Fast Hardware-Aware Neural Architecture Search

Li Lyna Zhang[1] Yuqing Yang[1] Yuhang Jiang[2] Wenwu Zhu[2] Yunxin Liu[1]
[1]Microsoft Research [2]Tsinghua University
{lzhani, yuqing.yang, yunxin.liu}@microsoft.com
jhy17@mails.tsinghua.edu.cn, wwzhu@tsinghua.edu.cn

Abstract

Designing accurate and efficient convolutional neural architectures for vast amount of hardware is challenging because hardware designs are complex and diverse. This paper addresses the hardware diversity challenge in Neural Architecture Search (NAS). Unlike previous approaches that apply search algorithms on a small, human-designed search space without considering hardware diversity, we propose HURRICANEthat explores the automatic hardware-aware search over a much larger search space and a two-stage search algorithm, to efficiently generate tailored models for different types of hardware. Extensive experiments on ImageNet demonstrate that our algorithm outperforms state-of-the-art hardware-aware NAS methods under the same latency constraint on three types of hardware. Moreover, the discovered architectures achieve much lower latency and higher accuracy than current state-of-the-art efficient models. Remarkably, HURRICANE achieves a 76.67% top-1 accuracy on ImageNet with a inference latency of only 16.5 ms for DSP, which is a 3.47% higher accuracy and a 6.35× inference speedup than FBNet-iPhoneX, respectively. For VPU, we achieve a 0.53% higher top-1 accuracy than Proxyless-mobile with a 1.49× speedup. Even for well-studied mobile CPU, we achieve a 1.63% higher top-1 accuracy than FBNet-iPhoneX with a comparable inference latency. HURRICANE also reduces the training time by 30.4% compared to SPOS.

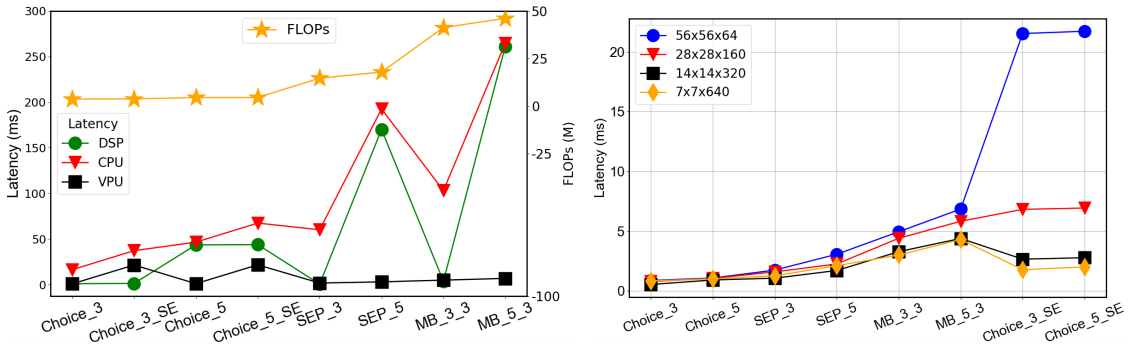
1. Introduction

Neural Architecture Search (NAS) is a powerful mechanism to automatically generate efficient Convolutional Neural Networks (CNN) without requiring huge manual efforts of human experts to design good CNN models [10, 29, 30, 24, 9, 1]. However, searching accurate and fast CNN for the massive smart devices is difficult by current NAS approaches due to the emergence of massive types of hardware devices and the intrinsic huge search cost.

Unaware of Hardware Diversity. Most of previous NAS methods focus on searching for the most accurate models. The common effort to guarantee the inference efficiency (e.g., model inference latency on real hardware) is to limit the model's FLOPs [1]. Some recent hardware aware NAS methods [9, 24, 5, 25] consider model-inference performance but they only aim at the same type of hardware, smart phones from different manufacturers but all based on ARM processors. Also, such hardware-aware approaches [4, 25, 23] use an identical manually elaborated search space for different types of hardware platforms. However, the emerging massive smart devices (e.g., IoT devices) are equipped with very diverse processors, such as GPU, DSP, FPGA, and various AI accelerators that have fundamentally different hardware designs. Such a big *hardware diversity* makes FLOPs an improper metric to predict model-inference performance and manual-designed search space not ideal for searching efficient models. As a result, it calls for new methods to automatically generate the hardware-aware search spaces that leverage the characteristics of every hardware platform and relax the reliance on human design effort.

To demonstrate it, we conduct an experiment to measure the performance of a set of widely used neural network operators (a.k.a. operations) on three types of mobile processors: Hexagon™ 685 DSP, Snapdragon 845 ARM CPU, and Movidius™ Myriad™ X Vision Processing Unit (VPU). Figure 1 shows the results and we make the following key observations. First, from Figure 1(a), we can see that even the operators have similar FLOPs, the same operator may have very different inference latency on different processors. For example, the latency of operator *Choice_3* is almost the same as *Choice_3_SE* on the DSP, but the difference on the VPU is more than 24×. Therefore, FLOPs is not the right metric to decide the inference latency on differ-

[1]In this paper, the definition of $FLOPs$ follows [28], i.e., the number of multiply-adds.



(a) Latency and FLOPs on different hardware

(b) Latency in different feature map sizes on VPU

Figure 1. Performance of widely used operators in NAS (c.f. Table 1). ***(a)***: Latency and FLOPs on three types of hardware: (1) DSP (Hexagon™ 685 DSP), (2) CPU (Snapdragon 845 ARM CPU), (3) VPU (Movidius™ Myriad™ X Vision Processing Unit). The input/output feature maps are all the same, equal to $56^2 \times 64$. ***(b)***: Latency in different input feature map sizes on VPU.

ent hardware. Second, the relative effectiveness of different operators on different processors is also different. For example, operator *Choice_3* has the smallest latency on the ARM CPU, but operator *SEP_3* has the smallest latency on the DSP. Thus, different processors should choose different operators for the best trade-off between model accuracy and inference latency. Furthermore, as shown in Figure 1(b), the computational complexity and latency of the same operator are also affected by the execution *context*, such as input feature map shapes, number of channels, etc. Such a context is determined by which layer the operator is placed on. That is, even for the same type of hardware, optimal operators may change at different layers of the network. Thus, it is difficult to cover hardware diversity using manually-designed search space.

Motivated by these observations, we argue that there is no one-size-fits-all model for different hardware platforms, and thus propose HURRICANE (shown in Figure 2) to generate different models tailored for different types of hardware. To cover the diversity of hardware platforms, we construct a much larger candidate operators pool (32 in our implementation) and propose a search space generation approach to automatically generate a *hardware-specialized search space* for each type of hardware. The key point here is to include much more hardware-efficient and accurate candidate architectures in search space without increasing the search cost. Our mechanism is based on profiled real performance on the target hardware as shown in Sec 3.1.

Moreover, we propose a two-stage search algorithm for the one-shot NAS[2] to further reduce the intensive search cost. Unlike previous works that search operators for all layers at one time, we search the complete architecture by a sequence of simpler searching of sub-networks. The method is inspired by the *layer diversity* (different layers have different impacts on inference latency [10] and model accuracy [27, 13]), we demonstrate that exploring more architecture selections in the layers close to classification output may help find better architectures with the limited sampling budget, and limiting the latency in the layers close to data input is critical to search for low-latency models.

In summary, we propose a novel approach that enables NAS to quickly search the accurate and fast architectures for different type of hardware platforms (not only the ARM CPU). The key technical innovations are: (i) automatically generate more effective search space for target hardware with minimal human design, (ii) explore more architectures in deeper layers and reduce search space size. We conduct comprehensive experiments on ImageNet 2012 and OUI-Adience-Age datasets over three hardware platforms (Figure 1(a)). Under all the three platforms, HURRICANE consistently achieves the better accuracy than state-of-the-art hardware-aware NAS methods with the same latency constraints. Specifically, HURRICANE improves the top-1 ImageNet accuracy by an average of 1.83% than Proxyless [4], and 1.35% than SPOS [9]. In addition, the searched architectures also outperform the current state-of-the-art efficient models. Remarkably, HURRICANE reduces the latency by 6.35× on DSP compared to FBNet-iPhoneX and 1.49× on VPU compared to Proxyless-mobile, respectively. Finally, HURRICANE reduces the training time by 30.4% on ImageNet comparing to SPOS.

2. Related Work

Hardware aware NAS. Recent methods [24, 25, 4, 9, 1] adopt a layer-level hierarchical search space with a fixed macro-structure allowing different layer structures at different resolution blocks of a network. The goal becomes searching operators for each layer so that the architecture achieves competitive accuracy under given constraints. To search hardware-efficient architectures, the search spaces have been built on increasingly more efficient building

[2]In this paper we adopt one-shot NAS because of its simplicity, however, our acceleration could also be combined with other NAS methods.

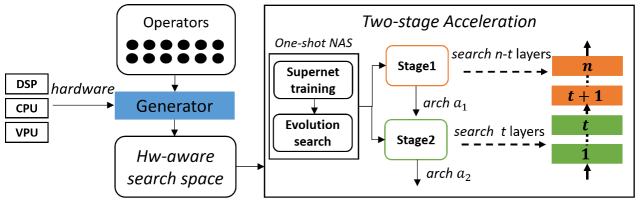


Figure 2. HURRICANE constructs hardware-specialized search space (by latency constraints) that contains more efficient architectures than previous NAS, and employs a two-stage search algorithm to reduce the search cost.

blocks. [24, 4, 23] built upon the MobileNetV2 [20] structure (*MB_k_e*). [25, 9] built search space by ShuffleNetV1 [28] and ShuffleNetV2 [15] (*Choice_k*). As these structures are primarily designed for mobile CPU, the efficiency of such manually-designed search space is unknown for other hardware.

To measure the model efficiency, many NAS methods [29, 30] adopt the hardware-agnostic metric FLOPs. However, architecture with lower FLOPs is not necessarily faster [22]. Recently, gradient-based methods [4, 23, 25] adopt direct metrics such as measured latency but only for mobile CPUs. They profile every operator's latency and build prediction model. The latency is then viewed as a differentiable regularization loss. However, the multi-objective loss is not optimal because accuracy changes much more dramatically with latency for small models, as [10] pointed out. Instead, we follow One-Shot NAS [9, 5] and apply the latency constraints directly.

One-Shot NAS. Starting from ENAS [16], weight sharing became popular as it accelerates the search process and makes search cost feasible. Recent one-shot methods encode the search space into an over-parameterized supernet, where each path is a stand-alone model. During the supernet training, architectures are sampled by different proxies (e.g., reinforcement learning) with weights updated. However, SPOS [9] and FairNAS [5] observe that such coupled architecture search and weight sharing could be problematic to fairly evaluate the performance of candidate architectures. SPOS [9] trains the supernet by an uniform path sampling method, and applies an evolutionary algorithm to efficiently search architectures directly without any fine tuning. As it's easy to train and fast to search, our work is built upon SPOS [9] with their officially open-sourced implementation [19].

3. Methodology

In this paper, HURRICANE aims to search the following architectures for a given hardware platform h (any of CPU, DSP, NPU, VPU, etc.) and the latency constant $\tau_c^{(h)}$:

$$\begin{aligned} \max \quad & \mathrm{ACC}_{\mathrm{val}}(a) \\ \text{s.t.} \quad & \tau(a,h) \le \tau_c^{(h)} \end{aligned} \tag{1}$$

We seek to find an architecture a that achieves the maximum accuracy $\mathrm{ACC}_{\mathrm{val}}(a)$ on the validation set while the inference latency $\tau(a,h)$ is under the constraint $\tau_c^{(h)}$.

3.1. Hardware-aware Search Space

We follow the design of layer-level hierarchical search space in recent hardware-aware NAS [25, 24]. Besides first and last three fixed layers, each learnable layer can choose an operator from a candidate pool. For each target hardware, we encode the specialized search space into a over-parameterized supernet for one-shot NAS.

Diverse Candidate Operators Pool. Compared with the small operator pool in previous works, we employ a much bigger pool of candidate operators from the primary building blocks of off-the-shelf networks. In our experiment, FLOPs and memory access cost of an operator leverage different impacts to the latency on three hardware platforms. As a result, our pool contains up to 32 operators (detailed in Table 2) with different levels of computation and memory complexity. They are built upon the following 4 basic structures from current efficient models:

- **SEP**: depthwise-separable convolution. Following DARTS [14], we applied the depthwise-separable convolution twice. *SEP* has a larger FLOP count than others, but less memory access complexity.
- **MB**: mobile inverted bottleneck convolution in MobileNetV2 [20]. *MB* has a medium memory access cost due to its shortcut and add operation. Its computation complexity is decided by the kernel size *k* and expansion rate *e*.
- **Choice**: basic building block in ShuffleNetV2 [15]. Following [9], we add a similar operator *ChoiceX*. *Choice* and *ChoiceX* have much smaller FLOPs than the others, but the memory complexity is high due to the channel split and concat operation.
- **SE**: squeeze-and-excitation network [11]. To balance the impacts in latency and accuracy, we follow the settings in MobileNetV3 [10]. We set the reduction ratio r to 4, and replace the original sigmoid function with a hard version of swish $hswish[x] = x\frac{ReLU6(x+3)}{6}$. We apply *SE* module to the above operators and generate new operators. The computation complexity of *SE* is decided by its insert position, while the memory access cost is relatively lower.

Hardware Aware Search Space Generation. The enrichment of candidate operators covers the diversity of hardware platforms. However, doing so increases the search

Output shape	Layer	DSP	CPU	VPU
$56^2 \times 64$	1-4	SEP_3, Choice_3 MB_3_1, ChoiceX	Choice_3, Choice_3_SE MB_3_1, ChoiceX	Choice_3, Choice_5 Choice_7, SEP_3
$28^2 \times 160$	5-8	Choice_3, ChoiceX MB_3_1, Choice_3_SE	Choice_3, ChoiceX Choice_5, MB_3_1	Choice_3, Choice_5 Choice_7, ChoiceX
$14^2 \times 320$	9-16	Choice_3, Choice_3_SE ChoiceX, MB_3_1	Choice_3, Choice_3_SE Choice_5 ,Choice_5_SE	Choice_3, Choice_5 Choice_7, ChoiceX
$7^2 \times 640$	17-20	Choice_3, Choice_3_SE ChoiceX ,MB_3_1, MB_3_3	Choice_3, Choice_5 Choice_3_SE, Choice_7, MB_5_1	Choice_3, Choice_5 Choice_7, MB_3_1, MB_7_1

Table 1. Hardware-aware search space for each mobile hardware. For layer at 1-16, it contains 4 operators for selection, for layer 17-20, each layer has 5 operators. The input/output channel and stride settings for each layer are the same with SPOS [9].

Operator	Variable range		Number
	Kernel (k)	Expansion (e)	
SEP_k	3,5,7	-	3
SEP_k_SE	3,5,7	-	3
MB_k_e	3,5,7	1,3,6	9
MB_k_e_SE	3,5,7	1,3,6	9
Choice_k	3,5,7	-	3
Choice_k_SE	3,5,7	-	3
ChoiceX	3	-	1
ChoiceX_SE	3	-	1

Table 2. Candidate operators. For depthwise convolution in each operator, we allow choosing k of 3 , 5, 7. For the expansion ratio e in *MB*, we allow it choosing of 1, 3, 6.

space by many orders of magnitude (e.g., the original large search space size is 10^{18} larger than SPOS), and thus leads to unacceptable search and training cost and may even cause non-convergence problem in one-shot NAS methods.

To reduce the cost while improving the search space efficiency, we propose a layer wise *hardware aware search space generation* approach to generate specialized search space for *every target hardware platform*. Unlike the previous methods that apply same operators for all layers, we select the most efficient operators for every layer by real hardware deployment score. We benchmark all the 32 operators layer-by-layer and sort each layers' candidate operators in non-increasing order of their scores in Equation 2:

$$score_{op}^{(i)} = (F_{op} \times P_{op})^{\alpha}(\tau_{\ell_i(a)=op}(a,h))^{-1} \qquad (2)$$

where F_{op} and P_{op} are the FLOPs count and number of parameters of operator op respectively, $\ell_i(a) = op$ means architecture a whose i-th learnable layer is op. Parameter α is non-negative constant. The score of candidate operator op at the i-th learnable layer ($score_{op}^{(i)}$) considers both representation capacity (approximately) and real hardware performance.

The operators listed upfront will be selected to construct the reduced search space. For each layer, we filter out the first p operators with highest score, and the size of search space would be n^p (n is the number of learnable layers). In our experiment, we choose the top $p = 4$ operators for each layer to keep the similar size with other NAS search space.

Exploring Operator. Inspired by the observations of *layer diversity*, some layers (commonly in the layers close to output) contribute small to the latency (due to the small feature map size) but impact largely on the accuracy. For these learnable layers, we add an extra exploring operator besides the first p operators. Since exploring operator is mainly for better accuracy, its score could be not so top ranked. For our backbone network (shown in Table 1), it is natural to add the exploring operator to the last 4 layers because of their smallest feature map size.

In summary, we construct three different search spaces for our hardware platforms as shown in Table 1. For every specialized search space, it contains $n = 20$ learnable layers, and each layer can choose from 4 or 5 candidate operators from the Table 1. Each search space contains $4^{16} \times 5^4 \approx 2 \times 10^{13}$ possible architectures, which is approximately twice the size of SPOS's search space.

3.2. Two-Stage NAS Acceleration

To search an architecture of n learnable layers, early NAS methods search for one cell structure and repeat it for all layers [29, 18], while the recent NAS methods search operators for the complete architecture [25, 4, 9, 24]. We adopt a different two-stage approach that each stage searches operators for part of the whole architecture. This strategy leverages the layer diversity in accuracy and latency, and further to reduce the one-shot search cost.

Layer Grouping. The phenomenon that different CNN layers reveal different sensitivity to accuracy has been observed in other domains [6, 13, 27]. NAS should take more efforts in searching the ideal operators for the critical layers as the operator selection for non-critical layers impacts less to the final accuracy. However, it's difficult to do the accuracy sensitivity analysis for individual layer in NAS scenario. Fortunately, some previous works [6, 8] have revealed different behaviors between the earlier layers (close to data input) and the later layers (close to classification output) in CNN models. The earlier layers extract low-level features from inputs (e.g., edges and colors), are computa-

tion intensive and demand more data to converge, while the later layers capture high-level class-specific features but are less computation intensive. Inspired by this, our intuition is that operator search for later layers is more critical than earlier layers. To this end, we group the n layers of the CNN model into two groups: the earlier t layers (less critical) and the later $n-t$ layers (more critical).

Two-Stage Search Algorithm. Algorithm 1 illustrates the main procedures of hardware-aware NAS with two-stage search acceleration. Each stage starts with a different winning architecture and runs a one-shot NAS to search the target group of learnable layers. We treat the rest group as the non-active fixed layers and use the corresponding layers' operator of the winning architecture. In the beginning, we set up the initial winning architecture a_{win0} with the operators of the highest scores in every layer (line 4-6).

First, *Stage1* searches the later $n-t$ layers for a_{win0}. We mark the later $n-t$ layers as active and the earlier t layers as non-active. The non-active layers are fixed to the corresponding layer structures of architecture a_{win0} (line 9), while the active layers will be searched from the generated operator list $l_i(\mathcal{A})$ (line 10-12). The one-shot NAS method itself is similar to the work [9], except that we constraint the search space with a hardware latency other than FLOPs (line 14). After a complete process of one-shot search, a new winning architecture a_{win1} would be generated.

Second, *Stage2* starts with the new winning architecture a_{win1} and searches for earlier t layers. The later $n-t$ layers are fixed to the corresponding layer operator of a_{win1} and the earlier t layers are active for another one-shot search. *Stage2* returns the final architecture a_{win2}.

Hyper-Parameter t. The layer grouping boundary t in Algorithm 1 impacts the effectiveness and efficiency of two-stage acceleration. Specifically, two-stage acceleration rolls back to the original one-shot NAS and searches for the complete n learnable layers when $t = 0$, and thus no search cost reduction is achieved. While larger t reduces the supernet training time a lot, it can harm the search for the optimal architectures. In this paper, we set $t = 8$ (only learnable layers counted) according to the natural resolution changes of the supernet. The search space size is reduced by ≈65,000 $\times$. According to our empirical results, two-stage search algorithm achieves better accuracy and promising search time reduction on two datasets when $t = 8$ (c.f. Table 5). We will study more about how to choose t in future works.

Algorithm 1 Hardware-aware NAS with acceleration

Input: hardware h, latency constraint $\tau_c^{(h)}$, hyper-parameter t

1: **function** TWOSTAGECONSTRAINEDNAS(A, t)
2: ▷ $\ell_i(a)$ denotes the i-th layer of architecture a
3: ▷ $\ell_i(A)$ denotes all the candidate operators in the i-th layer of search space A, sorted in non-increasing order of score
4: **for** $i \leftarrow 1$ to n **do**
5: $\ell_i(a_{win}) \leftarrow \ell_i(A)[0]$ ▷ Init with top-1 ranked operator
6: **end for**
7: ▷ Each stage searches 1 group layers by one-shot NAS.
8: **for** $iter \leftarrow 1$ to 2 **do** ▷ Stage1: iter=1, Stage2: iter=2
9: Fixed $\leftarrow [1, t]$ if $iter = 1$, else: $[t+1, n]$
10: **for** $i \leftarrow 1$ to n **do**
11: $\phi_i \leftarrow \{\ell_i(a_{win})\}$ if $i \in$ Fixed, else: $\ell_i(A)$
12: **end for**
13: $A_{iter} \leftarrow \{a \mid \ell_i(a) \in \phi_i, 1 \le i \le n\}$
14: $a_{win} \leftarrow$ CONSTRAINEDONESHOTNAS(A_{iter}, $\tau_c^{(h)}$)
15: **end for**
16: **return** a_{win}
17: **end function**
18: $\mathcal{A}_h \leftarrow$ HWAWARESEARCHSPACE(h, $\tau_c^{(h)}$) ▷ c.f. Sec 3.1
19: $a_{win} \leftarrow$ TWOSTAGECONSTRAINEDNAS($\mathcal{A}_h$, t)
20: retrain a_{win}

4. Evaluation

4.1. Experiment Setup

Hardware Platforms and Measurements. We target three representative mobile hardware that is widely used for CNN deployment: (1) DSP (Qualcomm's Hexagon™ 685 DSP), (2): CPU (Qualcomm's Snapdragon 845 ARM CPU), (3): VPU (Intel's Movidius™ Myriad™ X Vision Processing Unit). To make full utilization of these hardware at inference, we use the particular inference engine provided by the hardware vendor. Specifically, DSP and CPU latency are measured by Snapdragon Neural Processing Engine SDK [17] with int8 and float32 precision, respectively. VPU latency is measured by Intel OpenVINO™ Toolkit [12] with float16 implementation.

Latency Constraints. For better comparison with other works, we set the latency constraints to be smaller than the best latency of models from other works [24, 10, 25, 4], which are 310 ms (CPU), 17 ms (DSP) and 36 ms (VPU).

Hardware-aware One-Shot Search. As shown in Table 1, the hardware aware search space is generated according to the different characteristics of every hardware. The search space is then encoded into a over-parameterized supernet for two-stage acceleration search. Our two-stage search acceleration is built on top of SPOS [9, 19]. Once the supernet training finishes, we perform a 20-iterations evolution search for total 1,000 architectures as SPOS. To avoid measuring the latency of every candidate architecture during search, we build a latency-prediction model [3] with high accuracy: the average estimated error for DSP, CPU, VPU is 4.7%, 4.2%, and 0.08%, respectively.

[3] The prediction model is built with Bayesian Ridge Regression [2]

	Model	Search Method	Target Hardware	FLOPs (#)	DSP (ms)	CPU ‡ (ms)	VPU (ms)	Top-1 Acc (%)
Existing STOA Models	MobileNetV2 [20]	Manual	CPU	300M	10.1	432.4	45.2	72.00
	MobileNetV3-Large1.0 [10]	RL+NetAdapt [26]	CPU	219M	141.6	411.4	72.3	75.20
	MnasNet-A1 [24]	RL	CPU	312M	149.0	1056.1	52.4	75.20
	FBNet-iPhoneX [25]	Gradient	CPU	322M	105.0	313.0	45.6	73.20
	FBNet-S8 [25]	Gradient	CPU	293M	293.0	369.6	45.1	73.27
	Proxyless-R (mobile) [4]	Gradient	CPU	333M	534.6	616.5	53.1	74.60
	SPOS (block search) [9]	Oneshot	-	319M	270.6	455.8	38.7	74.30
Search for Our Hardware	Proxyless-R*	Gradient	DSP	421M	43.9	662.0	46.3	74.20
	SPOS*	Oneshot	DSP	366M	17.3	538.2	46.1	74.56
	HURRICANE-DSP (Ours)	Oneshot	DSP	709M	**16.5**	576.7	45.4	**76.67**
	Proxyless-R*	Gradient	CPU ‡	279M	182.8	392.3	37.8	73.40
	SPOS*	Oneshot	CPU ‡	302M	106.0	345.5	36.3	73.76
	HURRICANE-CPU (Ours)	Oneshot	CPU ‡	327M	80.1	**301.3**	38.9	**74.59**
	Proxyless-R*	Gradient	VPU	275M	264.7	464.9	35.6	73.30
	SPOS*	Oneshot	VPU	323M	372.9	693.1	36.1	74.02
	HURRICANE-VPU (Ours)	Oneshot	VPU	409M	390.8	645.3	**35.6**	**75.13**

Table 3. Compared with state-of-the-art hardware-aware NAS methods on ImageNet, HURRICANE is the only NAS method that consistently achieves high accuracy and low latency on all the target hardware. Latency numbers are measured on our hardware platforms. *: We run Proxyless-R and SPOS with their officially open-sourced implementations to search models on our hardware platforms . ‡: CPU latency is measured on a single CPU core with float32 precision.

Group	**NAS**	**Acc (%)**	**CPU (ms)**
Similar latency	FBNet-iPhoneX	73.20	313.0
	FBNet-S8	73.27	369.6
	HURRICANE-CPU	**74.59**	301.3
Similar accuracy	Proxyless-mobile	74.60	616.5
	FBNet-C	74.90	688.6
	MnasNet-A1	75.20	1056.1
	HURRICANE-CPU1	74.98	**381.2**

Table 4. Compared with models of same-level CPU inference latency, HURRICANE-CPU improves the top-1 accuracy from 73.27% to 74.59% on ImageNet. Compared with models of same-level top-1 accuracy, HURRICANE-CPU1 accelerates the CPU inference time by 1.62× - 2.77×.

4.2. Searching on ImageNet Dataset

Our comparisons are two-folds: (1) we compare HURRICANE searched models with various existing state-of-the-art efficient models that are primarily designed or searched for ARM CPU, to demonstrate that HURRICANE is able to generate different models suitable for different types of hardware; (2) we compare HURRICANE with two representative NAS methods, Proxyless-R [4] and SPOS [9] [4], to show that HURRICANE is able to search for better models at a lower cost, benefiting from the two-stage search algorithm. The primary metrics we care about are top-1 **accuracy** on the ImageNet dataset and **inference latency** on the three hardware.

Dataset and Training Details. Following [4], we randomly split the original training set into two parts: 50,000 images for validation (50 images for each class exactly) and the rest as the training set. The original validation set is used for testing, on which all the evaluation results are reported. We follow most of the training settings and hyper-parameters used in SPOS [9], with two exceptions: *(i)* For supernet training, the epochs change with different hardware-aware search spaces (listed in Table 1), and we stop at the same level training loss as SPOS. *(ii)* For architecture retraining, we change linear learning rate decay to cosine decay from 0.4 to 0. The batch size is 1,024. Training uses 4 *NVIDIA V100* GPUs. We implement Proxyless-R [3] and SPOS [19] on our hardware platforms to search for the models within the same latency constraints.

Results and Analysis. Table 3 and Table 4 summarize our experiment results on ImageNet. It demonstrates that it's essential to leverage hardware diversity in NAS to consistently achieve the high accuracy and low latency on different hardware platforms.

Firstly, HURRICANE surpasses existing state-of-the-art efficient models. Compared to MobileNetV2 (top-1 accuracy 72.0%), HURRICANE improves the accuracy by 2.59% to 4.03% on all target hardware platforms. Compared to state-of-the-art models searched by NAS, HURRICANE achieves the lowest inference latency on DSP, CPU, VPU, with better or comparable accuracy. Remarkably, HURRICANE-DSP achieves 76.67% accuracy, better than MnasNet-A1 (+1.47%), FBNet-iPhoneX (+3.47%), FBNet-

[4]Considering the reproducibility issue, we didn't test other hardware-aware NAS methods due to the lack of officially open-sourced code.

S8 (+3.4%), Proxyless-R (+2.07%), and SPOS (+2.37%). Regarding latency, HURRICANE-DSP is 16.5ms on DSP, that reaches a a 6.35× inference speedup than FBNet-iPhoneX. Interestingly, HURRICANE-DSP is faster than other NAS models but with a much larger FLOPs count. This is against the widely accepted belief that smaller FLOPs count results in lower latency. Our study for DSP indicates that small-kernel-sized complicated operators are most suitable for this platform, and the hardware aware search space fully takes advantage of this and benefits from such operators (c.f. Sec 3.1).

Secondly, HURRICANE outperforms the state-of-the-art NAS methods for the same target hardware platforms. Compared to SPOS, HURRICANE improves the accuracy by 2.11% (DSP), 0.83% (CPU), 1.11% (VPU) with slightly lower inference latency (DSP: -0.8ms, CPU: -44.2ms, VPU: -0.5ms). When compared with Proxyless-R, our method achieves higher accuracy of 2.47% (DSP), 1.49% (CPU), 1.83% (VPU) with less inference latency (DSP: -27.4ms, CPU: -91ms). We noted that the models searched by Proxyless-R are with lower accuracy and larger latency than SPOS and ours. One hypothesis is that the default hyper-parameter w in Proxyless-R that controls the trade-off between accuracy and latency might be not optimal for other hardware platforms. For instance, Proxyless-R searched many zero operators for the earlier layers on CPU and VPU.

Finally, HURRICANE also achieves competitive performance on the well-studied ARM CPU. To further compare the efficiency on CPU, we group related NAS models into same-level latency group and same-level accuracy group in Table 4. For fairness, we didn't compare MobileNetV3-Large1.0 as it adopts a second fine-grained search by NetAdapt on MnasNet-A1. Results in Table 4 suggests HURRICANE (CPU) achieves the highest accuracy in same-level CPU inference time group, and achieves 1.62× - 2.77× lower inference time in same-level accuracy group.

Search Cost Analysis. To compare the search cost, we report supernet training time reduction compared with SPOS instead of exact GPU search days as [4, 25] for two reasons: *(i)*: the GPU search days are highly relevant with the experiment environments (e.g., different GPU hardware) and the code implementation (e.g., ImageNet distributed training). *(ii)*: The primary time cost comes from supernet training in SPOS, as the evolution search is fast that architectures only perform inference.

Compared with SPOS, HURRICANE (*Stage1* + *Stage2*) reduces 30.4% supernet training time and finds models with better performance. Furthermore, HURRICANE (*Stage1*) already achieves better classification accuracy than other NAS methods (DSP: 76.57%, CPU: 74.59%, VPU: 74.63%) while reducing an average of 54.7% time, which is almost a 2× training time speedup (More analysis are in Sec 4.3.2). It demonstrates the effectiveness of exploring more architecture in the deeper CNN layers.

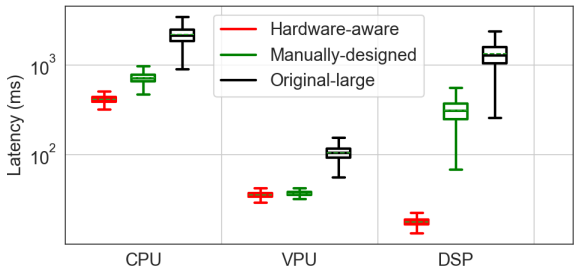


Figure 3. Architectures sampled in hardware specialized search spaces achieve lower latency than manually-designed and original-large search spaces. The y-axis is log-scaled for better comparison.

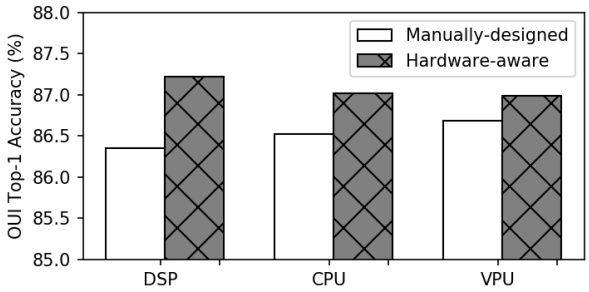


Figure 4. Compared to manually-designed search space, our hardware-aware search spaces achieve higher accuracy by SPOS.

4.3. Ablation Study and Analysis

We now further evaluate the efficiency of the proposed hardware-aware search space and two-stage acceleration algorithm. For simplicity, we run the experiments on the OUI-Adience-Age (OUI) dataset. OUI [7] is a small 8-class dataset consisting of 17,000 face images. We split OUI into train and test sets by 8:2. The settings for supernet training parameters are the same as ImageNet experiments except that we reduce the initial learning rate from 0.5 to 0.1, and the batch size reduced from 1024 to 64. Supernet trains until converge. For the architecture retraining, we train for 400 epochs and change the linear learning rate decay to Cyclic decay [21] with a $[0, 1]$ bound. We use 1 *NVIDIA Telsa P100* for training.

4.3.1 Effectiveness of Hardware-aware Search Space

Ideal search spaces contain many high accuracy architectures within the latency constraint so that the optimal ones are easily sampled by the search algorithm. We first investigate the latency distribution of architectures in a search space. We random sample 10 million architectures from three different search spaces: *(i)*: our hardware-aware search spaces (c.f. Table 1), *(ii) Manually-designed*: the search space used in SPOS [9] that designed by domain experts, *(iii) Original-large*: the search space generated by our large operator pool (c.f. Table 2) , and benchmark

Search Space	Hardware	SPOS (20 layers)		HURRICANE (Stage1: 12 layers)		HURRICANE (Stage1 + Stage2: 20 layers)	
		Acc (%)	Train iters (#)	Acc (%)	Train iters (#)	Acc (%)	Train iters (#)
Manually-designed	DSP	86.35	393,000	86.29	157,200	86.69	184,710
	CPU	86.52	393,000	86.32	157,200	86.75	183,400
	VPU	86.68	393,000	86.49	157,200	86.90	192,000
Hardware-aware	DSP	87.22	569,850	86.56	128,380	87.62	150,650
	CPU	87.02	476,840	86.75	144,100	87.33	168,990
	VPU	86.99	524,000	86.93	133,620	87.07	157,200

Table 5. Compared to SPOS [9], our two-stage search method achieves higher accuracy with much less search cost (51.1%-70%) on both manually-designed and hardware-aware search spaces. We list out the training iterations OUI (batchsize=64) for search cost comparison.

their inference latency on hardware. As Figure 3 shows, architectures in our hardware specialized search spaces are with much lower latency than the manually-designed and the original large search spaces. This indicates our search space is more compact and easier for constrained sampling.

To evaluate the accuracy gains by search space, we run the SPOS on both manually-designed and hardware-aware search spaces. We did not compare with the original-large search space here due to the unacceptable cost and non-convergence problem in supernet training. Figure 4 shows our hardware-aware search space consistently achieves higher accuracy than the manually-designed search space (+0.87% on DSP, +0.5% on CPU, +0.31% on VPU).

Hardware Insights. We share several important *insights* from search space generation.

- *Hexagon^{TM} 685 DSP*. Small kernel convolutions ($k \leq 3$) are well optimized on this platform. As a result, all the operators are of k=3 in search space. It also allows the search space to contain complicated operators (of large FLOPs) with small kernels, because their efficiency on this platform is better than those less-complicated operators but with bigger kernels. That's why HURRICANE (DSP) is faster than other NAS models but with a much larger FLOPs. On the contrary, the search spaces of Proxyless and SPOS contain many large kernel operators (i.e., k=5/7).
- *Myriad^{TM} X VPU*. The efficiency is strongly impacted by whether the operator is natively supported by the AI accelerator. For example, SE module is of low efficiency in this platform, because it has to roll back to relatively slow CPU execution. On the contrary, convolutions with bigger kernels ($k = 7$) are much more efficiently executed than on other platforms. This explains why the search space for this platform selects no SE operators but much more bigger kernel operators (especially in the earlier layers). We also observe that the operators applied in Proxyless and SPOS are all supported by the AI accelerator. Therefore, the VPU latency of searched models by Proxyless-R and SPOS are relatively low as shown in Table 3.
- *Snapdragon 845 ARM CPU*. Even with complex memory operator, Choice_3 (i.e., ShuffleNetV2 unit) is the most efficient operator on this platform.

4.3.2 Effectiveness of Two-Stage Search Algorithm

Different with previous NAS methods that globally search over all the learnable layers (e.g., SPOS searches $n = 20$ layers), our two-stage search algorithm groups CNN layers into 12 later and 8 earlier layers: Stage1 searches the later layers first, Stage2 searches the earlier layers for the winning architectures in Stage1. To demonstrate the effectiveness, we compare it with SPOS on both manually-designed and our hardware-aware search spaces.

Table 5 summarizes experiment results. On all search spaces, our proposed method could achieve not only higher accuracy but also less search cost for the target hardware under the latency constraint. In addition, only one step search (*Stage1*) of HURRICANE could achieve a comparable top-1 accuracy (with an average of 0.23% loss), but the number of training iterations is significantly reduced (60%-77.5%). This indicates that operators in later CNN layers are more critical for final accuracy.

If the computation budget (e.g., training time) allows, HURRICANE can further benefit from the second step search (*Stage2*). The accuracy is improved by 0.14%-1.06% with an additional cost of only 4.0%-8.8% of training iterations. Our gains mainly come from the reduced search space size by the two-stage search algorithm.

5. Conclusion

In this paper, we propose HURRICANE to address the challenge of hardware diversity in NAS. By exploring hardware-aware search space and two-stage search algorithm, we demonstrate that HURRICANE is able to search for better models specialized for different hardware platforms and outperforms the previous NAS methods by both accuracy and significant training time reduction.

References

[1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Understanding and Simplifying One-Shot Architecture Search. In *ICML*, 2017.

[2] Christopher M. Bishop. Pattern Recognition and Machine Learning, 2006.

[3] Han Cai, Ligeng Zhu, and Song Han. Github code: Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*. 2019.

[4] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*. 2019.

[5] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. arXiv preprint, arXiv:1907.01845, 2019.

[6] Matthew D.Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. ECCV, 2014.

[7] Eran Eldinger, Roee Enbar, and Tal Hassner. Age and Gender Estimation of Unfiltered Faces. In *TIFS*, 2014.

[8] Deeptha Girish, Vineeta Singh, and Anca Ralescu. Unsupervised clustering based understanding of cnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 9–11, 2019.

[9] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. arXiv:1904.00420, 2019.

[10] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. arXiv preprint, arXiv:1905.02244, 2019.

[11] Jie Hu, Li Shen, and Samuel Albanie. Squeeze-and-excitation networks. In *arXiv preprint, arXiv:1709.01507*, 2017.

[12] Intel. Intel distribution of openvino toolkit, 2018 r5 build. https://software.intel.com/en-us/openvino-toolkit, 2018.

[13] Hao Li, Asim Kadav, Igor Durdanovic, Hannan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*. 2017.

[14] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. ICLR, 2019.

[15] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. ECCV, 2018.

[16] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*, 2018.

[17] Qualcomm. Snapdragon neural processing engine sdk, version: 1.25.1. https://developer.qualcomm.com/docs/snpe/setup.html, 2019.

[18] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-Scale Evolution of Image Classifiers. In *ICML*, 2017.

[19] Megvii Research. Shufflenet series. urlhttps://github.com/megvii-model/ShuffleNet-Series,, 2019.

[20] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. CVPR, 2018.

[21] Leslie N. Smith. Cyclical learning rates for training neural networks. In *WACV*, 2017.

[22] Dimitrios Stamoulis, Ermao Cai, Da-Cheng Juan, and Diana Marculescu. Hyperpower: Power- and memory-constrained hyper-parameter optimization for neural networks. *The Journal of Machine Learning Research*, 2018.

[23] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. CVPR, 2018.

[24] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V.Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

[25] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Yajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*. 2019.

[26] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. ECCV, 2018.

[27] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? In *arXiv preprint arXiv:1902.01996v3*. 2019.

[28] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. CVPR, 2018.

[29] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. arXiv preprint, arXiv:1611.01578, 2016.

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