HSCoNAS: Hardware-Software Co-Design of Efficient DNNs via Neural Architecture Search

Xiangzhong Luo[1], Di Liu[2], Shuo Huai[3], Weichen Liu[4]

[1,3,4]*School of Computer Science and Engineering, Nanyang Technological University, Singapore*
[2,3,4]*HP-NTU Digital Manufacturing Corporate Lab, Nanyang Technological University, Singapore*
(xiangzho001[1], shuo001[3])@e.ntu.edu.sg, (liu.di[2], liu[4])@ntu.edu.sg

***Abstract*—In this paper, we present a novel multi-objective hardware-aware neural architecture search (NAS) framework, namely HSCoNAS, to automate the design of deep neural networks (DNNs) with high accuracy but low latency upon target hardware. To accomplish this goal, we first propose an effective hardware performance modeling method to approximate the runtime latency of DNNs on target hardware, which will be integrated into HSCoNAS to avoid the tedious on-device measurements. Besides, we propose two novel techniques, *i.e.*, dynamic channel scaling to maximize the accuracy under the specified latency and progressive space shrinking to refine the search space towards target hardware as well as alleviate the search overheads. These two techniques jointly work to allow HSCoNAS to perform fine-grained and efficient explorations. Finally, an evolutionary algorithm (EA) is incorporated to conduct the architecture search. Extensive experiments on ImageNet are conducted upon diverse target hardware, *i.e.*, GPU, CPU, and edge device to demonstrate the superiority of HSCoNAS over recent state-of-the-art approaches.**

I. Introduction

Deep neural networks (DNNs) have become the *de facto* engine of artificial intelligence (AI). Over the past few years, DNNs have achieved remarkable success in a wide range of real-world applications, such as person re-identification [8], autonomous driving [3], intelligent IoT [2], *etc*. However, to pursue competitive accuracy, DNNs are evolving deeply with more layers as well as widely with more channels, thereby incurring the *computational gap* between complicated DNNs and resource-limited hardware like edge devices, which are deemed as the key computing platform for future AI [5], [7]. Nonetheless, designing resource-efficient DNNs for less capable hardware still remains challenging since hardware-aware DNNs need to be small and fast, yet still accurate.

To tackle the above *computational gap*, unlike previous hardware-agnostic methods, we propose an efficient and unified hardware-software co-design NAS framework, namely HSCoNAS, to automatically design efficient DNNs with high accuracy but low latency upon diverse target hardware. The overview of HSCoNAS is illustrated in Fig. 1. Our main contributions are as follows:

1) We propose a hardware performance modeling method to approximate the runtime latency of DNNs upon target hardware while introducing negligible overheads.
2) We formulate a multi-objective NAS approach, *i.e.*, HSCoNAS. Besides, we propose a novel dynamic channel scaling scheme to enable the channel-level explorations. Also, we present a progressive space shrinking to refine the search space towards target hardware, followed by an evolutionary algorithm to perform efficient search.
3) We perform extensive experiments on ImageNet with three hardware devices, *i.e.*, GPU, CPU, and edge device, which demonstrate the superiority of HSCoNAS.

This work is partially supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (MOE2019-T2-1-071) and Tier 1 (MOE2019-T1-001-072), and partially supported by Nanyang Technological University, Singapore, under its NAP (M4082282) and SUG (M4082087).

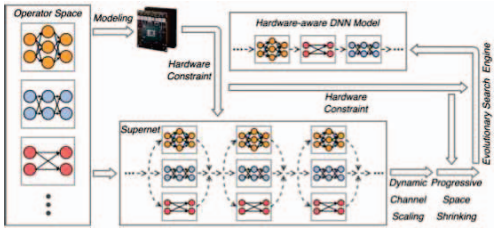

Fig. 1: Overview of the proposed HSCoNAS framework.

II. Preliminaries and Problem Formulation

A. Preliminaries

NAS has been regarded as a promising alternative to automate the design of competitive DNNs. Prior NAS works [6], [10], [14] usually construct an over-parameterized network with L layers, namely supernet $\mathcal{N}$, to ease the search of optimal neural architectures. As illustrated in Fig. 1, the supernet can be formulated as a directed acyclic graph (DAG) based on a set of K operators, *i.e.*, $\mathcal{O} = \{op_i\}_{i=1}^{K}$, where each layer has K different operators. Note the operator can be the basic convolution, pooling, or building blocks from manually-designed DNNs like ShuffleNetV2 [9] and MobileNetV2 [13]. Finally, an architecture candidate $arch$ can be sampled from the supernet by selecting one operator for each layer, *i.e.*, $arch = \{op^l\}_{l=1}^{L} \in \mathcal{N} = \{\mathcal{O}^l\}_{l=1}^{L}$. Note we set $L = 20$ and $K = 5$ across this work. Thus, once the supernet is well trained, we can evaluate architecture candidates (*i.e.*, subgraphs) with inherited weights from the supernet by means of the weight-sharing technique [10], thereby avoiding the considerable overheads of training vast stand-alone DNNs.

B. Problem Formulation

In deep learning driven platforms, they may trade latency for higher accuracy, and vice versa. Thus, to accomplish flexibility, we present a multi-objective formula to achieve the trade-off between accuracy and latency as follows:

$$\underset{arch \in \mathcal{A}}{\text{maximize}} \;\; ACC(arch) + \beta \times \left| \frac{LAT(arch)}{T} - 1 \right| \quad (1)$$

2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) | 978-3-9819263-5-4/21/$31.00 ©2021110.23919/DATE51398.2021.9473937

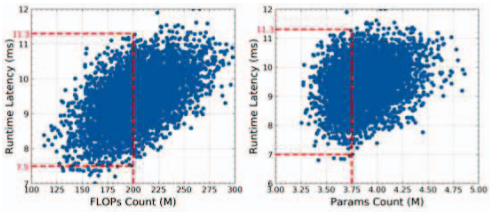

Fig. 2: Illustration of relationships between the runtime latency and FLOPs (*left*) / Params (*right*) count.

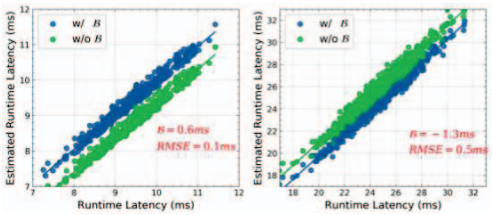

Fig. 3: Illustration of the effectiveness of proposed hardware performance modeling method on GPU (*left*) and CPU (*right*).

where $\mathcal{A}$ denotes the search space. $ACC(\cdot)$ and $LAT(\cdot)$ denote accuracy on target task and runtime latency on target hardware, respectively. T is the specified latency constraint on target hardware. $\beta < 0$ is the trade-off coefficient. For simplicity, we denote the above objective as $\mathcal{F}(arch, T)$. Therefore, we can either penalize the architecture with high latency or with low accuracy, thereby achieving an effective trade-off between accuracy and latency.

III. HSCoNAS Framework

The demonstrations of the proposed HSCoNAS are twofold. From *hardware's perspective*, we present an effective hardware performance modeling method to approximate the runtime latency of DNNs upon target hardware. From *software's perspective*, we introduce a multi-objective evolutionary algorithm (EA) based NAS approach, where a novel dynamic channel scaling scheme is integrated to enable HSCoNAS to perform channel-level explorations. Besides, we introduce a novel progressive space shrinking method to improve the quality of the search space towards target hardware.

A. Hardware Performance Modeling

As illustrated in Fig. 2, we observe that neural architectures with the same FLOPs or Params count significantly differ regarding the runtime latency. Therefore, the FLOPs or Params count is a hardware-agnostic metric and is inadequate to reflect the runtime performance upon target hardware. However, directly measuring the runtime performance on target hardware for $arch \in \mathcal{A}$ is prohibitively expensive since the search space of NAS is immensely large, *e.g.*, $|\mathcal{A}| \approx 9.5 \times 10^{33}$ in HSCoNAS. To tackle this, we analytically model the runtime latency for $arch \in \mathcal{A}$ using the following formulation:

$$LAT(arch) = \sum_{l=1}^{L} op^l + \mathcal{B} \tag{2}$$

where op^l represents the operator of l-th layer in $arch$, *i.e.*, $arch = \{op^l\}_{l=1}^{L}$. Here $\mathcal{B}$ is incorporated to compensate the communication overheads in sequential layers and can be empirically approximated as follows:

$$\mathcal{B} = \frac{1}{M}(\sum_{i=1}^{M} LAT^{+}(arch_i) - \sum_{i=1}^{M} LAT(arch_i)) \tag{3}$$

where $LAT^{+}(arch_i)$ denotes the on-device runtime latency of $arch_i$. M is the number of architectures sampled from $\mathcal{A}$.

We perform experiments on three hardware devices, *i.e.*, GPU (Nvidia Quadro GV100), CPU (Intel Xeon Gold 6136), and edge device (Nvidia Jetson Xavier). Please note we set the batch size as 1, 16, 32 for CPU, edge device, and GPU since small batch size will lead to resource under-utilization [1]. For the edge device, the power mode 6 is applied across this work. As illustrated in Fig. 3, we observe a strong correlation between the on-device and the estimated runtime latency after incorporating $\mathcal{B}$. Notably, the proposed method achieves an extremely low root-mean-squared-error (RMSE) of 0.1ms, 0.5ms, 1.7ms for CPU, GPU, edge device, respectively.

B. Dynamic Channel Scaling

In literature, the hardware-aware NAS works [1], [14], [15] merely search for the optimal configuration of the operator in each layer while keeping the number of channels in each operator fixed. However, as demonstrated in [13], [16], the number of channels has an essential impact on both accuracy and runtime efficiency upon target hardware. Nonetheless, the conventional channel scaling scheme [16] is performed after the neural architecture is determined, and a uniform scaling factor is imposed across layers as illustrated in Fig. 4 (*top*), thereby cannot achieve effective trade-offs between accuracy and efficiency. To alleviate these issues, we propose a dynamic channel scaling scheme as depicted in Fig. 4 (*bottom*), which is further integrated into HSCoNAS to enable the channel-level explorations, *i.e.*, determining the best channel configuration for each layer. To achieve this, we first define a list of n channel scaling factors in HSCoNAS as $C = \{c_1, c_2, ..., c_n\}$, *e.g.*, $\{0.1, 0.2, ..., 1.0\}$. We denote S^l as the maximum number of channels for the l-th layer.

Recall that NAS algorithms [1], [14] select one proper operator op^l from the operator set $\mathcal{O}$ for each layer l to generate an architecture candidate. To begin with, we initialize the number of channels as S^l for layer l in the supernet, *i.e.*, initialized with the maximum number. In practice, the dynamic channel scaling is implemented via scaling down from the maximum number of channels. The reason behind this is the scaling down method can avoid collapses during training the supernet since we need to reconstruct the supernet topology and reload the inherited weights into memory once the scaled number of channels is larger than the initialized one. Throughout training the supernet, we leverage a masking mechanism with the vector $\mathbb{I}^l \in \{0, 1\}^{S^l}$, where the scaling factor $c^l \in C$ is used to manipulate the number of channels for each operator within layer l, *i.e.*, assigning 1 to the selected channels and 0 to the masked ones. By means of the scaling down method, we can derive the output of op^l as

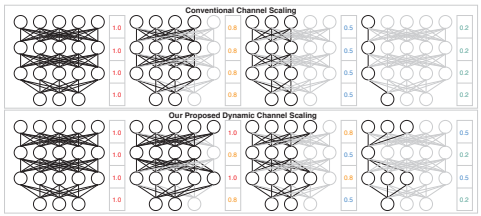

Fig. 4: Comparisons between the conventional (*top*) and the proposed dynamic channel scaling (*bottom*) scheme. Note the scaled number of channels is rounded (e.g., $5 \times 0.5 \approx 3$).

$\mathbb{I}^l \times op^l(x)$, where x denotes the output of the previous layer. To incorporate the dynamic channel scaling into HSCoNAS, we adjust the supernet training accordingly, *i.e.*, we change $arch \in \mathcal{A}$ from $\{op^l\}_{l=1}^{L}$ to $\{op^l, c^l\}_{l=1}^{L}$, where $c^l \in C$ is dynamically imposed across different layers. After the supernet is well trained, the proposed EA-based architecture search (refer to Section III-D) can automatically discover the optimal architecture candidate upon target hardware, *i.e.*, $arch^* = \{op^{l*}, c^{l*}\}_{i=1}^{L}$, using the weight-sharing technique.

C. Progressive Space Shrinking

Since the search space of NAS is combinatorially large, we propose an efficient space shrinking method to progressively prune and shrink the initial search space, which finally arrives at a well-designed subspace where it is much easier to find superior DNNs upon target hardware [11]. Thus, we only need to explore the well-designed subspace instead of the whole space, thereby significantly improving the search efficiency. In HSCoNAS, we characterize the quality of different subspaces upon target hardware with statistical distribution estimates.

Definition 1: Given a set of N architecture candidates uniformly sampled from a subspace $\mathcal{A}^{sub}$, the quality $Q(\mathcal{A}^{sub})$ of $\mathcal{A}^{sub}$ upon target hardware is defined as follows:

$$Q(\mathcal{A}^{sub}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{F}(arch_i, T), \text{ s.t. } arch_i \sim \mathcal{U}(\mathcal{A}^{sub}) \quad (4)$$

where $\mathcal{F}(\cdot)$ represents the objective defined in Eq (1). For instance, for two subspaces $\mathcal{A}^1$ and $\mathcal{A}^2$, if $Q(\mathcal{A}^1) > Q(\mathcal{A}^2)$, this indicates that we have a higher probability to find better DNNs in terms of the trade-off between latency and accuracy within subspace $\mathcal{A}^1$. We set N to 100 across our experiments, which is proven to be sufficient in [11].

The progressive space shrinking consists of three stages: the initial search space $\mathcal{A}$, the first space shrinking to $\mathcal{A}_{ss}^{1st}$, and the second space shrinking to $\mathcal{A}_{ss}^{2nd}$. These procedures are illustrated in Fig. 5 (*right*). To begin with, we train the supernet $\mathcal{N}$ for 100 epochs within the initial search space $\mathcal{A}$, which serves as the foundation for the subsequent space shrinking steps since we need to have architecture samples to approximate the quality of different subspaces. Then, we start the first space shrinking, where we sample a subspace for each operator within each layer and use Definition 1 to evaluate the quality of different subspaces. Finally, the operator with the highest quality is selected for that layer. For the first stage we apply this space shrinking to 20-th, 19-th, 18-th, 17-th, layer by

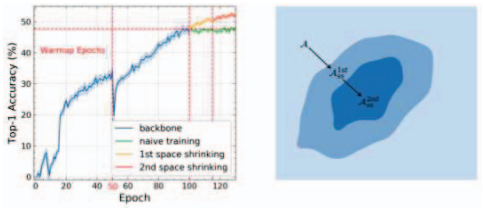

Fig. 5: Illustration of the progressive space shrinking scheme.

layer. Note that when we evaluate the subspaces of the current layer, the operator of its subsequent layer should be fixed. For example, when evaluating the 19-th layer, we fix the operator of 20-th layer according to the subspace quality. After applying the space shrinking to the four layers, we complete the first stage space shrinking, reaching a smaller design space $\mathcal{A}_{ss}^{1st}$, which reduces the space size by three orders of magnitudes. Furthermore, we tune the supernet within subspace $\mathcal{A}_{ss}^{1st}$ for 15 epochs and then conduct the second space shrinking in order of 16-th, 15-th, 14-th, 13-th layer in the same way. At the end, we arrive at $\mathcal{A}_{ss}^{2nd}$, further reducing the space size by another three orders of magnitudes.

In terms of complexity, if we evaluate the subspaces of four layers at the same time, it needs to evaluate 5^4 subspaces, whereas our method only needs to evaluate 5×4 subspaces. Moreover, as illustrated in Fig. 6 (*left*), we observe after each space shrinking the supernet obtains higher accuracy when compared with *naive training*, which indicates to continue training the supernet within the initial space $\mathcal{A}$.

D. Evolutionary Architecture Search

With all the key components introduced before, this section presents our architecture search method. EA and RL are two widely used algorithms in literature. However, RL incurs a high search cost since it is hard to converge [17]. Thus, we adopt EA across this work, which is as effective as RL but with higher efficiency [12]. HSCoNAS aims to search for the architecture candidate with the highest objective score (refer to Eq (1)), which can be formulated as follows:

$$arch^* = \arg\max_{arch \in \mathcal{A}} \mathcal{F}(arch, T) \quad (5)$$

where $arch = \{op^l, c^l\}_{l=1}^{L}$ represents the architecture candidate sampled from the supernet with inherited weights. We set the number of generations as 20, the size of population as 50, the number of parents as 20, respectively. During each evolution, crossover with a probability of 0.25 and mutation with a probability of 0.25 jointly work to yield efficient explorations not only on the operator level but also on the channel level. We take the evolutionary results on the edge device as an example, which is illustrated in Fig. 6 (*top*), where a specified latency requirement of 34ms is given. Notably, HSCoNAS discovers an optimal neural architecture with the runtime latency of 34.3ms, which approximately meets the specified latency constraint. We visualize the results in a histogram as seen in Fig. 6 (*bottom*), where we find EA can find more architecture candidates which have the runtime latency closed to the specified latency constraint, *i.e.*, 34ms.

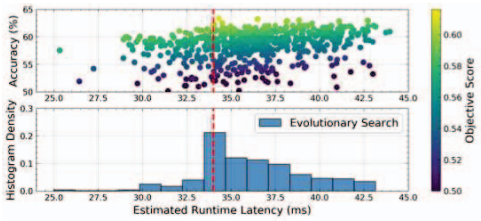

Fig. 6: Illustration of the evolutionary results. The red dashed line denotes the latency constraint for edge device, *i.e.*, 34ms.

IV. Experimental Settings and Results

In this section, we evaluate HSCoNAS on three hardware platforms, *i.e.*, GPU (Nvidia Quadro GV100), CPU (Intel Xeon Gold 6136), edge device (Nvidia Jetson Xavier), with the specified latency constraint of 9ms, 24ms, 34ms, respectively.

A. Experimental Settings

In this work, we apply ImageNet as the experimental dataset. In practice, we train the supernet backbone using the SGD optimizer with a momentum of 0.9, a weight decay of 3×10^{-5}, a norm gradient clipping of 5, a batch size of 512, a learning rate of 0.5 annealed down to zero following the cosine schedule for 100 epochs. The standard data augmentations are applied. After each stage of progressive space shrinking, the supernet is tuned within the shrunk search space for 15 epochs with an initial learning rate of 0.01 and 0.0035, respectively. We denote those architectures discovered by HSCoNAS as HSCoNets, which will be trained from scratch for fair comparisons. The training settings are the same as training the supernet backbone with two exceptions. The batch size is set to 1024 and the learning rate warm-up strategy is applied for the first five epochs. Afterward, we set the batch size as 1, 16, 32 for CPU, edge device, GPU to evaluate the runtime latency.

B. Experimental Results

Similar to [4], [13], we apply two channels layouts, *i.e.*, $[48, 128, 256, 512]$ and $[68, 168, 336, 672]$, to generate two different sizes of HSCoNets denoted as HSCoNet-A and HSCoNet-B, respectively. The search space consists of building blocks of ShuffleNetV2 [9] with different kernel sizes (*e.g.*, 3×3). Besides, a skip-connection operation is incorporated to allow flexible architecture search as seen in [1], [6], [14].

Results and comparisons with recent state-of-the-art works are summarized in TABLE I. Please note HSCoNet-Edge-A denotes the hardware-aware DNNs for the edge device, and vice versa. Remarkably, the HSCoNets outperform those manually-designed lightweight DNNs like ShuffleNetV2 [9] and MobileNetV2 [13] in terms of both accuracy and runtime latency on our three experimental hardware devices, respectively. Besides, HSCoNet-GPU-A obtains comparable accuracy as ProxylessNAS-GPU on ImageNet while being x1.3 faster on GPU. Besides, HSCoNet-GPU-B maintains similar runtime latency on GPU as ProxylessNAS-GPU but achieves +1.3% higher accuracy. Furthermore, HSCoNet-CPU-B yields the lowest top-1/5 error of 23.5%/6.8% among those state-of-the-art DNNs but with an inference speedup of x3.1 on CPU when compared with the hardware-agnostic NAS, *i.e.*, DARTS [6], which demonstrates the effectiveness of our hardware-software co-design paradigm.

TABLE I: Comparisons with state-of-the-art approaches.

	Test Error (%)		Runtime Latency (ms)		
	Top-1	Top-5	GPU	CPU	Edge
Manually-Designed Models					
MobileNetV2 1.0× [13]	28.0	-	11.5	25.2	61.9
ShufflenetV2 1.5× [9]	27.4	-	10.5	34.3	65.9
MobileNetV3 (*large*) [4]	24.8	-	12.2	31.8	61.1
State-of-the-art NAS Models					
DARTS [6]	26.7	8.7	17.3	81.4	68.7
MnasNet-A1 [14]	24.8	7.5	10.9	26.4	51.8
FBNet-A [15]	27.0	9.1	10.5	21.6	48.6
FBNet-B [15]	25.9	8.2	13.6	25.5	57.1
FBNet-C [15]	25.1	7.7	15.5	28.7	66.4
ProxylessNAS-GPU [1]	24.9	7.5	12.0	24.5	57.4
ProxylessNAS-CPU [1]	24.7	-	16.1	29.6	70.1
ProxylessNAS-Mobile [1]	25.4	7.8	11.5	26.4	53.5
Hardware-Aware Models Discovered by HSCoNAS					
HSCoNet-GPU-A	**25.1**	7.7	**9.0**	26.5	43.4
HSCoNet-CPU-A	25.3	**7.6**	10.1	**22.8**	43.1
HSCoNet-Edge-A	25.7	8.1	9.9	25.8	**34.9**
HSCoNet-GPU-B	23.6	6.9	**12.0**	31.6	76.9
HSCoNet-CPU-B	**23.5**	**6.8**	13.4	**26.4**	69.1
HSCoNet-Edge-B	23.8	6.9	12.9	31.8	**52.7**

V. Conclusion

In this work, we introduce a hardware-aware evolutionary algorithm (EA) based neural architecture search (NAS) framework, dubbed HSCoNAS. HSCoNAS is integrated with an effective hardware performance modeling method and two NAS improvements to automate the design of efficient deep neural networks (DNNs) upon target hardware devices. Extensive experimental results demonstrate HSCoNAS outperforms recent state-of-the-art methods in terms of latency and accuracy. In future, we plan to extend HSCoNAS, which will incorporate different hardware constraints like power consumption.

References

[1] H. Cai and et al. Proxylessnas: Direct neural architecture search on target task and hardware. *ICLR*, 2019.
[2] C. Hao and et al. Fpga/dnn co-design: An efficient design methodology for 1ot intelligence on the edge. In *DAC*, 2019.
[3] C. Hao and et al. Nais: Neural architecture and implementation search and its applications in autonomous driving. *ICCAD*, 2019.
[4] A. Howard and et al. Searching for mobilenetv3. In *ICCV*, 2019.
[5] D. Liu and et al. Bringing ai to edge: From deep learning's perspective. In *arXiv preprint arXiv:2011.14808*, 2020.
[6] H. Liu and et al. Darts: Differentiable architecture search. *ICLR*, 2019.
[7] X. Luo and et al. Edgenas: Discovering efficient neural architectures for edge systems. In *ICCD*, 2020.
[8] X. Luo and et al. Person re-identification via pose-aware multi-semantic learning. In *ICME*, 2020.
[9] N. Ma and et al. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.
[10] H. Pham and et al. Efficient neural architecture search via parameter sharing. *ICML*, 2018.
[11] I. Radosavovic and et al. On network design spaces for visual recognition. In *ICCV*, 2019.
[12] E. Real and et al. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.
[13] M. Sandler and et al. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.
[14] M. Tan and et al. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.
[15] B. Wu and et al. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.
[16] J. Yu and et al. Slimmable neural networks. *ICLR*, 2019.
[17] B. Zoph and et al. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.