Customized Cross-device Neural Architecture Search with Images

Yang Yao[1], Xin Wang[1,2,*], Yijian Qin[1], Ziwei Zhang[1], Wenwu Zhu[1,2,*], Hong Mei[3]
[1]Department of Computer Science and Technology, Tsinghua University
[2]Beijing National Research Center for Information Science and Technology, Tsinghua University
[3]Peking University
{yaoyang21,qinyj19}@mails.tsinghua.edu.cn, {xin_wang,zwzhang,wwzhu}@tsinghua.edu.cn, meih@pku.edu.cn

***Abstract*—Cross-device scenarios have become increasingly common, where non-independently and identically distributed (non-IID) data is generated and stored in different devices. However, the existing cross-device NAS methods only search for a fixed architecture for different devices, neglecting that different devices have varying hardware characteristics and data distributions. In this paper, we propose a novel NAS framework that can customize the most suitable architecture for each device and its associated dataset. Specifically, we propose a *decoupled data feature extractor* and a *device feature extractor* to characterize the complex distributions of the different datasets and diverse hardware features. Then, we propose a *prototype matcher* to customize the operators and shape selection parameters of architectures. Experiments on ImageNet and CIFAR-10 show that our method can discover more efficient and effective architectures in cross-device scenarios than the existing approaches. To the best of our knowledge, this is the first exploration on customized cross-device NAS problem.**

***Index Terms*—Neural architecture search, Cross-device, Non-IID**

I. Introduction

Deep neural networks have achieved great success in many fields, such as image classification, speech recognition, machine translation, and so on. In order to automatically design more effective and efficient model architectures, neural architecture search (NAS) [1] has been proposed as an emerging research field. The goal of NAS is to automatically design neural networks by searching for architectures that work the best for a given task. To date, NAS has achieved great success in designing neural network architectures [1]–[4]. Additionally, NAS can deal with resource-constrained scenarios for specific devices, e.g., by limiting the model size, inference latency or other metrics on a target device. With appropriate model designs and resource constraints, hardware-aware NAS methods [5]–[8] can discover efficient yet effective neural network architectures for various devices, such as mobile phones, CPU, and, GPU.

The proliferation of diverse devices in our daily lives has led to the availability of massive amounts of image data, which is generated and stored on various devices. For example, people take photos using different types of phones and process their images using different hardware ranging from edge devices to GPU servers. For these cross-device scenarios, different hardware have different characteristics and capabilities, leading to diverse architecture performance in terms of efficiency metrics such as latency. Moreover, the associated images stored in different devices are usually non-independently and identically distributed. For example, images in different devices may focus on different objects, have different backgrounds and shooting conditions, or are for different purposes.

Cross-device NAS is further proposed to automatically design architectures in cross-device scenarios [9], [10]. However, the existing cross-device NAS methods only search for a single architecture for all datasets, neglecting diverse device characteristics and associated complex data distributions. As a result, the searched architectures may not perform optimally in each device.

In this paper, we study customized cross-device NAS, which aims to automatically customize the most suitable architecture for each device and its associated dataset. This problem is highly non-trivial due to the following challenges. Firstly, the characteristics of diverse devices and datasets need to be properly modeled in a joint space to share knowledge across devices while customizing efficient and effective architectures for each device. Secondly, the non-IID nature of datasets makes it challenging to model the complex interaction between datasets, devices, and architecture performance.

To solve these challenges, we propose Customized Cross-Device Neural Architecture Search (CCDNAS) approach to customize the most suitable architecture for each device in the cross-device learning scenario under resource constraints. Specifically, we first design a *decoupled dataset feature extractor*, which captures the characteristics of the datasets at different scales to comprehensively model the non-IID dataset. Then, we design a *device feature extractor* to learn the characteristic and capability of the target device by measuring different neural network operators, which serve as important factors in customizing architectures in cross-device scenarios. We further project the dataset feature and device feature into a joint space, which can capture complex data distributions and diverse device properties. Lastly, we propose a *prototype matcher* to obtain the operator selection parameters and shape selection parameters from the extracted features, and customize latency-constrained neural architectures using

*Corresponding authors.

a hardware-aware latency regularizer. We also adopt a super-network in the performance estimation strategy to ensure the efficiency of our proposed method. Experimental results on ImageNet and CIFAR-10, two representative image datasets, demonstrate that our proposed method can search for more efficient and effective architectures than the existing NAS methods in cross-device scenarios.

Our contributions are summarized as follows:

- We study the problem of customizing the optimal neural architecture for each device in cross-device learning scenarios with non-IID datasets. To the best of our knowledge, we are the first to study this critical problem.
- We design a decoupled dataset feature encoder and a device feature extractor, which can effectively characterize the features of non-IID datasets and diverse devices, respectively.
- We further propose a prototype matching method together with a hardware-aware latency regularizer to search for efficient and effective architectures which are customized for different devices while sharing knowledge across devices.

II. Related Works

A. Neural architecture search

Neural architecture search, the process of automatically designing optimal neural network architectures, is an important branch of automated machine learning. [1], [3] used reinforcement learning to search for neural architectures, representing the architecture of a neural network as a set of parameters. An RNN is used as a controller to generate architecture parameters, which is optimized according to the performance of the architecture (such as accuracy) during the search process. To reduce the search cost, [2], [11] proposed differentiable method that constructs hybrid architectures by mixing the output of each possible architecture, enabling the use of gradient-based optimizers during searching. In addition to using reinforcement learning and differentiable strategies, [12], [13] and others have also explored the use of evolutionary algorithms in neural architecture search. Recently, some works focused on searching for efficient architectures for resource constrained scenarios, enabling the searched models to achieve the best results under resource-constraints, such as MobileNetV3 [14], ProxylessNAS [15], MnasNet [8], FBNetV1 [5], and FBNetV2 [6].

B. Neural architecture search on non-IID datasets

In many real-world scenarios, the data of a task often exists in the form of many non-IID datasets, which brings challenges for the design and training of the model. Some federated neural architecture search methods [9], [10] search for a common model architecture for these non-IID datasets under privacy constraints. A recent work [16] extends ProxylessNAS [15] into a federated learning framework while considering multiple devices, dividing several non-IID datasets into subsets with similar characteristics and searching for architectures for each subset. However, these methods do not consider how to customize architectures for each device, which is critical because the architecture that is optimal for the overall data is not necessarily optimal for individual datasets and devices.

III. Proposed Method

This section presents our CCDNAS approach, which is a NAS framework with cross-device under resource constraints. To obtain architectures that can be adapted to different devices and associated datasets, we learn device features and dataset features extracted by our feature extractors to customize the architecture choice. The architecture is generated from the features using a prototype-based customizer. During searching, the model parameters of different architectures are shared via a super-network containing all possible choices of operators and shapes, which is synchronized across devices. The framework of our method is demonstrated in Fig. 1, and the overall algorithm is available in the appendix.

A. Problem formulation

Let $M = \{m_j\}$ be a search space of neural network architectures and $\{(H_i, T_i, D_i^{tr}, D_i^{val})\}_{i=1}^{N}$ be a collection of N devices, where T_i is the hardware metric constraint of device H_i, D_i^{tr} and D_i^{val} represents the training and validation dataset on device H_i, respectively. Our goal is to find the best architectures for each device from the architecture space according to the validation accuracy while satisfying the specific hardware constraints. This problem can be formally described as follows:

$$\begin{aligned} m_i^* &= \text{argmax}_{m \in M_i} \mathcal{A}(m, \theta(m); D_i^{val}) \\ s.t. \quad M_i &= \{m \mid m \in M \wedge \mathcal{M}(m; H_i) \leq T_i\}, \\ \theta(m) &= \text{argmin}_\theta \mathcal{L}(m, \theta; D_i^{tr}) \end{aligned} \tag{1}$$

where $\mathcal{A}(m, \theta; D)$ represents the accuracy of architecture m with parameter θ on dataset D, $\mathcal{M}(m, H)$ represents the hardware metric (e.g., inference latency or energy) of architecture m measured on device H, $\mathcal{L}(m, \theta; D)$ represents the loss of architecture m with parameter θ on dataset D.

B. Decoupled dataset feature extractor

Cross-device scenarios contain diverse devices with different characteristics, capabilities, and non-IID datasets. Capturing the key features of these devices and datasets is important for cross-device NAS. We propose the *joint dataset and device feature extractor* to capture both datasets and device features in a joint manner.

The first part of the *joint dataset and device feature extractor* is the *Decoupled dataset feature extractor*. Considering that the dataset feature needs to be independent of the order of data in the dataset and the feature extractor needs to be applied to datasets with different sizes, we design a single-data feature extractor $F(\cdot)$ and a data feature aggregator $G(\cdot)$. In our work, we use a classical CNN backbone ResNet18 as the single-data feature extractor $F(\cdot)$ to generate the hidden features of the dataset. For the data feature aggregator $G(\cdot)$, we first take the average of all the single data features to capture the global feature of the dataset. Then we use fully-connected

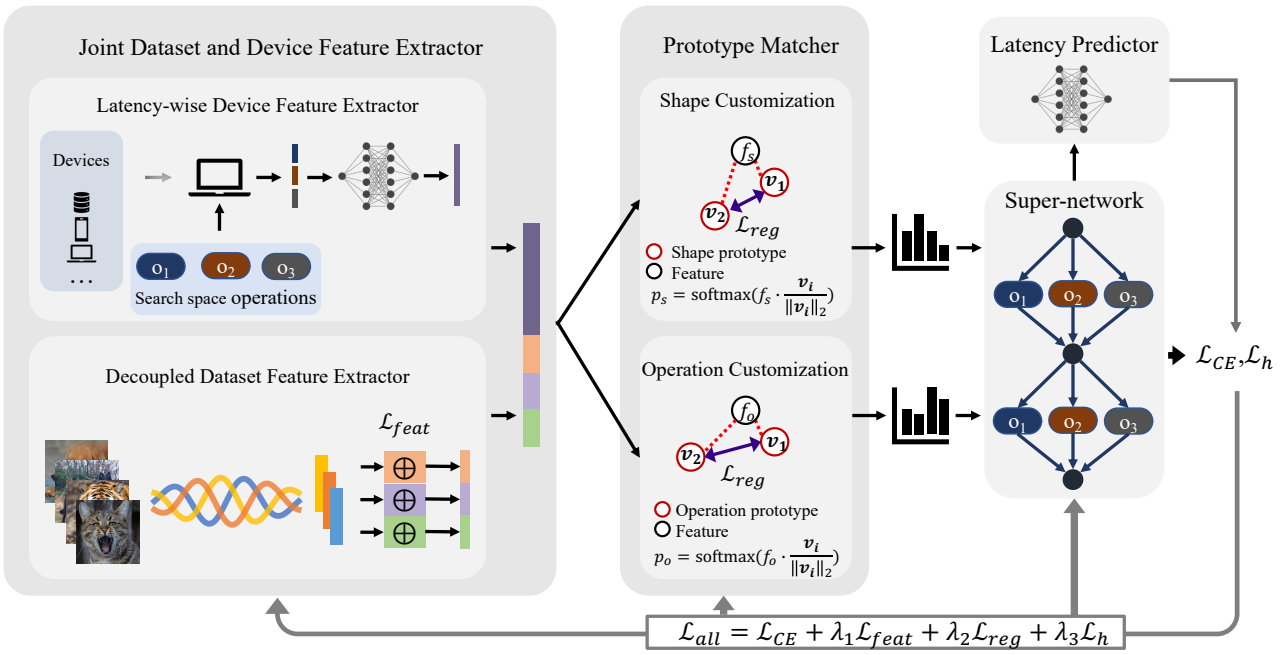


Fig. 1: The framework of CCDNAS. On each device, the features of the device and the associated dataset are extracted with the joint feature extractor. The prototype matcher then produces shape customization and operation customization. During searching, the weights of the architectures are shared via a super-network, which is synchronized across all devices, and the latency of the searched architectures are constrained by the hardware-aware latency regularizer.

layers to process the final feature, i.e., the dataset feature is obtained by applying $F(\cdot)$ and $G(\cdot)$ as follows:

$$f_D = G\left(\sum_{d\in D}\frac{F(d)}{|D|}\right). \tag{2}$$

In order to speed up the feature extraction process, we replace the global average with an exponential moving average during searching:

$$\begin{aligned} a_\eta(t) &= (1-\eta)a_\eta(t-1) + \eta F(d(t)) \\ f_\eta(t) &= G(a_\eta(t)). \end{aligned} \tag{3}$$

where t is the index of training iterations, $d(t)$ is the training data in this iteration, $f_\eta(t)$ is the extracted dataset feature in this iteration, and η is the decay coefficient.

In this process, η controls the influence of different batches of data. The choice of η can influence the characteristics of the extracted feature. A large choice of η only allows the feature extractor to see a small amount of data, making it more likely to produce features common to each individual data. On the contrary, a small choice of η can lead to features about the diversity of data. Therefore, in order to decouple the characteristics of different scales of the dataset, we construct multiple dataset feature extractors with different η values and combine their results as the final dataset features:

$$f_D = [f_{\eta_1}, f_{\eta_2}, \ldots, f_{\eta_k}]. \tag{4}$$

To ensure the consistency and order independence of the feature extractor, we further introduce a consistency loss to regularize the feature differences between iterations:

$$\mathcal{L}_{\text{feat}} = \sum_i \|f_{\eta_i}(t) - f_{\eta_i}(t-1)\|^2. \tag{5}$$

C. Latency-wise device feature extractor

The second part of the *joint dataset and device feature extractor* is the *Latency-wise device feature extractor*. The goal is to extract knowledge about device characteristics to help the search process, considering that operations perform differently on different devices, e.g., efficient operators on one device may become inefficient on another due to device structure and operator implementations. In addition, we also need detailed information about the efficiency of operators to search for architectures under specific resource constraints. Therefore, we propose latency-wise feature extractor to encode such device knowledge.

Device features need to reflect the performance of architectures on the given device. Because the architecture is composed of operators, the performance of the architecture can be reflected by the performance of the operators. In our proposed method, we focus on operator inference latency, which is an important metric in hardware-aware NAS [8], [14], [15]. Notice that our method can be easily extended to contain more device metrics, which we leave for future works. We measure the inference latency of all operators in the search space directly on all devices. These measured metrics

are collected into a vector as the representation of the device. We further use an MLP to process the vector into the final device features:

$$f_H = \text{MLP}([\mathcal{M}(o_1, H), \mathcal{M}(o_2, H), \ldots, \mathcal{M}(o_n), H]), \quad (6)$$

where $\{o_i\}$ are the operators in the search space, $\mathcal{M}(o, H)$ represents the inference latency of operator o measured on device H.

D. Architecture customization with prototype matcher

After jointly extracting dataset and device features, we search for customized architectures on different devices and their associated datasets through customizing proper operations and network shapes for them. Inspired by [17], we use the extracted features as the input of the prototype-based architecture customization module to obtain specific architectures. For each layer in the search space, we represent the candidate operators with learnable prototype vectors $v_{o,i}$. The operator choice $p_{o,i}$ is determined by calculating the cosine similarity between feature vectors and prototype vectors.

$$p_{o,i} = \frac{\exp(\hat{p}_{o,i})}{\sum_j \exp(\hat{p}_{o,j})}, \quad \hat{p}_{o,i} = [f_D, f_H] \cdot \frac{v_{o,i}}{\|v_{o,i}\|_2}, \quad (7)$$

where $[\cdot, \cdot]$ denotes concatenation. The choice of channel width $p_{s,i}$ is determined in a similar way using prototype vectors $v_{s,i}$.

In order to avoid mode collapse, i.e., all prototype vectors become very similar and indistinguishable, we use the distances between prototype vectors as a regularization loss $\mathcal{L}_{\text{reg}}$. For each layer of the hybrid architecture, its output is defined by mixing the outputs of all candidate operators and channel widths according to p_s and p_o. The parameters of the hybrid architecture are shared via a super-network to reduce the amount of computation during searching.

E. Hardware-aware latency regularizer

To satisfy the hardware metric constraints, we further design a hardware-aware latency regularizer $\mathcal{L}_h$ to limit the size of the generated network architectures. We treat the operator and shape choices as probability distributions and define $\mathcal{L}_h$ as the expectation of a cost function $\mathcal{C}(m; H_i)$ over architecture m and device H_i. We use an unbiased gradient estimate and compute $\mathcal{L}_h$ as follows:

$$\mathcal{L}_h = \frac{1}{k} \sum_{i=1}^{k} \log(p_o(m_i) p_s(m_i)) \mathcal{C}(m_i; H_i), \quad (8)$$

where $m_1, \ldots, m_k$ are architectures sampled from the distributions p_o and p_s. The cost function $\mathcal{C}$ depends on which hardware metric is used. In the case of inference latency:

$$\mathcal{C}(m; H_i) = \max(\mathcal{M}(m; H_i) - T_i, 0), \quad (9)$$

i.e., zero when the inference latency is below the constraint, and a linear function of the latency otherwise.

TABLE I: Result on ImageNet. "Acc." denotes model accuracy, while "Lat.' denotes inference latency in milliseconds.

Device	CPU		GPU	
Model	Acc.	Lat.	Acc.	Lat.
ResNet18	87.62	84.7	88.49	26.7
MnasNet	87.18	**27.7**	88.13	23.3
MobileNetV3	87.24	30.1	88.53	22.6
DARTS	87.12	59.1	87.84	37.2
One-Shot NAS	84.99	31.2	86.49	27.1
ProxylessNAS	87.79	43.0	88.75	30.5
FBNetV2	87.48	29.2	88.70	**20.2**
CCDNAS (ours)	**88.14**	30.9	**89.63**	23.7

F. Optimization procedure

By combining the standard cross-entropy loss $\mathcal{L}_{\text{CE}}$ for image classification and other losses defined in the subsections above, we can obtain the final loss objective. We use hyper-parameters λ_1, λ_2, and λ_3 to control the impact of losses:

$$\mathcal{L}_{\text{all}} = \mathcal{L}_{\text{CE}} + \lambda_1 \mathcal{L}_{\text{feat}} + \lambda_2 \mathcal{L}_{\text{reg}} + \lambda_3 \mathcal{L}_h \quad (10)$$

The parameters on each device are synchronized to their average value over all devices [18] during searching. In this way, knowledge learned from different devices and datasets can be shared, while customized architectures can be obtained for each device in a differentiable manner.

IV. Experiments

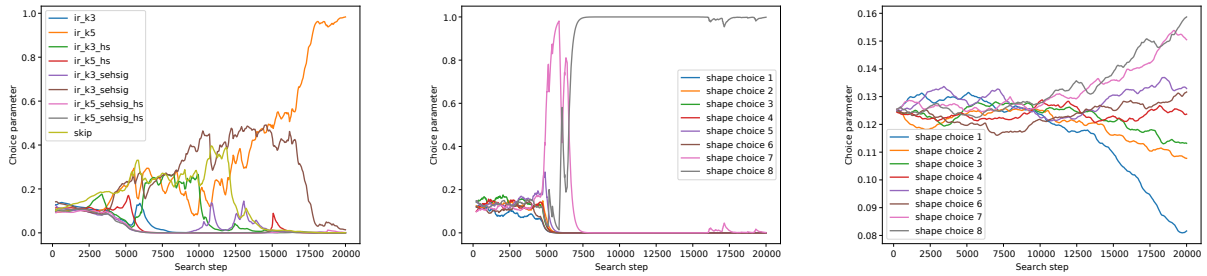
A. Experimental setup

To compare the performance of different NAS algorithms in cross-device scenarios, we construct non-IID image datasets with distribution differences and store different parts of the datasets in different devices, simulating the real cross-device scenario. Specifically, we adopt two classes of devices with large differences in performance, i.e., CPUs and GPUs, in our experiments. We provide the details as follows.

a) Datasets: We use two datasets, ImageNet and CIFAR-10, in our experiments. For each dataset, we divide it into multiple non-IID subsets, which are assigned to the devices in our experiments.

b) Baseline: To demonstrate the effectiveness of our method, we compare our method with a manually-designed architecture ResNet18 [19], several architectures found by NAS methods including MnasNet [8] and MobileNetV3 [14], NAS methods including DARTS [2] and One-Shot NAS [20], and hardware-aware NAS methods including ProxylessNAS [15] and FBNetV2 [6].

c) Search space: For ImageNet, we take the search space from FBNetV2 [6], which is a lightweight layer-wise space for efficient neural networks. For CIFAR-10, as the images are of smaller sizes (32×32), we modify the search space to fit its size, and the macro-architecture search space for CIFAR-10 is shown in the appendix.



(a) Operator selection parameters (ours) (b) Shape selection parameters (ours) (c) Shape selection parameters (FBNetV2)

Fig. 2: The change in selection parameters during searching. The results indicate that our proposed method can converge faster than FBNetV2 using the same search space.

d) Devices and hardware metric constraints: On ImageNet, we adopt two devices. The first device is a CPU (Intel(R) Xeon(R) Gold 6330). The second device is a GPU (NVIDIA GeForce RTX 3090). For CIFAR-10, we adopt three devices. The first device is a CPU (Intel(R) Xeon(R) Gold 6330 CPU). The second device is a CPU (AMD EPYC 7642 48-Core Processor). The third device is a GPU (NVIDIA GeForce RTX 3090). On both datasets, we choose inference latency as the hardware metric in our experiments.

B. Results on ImageNet

The results are shown in Table I. It can be seen that model accuracy does not increase monotonically with their inference latency. Instead, some architectures are able to achieve higher accuracy with lower latency. With similar performance in latency, the architecture searched by our method on each device achieves the highest accuracy compared to other methods. Moreover, we use the same search space as FBNetV2 on ImageNet, but the architectures we obtain are more efficient and effective. The results demonstrate that our method can integrate information from all devices and datasets during the search process and customize suitable architectures for each device and dataset.

Fig. 2 further shows the change in operator selection parameters and shape selection parameters during the search process. It can be seen that in the early stage of the search, there is no clear preference among the choices in operators and shapes. As the search process goes on, some operators and shapes show significantly higher choice parameters than others, indicating that our search process has learned preferences on these architecture designs. Besides, our method converges faster than FBNetV2 based on the same search space. A plausible reason is that our method can effectively learn device and dataset characteristics through the joint dataset and device feature extractor, making architecture search easier. We provide additional analysis on the choice of architectures in the appendix.

C. Results on CIFAR-10

The results are shown in Table II. Similar to the results on ImageNet, our proposed method obtains a better tradeoff between accuracy and latency in general. Although DARTS and ProxylessNAS have higher accuracy, their latency is very high. On the other hand, though One-Shot NAS can obtain efficient architectures, its search architectures are not as effective. Compared with FBNetV2, our method achieves better accuracy under similar latency limits. In comparison, the models we search for have much lower inference latency and overall higher accuracy. The results show that our method is indeed capable of customizing suitable models for different devices with latency constraints.

TABLE II: Result on CIFAR-10. "Acc." denotes model accuracy, while "Lat.' denotes inference latency in milliseconds.

Device	CPU1		CPU2		GPU	
Model	Acc.	Lat.	Acc.	Lat.	Acc.	Lat.
ResNet18	88.51	4.0	72.41	5.9	73.76	3.7
DARTS	90.70	66.1	73.98	90.3	77.30	47.4
One-Shot NAS	89.30	14.4	73.02	19.3	75.29	10.8
ProxylessNAS	91.51	189.3	75.98	262.5	79.72	161.8
FBNetV2	89.85	11.5	72.50	11.4	75.51	7.6
CCDNAS (ours)	90.23	13.8	74.30	12.2	76.61	5.2

TABLE III: Results of the ablation study. "Acc." denotes model accuracy.

Device	CPU	GPU
Method	Acc.	Acc.
No dataset features	85.77	87.14
No device features	81.33	83.49
No architecture customization	87.18	88.02
Single-device	87.50	88.62
CCDNAS	**88.14**	**89.63**

D. Ablation study

We conduct ablation experiments to explore the function of each module of our method. In the experiments, one or both of the feature extraction modules or the parameter synchronization mechanism is removed, while all other settings, such as devices and latency limits, are kept the same. The results for ImageNet are shown in Table III, while results on CIFAR show similar patterns. It can be seen that our method has the highest accuracy when all modules are used together, showing

that each module of our proposed framework is essential. Additionally, the results with no dataset features are better than with no device features, suggesting that device features, while being easier to learn, have a greater impact on architecture choices under the same latency limits. The results with no architecture customization are close to the single-device NAS methods in Table I, which shows that the dataset features and device features play a great role in our search process.

V. Conclusion

We propose a novel cross-device NAS method called CCD-NAS to customize the optimal neural architecture for each device in cross-device learning scenarios with non-IID image datasets. We design a joint dataset amd device feature extrator to learn the characteristics of device and dataset, and then use the prototype matching method to customize operator parameters and shape parameters of the model for each device. Experimental results show that our proposed method outperform the existing NAS methods on the ImageNet and CIFAR-10 dataset in cross-device scenarios.

Acknowledgment

This work was supported by National Science and Technology Major Project No. 2020AAA0106300, National Natural Science Foundation of China (No. 62250008, 62222209, 62102222), Beijing National Research Center for Information Science and Technology under Grant No. BNR2023RC01003, BNR2023TD03006, and Beijing Key Lab of Networked Multimedia.

References

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

[2] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *ICLR*, 2019.

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

[4] D. Lian, Y. Zheng, Y. Xu, Y. Lu, L. Lin, P. Zhao, J. Huang, and S. Gao, "Towards fast adaptation of neural architectures with meta learning," in *ICLR*, 2020.

[5] 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.

[6] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen, P. Vajda, and J. E. Gonzalez, "FBNetV2: Differentiable neural architecture search for spatial and channel dimensions," in *CVPR*, 2020.

[7] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha, "ChamNet: Towards efficient network design through platform-aware model adaptation," in *CVPR*, 2019.

[8] 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.

[9] S. Laskaridis, J. Fernandez-Marques, and L. Dudziak, "Cross-device federated architecture search," in *FL-NeurIPS*, 2022.

[10] C. He, M. Annavaram, and S. Avestimehr, "FedNAS: Federated deep learning via neural architecture search," *arXiv*, 2020. [Online]. Available: https://arxiv.org/abs/2004.08546

[11] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *CVPR*, 2019.

[12] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *IJCAI*, 2018.

[13] E. Real, S. Moore, A. Selle, S. Saxena, Y. I. Leon-Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *ICML*, 2017.

[14] A. Howard, R. Pang, H. Adam, Q. V. Le, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, and Y. Zhu, "Searching for MobileNetV3," in *ICCV*, 2019.

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

[16] C. Zhang, X. Yuan, Q. Zhang, G. Zhu, L. Cheng, and N. Zhang, "Towards tailored models on private AIoT devices: Federated direct neural architecture search," *arXiv*, 2022. [Online]. Available: https://arxiv.org/abs/2202.11490

[17] Y. Qin, X. Wang, Z. Zhang, P. Xie, and W. Zhu, "Graph neural architecture search under distribution shifts," in *ICML*, 2022.

[18] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[20] 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 *ECCV*, 2020.