CNN model is insensitive to quantizing it from floating-point to fixed-point representation. **Uniform Quantization (UQ)** is a method of adopting the same bit-width (global precision) across all the layers (weights and activations) of a CNN model. However, **Mixed Precision Quantization (MPQ)** assigns different bit-widths (e.g., 1, 2, 4, . . .) to different weight and activation parameters within a network and has achieved a remarkable improvement over the former method.

Due to the success of MPQ, several DNN accelerators such as Nvidia Turing GPU [7], BitFusion [113], Stripes [66], and BISMO [122] started offering support to mixed precision (1–8 bits) multiplication to improve the computation efficiency. The problem lies in deciding the optimal bit-width of each layer in the CNN model such that the computation cost is reduced through quantization while preserving the accuracy. For a CNN consisting of "L" layers, and "N" bit-widths to choose from, there exists N^L different configurations. It is practically impossible to try out every possible combination to find the optimal bit-width of every layer. Hence, the Mixed Precision Quantization can be formulated as a search problem, thereby applying the same principles and methodologies of various NAS algorithms.

9.1 Mixed Precision Quantization Search

Mixed Precision Quantization Search is a technique to search for different precision in each layer on a fixed backbone network (e.g., Mobilenet, Resnet18), with/without considering the underlying fixed hardware. The early pioneering works include **Hardware-Aware Automated Quantization (HAQ)** method [127] for layer-wise precision search, specific to the underlying hardware accelerators (BitFusion [113] and BISMO [122]). HAQ utilizes RL-based **Deep Deterministic Policy Gradient (DDPG)**, consisting of an actor-critic algorithm (Figure 14(a)) to optimize metrics such as latency, energy, and model size. The authors [127] incorporated the accelerator's feedback (metrics) directly into the RL-agent design loop using a hardware simulator. HAQ can reduce the latency by 1.4× to 1.9× with negligible accuracy loss compared to the Uniform 8-bit Quantization. Similar to specializing the architecture search to a specific hardware, the quantization policy tuned towards a specific accelerator may not be optimal on the other hardware.

Releq [30] also automates the bit allocation process using an LSTM-based RL method, where the reward function gives importance to the accuracy over network quantization. ReLeQ achieves 2.2× and 1.22× speedup over the 8-bit uniform quantization on CPU and Stripes [66], respectively. The differences between Releq and HAQ [127] are: **(1)** The ReLeQ method quantizes only the weight parameters of a network, while HAQ quantizes the weight and activation matrices of a single layer with different precisions. **(2)** ReLeQ has a wide range of bits (1,2, . . . , 8) in the search space, while HAQ consists of a limited space (2, 4, 6, 8), as supported by the underlying accelerator.

Huang et al. proposed an RL-based mixed-precision quantization method for ReRAM-based accelerators [56] that jointly targets weight, input, and the accumulated partial sum quantization (not a concern in the digital accelerators). The partial sum quantization is obtained by reducing the precision of ADCs at the array outputs. The cost function is the difference between the loss of quantized and full precision network, unlike the validation accuracy of the quantized model.

RaQu [107] proposed "array-aware quantization" on a ReRAM accelerator to automatically search for the same bit-width in the weight and activation matrix, and "array-wise grouping" to cluster a set of kernels in each layer such that weights in the same cluster share the same precision. A DDPG agent selects the precision of every layer and the group to be mapped on a ReRAM.

Gradient-based Precision Search Methods: Similar to the conventional RL-based architecture search methods, the mixed-precision search methods using RL are expensive, as the new quantized model is retrained for every new bit-width configuration. Hence, the discrete bit-width search space can be relaxed to a continuous space using gradient descent. Wu et al. [133] formulated the mixed-precision search problem using gradient-method (DNAS) to explore the search space consisting of "N" bit-widths. A stochastic supernet is constructed on a pretrained backbone network (e.g., ResNet20) with N paths at each layer, as shown in Figure 14(b). Each path in the supernet is a Convolution operation between quantized weight and activation with a unique precision. A differentiable sampler is used to sample the child architecture in every forward pass using Gumbel Softmax [59]. It also incorporates a cost function based on the number of FLOPS and bit widths in

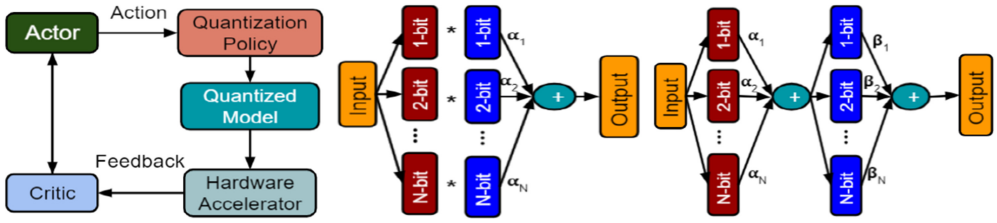


Fig. 14. Mixed precision quantization search algorithms: RL (HAQ) and gradient-based (DNAS & EdMIPS).

the loss function to maintain a balance between accuracy and efficiency. Gradient descent can be applied to select a path by alternatively updating weight and architectural parameters.

EdMIPS [10] alleviates the search burden in the supernetwork training of MPQS. The number of paths in the supernetwork proposed by Wu et al. [133] increases with an increase in the number of bit-width choices, thereby requiring one Convolution operation for each path. EdMIPS converts the parallel paths to a single Convolution, parameterized by the softmax-weighted sum of quantized weights (α) and activations (β). Therefore, there exists only one Convolution operation (Figure 14(c)), irrespective of the number of bit-width choices.

9.2 Architecture and Mixed Precision Quantization Co-search

Gong et al. [39] proposed an end-to-end co-optimization of neural architecture and precision in each layer to achieve good accuracy for the mixed-precision quantized network and energy consumption on the mixed-precision quantization supporting accelerator (BitFusion [113]). The dimensions of the architecture search space on MobileNetV2 backbone are kernel size {3, 5, 7} and expansion ratio {1, 3, 6}. The depthwise and bottleneck layers within a single MBConv block are assigned with different precisions within {2, 4, 6, 8}. The algorithm employs a Gumbel-Softmax [59] for the differentiable search method to find an optimal configuration of neural architecture and precision. The searched network outperforms HAQ [127] in terms of accuracy (%0.87 more accurate), energy (1.5× lower), and latency (1.6× faster). **Uncertainty Aware sEarch (UAE)** [137] is an RL-based technique to jointly search for the number of channels, filter dimensions, and bit-widths on Computing-in-Memory NN accelerators such as a ReRAM under an uncertain environment.

9.3 Neural Architecture, Precision, and Hardware Specifications Co-search

The optimal DNN execution on the accelerator requires a co-design of three aspects: neural architecture, precision in each layer, and hardware specifications. The co-exploration of any two dimensions together leads to sub-optimal performance in both the algorithm and hardware spectrum.

Efficient Differentiable DNN (EDD) [79] is a fully differentiable, single-path NAS-based [116] co-search method to co-explore neural architecture (on MobileNetV2), layer-wise precision, and hardware implementation. GPU implementation [45, 46] search space includes kernel fusion and optimizing memory accesses, whereas FPGA implementation search space consists of parallel factor and loop tiling. They use the Gumbel softmax [59] to sample non-differentiable components for both architecture hyperparameters (kernel size and expansion channels) and bit widths. The quantization search space for GPU includes 8, 16, and 32 bits for weights and only 32-bit for activations as they are supported by TensorRT. The precision search space for FPGA consists of 4, 8, and 16 bits for weights and only 16 bit for activations. EDD-GPU and EDD-FPGA achieve a latency of 11.17 ms and 7.96 ms on the Titan RTX GPU and ZCU102 FPGA, respectively. The GPU model is

1.4×, 1.6×, 2.0× faster and the FPGA model is 1.37×, 1.1×, 1.5× faster than ProxylessNAS-GPU [8], MNasNet [119], FBNet-C [132], respectively, with almost the same accuracy on ImageNet.

HAO [26] co-explores the combined space of architecture, quantization, and hardware design by formulating the search space as an integer programming problem to reduce the computation cost. HAO utilizes a latency model and an accuracy predictor to generate Pareto-optimal models for acceleration on embedded FPGAs. The integer programming method is used to prune the hardware space by minimizing the hardware latency, subject to resource constraints. HAO network achieves 72.5% accuracy on ImageNet dataset with 50 fps on Zynq ZU3EG FPGA, which is 60% and 135% faster than MnasNet [119] and FBNet [132], respectively.

Auto-NBA [37] jointly explores the network architecture (kernel size, expansion channels), precision (4, 6, 8, 12, 16 bits), and the accelerator specifications (memory size and tiling strategies) in a differentiable/gradient manner.

Neural-hardware Architecture Search [83] co-designs the accelerator (using evolution search [41]) and quantized neural algorithm (using one-shot hyper-net). The hardware space is the author's in-house simulator supporting mixed-precision quantized networks with search elements: number of rows and columns of the systolic array and input/output buffer size. The ResNet50 [47] network is considered as the neural backbone on which the kernel sizes {3, 5}, number of filters, and bit precisions in each layer (2/4/6 for weights and 4/6/8 bits for activations) are searched.

NACIM [62] is an RL-controller based co-search method for computing-in-memory devices to optimize: **(i) Neural Architecture:** number of filters and kernel size; **(ii) Quantization:** varying precision for weight and activation; **(iii) Dataflow:** weight/output/row stationary and no local reuse; **(iv) Circuit:** architecture of the core accelerator; **(v) Device Type:** ReRAM, FeFET, STT-MRAM.

10 OTHER FORMS OF HARDWARE-BASED NEURAL ARCHITECTURE SEARCH

The algorithmic development of HW-NAS and advancement on a wide range of devices paved the way for applying NAS methods on other hardware-based scenarios, apart from just searching for a neural architecture. This section explores some of these miscellaneous techniques.

10.1 Automatic Neural Network Pruning Using AutoML and NAS

CNN pruning is an effective approach to reduce the model size by removing the redundant parameters. Several handcrafted methods have been developed to efficiently compress and accelerate the networks on various hardware platforms. Hardware-aware pruning methods adapt the pruning process with respect to the target hardware. AutoML alleviates human efforts by automatically pruning the over-parameterized model via a feedback mechanism. These methods consider inference metrics such as the latency of the target device into the loop and prune filters iteratively. **AMC** [49] employs RL method for automatic model compression using a trained agent. **NetAdapt** [139] utilizes a hardware-aware pruning algorithm to fine-tune a pre-trained model by considering latency on mobile CPU and GPU. AMC achieves 1.53× speedup on a Titan XP GPU for MobileNetV1 network, while NetAdapt attains a speedup of 1.2× on mobile GPU of Samsung S8. **APQ** [128] moves one step forward to jointly search the architecture, pruning percentage, and mixed precision.

10.2 Dedicated NAS Processor

Traditionally, the static neural networks and supernetworks in the search process are trained on GPU, thereby consuming hundreds of GPU hours. Ma et al. proposed NASA [96], a high-throughput hardware accelerator specially designed to accelerate the one-shot based NAS. Unlike the accelerator co-designed in the previous section, the NASA processor supports the training

of the supernetworks, where the former is mainly for inference of the final search model. NASA achieves a speedup of 33.52× and decreased energy cost by 214.33× compared to the conventional CPU-GPU system.

10.3 Searching for Winograd Convolutions

The traditional Convolution operation can be reformulated into other forms such as FFT or Winograd to obtain speedup while implementing CNNs on various devices. Winograd transformation of Convolution [72] is the fastest algorithm for spatially small Convolution operations. Although Winograd Convolution is faster than the traditional im2row method, it introduces numerical error in the product, leading to an accuracy loss. Hence, choosing the optimal Winograd configuration or im2row in each layer becomes a combinatorial problem, thereby allowing us to use the NAS algorithms. Fernandez et al. proposed **wiNAS** [34] to search for im2row or Winograd configuration from the pool of F(2 × 2, 3 × 3), F(4 × 4, 3 × 3), F(6 × 6, 3 × 3) using ProxylessNAS [8]. The wiNAS-Resnet network is 1.54× faster than the im2row method using INT8 representation.

10.4 NAS for RL Agent and Accelerator Co-search

Automated Agent Accelerator Co-search (A3C-S) [36] co-explores the **Deep Reinforcement Learning (DRL)** agent, an integration of RL algorithm with DNNs, and corresponding accelerator to maximize DRL score and performance efficiency in applications like self-driving vehicles.

10.5 Fault-tolerant Neural Architecture Search

Hardware systems are often prone to soft errors or permanent faults due to external conditions or internal scaling. Li et al. proposed **FTT-NAS** [77] to search for networks that are dependable on various faulty conditions such as random bit-flip, Gaussian, and so on, by combining **fault tolerant NAS (FT-NAS)** and **fault tolerant training (FTT)**. The REINFORCE method [130] optimizes the multi-objective function: R = $(1-\alpha_r)$*acc_{clean} + α_r*acc_{faulty}, operating on clean (acc_{clean}) and faulty accuracy (acc_{faulty}). The experiments on CIFAR-10 suggest that the searched network outperforms ResNet, Vgg16, Mobilenet in terms of accuracy under the different fault settings.

10.6 NAS for Electronic Design Automation (EDA)

CNNs are used in EDA applications to automate the modern digital IC design process. Pan et al. [104] applied NAS to aid the CNN design pipeline for routability prediction, a method to estimate the routability of solutions at the placement stage. Evolutionary search algorithm is applied on ResNet search space to outperform the hand-crafted ResNet-18 model with 5.6% higher Kendall's τ.

10.7 Adversarial Neural Architecture Search

Adversarial examples mislead a high accurate deep learning model using a **Generative Adversarial Network (GAN)**. **AdversarialNAS** [38] is a method to concurrently search for generator and discriminator architecture in GANs on the unconditional image generation task. **NASGuard** [129] is a dedicated accelerator for robust NAS models for adversarial examples that can fully utilize the accelerator resources and enhance data locality.

10.8 Hardware-aware Neural Architecture Search Benchmarks

The need for computational resources (GPUs) is a huge obstacle to the development of novel NAS algorithms, both in terms of accuracy and performance evaluation. As a result, several NAS-related benchmarks have been released, which act as a LUT/dataset for various networks and corresponding accuracy/performance metrics. **NAS-Bench-101** [140] is the first large-scale dataset for NAS

where 423K cell-based architectures are trained end-to-end on CIFAR-10 to capture the test accuracy and training time. **NAS-Bench-201** [25] extends the NAS-Bench-101 dataset by including possible networks generated by four nodes and five operations on three different datasets. **NAS-Bench-301** [115] initially generates 10^{18} unique neural architectures on the DARTS-search space and samples only 60K networks as a subset for training based on a surrogate fitted model.

The hardware-agnostic NAS benchmarks provide only the training time and do not measure the latency of each network on the hardware. Therefore, these benchmarks limit the usage in developing HW-aware NAS algorithms, especially for non-hardware researchers. **LatBench** [28] is a large-scale dataset of latency measurement of NAS-Bench-201 models [25] on a wide range of devices including desktop-, mobile-, embedded-CPU/GPU, TPU, and DSP. **HW-NAS-Bench** [75] estimates the latency of various networks on the cell-based architecture (NAS-Bench-201) and layer-wise network (FBNet search space) on Jetson TX2 Edge GPU, Raspberry Pi 4, EdgeTPU, Pixel 3, Eyeriss, and XilinxZC706 FPGA.

11 CONCLUSION AND FUTURE WORK

In recent years, HW-NAS has completely redesigned the way neural networks are traditionally designed for various devices. HW-NAS provided a way to have hybrid operations within a single network has not been the case in the earlier models such as ResNet50 [47]. We, in this article, have provided an overview and summary of several HW-NAS methods targeting resource constraint devices to high-performance systems, followed by a co-search of several aspects of accelerator and algorithm. We stress the importance of hardware-aware search space, search space construction, and hardware-based multi-objective search while designing robust and effective architectures. The automated-designed networks outperform the manually built models, and the co-searched accelerator-network pair outperform the manually co-deign designs in terms of both accuracy and performance. Going forward, we believe more research is needed in the following areas:

(1) **MCU-NAS:** We are still in the early stages of NAS for efficient inference on MCUs with reasonable accuracy. MCUNet [81] is the only method in our literate that provides accuracy close to 70% on ImageNet. More efficient algorithms and compiler design to improve accuracy and performance on the limited computing system are needed.
(2) **Efficient Search Space:** The search elements often limit the NAS algorithm to find more efficient models. The majority of the HW-NAS methods choose mobile search space (MobileNetV2, ShuffleNetV2) as the backbone network, even on high-end devices. The accuracy range of models searched on this search space is 72%–76%, even on the mixed search space [147]. More efficient spaces such as EfficientNet may lead the way to the development of a more robust and efficient search space. Recently, transformers [68] are being used to replace the traditional Convolutions for vision applications. Transformer architecture and accelerator co-search could be very efficient to achieve accuracy higher than SOTA designs.
(3) **Co-design of Algorithm and Accelerator: Auto-NBA** [37] targets architecture, precision, and accelerator micro-architecture while ignoring dataflow. **NAAS** [84] targets neural architecture and accelerator (micro-architecture and compiler mapping), while leaving the quantization aspect. Hence, efficient differential co-search methods for all four dimensions, i.e., algorithm (architecture and precision) and accelerator (hardware architecture and compiler routines), are needed to deploy DNNs in many use cases.
(4) **Co-searching Sparse CNN and Accelerator:** The current HW-NAS methods search only for regular/dense Convolution layers while ignoring the sparse matrices and do not consider the sparsity-supporting Tensor Core hardware in the latest Nvidia A100 GPUs. Sparse CNN

and sparsity-supporting accelerator search could be very efficient research to achieve more model compression and acceleration than the current searched regular accelerator-CNN pair.

(5) **Benchmarks:** There exists a variety of hardware such as MCUs, server-level CPUs and multi-GPUs, backbone architectures such as MobileNetV3, EfficientNet, and datasets for which HW-NAS benchmarks can be developed. As quantization is very important for real-time inference, HW-quantized NAS benchmarks should be given priority in the future, as the current benchmarks do not include precision. HW accelerator benchmarks for NAS could be taken into consideration as the co-design process involves both dimensions.

(6) **NAS for other Purposes:** Initially, NAS algorithms were designed and targeted only for searching efficient architectures. Later, as NAS became an approachable concept, it has been utilized for other purposes, such as pruning, quantization, and Winograd search. Hence, the potential of NAS can be expanded for other applications and use cases as well.

(7) **HW-NAS for other Applications:** While HW-NAS has been used to develop latency-efficient models for image classification task, there has been a very little focus on other tasks, such as **MobileDets** [135] for Object Detection, **Hardware-aware Transformers** [125] for NLP, **SqueezeNAS** [114] for Semantic Segmentation. Exploration to optimize and build custom DNN-hardware designs for other vision applications still remains an open challenge.

REFERENCES

[1] Mohamed S. Abdelfattah et al. 2020. Best of both worlds: AutoML codesign of a CNN and its hardware accelerator. In *57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[2] Paniti Achararit et al. 2020. APNAS: Accuracy-and-performance-aware neural architecture search for neural hardware accelerators. *IEEE Access* 8 (2020), 165319–165334.

[3] Kai Arulkumaran et al. 2017. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866* (2017).

[4] Bowen Baker et al. 2017. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823* (2017).

[5] Colby Banbury et al. 2020. MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers. *arXiv preprint arXiv:2010.11267* (2020).

[6] Hadjer Benmeziane et al. 2021. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336* (2021).

[7] John Burgess. 2020. RTX on—The NVIDIA Turing GPU. *IEEE Micro* 40, 2 (2020), 36–44.

[8] Han Cai et al. 2018. ProxylessNAS: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332* (2018).

[9] Han Cai et al. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791* (2019).

[10] Zhaowei Cai et al. 2020. Rethinking differentiable search for mixed-precision neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2349–2358.

[11] Dong Chen et al. 2021. DcaNAS: Efficient convolutional network design for desktop CPU platforms. *Appl. Intell.* (2021), 1–14.

[12] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-state Circ.* 52, 1 (2016), 127–138.

[13] Hsin-Pai Cheng et al. 2019. MSNet: Structural wired neural architecture search for internet of things. In *IEEE/CVF International Conference on Computer Vision Workshops*.

[14] Krishna Teja Chitty-Venkata and Arun K. Somani. 2021. Array-aware neural architecture search. In *IEEE 32nd International Conference on Application-specific Systems, Archit. Process.* IEEE, 125–132.

[15] Choi et al. 2020. DANCE: Differentiable accelerator/network co-exploration. *arXiv preprint arXiv:2009.06237* (2020).

[16] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*. 1251–1258.

[17] Grace Chu et al. 2021. Discovering multi-hardware mobile models via architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3022–3031.

[18] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. 2020. MoGA: Searching beyond MobileNetV3. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 4042–4046.

[19] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. 2019. FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845* (2019).
[20] Xiaoliang Dai et al. 2019. ChamNet: Towards efficient network design through platform-aware model adaptation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11398–11407.
[21] Robert David et al. 2020. TensorFlow Lite Micro: Embedded machine learning on TinyML systems. *arXiv preprint arXiv:2010.08678* (2020).
[22] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolut. Computat.* 6, 2 (2002), 182–197.
[23] Jia Deng et al. 2009. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 248–255.
[24] Ciarán Donegan et al. 2021. VPU specific CNNs through neural architecture search. In *25th International Conference on Pattern Recognition (ICPR)*. IEEE, 9772–9779.
[25] Xuanyi Dong and Yi Yang. 2020. NAS-Bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326* (2020).
[26] Zhen Dong et al. 2021. HAO: Hardware-aware neural architecture optimization for efficient inference. *arXiv preprint arXiv:2104.12766* (2021).
[27] Zidong Du et al. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *42nd Annual International Symposium on Computer Architecture*. 92–104.
[28] Łukasz Dudziak et al. 2020. BRP-NAS: Prediction-based NAS using GCNs. *arXiv preprint arXiv:2007.08668* (2020).
[29] Thomas Elsken et al. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).
[30] Ahmed Elthakeb et al. 2019. ReLeQ: An automatic reinforcement learning approach for deep quantization of neural networks. In *NeurIPS ML for Systems Workshop*.
[31] Hongxiang Fan et al. 2020. Optimizing FPGA-based CNN accelerator using differentiable neural architecture search. In *IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 465–468.
[32] Jiemin Fang et al. 2020. Densely connected search space for more flexible neural architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10628–10637.
[33] Igor Fedorov, Ryan P. Adams, Matthew Mattina, and Paul N. Whatmough. 2019. SpArSe: Sparse architecture search for CNNs on resource-constrained microcontrollers. *arXiv preprint arXiv:1905.12107* (2019).
[34] Fernandez et al. 2020. Searching for Winograd-aware quantized networks. *arXiv preprint arXiv:2002.10711* (2020).
[35] Cheng Fu et al. 2020. Enhancing model parallelism in neural architecture search for multidevice system. *IEEE Micro* 40, 5 (2020), 46–55.
[36] Yonggan Fu et al. 2021. A3C-S: Automated agent accelerator co-search towards efficient deep reinforcement learning. *arXiv preprint arXiv:2106.06577* (2021).
[37] Yonggan Fu et al. 2021. Auto-NBA: Efficient and effective search over the joint space of networks, bitwidths, and accelerators. *arXiv preprint arXiv:2106.06575* (2021).
[38] Chen Gao et al. 2020. AdversarialNAS: Adversarial neural architecture search for GANs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5680–5689.
[39] Chengyue Gong et al. 2019. Mixed precision neural architecture search for energy efficient deep learning. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–7.
[40] Kaiyuan Guo et al. 2017. A survey of FPGA-based neural network accelerator. *arXiv preprint arXiv:1712.08934* (2017).
[41] Zichao Guo et al. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*. Springer, 544–560.
[42] Gupta et al. 2020. Accelerator-aware neural network design using automl. *arXiv preprint arXiv:2003.02838* (2020).
[43] Suyog Gupta. 2021. Improved On-Device ML on Pixel 6, with Neural Architecture Search. Retrieved from https://ai.googleblog.com/2021/11/improved-on-device-ml-on-pixel-6-with.html.
[44] Cong Hao et al. 2019. FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge. In *56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[45] Cong Hao et al. 2019. NAIS: Neural architecture and implementation search and its applications in autonomous driving. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
[46] Cong Hao et al. 2020. Effective algorithm-accelerator co-design for AI solutions on edge devices. In *Great Lakes Symposium on VLSI*. 283–290.
[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[48] Xin He et al. 2019. AutoML: A survey of the state-of-the-art. *arXiv preprint arXiv:1908.00709* (2019).
[49] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*. 784–800.

[50] Andrew Howard et al. 2019. Searching for MobileNetV3. In *IEEE/CVF International Conference on Computer Vision*. 1314–1324.
[51] Andrew G. Howard et al. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
[52] Chi-Hung Hsu et al. 2018. MONAS: Multi-objective neural architecture search using reinforcement learning. *arXiv preprint arXiv:1806.10332* (2018).
[53] Jie Hu et al. 2018. Squeeze-and-excitation networks. In *IEEE Conference on Computer vision and Pattern Recognition*. 7132–7141.
[54] Yibo Hu, Xiang Wu, and Ran He. 2020. TF-NAS: Rethinking three search freedoms of latency-constrained differentiable neural architecture search. *arXiv preprint arXiv:2008.05314* (2020).
[55] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4700–4708.
[56] Sitao Huang et al. 2021. Mixed precision quantization for ReRAM-based DNN inference accelerators. In *26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 372–377.
[57] X. Inc. 2019. *CHaiDNNv2 - HLS based DNN accelerator library for Xilinx ultrascale+ MPoCs*. Retrieved from https://github.com/Xilinx/CHaiDNN.
[58] Intel. 2016. *Intel Movidius Myriad Vision Processing Unit*. Retrieved from: https://software.intel.com/content/www/us/en/develop/topics/iot/hardware/vision-accelerator-movidius-vpu.html.
[59] Eric Jang et al. 2016. Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).
[60] Qian Jiang et al. 2021. EH-DNAS: End-to-end hardware-aware differentiable neural architecture search. *arXiv preprint arXiv:2111.12299* (2021).
[61] Weiwen Jiang et al. 2019. Accuracy vs. efficiency: Achieving both through FPGA-implementation aware neural architecture search. In *56th Annual Design Automation Conference*. 1–6.
[62] Weiwen Jiang et al. 2020. Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Trans. Comput.* (2020).
[63] Weiwen Jiang et al. 2020. Hardware/software co-exploration of neural architectures. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 39, 12 (2020), 4805–4815.
[64] Weiwen Jiang et al. 2020. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 39, 11 (2020), 4154–4165.
[65] Norman P. Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *44th Annual International Symposium on Computer Architecture*. 1–12.
[66] Patrick Judd et al. 2016. Stripes: Bit-serial deep neural network computing. In *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.
[67] Donghyun Kang et al. 2019. A novel convolutional neural network accelerator that enables fully-pipelined execution of layers. In *IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 698–701.
[68] Salman Khan et al. 2021. Transformers in vision: A survey. *arXiv preprint arXiv:2101.01169* (2021).
[69] Alex Krizhevsky, Geoffrey Hinton et al. 2009. Learning multiple layers of features from tiny images. (2009).
[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Conference on Advances in Neural Information Processing Systems*. 1097–1105.
[71] Liangzhen Lai et al. 2018. CMSIS-NN: Efficient neural network kernels for arm Cortex-M CPUs. *arXiv preprint arXiv:1801.06601* (2018).
[72] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*. 4013–4021.
[73] Jaeseong Lee et al. 2020. S3NAS: Fast NPU-aware neural architecture search methodology. *arXiv preprint arXiv:2009.02009* (2020).
[74] Li et al. 2021. Searching for fast model families on datacenter accelerators. *arXiv preprint arXiv:2102.05610* (2021).
[75] Chaojian Li et al. 2021. HW-NAS-bench: Hardware-aware neural architecture search benchmark. *arXiv preprint arXiv:2103.10584* (2021).
[76] Guihong Li et al. 2021. FLASH: Fast neural Architecture Search with Hardware Optimization. *ACM Trans. Embed. Comput. Syst.* 20, 5s (2021), 1–26.
[77] Wenshuo Li et al. 2020. FTT-NAS: Discovering fault-tolerant neural architecture. In *25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 211–216.
[78] Xin Li et al. 2019. Partial order pruning: For best speed/accuracy trade-off in neural architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9145–9153.
[79] Yuhong Li et al. 2020. EDD: Efficient differentiable DNN architecture and implementation co-search for embedded AI solutions. In *57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[80] Edgar Liberis, Łukasz Dudziak, and Nicholas D. Lane. 2021. μNAS: Constrained neural architecture search for microcontrollers. In *1st Workshop on Machine Learning and Systems*. 70–79.
[81] Ji Lin et al. 2020. MCUNet: Tiny deep learning on IoT devices. *arXiv preprint arXiv:2007.10319* (2020).
[82] Ji Lin et al. 2021. MCUNetV2: Memory-efficient patch-based inference for tiny deep learning. *arXiv preprint arXiv:2110.15352* 1 (2021).
[83] Yujun Lin et al. 2019. Neural-hardware architecture search. *NeurIPS WS* (2019).
[84] Yujun Lin et al. 2021. NAAS: Neural accelerator architecture search. *arXiv preprint arXiv:2105.13258* (2021).
[85] Chenxi Liu et al. 2018. Progressive neural architecture search. In *European Conference on Computer Vision (ECCV)*. 19–34.
[86] Hanxiao Liu et al. 2018. DARTS: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[87] Yuqiao Liu et al. 2020. A survey on evolutionary neural architecture search. *arXiv preprint arXiv:2008.10937* (2020).
[88] Mohammad Loni et al. 2020. DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocess. Microsyst.* 73 (2020), 102989.
[89] Javier García López et al. 2021. E-DNAS: Differentiable neural architecture search for embedded systems. In *25th International Conference on Pattern Recognition (ICPR)*. IEEE, 4704–4711.
[90] Bingqian Lu et al. 2021. One proxy device is enough for hardware-aware neural architecture search. *arXiv preprint arXiv:2111.01203* (2021).
[91] Haodong Lu et al. 2021. An adaptive neural architecture search design for collaborative edge-cloud computing. *IEEE Netw.* 35, 5 (2021), 83–89.
[92] Qing Lu et al. 2019. On neural architecture search for resource-constrained hardware platforms. *arXiv preprint arXiv:1911.00105* (2019).
[93] Xiangzhong Luo, Di Liu, Hao Kong, and Weichen Liu. 2020. EdgeNAS: Discovering efficient neural architectures for edge systems. In *IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 288–295.
[94] Bo Lyu, Hang Yuan, Longfei Lu, and Yunye Zhang. 2021. Resource-constrained neural architecture search on edge devices. *IEEE Trans. Netw. Sci. Eng.* (2021).
[95] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical guidelines for efficient cnn architecture design. In *European Conference on Computer Vision (ECCV)*. 116–131.
[96] Xiaohan Ma et al. [n.d.]. NASA: Accelerating neural network design with a NAS processor. ([n. d.]).
[97] Daniel Manu et al. 2021. Co-exploration of graph neural network and network-on-chip design using AutoML. In *Great Lakes Symposium on VLSI*. 175–180.
[98] Hashan Roshantha Mendis et al. 2021. Intermittent-aware neural architecture search. *ACM Trans. Embed. Comput. Syst.* 20, 5s (2021), 1–27.
[99] Keith G. Mills et al. 2021. Profiling neural blocks and design spaces for mobile neural architecture search. In *30th ACM International Conference on Information & Knowledge Management*. 4026–4035.
[100] Bert Moons, Parham Noorzad, Andrii Skliar, Giovanni Mariani, Dushyant Mehta, Chris Lott, and Tijmen Blankevoort. 2021. Distilling optimal neural networks: Rapid search in diverse spaces. In *IEEE/CVF International Conference on Computer Vision*. 12229–12238.
[101] Niv Nayman et al. 2021. HardCoRe-NAS: Hard constrained differentiable neural architecture search. *arXiv preprint arXiv:2102.11646* (2021).
[102] Shubham Negi et al. 2021. NAX: Co-designing neural network and hardware architecture for memristive Xbar based computing systems. *arXiv preprint arXiv:2106.12125* (2021).
[103] NVIDIA. 2017. *NVDIA deep learning accelerator*. Retrieved from http://nvdla.org.
[104] Jingyu Pan et al. 2020. Automatic routability predictor development using neural architecture search. *arXiv preprint arXiv:2012.01737* (2020).
[105] Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. 2020. A flexible framework for multi-objective Bayesian optimization using random scalarizations. In *Uncertainty in Artificial Intelligence*. PMLR, 766–776.
[106] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*. PMLR, 4095–4104.
[107] Songyun Qu et al. 2020. RaQu: An automatic high-utilization CNN quantization and mapping framework for general-purpose RRAM accelerator. In *57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[108] Ilija Radosavovic et al. 2020. Designing network design spaces. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10428–10436.
[109] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*. 4780–4789.
[110] Pengzhen Ren et al. 2020. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903* (2020).

[111] Sergio Rivas-Gomez et al. 2018. Exploring the vision processing unit as co-processor for inference. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 589–598.
[112] Mark Sandler et al. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.
[113] Hardik Sharma et al. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 764–775.
[114] Albert Shaw et al. 2019. SqueezeNAS: Fast neural architecture search for faster semantic segmentation. In *IEEE/CVF International Conference on Computer Vision Workshops*.
[115] Julien Siems et al. 2020. NAS-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777* (2020).
[116] Dimitrios Stamoulis et al. 2019. Single-path NAS: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 481–497.
[117] Mingxing Tan et al. 2019. MixConv: Mixed depthwise convolutional kernels. *arXiv preprint arXiv:1907.09595* (2019).
[118] Mingxing Tan et al. 2021. EfficientNetV2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298* (2021).
[119] Mingxing Tan, Bo Chen, et al. 2019. MnasNet: Platform-aware neural architecture search for mobile. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2820–2828.
[120] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 6105–6114.
[121] Chris Thornton et al. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 847–855.
[122] Yaman Umuroglu et al. 2018. BISMO: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 307–3077.
[123] Arash Vahdat et al. 2020. UNAS: Differentiable architecture search meets reinforcement learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11266–11275.
[124] Ilja van Ipenburg et al. [n.d.]. Exploring cell-based neural architectures for embedded systems. ([n. d.]).
[125] Hanrui Wang et al. 2020. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187* (2020).
[126] Ke Wang et al. 2020. Neural architecture search for robust networks in 6G-enabled massive IoT domain. *IEEE Internet Things J.* 8, 7 (2020), 5332–5339.
[127] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-aware automated quantization with mixed precision. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8612–8620.
[128] Tianzhe Wang et al. 2020. APQ: Joint search for network architecture, pruning and quantization policy. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2078–2087.
[129] Xingbin Wang et al. [n.d.]. NASGuard: A novel accelerator architecture for robust neural architecture search (NAS) networks. *Cell* 1 ([n. d.]), 1.
[130] Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* 8, 3–4 (1992), 229–256.
[131] Martin Wistuba et al. 2019. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392* (2019).
[132] Bichen Wu et al. 2019. FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10734–10742.
[133] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed precision quantization of ConvNets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090* (2018).
[134] Xia et al. 2020. HNAS: Hierarchical neural architecture search on mobile devices. *arXiv preprint arXiv:2005.07564* (2020).
[135] Yunyang Xiong et al. 2020. MobileDets: Searching for object detection architectures for mobile accelerators. *arXiv preprint arXiv:2004.14525* (2020).
[136] Yuhui Xu et al. 2020. Latency-aware differentiable neural architecture search. *arXiv preprint arXiv:2001.06392* (2020).
[137] Zheyu Yan et al. 2021. Uncertainty modeling of emerging device based computing-in-memory neural accelerators with application to neural architecture search. In *26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 859–864.
[138] Lei Yang et al. 2020. Co-exploration of neural architectures and heterogeneous ASIC accelerator designs targeting multiple tasks. In *57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[139] Tien-Ju Yang et al. 2018. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *European Conference on Computer Vision (ECCV)*. 285–300.
[140] Chris Ying et al. 2019. NAS-Bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*. PMLR, 7105–7114.
[141] Yongan et al. 2020. DNA: Differentiable network-accelerator co-search. *arXiv preprint arXiv:2010.14778* (2020).

[142] Shan You et al. 2020. GreedyNAS: Towards fast one-shot NAS with greedy supernet. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1999–2008.
[143] Steven R. Young et al. 2019. Evolving energy efficient convolutional neural networks. In *IEEE International Conference on Big Data (Big Data)*. IEEE, 4479–4485.
[144] Fisher Yu et al. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).
[145] Zhihang Yuan et al. 2021. NAS4RRAM: Neural network architecture search for inference on RRAM-based accelerators. *Sci. China Inf. Sci.* 64, 6 (2021), 1–11.
[146] Shulin Zeng et al. 2020. Black box search space profiling for accelerator-aware neural architecture search. In *25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 518–523.
[147] Li Lyna Zhang et al. 2020. Fast hardware-aware neural architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 692–693.
[148] Yongan Zhang et al. 2021. G-CoS: GNN-accelerator co-search towards both better accuracy and efficiency. *arXiv preprint arXiv:2109.08983* (2021).
[149] Yanqi Zhou et al. 2021. Rethinking co-design of neural architectures and hardware accelerators. *arXiv preprint arXiv:2102.08619* (2021).
[150] Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

Received 23 July 2021; revised 23 February 2022; accepted 8 March 2022