HARDWARE-AWARE TRANSFORMABLE ARCHITECTURE SEARCH WITH EFFICIENT SEARCH SPACE

Yuhang Jiang[1], *Xin Wang*[2], *Wenwu Zhu*[1,2]

[1]Tsinghua-Berkeley Shenzhen Institute, Tsinghua University
[2] Department of Computer Science and Technology, Tsinghua University
jyh17@mails.tsinghua.edu.cn, {xin_wang, wwzhu}@tsinghua.edu.cn

ABSTRACT

While Neural Architecture Search (NAS) discovers the optimal topology structure, Transformable Architecture Search (TAS) aims to search for the best width and depth, which is more challenging due to the larger search space. Since FLOPs is inconsistent with the actual latency, hardware-aware TAS uses the inference latency to evaluate the efficiency. However, most existing work focuses on the search strategy, ignoring the critical role of the search space in affecting the actual efficiency. Motivated by it, we study hardware-aware TAS by considering the search space, to the best of our knowledge, for the first time. We propose a hardware-aware transformable architecture search (HTAS) framework to discover the optimal architecture for different hardware. The core of our method is a novel hardware-aware search space, which provides efficient channel choices for the search strategy to sample efficient architectures. Experiments on CIFAR datasets demonstrate the superiority of HTAS over the state-of-the-art method.

Index Terms— Deep Learning; Neural Architecture Search; Transformable Architecture Search; Hardware-aware

1. INTRODUCTION

Deep neural networks have become indispensable in lots of application. The performance of neural networks heavily relies on their architectures, given that different designs of the neural architectures have significant impacts on the performance. Nevertheless, finding the optimal architecture among a large number of candidates is intractable to human, motivating the advent of Neural Architecture Search (NAS) [1, 2, 3, 4] which is capable of automatically searching for the best neural network with the optimal structure.

Most existing work on NAS targets at discovering the optimal topological structure of a neural network including the operator type and the kernel size in each layer, while the width (number of output channels) and depth (number of layers) of the backbone architecture are fixed. However, the width and depth also have a significant influence on the performance of the neural network. The neural network with a wider width and deeper depth usually has higher accuracy but also more computation cost. Therefore, to achieve a trade-off between accuracy and efficiency or adapt to different tasks, Transformable Architecture Search (TAS) [5] adjusts the network size, through searching for the best width and depth configuration for the network backbone. However, the number of candidate width values is usually much larger than that of candidate operator types, which makes the search space of TAS much larger and more challenging to explore than that of NAS.

Besides, in the real application, the actual efficiency of the model such as inference latency varies on different hardware devices, due to their different characteristics. As rapidly increasing kinds of hardware are equipped with neural networks nowadays, there yields a strong urge for studying the more challenging hardware-aware TAS problem in both academy and industry. However, most existing works on TAS adopt hardware-agnostic metrics such as FLOPs to measure the model efficiency or estimate the inference latency, causing inconsistency between these hardware-agnostic metrics and the actual efficiency on real hardware. For example, lower FLOPs does not mean lower inference latency or energy. Other works on hardware-aware TAS [6], consider the inference latency primarily through improving the *search strategy* by regularizing the loss function with hardware-aware regularizers, such as the expected inference latency, which ignores that the search space plays an important role in affecting the actual efficiency of the searched networks [1, 7].

To tackle these challenges, we study hardware-aware TAS by considering the *hardware-aware search space* simultaneously in this paper. We propose a hardware-aware transformable architecture search (HTAS) method to discover the optimal architectures for different hardware. Specifically, we first expand the global search space to include more width candidates. Then we select a subset of efficient width choices from the global search space, and construct a hardware-aware search space for TAS, which will be detailedly discussed in section 3.2. Finally, we utilize the differentiable search strategy to find the best width as well as depth configurations for the neural network.

978-1-7281-1331- 9/20/$31.00 ©2020 IEEE

We evaluate our proposed method on two CIFAR datasets, and the experimental results show that HTAS can discover more efficient architectures with competitive or higher accuracy than the state-of-the-art method. For ResNet110 on CIFAR100, our method is $2.2\times$ faster than TAS [5] on both GPU and CPU, while improving the accuracy by more than 1.3%. For ResNet32 on CIFAR10, HTAS also reaches $1.9\times$ acceleration with higher accuracy.

We summarize our contributions as follows:

- We propose a hardware-aware transformable architecture search (*HTAS*) method. It constructs a novel hardware-aware search space for the search strategy to dicover the optimal neural architecture for the hardware, while requires no detailed knowledge of the hardware. To the best of our knowledge, this is the first exploration of hardware-aware TAS by considering the efficient search space to find the optimal archietcture.
- Experiments show that HTAS outperforms the state-of-the-art method on CIFAR datasets and different hardware, demonstrating the effectiveness of our method. We also compare the models searched on different hardware, which shows the necessity to design the neural network specialized for different target hardware.

2. RELATED WORK

It is a vibrant field to design efficient neural networks. We summarize the related work in this section.

Channel pruning studies the pruning of filters and thus reduces the width of networks. Most channel pruning methods [8, 9, 10, 11] utilize heuristics as the importance metrics to determine which channels to be pruned. For example, [8] leverages the scaling factor of the batch normalization layer. ThiNet [9] guides the channel pruning with the feature map of the next layer. However, the human-designed rules may lead to a sub-optimal architecture, so our method applies the search method to explicitly learn the optimal width and depth of efficient neural architectures.

Neural architecture search (NAS) [2, 1, 12, 3, 4, 13] mostly aims to search for the topology structure of a neural network. [13, 12] propose to utilize evolutionary algorithm to search for neural architecture, while [2, 14] use reinforcement learning. These methods usually require high computation cost for search, while [1, 4, 3] apply differentiable methods to explore the search space, which reduces the search cost dramatically. Apart from the topology structure, the width and depth are also crucial to the performance of the network. Hence, we target at discovering the best width and depth for the neural network automatically.

Transformable architecture search (TAS) [5, 15, 6, 16] adjusts the width and depth of networks to achieve better performance. TAS [5] applies differentiable NAS to reduce the width and depth to get more compact networks, and perform knowledge transfer to the searched architecture. MorphNet [15] shrinks and expands the neural network alternatively to discover the optimal width under resource constraints. NetAdapt[6] adapts the network to different hardware with the hardware-aware resource budgets on the loss function. Most of these methods focus on improving the search strategy to discover the efficient networks under resource constraints, while without much effort on the search space. Different from them, we pay more attention to the design of the search space by constructing a hardware-aware search space, which provides efficient choices for the search strategy to discover efficient architectures for different hardware.

3. PROPOSED METHOD

In this section, we present our *Hardware-aware Transformable Architecture Search* (HTAS) method. HTAS consists of three major steps as shown in Fig. 1. First, we expand the global search space. Second, we select those hardware-friendly channel numbers as the choices to construct hardware-aware search space. Third, we use differentiable search strategy with latency constraints to search for the optimal width and depth of the neural network.

3.1. Global search space expansion

The global search space contains all the possible output channel candidates. Previous transformable architecture search [5, 17] uses a search space limited in the size of the base network, as each layer of a candidate network will not be wider than that of the base network. However, even if a more efficient architecture has smaller size, it can be wider than the original network structure in some layers. It is impossible to find such an architecture in the limited search space and thus may lead to finding a sub-optimal neural architecture. Hence, we expand the global search space to be able to find these better neural architectures. Motivated by MorphNet [15], we expand the width of the base network by e times as the upper bound of the global search space, which makes the global search space larger and more flexible.

Tab. 1 lists the expanded global search space in different stages of ResNets [18] on CIFAR datasets. For example, we set e as 2, and the global search space of the three stages is from 1 to 32, 64 and 128 respectively.

3.2. Hardware-aware search space construction

Since the size of the global search space is usually huge, the search space for the search strategy only consists of the selected choices from the global search space. The search space of existing TAS methods [5] is mostly generated by the product of a set of step-wise ratios and the layer width of the base network. For example, the channel number of the layers in

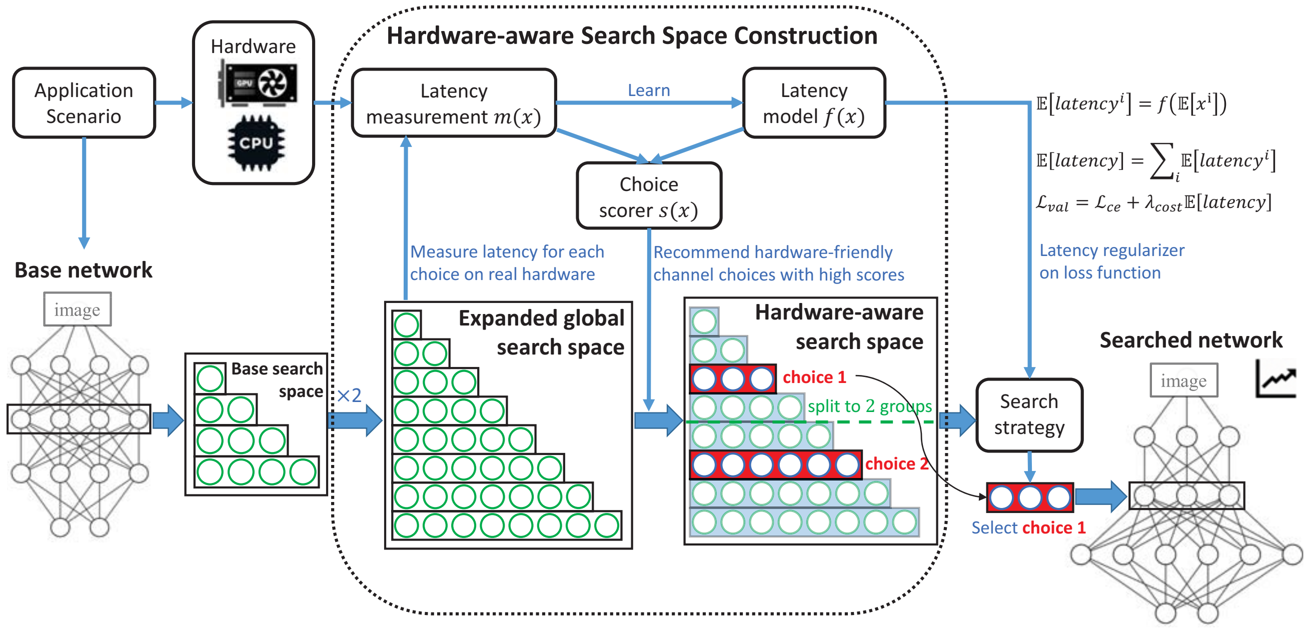
Hardware-aware Search Space Construction
Application Scenario
Hardware
GPU
CPU
Latency measurement $m(x)$
Learn
Latency model $f(x)$
Choice scorer $s(x)$
$\mathbb{E}[latency^i] = f(\mathbb{E}[x^i])$
$\mathbb{E}[latency] = \sum_i \mathbb{E}[latency^i]$
$\mathcal{L}_{val} = \mathcal{L}_{ce} + \lambda_{cost}\mathbb{E}[latency]$
Measure latency for each choice on real hardware
Recommend hardware-friendly channel choices with high scores
Latency regularizer on loss function
Base network
image
Base search space
×2
Expanded global search space
Hardware-aware search space
choice 1
split to 2 groups
choice 2
Search strategy
Select choice 1
Searched network
image

Fig. 1. The framework of hardware-aware TAS searching for width and depth of a neural network. We first expand the global search space by e=2 times. Then based on the latency measurements over the channel numbers, the choice scorer recommends the hardware-friendly channel choices to construct the hardware-aware search space. The final architecture is chosen by the search strategy with the latency regularizer.

Table 1. Global search space expansion for ResNets on CIFAR datasets. "Output size" and "Width" denote the resolution and number of output feature maps in the layer.

Stage	Output size	Width	Global search space
1	32×32	16	$\{n \mid 1 \leq n \leq 16 \times e\}$
2	16×16	32	$\{n \mid 1 \leq n \leq 32 \times e\}$
3	8×8	64	$\{n \mid 1 \leq n \leq 64 \times e\}$

the first stage of Resnet32 is 16, and the ratio can be select from (0.25, 0.50, 0.75, 1.0), so the search space of the layers is (4, 8, 12, 16).

However, this kind of search space is trivial and hardware-agnostic, which ignores the actual efficiency difference between the channel numbers on the hardware. To be more specific, it is common to use FLOPs as the metric of the computation cost to estimate the actual inference latency, but some channel numbers may be more efficient with lower latency than their FLOPs expects. In addition, the most efficient channel choices are usually different on different hardware due to their diverse hardware designs. Motivated by this, we construct the hardware-aware search space by selecting the efficient channel choices for different hardware.

Here we use how much the actual latency is faster the estimated latency as the metric to evaluate the efficiency of the channel choice. To estimate the latency, we measure the inference latency of different channel numbers in the global search space, and then build a latency model $f(x)$ to learn the relationship between the actual latency and the channel numbers.

We use the following model to estimate the inference latency over the output channel numbers.

$$f(x) = kx^{\alpha} + b \tag{1}$$

where x is the output channel number and parameter α is application-specific constant. Fig. 2 shows the inference latency and the estimated latency over the output channel numbers on CPU when the size of the output feature map is 32×32 and α is set as 1.

With the latency model, we can score the channel number by evaluating the efficiency of the output channel numbers, which is defined as the difference between the estimated latency and the actual latency. The scoring function $s(x)$ is as follows.

$$s(x) = f(x) - m(x) \tag{2}$$

where $m(x)$ is the measured actual inference latency on real hardware.

The complexity of different tasks is usually not the same, which requires models with different capacities to learn. For instance, the model adapting to ImageNet usually needs a larger capacity than that on CIFAR-10. On the other hand, the capacity of a layer in the neural network usually increases over the output channel number with the input channel number fixed. To adapt to different tasks, we split the global search space into g groups in each layer, and each group has different width choices with different capacities. In each group, the most efficient channel choice that has the highest

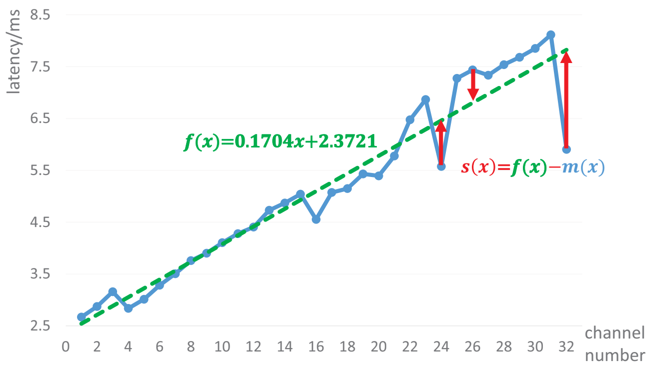

Fig. 2. Measure the latency over the global search space and construct the latency model. Some channel choices are more efficient, as their actual latency (solid) is lower than the estimated latency (dashed). The difference between the actual latency and the estimated latency is the choice score (arrow).

score will be recommended by the scoring function.

$$choice_j = \arg\max_{x\in X_j} s(x) \tag{3}$$

where $choice_j$ is the channel choice selected from the j-th group X_j of the global search space X. The scores over the output channel numbers and the selected channel choices are shown in Fig. 3.

Finally, we gather these recommended channel choices together to construct the hardware-aware search space HS.

$$HS := \cup_{j=1}^{g}\{choice_j\} \tag{4}$$

Tab. 2 lists the constructed hardware-aware search space in each stage of the ResNets on CPU and GPU.

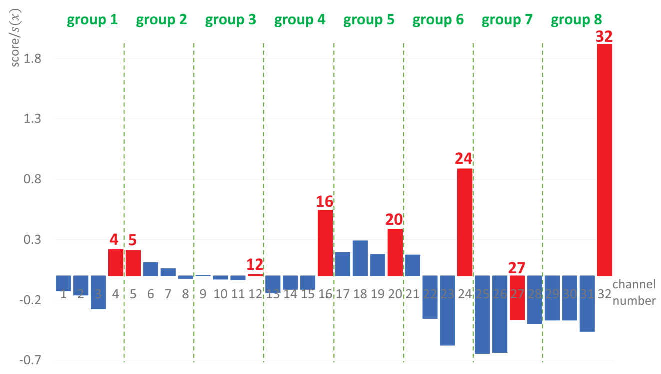

Fig. 3. Select hardware-friendly output channel choices to construct the hardware-aware search space for CPU. The dashed lines split the global search space into g=8 groups. The numbers above the red bars indicate the selected efficient channel choices with the highest score in each group.

3.3. The search strategy

We apply differentiable search strategy in [5] to explore the hardware-aware search space in section 3.2. Transformable Architecture Search aims to discover the best width and depth for an architecture $\mathcal{A}$, which minimizes the validation loss $\mathcal{L}_{val}$ with the network weights optimized on the training dataset. The above problem can be formulated as

$$\min_{\mathcal{A}} \mathcal{L}_{val}(\omega_{\mathcal{A}}^{*}, \mathcal{A}) \quad \text{s.t.} \quad \omega_{\mathcal{A}}^{*} = \arg\min_{\omega} \mathcal{L}_{train}(\omega, \mathcal{A}), \tag{5}$$

where $\omega_{\mathcal{A}}^{*}$ indicates the optimized weights of $\mathcal{A}$. The training loss is the cross-entropy classification loss of the networks. $\mathcal{L}_{val}$ consists of two parts, one is the loss of cross entropy $\mathcal{L}_{ce}$, another is the loss of computation cost, which is the expected latency of the network $\mathbb{E}[latency]$ in this paper. Note that we can also evaluate the computation cost with other metrics, such as energy.

$$\mathcal{L}_{val} = \mathcal{L}_{ce} + \lambda_{cost}\mathbb{E}[latency], \tag{6}$$

$$\mathbb{E}[latency] = \sum_i \mathbb{E}[latency^i] \tag{7}$$

$$\mathbb{E}[latency^i] = f(\mathbb{E}[x^i]) \tag{8}$$

where $\mathbb{E}[latency]$ is estimated by the latency model $f(x)$ in section 3.2, $\mathbb{E}[latency^i]$ is the expected latency and $\mathbb{E}[x^i]$ is the expected width of the i-th layer.

TAS [5] introduces architecture parameters β for the width and depth choices, which generate probabilities p for them. The output is the weighted sum of the output of the choices, so that they can use gradient-based methods to learn the architecture parameters β and the weight parameters ω of the network. The final architecture is generated by selecting the choices with the maximum β. To reduce the search cost, we choose the differentiable search strategy, while it is easy to combine our proposed search space with many other search strategies such as evolutionary algorithms.

Table 2. Hardware-aware search space for CPU and GPU.

Hardware	Output size	Hardware-aware search space
CPU	32×32	4, 5, 12, 16, 20, 24, 27, 32
	16×16	8, 13, 20, 27, 35, 45, 52, 64
	8×8	15, 20, 46, 50, 70, 92, 107, 128
GPU	32×32	4, 6, 12, 16, 20, 24, 28, 32
	16×16	6, 16, 24, 32, 40, 48, 56, 64
	8×8	16, 32, 47, 64, 80, 96, 112, 128

4. EXPERIMENTS AND DISCUSSIONS

4.1. Datasets and settings

Datasets. We demonstrate the effectiveness of our proposed method on CIFAR-10 and CIFAR-100. CIFAR-10 has 10 classes, including 50K training images and 10K test images with a resolution of 32×32. CIFAR-100 is similar to CIFAR-10 except that it contains 100 classes.

Searching and training settings. To construct the hardware-aware search space, we expand the global search space by e=2 times. We split the global search space into g=8 groups. We set α as 1 to learn the latency model and score the channel choices. When searching the neural architecture and retraining of the searched neural architecture, we use the same searching and training setting (including batch size, learning rate schedule, etc) as [5]. For fair comparison, we utilize the same knowledge distillation method [19] as in TAS [5] to train the searched neural architecture. With the pretrained base network as the teacher model, it will transfer knowledge to the searched architecture.

Latency measurements. We conduct the experiments on GPU and CPU with PyTorch. Since the image resolution of CIFAR is 32×32 and relatively small, we use a batch of pictures instead of a single one as the input to measure the inference latency. Furthermore, as GPU is under-utilized with small batch size, the batch size on GPU is larger than that on CPU. Concretely, we measure the GPU latency with a batch size of 1024 on one TitanXP GPU, and the CPU latency under batch size 4 on a server with two 2.30GHz Intel(R) Xeon(R) CPU E5-2630 0. Each reported latency number is the mean of 2000 latency measurements.

Baseline. Prevailing hardware-aware TAS methods mainly focus on the search strategy, which is orthogonal to our work focusing on the search space. Hence, we do not compare HTAS model with those studying the search strategy. Instead, we choose one state-of-the-art method TAS [5] for comparison, and apply the same search strategy to explore the benefit of our proposed search space.

4.2. Comparison with the state-of-the-art method

Results on CIFAR-10 in Tab. 3.We apply HTAS to search for ResNets32, 56, 110 on CIFAR-10 and GPU. With competitive or higher accuracy, our method outperforms TAS [5] in terms of the inference latency for different depths of ResNet. For ResNet32, although the FLOPs of HTAS is higher than TAS, our inference latency of 18.4ms is 1.9× faster than 35.0ms of TAS with higher accuracy as well. The experiment of searching for ResNet110 shows similar improvements, which also validates that the FLOPs is inconsistent with the actual efficiency. The speed-up is contributed to the hardware-aware search space, which provides efficient channel choices that have lower latency than their FLOPs expect.

Results on CIFAR-100 in Tab. 4. We also search for ResNets on CIFAR-100 and different hardware, and HTAS achieves better performance in most cases on both GPU and CPU as well. When searching for ResNet110 on GPU, the inference latency of TAS [5] is 90.9ms, while our method obtains a 2.2× speed-up with 41.0ms and 1.35% improvement in accuracy with lower FLOPs. Besides GPU, we observe that our method also reaches superior performances on CPU. Notably, we obtain 74.93% accuracy for ResNet110, which exceeds TAS [5] by 1.77%, and the inference latency of our searched architecture is 51.5ms, which is also 2.2× faster than 113.6ms of TAS [5]. Our model searched for ResNet32 is slightly worse in accuracy but still faster in latency. The possible reason is that the steps are different between the selected choices in the hardware-aware search space, which may make the search strategy more difficult to explore it than the regular search space with the same step.

Table 3. Comparison with the state-of-the-art method for ResNets on CIFAR-10. ”Dep.” denotes the depth of ResNet. ”CIFAR10” denotes the top-1 accuracy on CIFAR-10. ”GPU” denotes the inference latency measured on GPU. ”FLOPs” denotes the number of multiply-adds. ”Param” means the number of network parameters. Our HTAS achieves lower inference latency with competitive or higher accuracy for different depths of ResNet.

Dep.	Method	CIFAR10	GPU	FLOPs	Param
110	TAS [5]	**94.33%**	81.6ms	124.0M	0.80M
	Ours	**94.33%**	**44.8ms**	119.3M	1.57M
56	TAS [5]	**93.69%**	42.4ms	59.5M	0.45M
	Ours	93.42%	**31.5ms**	58.5M	0.73M
32	TAS [5]	93.16%	35.0ms	35.0M	0.32M
	Ours	**93.34%**	**18.4ms**	37.7M	0.48M

We demonstrate the benefits of considering hardware for the design of the search space through these strong empirical results of our HTAS. In addition, since the size of our hardware-aware search space is the same as the search space in [5], the performance improvement of our method requires no extra searching or training cost.

4.3. Searching models for different hardware

Results of searching models for different hardware in Tab. 5. Previous hardware-agnostic TAS methods deploy the same architecture to different hardware. Nevertheless, the different characteristics of different hardware devices may lead to sub-optimal designs of the neural network.

To validate it, we compare two models searched for GPU and CPU on CIFAR-100. Then we measure the inference latency of the two models both on GPU and CPU. The searched models on different hardware achieve the similar accuracy as 72.34% and 72.39%. For ResNet32-G, the inference latency on its target hardware (GPU) is 21.6ms, while its inference latency hikes to 26.8ms when deployed to CPU. In contrast, for ResNet32-C, the inference latency on GPU is 23.5ms, higher than ResNet32-G by 1.9ms. However, when deployed to its target hardware (CPU), ResNet32-C reaches an inference latency of 24.4ms, which is faster than ResNet32-G by 2.4ms. This shows that it is necessary to search the specialized architecture for different target hardware.

Table 4. Comparison with the existing method for ResNets on CIFAR-100. ”Manual” denotes the manually designed ResNets without searching. ”GPU” and ”CPU” represent that models are searched and measured on GPU and CPU respectively. ”CIFAR100” denotes the top-1 accuracy on CIFAR-100, ”Latency” denotes the inference latency measured on this hardware. Our HTAS finds more efficient architecture with comparable or improved accuracy on both GPU and CPU.

Depth	Method	GPU				CPU			
		CIFAR100	Latency	FLOPs	Param	CIFAR100	Latency	FLOPs	Param
110	Manual	**74.51%**	144.0ms	253.2M	1.73M	74.51%	196.2ms	253.2M	1.73M
	TAS [5]	73.16%	90.9ms	119.6M	0.84M	73.16%	113.6ms	119.6M	0.84M
	Ours	**74.51%**	**41.0ms**	119.1M	1.71M	**74.93%**	**51.5ms**	126.0M	0.84M
56	Manual	73.18%	73.0ms	125.8M	0.86M	**73.18%**	100.0ms	125.8M	0.86M
	TAS [5]	72.25%	44.1ms	61.2M	0.47M	72.25%	59.1ms	61.2M	0.47M
	Ours	**74.22%**	**29.7ms**	72.9M	0.99M	73.01%	**34.9ms**	66.8M	0.47M
32	Manual	70.44%	41.8ms	69.1M	0.47M	70.44%	56.6ms	69.1M	0.47M
	TAS [5]	**72.41%**	33.7ms	42.5M	0.43M	**72.41%**	42.8ms	42.5M	0.43M
	Ours	72.34%	**21.6ms**	47.0M	0.58M	72.39%	**24.4ms**	47.0M	0.32M

Table 5. HTAS searches efficient models for different hardware on CIFAR-100. ”ResNet32-G” denotes the searched ResNet32 for GPU, ”ResNet32-C” denotes the searched ResNet32 for CPU. ”(t)” denotes the searched model is measured on its target hardware. Both of the searched models are faster on their target hardware.

Model	GPU	CPU	Top-1	FLOPs
ResNet32-G	**21.6ms(t)**	26.8ms	72.34%	47.0M
ResNet32-C	23.5ms	**24.4ms(t)**	72.39%	47.0M

5. CONCLUSION

In this paper, we propose HTAS, a hardware-aware transformable architecture search framework. It constructs a novel hardware-aware search space for TAS, in which the search strategy can discover the best width and depth for different hardware. To the best of our knowledge, this work is the first effort to explore TAS with the hardware-aware search space. Our experiments show that HTAS outperforms the state-of-the-art method on two CIFAR datasets and different hardware, demonstrating the advantage of our method. In the future, we hope to improve the method of constructing the hardware-aware search space and the search strategy.

Acknowledgements. This research is supported by National Natural Science Foundation of China Major Project No.U1611461 and Shenzhen Nanshan District Ling-Hang Team Grant under No.LHTD20170005. Xin Wang and Wenwu Zhu are corresponding authors.

6. REFERENCES

[1] H. Liu, K. Simonyan, and Y. Yang, “Darts: Differentiable architecture search,” in *ICLR*, 2019.

[2] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in *ICLR*, 2017.

[3] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search,” in *CVPR*, 2019, pp. 10734–10742.

[4] H. Cai, L. Zhu, and S. Han, “ProxylessNAS: Direct neural architecture search on target task and hardware,” in *ICLR*, 2019.

[5] X. Dong and Y. Yang, “Network pruning via transformable architecture search,” in *NeurIPS*, 2019.

[6] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, “Netadapt: Platform-aware neural network adaptation for mobile applications,” in *ECCV*, 2018, pp. 285–300.

[7] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in *CVPR*, 2019, pp. 2820–2828.

[8] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” in *CVPR*, 2017, pp. 2736–2744.

[9] J.-H. Luo, J. Wu, and W. Lin, “Thinet: A filter level pruning method for deep neural network compression,” in *CVPR*, 2017, pp. 5058–5066.

[10] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” in *ICLR*, 2017.

[11] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, “Filter pruning via geometric median for deep convolutional neural networks acceleration,” in *CVPR*, 2019, pp. 4340–4349.

[12] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, “Single path one-shot neural architecture search with uniform sampling,” in *arXiv*, 2019.

[13] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in *AAAI*, 2019, vol. 33, pp. 4780–4789.

[14] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, “Efficient neural architecture search via parameter sharing,” in *ICML*, 2018.

[15] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, “Morphnet: Fast & simple resource-constrained structure learning of deep networks,” in *CVPR*, 2018, pp. 1586–1595.

[16] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, “Amc: Automl for model compression and acceleration on mobile devices,” in *ECCV*, 2018, pp. 784–800.

[17] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, “Slimmable neural networks,” in *ICLR*, 2019.

[18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016, pp. 770–778.

[19] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in *NeurIPS-W*, 2014.