Hardware-Aware Neural Architecture Search: Survey and Taxonomy

Hadjer Benmeziane [1*], **Kaoutar El Maghraoui**[3], **Hamza Ouarnoughi** [2], **Smail Niar** [2], **Martin Wistuba** [4] and **Naigang Wang** [3]
[1]Université Polytechnique Hauts-de-France, LAMIH/CNRS, Valenciennes, France.
[2]Université Polytechnique Hauts-de-France, LAMIH/CNRS, INSA, Valenciennes, France.
[3] IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA.
[4] IBM Research AI, IBM Technology Campus, Dublin, Ireland.
{firstname.lastname}@uphf.fr, {kelmaghr, nwang}@us.ibm.com, martin.wistuba@ibm.com

Abstract

There is no doubt that making AI mainstream by bringing powerful, yet power hungry Deep Neural Networks (DNNs) to resource-constrained devices would require an efficient co-design of algorithms, hardware and software. The increased popularity of DNN applications deployed on a wide variety of platforms, from tiny microcontrollers to datac enters, have resulted in multiple questions and challenges related to constraints introduced by the hardware. In this survey on Hardware-Aware Neural Architecture Search (HW-NAS), we present some of the existing answers proposed in the literature for the following questions: "Is it possible to build an efficient Deep Learning (DL) model that meets the latency and energy constraints of tiny edge devices?", "How can we find the best trade-off between the accuracy of a DL model and its ability to be deployed in a variety of platforms?". The survey provides a new taxonomy of HW-NAS and assesses the hardware cost estimation strategies. We also highlight the challenges and limitations of existing approaches and potential future directions. We hope that this survey will help to fuel the research towards efficient deep learning.

1 Introduction

The past years have witnessed a remarkable development in neural network models and algorithms targeting multiple tasks from computer vision and natural language processing to games. Given the difficulty to manually design a Deep Learning (DL) model, many researchers turn to Neural Architecture Search (NAS). Current state-of-the-art model in image classification and object detection, EfficientNet-L2 [Pham *et al.*, 2020], was obtained by automatically searching for the hyperparameters and operators of the architecture.

A typical NAS process is composed of three fundamental parts. First, the *search space* which represents a considerable number of different architectures and different value ranges for each hyperparameter. Second, the *search algorithm* that explores the search space looking for the best architecture that maximizes the objective function. Finally, as we need a way to measure an architecture's performance to compute the objective function, a third component, *the evaluator*, is used to obtain the model's accuracy.

While considerable efforts have focused on the accuracy, the resulting models tend to be exponentially complex and require increasing memory and compute. Therefore, in real-world applications such as autonomous driving cars, hardware constraints come out as a critical limiting factor to exploit DL models at their full potential. Besides, an architecture's manual design is significantly more difficult if we consider the hardware variety and limitations. This is why, since 2017, we have seen a new wave of NAS algorithms, hardware-aware neural architecture search (HW-NAS), that incorporate the hardware constraints into their objective function and optimize their search space according to a target hardware platform.

The goal of this survey is to provide a comprehensive summary of HW-NAS by highlighting the following attributes:

- The survey proposes a special taxonomy for HW-NAS according to the final hardware target in section 2.
- It explains how the multi-objective problem is formulated and how the hardware constraints such as the latency, energy and memory footprint are incorporated into the objective function (see section 3).
- The survey also describes, in section 4, the Hardware Search Space, a new component in HW-NAS, as opposed to the conventional Architecture Search Space used in NAS.
- One crucial component is how to profile the hardware metrics and collect them. We discuss these in section 5.
- Finally, in section 6, the paper focuses on helping quickly ramp up new ideas in HW-NAS by exposing the limitations and the future directions towards building efficient deep learning models.

2 HW-NAS Taxonomy

HW-NAS has multiple objectives and multiple views of the problem emerging from the variety of hardware platforms available. As shown in figure 1, we can classify these goals into three categories, described in the following subsections.

*Contact Author

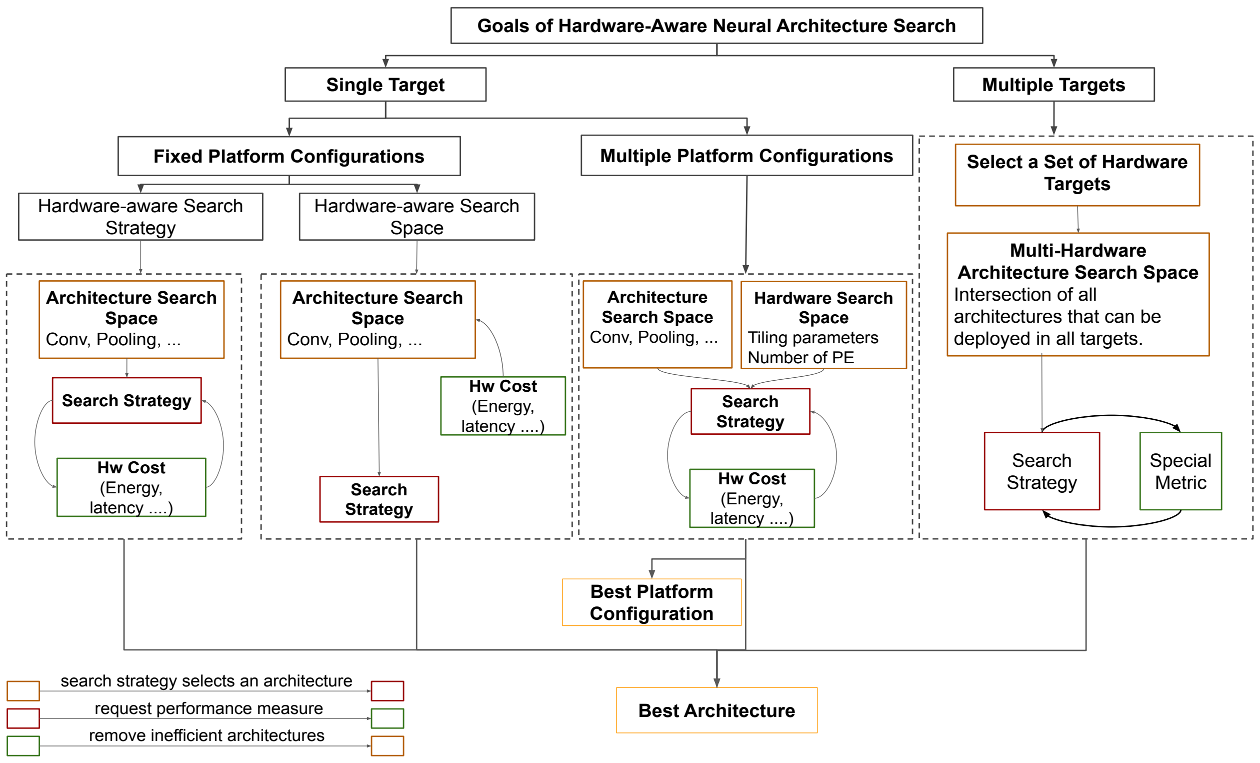
Goals of Hardware-Aware Neural Architecture Search
Single Target
Multiple Targets
Fixed Platform Configurations
Multiple Platform Configurations
Hardware-aware Search Strategy
Hardware-aware Search Space
Architecture Search Space
Conv, Pooling, ...
Search Strategy
Hw Cost
(Energy, latency ....)
Hardware Search Space
Tiling parameters
Number of PE
Select a Set of Hardware Targets
Multi-Hardware Architecture Search Space
Intersection of all architectures that can be deployed in all targets.
Special Metric
Best Platform Configuration
Best Architecture
search strategy selects an architecture
request performance measure
remove inefficient architectures

Figure 1: Taxonomy of HW-NAS

2.1 Single Target, Fixed Configuration.

Most of the existing HW-NAS fall under this category. The goal is to find the best architecture in terms of accuracy and hardware efficiency for one fixed target hardware. Let's suppose that our hardware platform is an edge NVIDIA GPU, the HW-NAS will explore the space of architectures to look for the one that optimize accuracy, latency, energy consumption, etc. Within this category, two approaches are adopted:

-*Hardware-aware search strategy* where the search is cast as a multi-objective optimization problem [Tan *et al.*, 2018; Cai *et al.*, 2018; Wu *et al.*, 2019]. While searching for the best architecture, the search algorithm calls the accuracy evaluator component to get the accuracy of the generated architecture and a special evaluator that measures the hardware cost metric (e.g., latency, memory usage, energy consumption). Both model accuracy and hardware cost guide the search and enable the NAS to find the most efficient architecture.

-*Hardware-aware Search Space* uses a restricted pool of architectures. Before the search, we either measure the operators' performance on the target platform or define a set of rules that will refine the search space; eliminate all the architectures' operators that do not perform well on the target hardware. The prior empirical study helps define the rules. HURRICANE [Zhang *et al.*, 2020] uses different operator choices for three types of mobile processors: Hexagon DSP, ARM CPU and Myriad Vision Processing Unit (VPU). Accumulated domain knowledge from prior experimentation on a given hardware platform help narrow down the search space. For instance, they do not use depthwise convolutions for CPU, squeeze and excitation mechanisms for VPU and they do not lower the kernel sizes for a DSP.

2.2 Single Target, Multiple Configurations.

This category aims not only to get the most optimal architecture based on the accuracy but also to get an optimal architecture with latency guaranteed to meet the target hardware specification. For example, the authors of FNAS [Jiang *et al.*, 2019b] define a new hardware search space containing the different FPGA specifications (e.g., tiling configurations). They also use a performance abstraction model to measure the searched neural architectures' latency without training. This allows them to quickly prune architectures that do not meet the target hardware specifications. In [Yang *et al.*, 2020], the authors use the same ASICs approach and define a hardware search space containing various ASIC templates.

2.3 Multiple Targets.

In this third category, the goal is to find the best architecture when given a set of hardware platforms to optimize for. In other words, we try to find a single model that performs relatively well across different hardware platforms. This approach is the most favourable choice, especially in mobile development, as it provides more portability. This problem was tackled by [Chu *et al.*, 2020; Jiang *et al.*, 2020b] by defining a multi-hardware search space. The search space contains the intersection of all the architectures that can be deployed in the different targets. Note that, targeting multiple hardware specifications is more challenging as the best model for a GPU, can be very different to the best model for a CPU (i.e., for GPUs wider models are more appropriate while for CPUs deeper models are).

3 HW-NAS Problem Formulation

The Hardware-aware NAS is cast as a multi-objective optimization problem in which we consider the accuracy and hardware constraints, such as the latency, memory footprint and energy consumption, as different objectives. In this section, we classify the formulations into two classes: single and multi-objective optimization.

We denote the space of all feasible architectures as A, also called architecture search space. The optimization method is looking for the architecture α that maximizes the performance metric denoted by f for a given dataset δ.

1. Single-Objective Optimization:

   A) Two-Stage optimization: In this formulation (equation 1), the search is solely based on the accuracy f. Once an architecture is found, different compression techniques are applied to specialize it. [Han *et al.*, 2019] applies a reinforcement learning agent to find the best quantization bitwidth and pruning level after selecting the most accurate model. [Cai *et al.*, 2020] searches over a pre-trained and selected architecture to find the most efficient one in terms of latency and energy consumption in the second stage.

   $$\max_{\alpha \in A} f(\alpha, \delta) \tag{1}$$

   B) Constrained optimization: As illustrated in equation 2, we consider in this formulation the hardware metrics g_i as constraints to our optimization problem. The threshold T_i and the trade-off between different constraints can be adapted to practical requirements. As most of the optimization methods used by NAS (i.e. reinforcement learning and evolutionary algorithms) were designed for unconstrained optimization problems, this formulation is hard to be adopted directly. Therefore, many researchers turned to penalty methods to transform the equation into a single objective function that contains the hardware constraints, and the accuracy measurement [Tan *et al.*, 2018; Wu *et al.*, 2019; Cai *et al.*, 2018]. For example, MNASNet [Tan *et al.*, 2018] defines g as the latency of the model and uses a learnable parameter to control the effect of the hardware constraints on the global objective function.

   $$\begin{aligned} &\max_{\alpha \in A} f(\alpha, \delta) \\ &\text{subject to } g_i(\alpha) \leq T_i \ \forall i \in I \end{aligned} \tag{2}$$

2. Multi-objective Optimization:

   A) Scalarization Methods: In this approach, we use a parameterized aggregation function h to transform the multi-objective optimization problem into a single-objective optimization problem. The function h can be a weighted sum, a weighted exponential sum, a weighted min-max or a weighted product. However, in most situations, finding all pareto optimal solutions in solving this problem with a fixed setting of the weights is not possible. Therefore, the problem is solved for multiple values of the vector w, which requires multiple optimization runs. [Hsu *et al.*, 2018] proposes to use the weighted sum as the objective function. The proposed formulation of this function is described by equation 3. *ACC* refers to the accuracy metric, *E* refers to the energy consumed by the architecture α and *w* is a learned parameter to adjust the effect of the energy on the reward function.

   $$\max_{\alpha \in A} \ w \cdot ACC(\alpha, \delta) - (1 - w) \cdot E(\alpha) \tag{3}$$

   Since the accuracy is not differentiable, some search strategies, including gradient-based once [Cai *et al.*, 2018], use the cross-entropy to which they add the hardware constraint.

   B) NSGA-II: An alternative approach is to use the elitist evolutionary algorithm NSGA-II [Deb *et al.*, 2002]. Hardware-aware NAS works [Lu *et al.*, 2019b; Chu *et al.*, 2019] have been using NSGA-II algorithm to ensure the exploration of diverse architectures in the search space. Moreover, NSGANet [Lu *et al.*, 2019b] uses Bayesian Optimization to profit from search history. MoreMNAS [Chu *et al.*, 2019] uses a hybrid search strategy combining NSGA-II with reinforcement learning to regulate random mutations.

   $$\max_{\alpha \in A} \ f_1(\alpha, \delta), f_2(\alpha, \delta), ..., f_n(\alpha, \delta) \tag{4}$$

4 Hardware Search Space

When the HW-NAS targets a single hardware platform with multiple configurations, the NAS process includes a new component that we call the Hardware Search Space (HSS), as shown in figure 1: "Single Target, Multiple Platform Configurations". This section describes this search space's characteristics and proposes a taxonomy based on the existing works. We refer the readers to [Wistuba *et al.*, 2019] to read more about the architecture search space.

First of all, we need to discuss the different hardware platforms used for DL applications. We classify them according to these three categories.

Server Processors. This type of platforms is found in cloud data centers, on-premise data centers, edge servers, or supercomputers. They provide abundant computational resources and can vary from CPUs, GPUs, FPGAs and ASICs. When available, machine learning researchers focus on accuracy. Many HW-NAS works target server processors to speed up the training process.

Mobile Devices. With billions of users worldwide, ML researchers' focus has shifted to enable fast and efficient DL on mobile devices. As these devices are heavily constrained with respect to their memory and computational capabilities, ML researchers' objective becomes to assess the trade-off between accuracy and efficiency. Many HW-NAS algorithms target smartphones including FBNet [Wu *et al.*, 2019] and ProxylessNAS [Cai *et al.*, 2018].

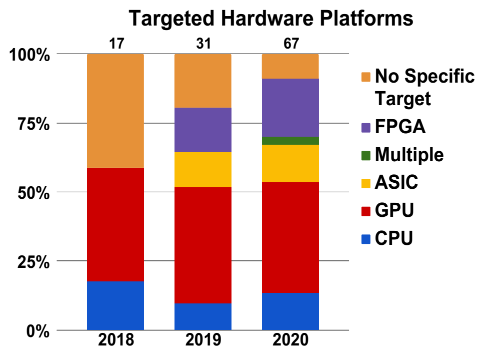

Figure 2: Statistics on the targeted platforms. *No specific target* categorizes the HW-NAS that incorporate model size or number of parameters in their objective function. *Multiple* categorizes the HW-NAS that targets various platforms at ones.

Tiny Devices . The strong growth in the use of microcontrollers and IoT applications gave rise to TinyML [TX Zhuo and Collins, 2020]. TinyML refers to all machine learning algorithms dedicated to tiny devices, i.e., capable of on-device inference at extremely low power. One relevant HW-NAS method that targets tiny devices is MCUNet [Lin *et al.*, 2020], which optimizes the search space and handles various constraints such as device, latency, energy, memory under low search costs. Thanks to the efficient search, MCUNet is the first to achieve >70% ImageNet top-1 accuracy on an off-the-shelf commercial microcontroller.

Figure 2 shows the types of hardware targeted by HW-NAS from 2017. We can see an increasing variety of platforms with the rise of domain-specific processors and FPGAs for inference. *'no specific target'* means that the hardware metrics are either the FLOPs or the number of parameters.

The HSS generates different hardware specifications and optimizations by applying various algorithmic transformations to fit the hardware design. Although the co-exploration is effective, it increases the search space-time complexity significantly as the search strategy looks for different pairs of (architecture, platform configuration).

Hardware Search Space (HSS) can be further categorized as follows.

Parameter-based. A set of different parameter configurations formalizes the search space. If we take FPGAs as an example, their design space may include IP instance categories, IP reuse strategies, quantization schemes, parallel factors, data transfer behaviours, tiling parameters, and buffer sizes. **FNAS** [Jiang *et al.*, 2019b] and **FNASs** [Yu *et al.*, 2019] are two HW-NAS that builds on this representation. The authors in [Jiang *et al.*, 2019a; Lu *et al.*, 2019a] used a multi-FPGA hardware search space. The search consists of dividing the architecture into pipeline stages that can be assigned to an FPGA according to its memory and DSP slices, in addition to applying an optimizer that adjusts the tiling parameters. Another example is [Mohamed *et al.*, 2020], where the adopted approach takes the global structure of an FPGA and adds all possible parameters to its hardware search space including the input buffer depth, memory interface width, filter size and ratio of the convolution engine.

Template-based. In this category, the search space is defined as a set of pre-configured templates. For example, **NASAIC** [Yang *et al.*, 2020] integrates NAS with Application-Specific Integrated Circuits (ASIC). Their search space includes templates of several existing successful designs. The goal is to find the best model with the different possible parallelizations among all templates. In addition to the tiling parameters and bandwidth allocation, the authors in [Jiang *et al.*, 2020a] define a set of FPGA platforms, and the search finds a coupling of the architecture and FPGA platform that fits a set of pre-defined constraints (e.g., max latency 5ms)

5 Hardware Metrics Collection

In this section, we will review the methods to collect hardware performance. First of all, let's discuss the different metrics used to judge if a deep learning model is efficient or not.

FLOPs. HW-NAS methods presented in 2016 and 2017 use [Smithson *et al.*, 2016; Gordon *et al.*, 2017] as an analytical function to minimize the number of parameters and the number of FLOPs. Such strategies presume that the number of operations is positively correlated with the execution time. However, recent work has shown that two models may have the same number of FLOPs, but different latencies [Bouzidi *et al.*, 2021; Zhang *et al.*, 2020; Wang *et al.*, 2020a]. For example, NASNet-A and MobileNetV1 have about the same number of FLOPs, but NASNet-A may have slower latency due to its hardware-unfriendly structure. Therefore, using FLOPs as a hardware cost metric is not effective and can return sub-optimal architectures.

Latency. Low-latency architectures at inference time are crucial on edge scenarios where the edge devices impose a constraint on the latency. Therefore, many works consider latency in their objective function and attempt a trade-off between inference time and accuracy.

Energy Consumption. Energy is often profiled by NVIDIA's given hardware platform profilers, such as nvprof. We can formalize energy either as peak or average power usage, and the two metrics are used by various HW-NAS works, including [Hsu *et al.*, 2018; Yang *et al.*, 2018; Gong *et al.*, 2019].

Area. Another metric that is of concern to chip manufacturers is the chip's area. The aim is to get the smallest processor to run the best model. [Yang *et al.*, 2020] uses MAESTRO [Kwon *et al.*, 2019] to profile the area and power consumption. They posit that the area of the circuit is also a good indicator of static power consumption. These two values are correlated.

Memory Footprint. This metric corresponds to the exact amount of memory a DL model uses. Rather than calculating the number of parameters an architecture has, one can profile the memory footprint and use it as an objective term. To reduce the memory footprint in edges devices, model compression techniques [Deng *et al.*, 2020] are usually applied. We can classify the types of methods that collect the hardware metrics into four categories:

1. *Real-time Measurements* in which the explored models are executed on the target hardware while searching.

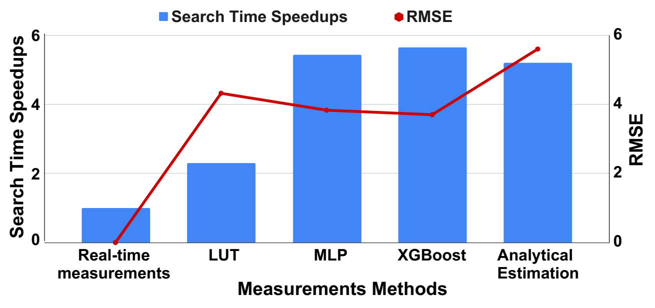

Figure 3: Comparison of hardware cost measurement methods. LUT stands for Look Up Table. The speedups are calculated w.r.t the real-time measurements.

2. *Lookup Table Models* where a lookup table is created beforehand and filled with each operator hardware metrics on the targeted hardware. Once the search starts, the system calculates the overall cost from the lookup table.
3. *Analytical Estimation* which consists of computing a rough estimate using the processing time, the stall time, and the starting time.
4. *Prediction Model* where we build an ML model to predict the cost using architecture and dataset features.

Real-world measurements provide high accuracy in measuring the hardware efficiency of an architecture. MnasNet [Tan *et al.*, 2018] uses this method in the exploration. It achieves 75.2% top-1 accuracy with 78ms latency on a Pixel phone platform, which is 1.8x faster than MobileNetV2 with 0.5% higher accuracy. However, this method considerably slows down the search algorithm by averaging hundreds of runs to get precise measurements. Additionally, this strategy is not scalable and requires that all the hardware platforms are available. That's why many works tend to use a prediction model [Cai *et al.*, 2018; Zhang *et al.*, 2020; Jiang *et al.*, 2019a; Lu *et al.*, 2019a; Zhang *et al.*, 2019; Hao *et al.*, 2019] or a pre-collected lookup table [Wu *et al.*, 2019; Mohamed *et al.*, 2020; Jiang *et al.*, 2020a] or computing an analytical estimation [Zhang *et al.*, 2019; Marchisio *et al.*, 2020]. Although these techniques are efficient, they require hardware knowledge to build the models.

To fairly compare each method's accuracy, we computed, in figure 3, each architecture's latency in NAS-Bench-101 [Ying *et al.*, 2019] and compare it to the real-time measurements, according to three collecting methods: lookup table, prediction model and analytical estimation (Figure 3). For the lookup table, we calculated each operator's latency in the cell of the benchmark. When a cell is generated, we sum the constructing operators' latency, and we get the whole cell's latency. For the prediction model, we used two models: a simple MLP and XGBoost, both trained on a portion of the benchmark's real-time measurements. We choose these two methods because they are both used by popular HW-NAS in [Cai *et al.*, 2018] and [Wu *et al.*, 2019] respectively. Lastly, for the analytical estimation, we computed the number of MAC for the cell and multiplied that by the latency of one multiply-add tensor instruction. We ran this experiment on a Tesla K80 GPU, the prediction model MLP was trained for 50 epochs with early stopping. NAS-Bench-101 defines more than 400k cells. The search algorithm used to calculate the search time is an evolutionary algorithm based on the benchmark's validation accuracy and the different methods' latency. As expected, the analytical estimation does not produce good results compared to the prediction models or the lookup table method. We expected the analytical estimation to be the fastest, but looping through the model graph to extract the number of MAC is time consuming. Even with a simple XGBoost, the prediction models give the best results and accelerate the search more than five times compared to the real-time measurements. However, it is worth mentioning that prediction methods require large dataset and consume a considerable preparation time in order to collect the data and train the model.

6 Challenges and Limitations

In this section, we set out the critical obstacles that keep from exploiting the HW-NAS's full potential. The majority of problems often apply to general NAS approaches as well.

6.1 Benchmarking and Reproducibility

To compare different search algorithms, we need to unify the search spaces. To that end, many benchmarks were released [Ying *et al.*, 2019; Siems *et al.*, 2020]. These benchmarks present a dataset of architectures with their respective accuracy and latency metrics. However, two limitations are relevant in the context of HW-NAS. (1) most of them define ConvNets architectures with a cell-based approach, i.e., the architectures aren't flexible enough for different hardware platforms, (2) most of the performance are measured on CPU or GPU which isn't suited for HW-NAS. A more recent paper introduced the first hardware-aware NAS benchmark, **HW-NAS-Bench** [Li *et al.*, 2021]. This work extends the number of hardware metrics and records the latency, the energy consumed on six hardware devices, including commercial edge devices, FPGA, and ASIC. Its search space is a combination of FBNet [Wu *et al.*, 2019] search space and NAS-Bench-201 [Dong and Yang, 2020].

6.2 Transfer Learning

Transfer Learning is one of the fundamental concepts in DL. It consists on training a model on a proxy dataset, then fine-tuning its weights on the target dataset. Using this technique, better precision with the final model is obtained. However, not all architectures are easily transferable from one dataset to another. Cell-based search space [Zoph *et al.*, 2018] enhance transferability by adding more repeating cells to the model and updating a small number of hyperparameters. However, stacking the same blocks seems to be not efficient when incorporating hardware constraints. As MNASNet [Tan *et al.*, 2018] argued, restricting cell diversity is critical for achieving high accuracy and low latency on mobile settings.

Many NAS works [Liu *et al.*, 2018; Nayman *et al.*, 2019] have included dedicated evaluations of the transferability of their final model. XNAS [Nayman *et al.*, 2019] transferred their final cell structure on six popular classification benchmarks surpassing other conventional NAS methods while taking into account the hardware constraints.

NAT [Lu *et al.*, 2020] leverages the NAS process to find transferable weights directly and get rid of the fine-tuning stage. NAT's key idea is that they start from a supernetwork and adaptively modify it to obtain a task-specific supernetwork. This latter can then be used directly to search for architectures within one task without the additional training cost. They demonstrated the efficacy of NAT on 11 benchmarks, including ImageNet, on mobile settings.

6.3 Transferability of HW-NAS Across Multiple Platforms

HW-NAS suffers from conditional optimality due to the variety of existing devices. Ideally, we should design different architectures for different platforms. However, in real situations, given the prohibitive cost of the search and the cost of training on multiple architectures, we often resort to designing one architecture and deploying it anywhere. Transferring a model from one platform to another or producing hardware transferable models via the NAS process is an exciting challenge for HW-NAS. In the following section, we discuss two popular approaches. Each approach has its pros and cons, as discussed below.

Transfer the entire NAS process. It consists of re-executing the single-target HW-NAS to suit a new target. This includes finding a collection method that is scalable and flexible to multiple hardware platforms. When using real-world measurements, [Han *et al.*, 2019] ran the NAS search for three hardware platforms. However, using real-world measurements considerably slows down the search algorithm and requires the targeted hardware's availability during the search time. Using other collection methods such as the lookup table or the prediction model, we'll need to collect data from the new platforms by rerunning the entire set of operators. For example, Once-for-all [Cai *et al.*, 2020] created a lookup table with the reported inference latency on each tested hardware platforms. According to the used measurement method, transferring the NAS process to target another platform is increasingly difficult and not scalable.

Transfer the final model. This approach consists of finding the best model for one hardware platform and then specialize it for another one. It has been proposed by [Cai *et al.*, 2020; Cai *et al.*, 2018; Nayman *et al.*, 2019]. The specialization is usually done by compressing the model using quantization, enabling the model to fit in tiny devices. However, specialization presents the following challenges:

- An operator may be efficient for one platform and less efficient in another: In [Chu *et al.*, 2020], the authors argued that separable convolutions give great results when ran on GPUs but perform badly on CPUs.
- Limits of the compression methods: We consider here the quantization and pruning. We know that theoretically, the compression ratio has a threshold that cannot be surpassed for these two methods. For example, quantizing a model implies encoding its activations and weights into the minimum possible bit length. Theoretically, this length is 2 (one bit).

6.4 Outlook and Future Directions

HW-NAS suffers from the same limitations as NAS. The main one is the cost of the search algorithm. The current popular way to speed up the process is to use differentiable NAS and create a supernetwork that can represent the whole search space. This supernetwork is trained once to get the weights of all the sub-networks and thus avoids training each sampled architecture. This technique reduces the search time from several days to hours. However, a major disadvantage of this network is the restriction it makes on the targeted task and domain. More research is needed to explore efficient ways and tricks to make HW-NAS more practical and more amenable to a diverse set of tasks and domains. This would especially be useful in commercial settings.

There is no doubt that the future of mobile and handheld devices is AI. AI-focused mobile chips from top manufacturing companies like Apple, Samsung, Huwaei and others are making their ways into the mainstream. These devices use SoCs that take advantage of multiple platforms (e.g., GPUs, CPUs and NPU in the same chip). However, this heterogeneity of platforms needs to be well understood to speed up the inference time of deep neural networks [Wang *et al.*, 2020b]. This motivates further the research community to simultaneously explore both the architecture search space and the hardware design space to identify the best neural architecture and hardware pairs that maximize both test accuracy and hardware efficiency. Such co-exploration will be key to allow designing architectures that can be deployed efficiently on a variety of platforms: data center, edge, mobile, and embedded. The HW-NAS methods should co-explore compression techniques and models search spaces to find the best trade-off between model size, inference time and accuracy.

HW-NAS should also look at exploring neural architecture search with emerging computing paradigms such as in-memory-computing [Jiang *et al.*, 2021]. These new non-von-Neumann paradigms present novel solutions to AI computing based on emerging nano-devices called Phase-Change Memory (PCM) [Sebastian *et al.*, 2019]. The optimization space spans multiple design points that range from device types, to circuit topologies, to device non-idealities and variations, to neural architectures.

While DNNs have clear commercial use cases, the next AI breakthrough may require an entirely different combination of algorithm, hardware and software. HW-NAS offers a paradigm that opens up the design space and pushes forward the Pareto frontier between hardware efficiency and model accuracy for efficient and improved hardware/software co-design, hence pushing AI to its next frontiers.

7 Conclusion

Efficient deep learning techniques are increasingly attracting both academia and industry researchers. This paper explores one of the trending approaches to build a model that respects the trade-off between accuracy and hardware constraints. We point out several potential improvements and new directions in this area, such as accounting for new hardware platforms (e.g., heterogeneous systems-on-chip and in-memory computing) and extending it to new application domains.

References

[Bouzidi *et al.*, 2021] Halima Bouzidi, Hamza Ouarnoughi, Smaïl Niar, and Abdessamad Ait El Cadi. Performance prediction for convolutional neural networks on edge gpus. In *CF '21: Computing Frontiers Conference*, pages 54–62. ACM, 2021.

[Cai *et al.*, 2018] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *CoRR*, abs/1812.00332, 2018.

[Cai *et al.*, 2020] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *8th International Conference on Learning Representations, ICLR*, 2020.

[Chu *et al.*, 2019] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Hailong Ma. Multi-objective reinforced evolution in mobile neural architecture search. *CoRR*, abs/1901.01074, 2019.

[Chu *et al.*, 2020] Grace Chu, Okan Arikan, Gabriel Bender, Weijun Wang, Achille Brighton, Pieter-Jan Kindermans, Hanxiao Liu, Berkin Akin, Suyog Gupta, and Andrew Howard. Discovering multi-hardware mobile models via architecture search. *CoRR*, abs/2008.08178, 2020.

[Deb *et al.*, 2002] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[Deng *et al.*, 2020] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE*, 108(4):485–532, 2020.

[Dong and Yang, 2020] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

[Gong *et al.*, 2019] Chengyue Gong, Zixuan Jiang, Dilin Wang, Yibo Lin, Qiang Liu, and David Z Pan. Mixed precision neural architecture search for energy efficient deep learning. In *ICCAD*, 2019.

[Gordon *et al.*, 2017] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. *CoRR*, abs/1711.06798, 2017.

[Han *et al.*, 2019] Song Han, Han Cai, Ligeng Zhu, Ji Lin, Kuan Wang, Zhijian Liu, and Yujun Lin. Design automation for efficient deep learning computing. *CoRR*, page arXiv:1904.10616, 2019.

[Hao *et al.*, 2019] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge. In *Proceedings of the 56th Annual Design Automation Conference*, 2019.

[Hsu *et al.*, 2018] Chi-Hung Hsu, Shu-Huan Chang, Da-Cheng Juan, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Shih-Chieh Chang. MONAS: multi-objective neural architecture search using reinforcement learning. *CoRR*, abs/1806.10332, 2018.

[Jiang *et al.*, 2019a] Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Shouzhen Gu, Yiyu Shi, and Jingtong Hu. Hardware/software co-exploration of neural architectures. *CoRR*, abs/1907.04650, 2019.

[Jiang *et al.*, 2019b] Weiwen Jiang, Xinyi Zhang, Edwin Hsing-Mean Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. In *Proceedings of the 56th Annual Design Automation Conference*. ACM, 2019.

[Jiang *et al.*, 2020a] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start, 2020.

[Jiang *et al.*, 2020b] Yuhang Jiang, Xin Wang, and Wenwu Zhu. Hardware-aware transformable architecture search with efficient search space. In *IEEE International Conference on Multimedia and Expo, ICME*, 2020.

[Jiang *et al.*, 2021] Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, and Yiyu Shi. Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Trans. Computers*, 2021.

[Kwon *et al.*, 2019] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[Li *et al.*, 2021] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021.

[Lin *et al.*, 2020] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. In *Advances in Neural Information Processing Systems (NeurIPS'20)*, 2020.

[Liu *et al.*, 2018] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[Lu *et al.*, 2019a] Qing Lu, Weiwen Jiang, Xiaowei Xu, Yiyu Shi, and Jingtong Hu. On neural architecture search for resource-constrained hardware platforms. *CoRR*, abs/1911.00105, 2019.

[Lu *et al.*, 2019b] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and

Wolfgang Banzhaf. Nsga-net: Neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019.

[Lu *et al.*, 2020] Zhichao Lu, Gautam Sreekumar, Erik D. Goodman, Wolfgang Banzhaf, Kalyanmoy Deb, and Vishnu Naresh Boddeti. Neural architecture transfer. *CoRR*, abs/2005.05859, 2020.

[Marchisio *et al.*, 2020] Alberto Marchisio, Andrea Massa, Vojtech Mrazek, Beatrice Bussolino, Maurizio Martina, and Muhammad Shafique. Nascaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks. In *ICCAD*, 2020.

[Mohamed *et al.*, 2020] Abdelfattah Mohamed, Lukasz Dudziak, Thomas C. P. Chau, Royson Lee, Hyeji Kim, and Nicholas D. Lane. Best of both worlds: Automl codesign of a cnn and its hardware accelerator. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, 2020.

[Nayman *et al.*, 2019] Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik-Manor. XNAS: neural architecture search with expert advice. In *Advances in Neural Information Processing System 32 (NeurIPS)*, pages 1975–1985, 2019.

[Pham *et al.*, 2020] Hieu Pham, Qizhe Xie, Zihang Dai, and Quoc V. Le. Meta pseudo labels. *CoRR*, abs/2003.10580, 2020.

[Sebastian *et al.*, 2019] Abu Sebastian, Irem Boybat, Martino Dazzi, Iason Giannopoulos, Vara Prasad Jonnalagadda, Vinay Joshi, Geethan Karunaratne, Benedikt Kersting, Riduan Khaddam-Aljameh, S. R. Nandakumar, Anastasios Petropoulos, Christophe Piveteau, Theodore Antonakopoulos, Bipin Rajendran, Manuel Le Gallo, and Evangelos Eleftheriou. Computational memory-based inference and training of deep neural networks. In *2019 Symposium on VLSI Circuits*. IEEE, 2019.

[Siems *et al.*, 2020] Julien Niklas Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. {NAS}-bench-301 and the case for surrogate benchmarks for neural architecture search, 2020.

[Smithson *et al.*, 2016] Sean C. Smithson, Guang Yang, Warren J. Gross, and Brett H. Meyer. Neural networks designing neural networks: Multi-objective hyper-parameter optimization. *CoRR*, abs/1611.02120, 2016.

[Tan *et al.*, 2018] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018.

[TX Zhuo and Collins, 2020] Fika Ventures TX Zhuo and Huston Collins. Why tinyml is a giant opportunity. https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/, 2020. Accessed: 2020-01-20.

[Wang *et al.*, 2020a] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[Wang *et al.*, 2020b] Siqi Wang, Anuj Pathania, and Tulika Mitra. Neural network inference on mobile socs. *IEEE Des. Test*, 2020.

[Wistuba *et al.*, 2019] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *CoRR*, abs/1905.01392, 2019.

[Wu *et al.*, 2019] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Conference on Computer Vision and Pattern Recognition, CVPR*. Computer Vision Foundation / IEEE, 2019.

[Yang *et al.*, 2018] Tien-Ju Yang, Andrew G. Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *ECCV 15th European Conference Computer Vision*, 2018.

[Yang *et al.*, 2020] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. Co-exploration of neural architectures and heterogeneous ASIC accelerator designs targeting multiple tasks. In *57th Design Automation Conference, DAC*. IEEE, 2020.

[Ying *et al.*, 2019] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML*, Proceedings of Machine Learning Research. PMLR, 2019.

[Yu *et al.*, 2019] Ye Yu, Yingmin Li, Shuai Che, Niraj K. Jha, and Weifeng Zhang. Software-defined design space exploration for an efficient AI accelerator architecture. *CoRR*, abs/1903.07676, 2019.

[Zhang *et al.*, 2019] Xinyi Zhang, Weiwen Jiang, Yiyu Shi, and Jingtong Hu. When neural architecture search meets hardware implementation: from hardware awareness to co-design. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019.

[Zhang *et al.*, 2020] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search. In *Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE, 2020.

[Zoph *et al.*, 2018] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition, 2018.