HARDWARE-AWARE ONE-SHOT NEURAL ARCHITECTURE SEARCH IN COORDINATE ASCENT FRAMEWORK

Anonymous authors
Paper under double-blind review

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 HURRICANE that explores the automatic hardware-aware search over a much larger search space and a multistep search scheme in coordinate ascent framework, to generate tailored models for different types of hardware. Extensive experiments on ImageNet show that our algorithm consistently achieves a much lower inference latency with a similar or better accuracy than state-of-the-art NAS methods on three types of hardware. Remarkably, HURRICANE achieves a 76.63% top-1 accuracy on ImageNet with a inference latency of only 16.5 ms for DSP, which is a 3.4% higher accuracy and a 6.35× inference speedup than FBNet-iPhoneX. For VPU, HURRICANE achieves a 0.53% higher top-1 accuracy than Proxyless-mobile with a 1.49× speedup. Even for well-studied mobile CPU, HURRICANE achieves a 1.63% higher top-1 accuracy than FBNet-iPhoneX with a comparable inference latency. HURRICANE also reduces the training time by 54.7% on average compared to SinglePath-Oneshot.

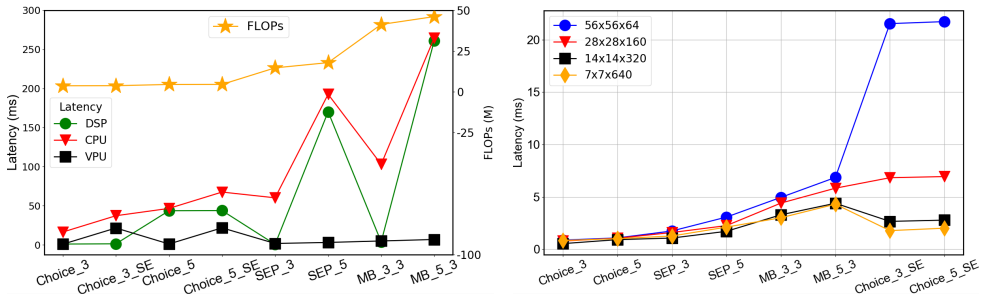
1 INTRODUCTION

Neural Architecture Search (NAS) is a powerful mechanism to automatically generate efficient Convolutional Neural Networks (CNNs) without requiring huge manual efforts of human experts to design good CNN models (Zoph & Le, 2016; Zoph et al., 2018; Tan et al., 2019; Guo et al., 2019; Bender et al., 2017). However, most existing NAS methods focus on searching for a single DNN model of high accuracy but pay less attention on the performance of executing the model on hardware, e.g., inference latency or energy cost. Recent NAS methods (Guo et al., 2019; Chu et al., 2019; Cai et al., 2019; Stamoulis et al., 2018b; Wu et al., 2019) start to consider model-inference performance but they use FLOPs [1] to estimate inference latency or only consider the same type of hardware, e.g., smartphones from different manufacturers but all ARM-based. However, the emerging massive smart devices are equipped with very diverse processors, such as CPU, 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 calls for new trade-offs and designs for NAS to generate efficient models for diverse hardware.

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 *SEP_5* is nearly 12× higher than that of operator *Choice_3* on the ARM CPU, but the difference on the VPU is less than 4×. Therefore, FLOPs is not the right metric to decide the inference latency on different hardware. Second, the relative effectiveness of different operators on different processors is also different. For example, operator *SEP_3* has the smallest latency on the DSP, but operator *Choice_3* has the

[1] In this paper, the definition of $FLOPs$ follows (Zhang et al., 2018), i.e., the number of multiply-adds.

smallest latency on the VPU. 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. As a result, even on the same hardware, optimal operators may change at different layers of the network.



(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^{TM} 685 DSP), (2) CPU (Snapdragon 845 ARM CPU), (3) VPU (Movidius^{TM} Myriad^{TM} 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.

In addition, we observe that the existing NAS methods tends to handle all the layers equally. For instance, the uniform sampling in one-shot NAS (Brock et al., 2018; Guo et al., 2019) will give the same sampling opportunities to every layer. However, not all the layers are the same: different layers may have different impacts on inference latency and model accuracy. Indeed, some previous works (D.Zeiler & Fergus, 2014; Girish et al., 2019) have revealed different behaviors between the earlier layers (close to data input) and the latter layers (close to classification output) in CNN models. The earlier layers extract low-level features from inputs (e.g., edges and colors), are computation intensive and demands more data to converge, while the latter layers capture high-level class-specific features but are less computation intensive. From these findings, we argue that *exploring more architecture selections in the latter layers may help find better architectures with the limited sampling budget, and limiting the latency in the earlier layers is critical to search for low-latency models.* To this end, it is desirable to explore how to leverage this *layer diversity* for better architecture sampling in NAS.

Motivated by these observations, we argue that there is no one-size-fits-all model for different hardware, and thus propose and develop a novel hardware-aware method, called HURRICANE (Hardware aware one-shot neUral aRchitecture seaRch In Coordinate AsceNt framEwork), to tackle the challenge of hardware diversity in NAS. Different from the existing hardware-aware NAS methods that use a small set of operators (e.g., 6 or 9) manually selected for a specific hardware platform, HURRICANE is initialized with a large-size (32 in our implementation) candidate operators set to cover the diversity of hardware platforms. However, doing so increases the search space by many orders of magnitude and thus leads to unacceptable search and training cost and may even cause non-convergence problem. To reduce the cost, we propose *hardware-aware search space reduction at both operator level and layer level*. In the *operator-level search space reduction*, a toolkit is developed to automatically score every layer's candidate operators on target hardware platforms, and choose a sub-set of them with low latency for further utilization. In the *layer-level search space reduction*, we split the layers into two groups, the earlier group and the latter group according to their locations in the network. Based on a *coordinate ascent framework* (Wright, 2015) (Appendix A), we propose a multistep search scheme, which searches the complete architecture by a sequence of simpler searching of sub-networks. In each iteration (step), we alternatively fix one group of layers and optimize the other group of layers to maximize the validation accuracy by a *one-shot NAS* [2]. The searching of sub-networks is much easier to complete because of the much smaller size of

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

search space, and the better architectures are reached by a sequence of iterations. This *layer-level search space reduction* is inspired by the *layer diversity* mentioned above. We choose most latency-effective operators for earlier layers and allocate more sampling opportunities to latter layers. As a result, we are able to search for models with both low latency and high accuracy.

We evaluate the effectiveness of our proposed approach on ImageNet 2012 dataset and a small OUI-Adience-Age dataset with the above three mobile hardware platforms (DSP/CPU/VPU). Under all the three platforms, HURRICANE consistently achieves the same level (or better) accuracy with much lower inference latency than state-of-the-art hardware-aware NAS methods. Remarkably, HURRICANE reduces the inference latency by 6.35× on DSP compared to FBNet-iPhoneX and 1.49× On VPU compared to Proxyless-mobile, respectively. Compared to Singlepath-Oneshot, on average HURRICANE reduces the training time by 54.7% on ImageNet.

2 Related Work

Neural Architecture Search (NAS). (Zoph & Le, 2016; Zoph et al., 2018) first proposed to use reinforcement learning (RL) to search for competitive architectures with low FLOPs. As a full configurable architecture search space grows exponentially, early works (Liu et al., 2019; Zoph et al., 2018; Pham et al., 2018; Real et al., 2018) search for a cell level structure as the building block and the same cell is reused in all layers. However, many cell structures are very complicated and fragmented, and thus are slow when deployed to a device. Recent methods (Tan et al., 2019; Wu et al., 2019; Cai et al., 2019; Guo et al., 2019; Bender et al., 2017) adopt a layer-level hierarchical search space with a back-bone 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 blocks. MnasNet (Tan et al., 2019), ProxylessNAS (Cai et al., 2019) and Single-path NAS (Stamoulis et al., 2018b) built upon the MobileNetV2 (Sandler et al., 2018) structure (*MB_k_e*). FBNet (Wu et al., 2019) and Singlepath-Oneshot (Guo et al., 2019) built search space by ShuffleNetV1 (Zhang et al., 2018) and ShuffleNetV2 (Ma et al., 2018) (*Choice_k*). As these structures are primarily designed for mobile CPU, the efficiency of such manually-designed search space is unknown for other hardware.

Hardware aware NAS. Early NAS methods (Zoph & Le, 2016; Zoph et al., 2018) adopt hardware-agnostic metric FLOPs to measure the efficiency. However, architecture with lower FLOPs is not necessarily faster (Stamoulis et al., 2018a). Recently, many methods (Cai et al., 2019; Stamoulis et al., 2018b; Wu et al., 2019) adopt direct metrics such as measured latency but only for mobile CPUs. (Cai et al., 2019) builds a latency prediction model, (Stamoulis et al., 2018b; Wu et al., 2019) profiles every operator's latency, and then the latency metric is viewed as regularization loss. Such design is also not optimized for small mobile models because accuracy changes much more dramatically with latency for small models, as (Howard et al., 2019) pointed out.

One-Shot NAS. Starting from ENAS (Pham et al., 2018), 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, gradient-based) with weights updated. However, Singlepath-Oneshot Guo et al. (2019) and FairNAS (Chu et al., 2019) observe that such coupled architecture search and weight sharing could be problematic to fairly evaluate the performance of candidate architectures. Our work is built upon Singlepath-Oneshot (Guo et al., 2019). It decouples the supernet training and evolution-based architecture search by uniform sampling.

3 Methodology

3.1 Problem Formulation

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 LatC_h:

$$\begin{aligned} \max \quad & \text{ACC}_{\text{val}}(a) \\ \text{s.t.} \quad & \text{Latency}(a, h) \leq \text{LatC}_h \end{aligned} \tag{1}$$

HURRICANE can find a architecture a that achieves the maximum accuracy $\text{ACC}_{\text{val}}(a)$ on the validation set and the inference latency $\text{Latency}(a, h)$ is under the constraint LatC_h.

Without loss of generality, $\mathcal{A}$ denotes the network architecture search space. Like other recent works (Guo et al., 2019; Wu et al., 2019; Tan et al., 2019; Cai et al., 2019; Chu et al., 2019), $\mathcal{A}$ is constructed by a back-bone network architecture, n layers of which could be chosen from a set of candidate operators $\mathcal{O}$.

$$\mathcal{A} = \{a \mid a[l] \in \mathcal{O}_l, 1 \leq l \leq n\} \tag{2}$$

Here $a[l]$ denotes the l-th layer of architecture a, and $\mathcal{O}_l$ is the set of candidate operators specialized for l-th layer according to its context (e.g. input shape, number of channels).

3.2 Algorithm Overview

Algorithm 1 briefs the *hardware aware one-shot neural architecture search in coordinate ascent framework*. We employ the same back-bone architecture with (Guo et al., 2019) but a much bigger collection of candidate operators from the primary blocks of off-the-shelf networks. The set of candidate operators ($\mathcal{O}$) contains up to 32 operators detailed in Section 3.3.

Algorithm 1 Hardware aware one-shot NAS in coordinate ascent framework

Require: h (hardware platform), $LatC_h$ (latency constraint), $\mathcal{O}$ (candidate operators)
Require: I (number of iterations), t, e (hyper parameter)
▷ Operator level search space reduction
1: **for** $l \leftarrow 1$ to n **do**
2: $\mathcal{O}'_l \leftarrow$SORT($\mathcal{O}_l$) in the descending order of SCORE($\mathcal{O}_l$, h)
3: $\mathcal{O}^*_l \leftarrow \{\mathcal{O}'_l[0], \ldots, \mathcal{O}'_l[3]\}$
4: **if** $l > n - 4$ **then**
5: $\mathcal{O}^*_l \leftarrow \mathcal{O}^*_l \cup \{\mathcal{O}'_l[e]\}$, e is index of extra exploring operator and $e \geq 4$
▷ Layer level search space reduction and coordinate ascent
6: Initialize winning architecture a_{win} with $a_{win}[k] \leftarrow \mathcal{O}'_k[0]$, $1 \leq k \leq n$
7: **for** $i \leftarrow 1$ to I **do**
8: $L_{active} \leftarrow [t+1, n]$ if i is odd else $[1, t]$
9: $\mathcal{A}_i \leftarrow \{a \mid a[l] \in \mathcal{O}^*_l, l \in L_{active}; a[l] \leftarrow a_{min}[l], l \notin L_{active}\}$
10: $W_{\mathcal{A}_i} = \arg\min_W \mathbb{E}_{a \sim \Gamma_h(\mathcal{A}_i)}[\mathcal{L}(\mathcal{N}(a, W(a)))]$
11: $a_{win} = \arg\max_{a \in \Gamma_h(\mathcal{A}_i)} ACC_{val}(\mathcal{N}(a, W_{\mathcal{A}_i}(a)))$
12: Retrain a_{win}

Firstly, we leverage the hardware performance to *reduce the search space on operator level*. We develop a toolkit to automatically score the candidate operators $\mathcal{O}_l$ on different hardware platforms. The toolkit profiles the real performances (including but may not limited to inference latency), and give a comprehensive score for every candidate operator. Then we select top 4 or 5 operators ($\mathcal{O}^\star_l$) in the non-increasing order of scores for l-th layer.

Secondly, we *reduce the search space on layer level*. At the beginning of coordinate ascent, we setup the winning architecture with the operators of highest scores in every layer. In first iteration, we mark the latter $n - t$ layers as active and the earlier t layers as non-active. The non-active layers are fixed to the corresponding layer structures of current winning architecture, while the active layers will be chosen from the 4 highest scored operators (an extra operator for exploring is added in the last 4 layers) (Line 8 and Line 9). The one-shot NAS method itself is similar to the work (Guo et al., 2019), except that we constraint the search space with a hardware latency [3] other than FLOPs.

$$\Gamma_h(\mathcal{A}) = \{a \in \mathcal{A} \text{ and } \text{Latency}(a, h) \leq \text{LatC}_h\} \tag{3}$$

We encode all candidate architectures of search space $\mathcal{A}$ into a over parameterized supernet (and its weights W), and sample a specific architecture a (and its weights $W(a)$). The sampled network is denoted by $\mathcal{N}(a, W(a))$. For every mini batch $\mathcal{B}$ in training dataset $\mathcal{D}_{tr}$, the supernet weights W is updated by the gradient $(\nabla \mathcal{L}_{\mathcal{B}}(\mathcal{N}(a, W_i(a))))$ of the training loss of network $\mathcal{N}(a, W(a))$ (Brock et al., 2018; Guo et al., 2019). After the supernet is trained (only once), all candidate architectures a inherit weights from the supernet $W_{\mathcal{A}}$, and thus weight $W_{\mathcal{A}}(a)$ is used to approximate the optimal weight of architecture a. The weights will not be updated during the process of architecture

[3] To do this, we build a latency-prediction model and details can be found in Appendix D.

searching. We adopt evolutionary search (Guo et al., 2019) to find the winning architecture with the highest validation accuracy (Line 11).

After a complete process of one-shot NAS, a new winning architecture would be generated. The new winning architecture would remain the same structure in the non-active layers and update to new operators in the active layers. A new iteration is started with the new winning architecture until the termination test is satisfied.

3.3 GLOBAL SEARCH SPACE

Candidate Operator Pool $\mathcal{O}$. As is known, the computation complexity (e.g., FLOPs) and memory access cost of an architecture are key factors in hardware latency efficiency (Ma et al., 2018). To cover the hardware diversity, our initial search space consists of 32 candidate operators (detailed structures are in Appendix Figure 3) that leverage different computation and memory complexity. They are built upon the following 4 basic structures from current efficient models:

- **SEP**: depthwise-separable convolution. Following DARTS (Liu et al., 2019), we applied the depthwise-separable convolution twice. We allow choosing kernel size *k* of 3, 5 or 7 for the depthwise convolution and generate 3 operators *SEP_k*. This rule with different kernel size for depthwise convolution also applies to other operator generation. *SEP_k* has a larger FLOP count than others, but less memory access complexity.
- **MB**: mobile inverted bottleneck convolution in MobileNetV2 (Sandler et al., 2018). It's widely used in recent one-shot NAS (Chu et al., 2019; Cai et al., 2019). We generate 9 operators *MB_k_e* based on it, that we allow choosing *k* of 3, 5, 7 and channel expansion rate *e* of 1, 3, 6. *MB_k_e* 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 (Ma et al., 2018). We construct 3 operators *Choice_k* (*k*=3, 5, 7). Following (Guo et al., 2019), we also adds a similar operator *ChoiceX*. *Choice_k* 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 (Hu et al., 2017). To balance the impacts in latency and accuracy, we follow the settings in MobileNetV3 (Howard et al., 2019). 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 16 operators. The computation complexity of *SE* is decided by its insert position, while the memory access cost is relatively lower.

As a result, our global search space consists of $n^{|\mathcal{O}|}$=20^{32} (n=20 layers) candidate architectures, which is exponentially larger than current one-shot NAS. For each searchable layer, there are total of 32 operators to choose: *SEP_k, SEP_k_SE,MB_k_e, MB_k_e_SE, Choice_k,Choice_k_SE,ChoiceX, ChoiceX_SE*, where k=3,5,7, and e=1,3,6.

3.4 HARDWARE AWARE ONE-SHOT NEURAL ARCHITECTURE SEARCH IN COORDINATE ASCENT FRAMEWORK

Hardware aware profiling and sorting. As shown in Figure 1(a), the real performances of operators vary significantly on different hardware platforms. Without loss of generality, the scoring function should consider both representation capacity and real hardware performance. There are many methods to approximate the scoring function, such as a customized weighted product of FLOPs and number of parameters (#Params)

$$(\text{FLOPs} \times \#\text{Params}^{\beta})^{\alpha} \times \text{Latency}^{-1} \tag{4}$$

where α and β are non-negative constants.

Construct search space with high scored operators. For each layer, we filter out the top 4 operators with highest scores first. This make the search space reduced to a comparable size with previous works, but specialized for the target platform. Another exploring operator is added to the candidate

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.

operator set of the last 4 layers. Only adding to the last layers is because (1) the latency degradations of these layers are acceptable due to their smallest feature size; (2) sophisticated operators could help these layers better capture high-level class-specific patterns.

Hyper-parameters. There are two important hyper-parameters in Algorithm 1: the layer grouping boundary t and the number of iterations I. We set $t = 8$ (only searchable layers counted) according to the natural resolution changes of the supernet (Table 4 in Appendix). Experiment discussed in Appendix C shows the rationality of it. For simplicity, we tested the simplest one-iteration case ($I = 1$) and adding an extra iteration ($I = 2$).

4 EVALUATION

4.1 HARDWARE PLATFORMS AND MEASUREMENTS

Diverse hardware platforms. HURRICANE targets 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 (Qualcomm, 2019), while VPU latency is measured by Intel OpenVINO™ Toolkit (Intel, 2019).

Operators profiling and sorting. Table 1 lists the top scored operators on different hardware platforms when $\alpha = 0$. The hardware-aware search space constructed by scoring the operators on target hardware platforms and then selecting the best 4 operators (layer 17-20 has another extra exploring operator with $e = 10$). We share new important insights from hardware profiling: **(i)** depthwise convolutions with kernel size $k \leq 3$ are well optimized on Hexagon™ 685 DSP. As a result, all the operators are of k=3 in search space. **(ii)** SE module is not supported by the AI accelerator of Myriad™ X VPU, and thus rolled back to relatively slow CPU execution. **(iii)** Even with complex memory operator, $Choice\_3$ (i.e., ShuffleNetV2 unit) is the most efficient operator on VPU and CPU due to its much smaller FLOPs count.

Latency predictor. To avoid measuring the latency of every candidate architecture, we build a latency-prediction model with high accuracy: the RMSE (root mean square error) of prediction model is 0.82ms, 21.84ms, and 0.03ms on DSP, CPU, VPU, respectively, which means an average 4.7%, 4.2%, and 0.08% latency estimated error for DSP, CPU and VPU. It suggests the latency prediction model can be used to replace the expensive direct hardware measurement with little error introduced. More details of latency predictor is in Appendix D.

4.2 EXPERIMENT SETUP - ONE-SHOT NEURAL ARCHITECTURE SEARCH

Latency constraints. As discussed in Appendix B, the latency constraint on a given platform should be in a meaningful range. For better comparison with other works, we set the latency constraints to be smaller than the best latency of models from other works, which are 310 ms (CPU), 17 ms (DSP) and 36 ms (VPU).

Over-parameterized Supernet and Architecture Search. HURRICANE is built on top of Singlepath-Oneshot (Guo et al., 2019; Research, 2019). The supernet architecture (in Appendix

Model	FLOPs	Acc (%)	DSP (ms)	CPU (ms)	VPU (ms)	Supernet training Time Reduction (%) $I = 1$ / $I = 2$ †
FBNet-iPhoneX	322M	73.20	105.0	313.0	45.6	-
FBNet-S8	293M	73.27	293.0	369.6	45.1	-
Proxyless-R (mobile)	333M	74.60	534.6	616.5	53.1	-
Singlepath-Oneshot*	319M	74.30	270.6	455.8	38.7	0
***HURRICANE* (DSP)**	709M	**76.63**	**16.5**	576.7	45.4	61.4 (*76.57*) / 36.4 (*76.63*)
***HURRICANE* (CPU)**	327M	**74.59**	80.1	**301.3**	38.9	57.7 (*74.59*) / 33.9 (*74.59*)
***HURRICANE* (VPU)**	409M	**75.13**	390.8	645.3	**35.6**	45.0 (*74.63*) / 20.8 (*75.13*)

Table 2: Comparisons with various state-of-the-art efficient NAS on ImageNet. We measure all the model latency on our hardware platforms. *: For fair comparison, we use the block search model instead of the block search+ channel search. †: In the form "x(*y*)", where "x" means the training time reduction and "*y*" means the accuracy achieved.

Table 4) consists of stem layers and n=20 searchable layers. Once the supernet training finishes, we perform a 20-iterations evolution search for total 1,000 architectures as Singlepath-Oneshot. For better fairness, the supernet is re-initialized randomly before every iteration.[4]

4.3 Searching on ImageNet Dataset

Dataset. Following (Cai et al., 2019), 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.

Training Details. We follow most of the training settings and hyper-parameters used in Singlepath-Oneshot (Guo et al., 2019), 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 Singlepath-Oneshot. *(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.

Results and Search Cost Analysis. Table 2 summarizes our experiment results on ImageNet. HURRICANE surpasses state-of-the-art models, both manually and automatically designed: 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 models searched automatically, HURRICANE demonstrates that it's essential to leverage hardware diversity in NAS to achieve the best efficiency on different hardware platforms. Specially, compared to the most efficient models searched by NAS, HURRICANE (DSP) reaches a 6.35$\times$ inference speedup than FBNet-iPhoneX with 3.43% accuracy improvement, HURRICANE (CPU) achieves 1.39% higher accuracy with 11.7ms latency reduction, HURRICANE (VPU) achieves 0.83% higher accuracy with 3.1ms latency reduction. Remarkably, HURRICANE is the only hardware-aware NAS method that searches the better accuracy with much lower latency on all diverse hardware platforms.

To compare the search cost, we report supernet training time reduction compared with Singlepath-Oneshot instead of exact GPU search days as (Cai et al., 2019; Wu et al., 2019) 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 Singlepath-Oneshot, as the evolution search is fast that architectures only perform inference. Table 2 shows that our method reduces an average 54.7% time if executes one step search in coordinate ascent ($I = 1$), which is almost a 2x training time speedup. In addition, HURRICANE already achieves better classification accuracy than other NAS methods at this step. It demonstrates the effectiveness of exploring more architecture selections in the latter layers. With one more step search, HURRICANE usually search better architectures but takes longer tme. Results suggest our method can still save an average 30.4% time ($I = 2$).

Table 2 also indicates that HURRICANE (DSP) achieves the highest accuracy than other hardware, but with much more latency speedup. The contributions mainly come from the larger computation

[4]We also test different reset methods (a) re-initialized randomly, (b) reset the same random values every time, (c) keep the values and continue training. The results show no noticeable differences.

Task	Singlepath-Oneshot		HURRICANE ($I = 1$)		HURRICANE ($I = 2$)	
	Acc (%)	Train iters (#)	Acc (%)	Train iters (#)	Acc (%)	Train iters (#)
FLOPs constrained (OUI)	86.41	235,800	86.44	112,660	86.90	133,620
FLOPs constrained (ImageNet)	73.72	144,360	74.01	72,180	74.16	105,864
latency constrained for DSP (OUI)	87.22	569,850	86.56	128,380	87.62	150,650
latency constrained for CPU (OUI)	87.02	476,840	86.75	144,100	87.33	168,990
latency constrained for VPU (OUI)	86.99	524,000	86.93	133,620	87.07	157,200

Table 3: Comparisons with Singlepath-Oneshot by different search spaces. We list out the training iterations on ImageNet (batchsize=1,024) and OUI (batchsize=64) for search cost comparison.

FLOPs (higher accuracy) and the DSP hardware characteristics leverage in search space (lower latency). It also demonstrates again that FLOPs is an improper metric for hardware aware NAS.

4.4 Effectiveness Analysis of HURRICANE

To further demonstrate the effectiveness HURRICANE, we compare it with recent work (Guo et al., 2019) on a sequence of tasks. These tasks include the original FLOPs (330M) constrained architecture search in (Guo et al., 2019), the hardware latency constrained architecture search for different hardware platforms described in this paper. We do the experiments on ImageNet and also the OUI-Adience-Age (OUI) for simplicity. For the fairness, we let the methods use the same search space for a single task. For the FLOPs constrained tasks, we use the original search space of (Guo et al., 2019). For those hardware latency constrained tasks, we let the methods use our specialized search space listed in Table 1.

Dataset and Training Details. OUI-Adience-Age (Eldinger et al., 2014) is a small 8-class dataset consisting of 17,000 face images. We split the images into training and testing test by 8:2 for architecture retraining. For architecture search, we randomly split the training set into two parts: 5,567 images for validation and the rest as the training set for supernet. We adopt the same hyper-parameter settings as Singlepath-Oneshot, 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 original linear learning rate decay to Cyclic decay (Smith, 2017) with a $[0, 1]$ bound. We use 1 *NVIDIA Telsa P100* for training.

Results and Search Cost Analysis. Table 3 summarizes experiment results. The searched models outperform the manual designed light-weight models, such as MobileNetV2 (top-1 acc: 72.00% on ImageNet, 85.67% on OUI-Adience-Age). For every row of the table, the proposed method could achieve not only higher accuracy but also better hardware efficiency for all the tasks. As shown in the column of Singlepath-Oneshot, our hardware-aware search space could also improve the accuracy (0.6%-0.8% on OUI-Adience-Age dataset).

To illustrate the cost of supernet training, we listed the number of iterations consumed. As shown in Table 3, in most tasks, only one iteration ($I = 1$) of HURRICANE could achieve a comparable top-1 accuracy (or even better in some tasks), but the number of training iterations is significantly reduced (50%-77.5%). If the computation budget (e.g. training time) allows, HURRICANE can benefit from another iteration ($I = 2$). The accuracy is improved by 0.15%-1.06% with an additional cost of only 4.0%-23.3% of training iterations.

5 Conclusion

In this paper, we propose HURRICANE to address the challenge of hardware diversity in NAS. By exploring hardware-aware search space and a multistep search scheme based on coordinate ascent framework, our solution achieves better accuracy and much lower latency on three hardware platforms than state-of-the-art hardware-aware NAS. And the searching cost (searching time) is also significantly reduced. For future work, we plan to support more diverse hardware and speed up more NAS methods.

REFERENCES

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

Christopher M. Bishop. Pattern Recognition and Machine Learning, 2006.

Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas J Weston. Smash: One-shot model architecture search through hypernetworks. In *6th International Conference on Learning Representations*, 2018.

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

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

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

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

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*, pp. 9–11, 2019.

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.

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, 2019.

Jie Hu, Li Shen, and Samuel Albanie. Squeeze-and-excitation networks. In *arXiv preprint*, 2017.

Intel. Intel distribution of openvino toolkit. `https://software.intel.com/en-us/openvino-toolkit`, 2019.

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

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

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

Qualcomm. Snapdragon neural processing engine sdk. `https://developer.qualcomm.com/docs/snpe/setup.html`, 2019.

Carl Edward Rasmussen and Christopher K.I. Williams. Gaussian processes for machine learning (adaptive computation and machine learning), 2005.

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.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized Evolution for Image Classifier Architecture Search. In *ICML AutoML Workshop*, 2018.

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

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

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

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*, 2018a.

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, 2018b.

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.

Robert Tibshirani. Regression shrinkage and selection via the lasso, 1994.

Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, Jun 2015. ISSN 1436-4646. doi: 10.1007/s10107-015-0892-3. URL https://doi.org/10.1007/s10107-015-0892-3.

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.

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

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. arXiv preprint, 2016.

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

A COORDINATE DESCENT (ASCENT) ALGORITHMS

Coordinate descent (CD, or coordinate ascent, CA) (Wright, 2015) are optimization algorithms to find the minimum (or maximum) of a function by successively performing approximate minimization (maximization) along coordinate directions or coordinate hyperplanes. They are iterative methods, in which we optimize along a selected coordinate direction or hyperplane at a time while keeping all others fixed. Though not sophisticated, the CD or CA methods could be surprisingly efficient and scalable in many real world scenarios. As it's extremely difficult to an optimal layer-sampling rates sequence with respect to different layers' importance for final accuracy, CA algorithm is much easier to achieve competitive results with careful implementations.

B ADJUSTMENT ACCORDING FOR DIFFERENT LATENCY CONSTRAINTS

In the body of this paper, we assume that the latency constraint should be in a meaningful range, which means the constraint should neither be too small to achieve nor too great to make the problem degraded to an unbounded optimization for accuracy only. The proposed solution doesn't guarantee the performance in case that the latency constraint is out of the meaningful range. However, this could be patched by

- for those constraints too small to achieve, we could reduce the number of layers by adding a skip operator to some of the layers.
- for those constraints too big, we could adjust the scoring function to let the algorithm cares more about representation capacity other than latency, or even add additional layers.

C SELECTION OF HYPER-PARAMETER t

As illustrated in Section 3, we group the 20 layers into earlier t layers and latter 20-t layers. As the changes of resolution are natural boundary due to the feature map size changes, we split the groups at the second resolution downsampling and set t=8. Figure 2 also demonstrates that HURRICANE achieves comparable accuracy on OUI-Adience-Age with MobileNetV2 when t=8, with 74.4% time reduction of supernet training.

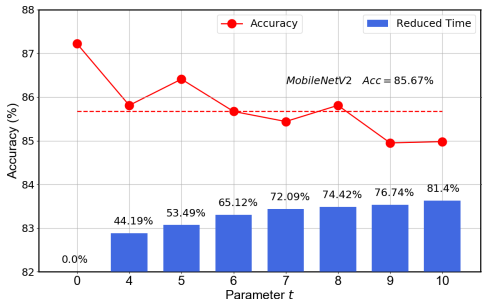


Figure 2: Accuracy vs. supernet training reduced time of HURRICANE-iter1 by different t on OUI-Adience-Age for DSP. When $t \leq 8$, HURRICANE-iter1 achieves a same level accuracy with MobileNetV2.

D LATENCY PREDICTOR

In a constrained optimization problem, it is often required to check whether an architecture exceeds the latency constraint. To reduce the cost and complexity of connecting hardware frequently, we developed the latency predictor, which consists of multiple independent hardware specific predictors. Each predictor takes the sequence of operators in all the layers as the input and predict the real latency of the whole architecture on the target hardware platform.

To build the latency prediction model, we uniformly sample 2,000 candidate architectures from our three hardware-aware search space, where 80% of them are used to build the latency model and

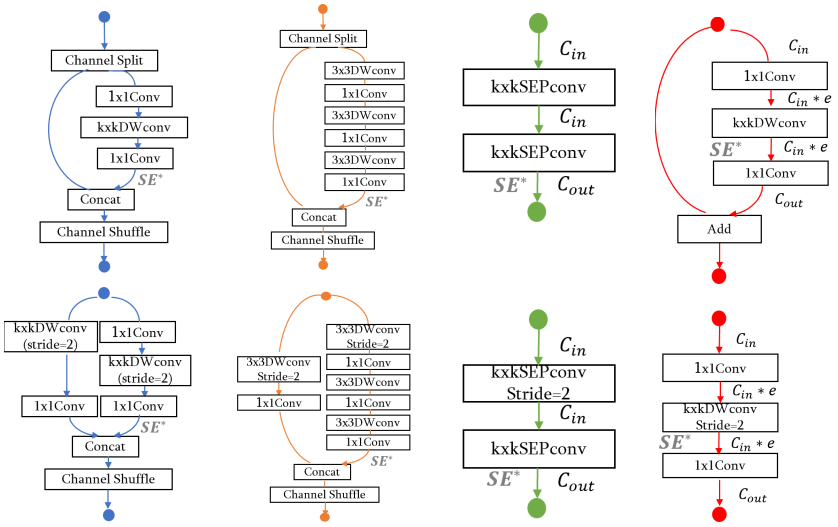
the rest are used for test. We encode the architectures into a 84-dimension binary vector, where the binary value indicates the occurrence of a corresponding operator. Different regression models are selected for latency prediction on diverse hardware. Specifically, We build GaussianProcess-Regressor with Matern kernel ($length\_scale$=1.5, nu=0.35) (Rasmussen & Williams, 2005), Lasso Regression model ($alpha$=0.01) (Tibshirani, 1994) and Bayesian Ridge Regression (Bishop, 2006) models for DSP, CPU and VPU, respectively.

E SUPERNET BACK-BONE ARCHITECTURE

Input shape	Block	channels	repeat	stride
224^2x3	3 x 3 conv	16	1	2
112^2x16	TBS	64	4	2
56^2x64	TBS	160	4	2
28^2x160	TBS	320	8	2
14^2x320	TBS	640	4	2
7^2x640	1 x1 conv	1024	-	-
7^2x1024	GAP	-	1	1
1024	fc	1000	1	1

Table 4: Supernet architecture. Column-"Block" denotes the block type. "TBS" means layer type needs to be searched. The "stride" column represents the stride of the first block in each repeated group. In our paper, we search the operations for total 20 layers.

F THE STRUCTURES OF OPERATIONS IN OUR SEARCH SPACE



(a) Choice_k (b) ChoiceX (c) SEP_k (d) MB_k_e

Figure 3: Operations in Section 3. SE^* indicates the position to add squeeze-and-excitation block. When SE^* is enabled, the operations are Choice_k_SE, ChoiceX_SE, SEP_k_SE, MB_k_e_SE. k indicates kernel size, where $k = 3, 5, 7$. e indicates the expansion rate, where $e = 1, 3, 6$.

G STRUCTURES OF SEARCHED ARCHITECTURES ON IMAGENET

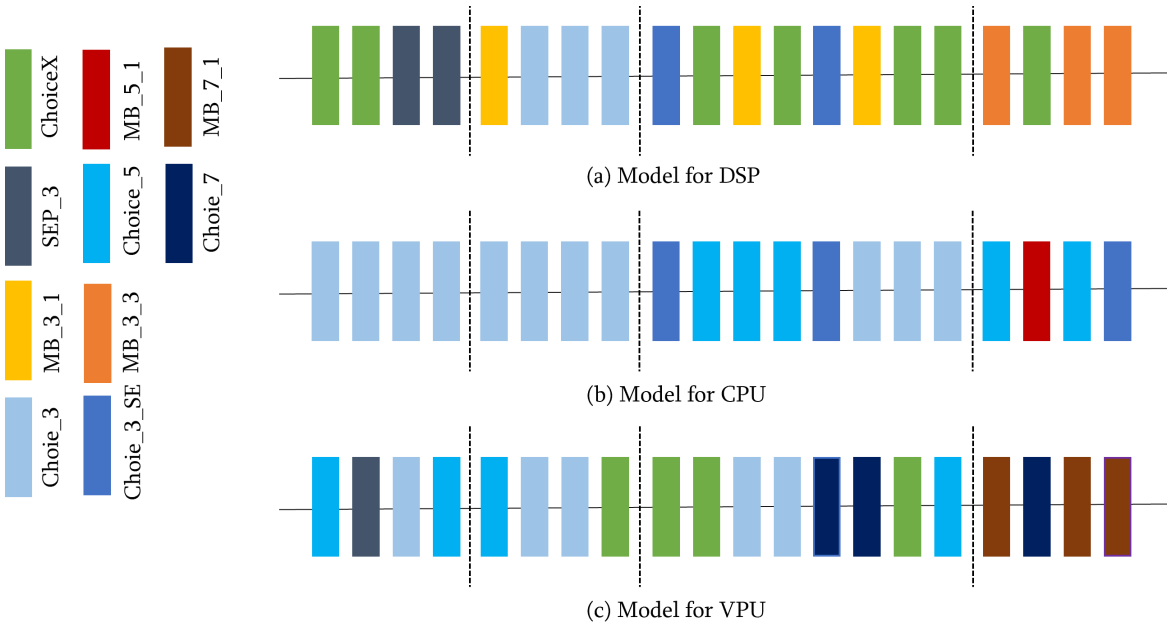


Figure 4: Structures of searched architectures on ImageNet in Table 2.