DATA: Differentiable ArchiTecture Approximation With Distribution Guided Sampling

Xinbang Zhang, Jianlong Chang, Yiwen Guo, Gaofeng Meng, *Senior Member, IEEE*, Shiming Xiang, Zhouchen Lin, *Fellow, IEEE*, and Chunhong Pan

Abstract—Neural architecture search (NAS) is inherently subject to the gap of architectures during searching and validating. To bridge this gap effectively, we develop *Differentiable ArchiTecture Approximation* (DATA) with *Ensemble Gumbel-Softmax* (EGS) estimator and *Architecture Distribution Constraint* (ADC) to automatically approximate architectures during searching and validating in a differentiable manner. Technically, the EGS estimator consists of a group of Gumbel-Softmax estimators, which is capable of converting probability vectors to binary codes and passing gradients reversely, reducing the estimation bias in a differentiable way. To narrow the distribution gap between sampled architectures and supernet, further, the ADC is introduced to reduce the variance of sampling during searching. Benefiting from such modeling, architecture probabilities and network weights in the NAS model can be jointly optimized with the standard back-propagation, yielding an end-to-end learning mechanism for searching deep neural architectures in an extended search space. Conclusively, in the validating process, a high-performance architecture that approaches to the learned one during searching is readily built. Extensive experiments on various tasks including image classification, few-shot learning, unsupervised clustering, semantic segmentation and language modeling strongly demonstrate that DATA is capable of discovering high-performance architectures while guaranteeing the required efficiency. Code is available at https://github.com/XinbangZhang/DATA-NAS

Index Terms—Neural architecture search(NAS), ensemble gumbel-softmax, distribution guided sampling

1 Introduction

In the last decade, deep learning has shown remarkable passion and potential for AI applications, such as image classification [1], [2], [3], object detection [4], [5] and semantic segmentation [6], [7]. Inspired by its remarkable representation power, deep neural networks have raised the wave of end-to-end learning and transform the dominant factor of these applications from features extraction to architecture design [8]. Unfortunately, deep neural architectures usually need to be elaborately designed for specific tasks, leading to the emergence of another tedious work, *i.e.*, "network engineering". Besides, neural architecture is always treated as a black box due to the lack of interpretability, which indicates that designing suitable architecture typically still requires tremendous efforts from human experts.

In order to eliminate such exhausting engineering, many neural architecture search methods have been invented to accomplishing the task automatically and raise a new steam for AutoML [9], [10], *i.e.*, evolution-based NAS which searches architecture with evolution algorithms [11], [12], [13], [14], [15], [16], [17], [18], reinforcement learning-based NAS which applies an RNN controller trained by RL algorithms to generate coded architectures [19], [20], [21], [22], [23], [24], [25]. However, as various child models are required to be evaluated by these sampling-based methods, they are subject to limited computing resources. Another stream of researchers propose the gradient-based NAS which parameterizes architecture with learnable parameters and updates them efficiently with gradients [26], [27], [28], [29], [30]. Benefiting from such efforts, searching cost is reduced remarkably and significant successes have been achieved in a multitude of fields, including image classification [31], [32], [33], [34], semantic segmentation [35], [36], and object detection [37], [38], [39], [40].

Although the achievements in the literature are brilliant, these methods suffer from the gap between architectures during searching and validating. Subjecting to the continuity requirement of gradients, most gradient-based methods build a continuous supernet containing all candidates by assigning architecture parameters to every possible path. During searching, network weights and architecture parameters can be optimized with gradients. After searching, the final architecture is obtained by simply selecting the path with the highest architecture parameter. In spite of the efficiency demonstrated by gradient-based methods, this pipeline may suffer from several problems. First, feasible

- *Xinbang Zhang, Gaofeng Meng, and Shiming Xiang are with the Department of National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Science, Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: {xinbang.zhang, gfmeng, smxiang}@nlpr.ia.ac.cn.*
- *Chunhong Pan is with the Department of National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Science, Beijing 100190, China. E-mail: chpan@nlpr.ia.ac.cn.*
- *Jianlong Chang is with the Huawei Cloud & AI, and the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences , Beijing 100095, China. E-mail: jianlong.chang@huawei.com.*
- *Yiwen Guo is with the Bytedance AI Lab, Beijing 100190, China. E-mail: guoyiwen.ai@bytedance.com.*
- *Zhouchen Lin is with the Key Lab. of Machine Perception (MoE), School of EECS, Peking University, Beijing 100871, China. E-mail: zlin@pku.edu.cn.*

Manuscript received 13 Mar. 2020; revised 20 June 2020; accepted 12 Aug. 2020. Date of publication 31 Aug. 2020; date of current version 4 Aug. 2021. (Corresponding author: Jianlong Chang.) Recommended for acceptance by H. J. Escalante, J. Vanschoren, W.-W. Tu, Y. Yu, S. Escalera, N. Pillay, R. Qu, N. Houlsby, and T. Zhang. Digital Object Identifier no. 10.1109/TPAMI.2020.3020315

paths in a learnable supernet are dependent on each other and become deeply coupled during searching, architecture parameters cannot indicate the importance of relative operations. Estimating the contribution of single operation by relative weight in the supernet may introduce estimation bias on the importance of operations. Second, as the optimization of network weights and architecture parameters is entirely based on the continuous supernet, obtaining child networks through discretizing the supernet may naturally destroy the completeness of the original network. Due to these gap during searching, the effectiveness and efficiency of these search algorithms may be degraded. In order to eliminate these limitations, Differentiable ArchiTecture Approximation is proposed to minimize the gap of architectures during searching and validating. Instead of optimizing architecture in a supernet, we propose to sample child architectures in the searching process and optimize relative weights and architecture probability. Therefore, different paths are decoupled through sampling and impact from completeness destruction can be avoided. To achieve this goal, we develop a sampling-based estimator *Ensemble Gumbel-Softmax* (EGS) estimator, an ensemble of a group of Gumbel-Softmax estimators, which is in a position to sample multi-operations architectures that approaches the supernet during searching as close as possible, while maintaining the differentiability of a promising NAS pipeline for required efficiency. Besides, *Architecture Distribution Constraint* is proposed to narrow the distribution gap between sampled architectures and supernet as well as improving the stability of sampling. That is, our EGS estimator suffices to not only bridge the gap between searching and validating but also pass back-propagated gradients seamlessly, yielding an end-to-end mechanism of searching suitable neural architectures in a more complex and challenging search space.

To sum up, the main contributions of this work are:

- Though generalizing the Gumbel-Softmax estimator, we develop the EGS estimator, which is capable of performing multi operations sampling in the searching process with high effectiveness and efficiency.
- The EGS estimator enable our algorithm to search architecture with the standard back-propagation and seamlessly bridging the estimating gap of architectures between searching and validating, yielding an end-to-end mechanism of searching deep models in a larger while more challenging search space.
- To minimize the distribution discrepancy between the supernet and the sampled child network, a regulation termed as architecture distribution constraint (ADC) is introduced, which is in a position to reduce the variance of sampling during searching.
- Extensive experiments demonstrate that our algorithm outperforms current NAS methods in searching high-performance convolution and recurrent architectures for image classification, semantic segmentation, few shot learning, unsupervised clustering and language modeling.

It should be noted that a previous version of this work has been published in NeurIPS 2019 [41], we further extend our previous work methodologically and empirically. For the methodological aspect, we improve DATA by introducing an architecture constraint (ADC) to bridge the neural architecture distribution gap caused by completeness destruction. With the help of ADC, architecture distribution appears ideally categorical distribution. As for the empirical aspect, extensive experiments are conducted to demonstrate the effectiveness of our method. First, we extend our method from image classification to other crucial while challenging tasks including few-shot learning and unsupervised clustering with significant improvement, demonstrating the universality of our method. Second, numerous ablation experiments, visualizations and analyses are presented to comprehensively evaluate the proposed method.

The remainder of this paper is organized as follows: a brief review on the related work of sampling-based NAS methods including evolution-based NAS as well as reinforcement learning-based NAS, and gradient-based NAS methods is given in Section 2. We formulate the problem of differentiable neural architecture search in Section 3, followed by the developed NAS method in Section 4. Comprehensive experimental results are reported and analyzed in Section 5. Finally, this paper is concluded in Section 6.

2 Related Work

Recently, discovering neural architecture automatically has raised great interest in both academia and industry [19], [40], [42], [43], [44], [45]. Nowadays, NAS methods can be roughly divided into two classes according to searching strategies [9], *i.e.*, the sampling-based NAS method and the gradient-based NAS method.

2.1 Sampling-Based NAS

Sampling-based NAS methods sample child architectures from particular search spaces and apply nested optimization to search suitable architectures based on the performance of sampled architectures. Typical sampling-based NAS methods include the evolution-based NAS methods and the reinforcement learning-based NAS methods. Historically, evolutionary algorithms have already been heavily investigated and applied in evolving neural architectures [46], [47], [48]. Recently, [11], [13], [14], [15], [16] bring this idea back by applying evolutionary algorithms in a CNN search space. Modifications like inserting layer, adjusting filter size and adding identity mapping are designed as mutations.

Another stream of researches utilize an RNN network as agent to generate architectures automatically. Reinforcement learning is applied to train the agent based on the performance of child architectures. In the pioneering work [24], RNN networks serve as controllers to decide the types and parameters of layers sequentially. The performance of the generated architectures work as rewards for reinforcement learning to train the controller in turn. Although such method achieves remarkable results, 800 GPUs are used to obtain a suitable architecture on the CIFAR-10 dataset, which is extremely computationally expensive. Based on this pipeline, many NAS methods are proposed to accelerate searching process. Specifically, [23], [25] apply a diminished search space, they search the architecture of a single block and stack the searched block structure to generate final network. In [22], network weights are shared among

child networks, saving searching time by reducing the cost of evaluating child networks individually. Additionally, a series of well-performing methods have also been explored, including progressive search [34] and multi-objective optimization [21], [39].

In spite of the high interpretability and feasibility of the sample-based methods, both evolution-based and reinforce learning-based NAS methods have the following two limitations. First, most of them are source-hunger and computationally expensive. An inherent cause is that they recast NAS as a black box optimization problem, which indicates that numerous architectures are required to be validated during searching. Second, the search spaces are constrained, *i.e.*, the number of operations needs to be predefined. Consequently, the performance of these methods will be degraded because of the limited search spaces.

2.2 Gradient-Based NAS

Contrary to treating architecture search as a black-box optimization problem, gradient-based NAS methods utilize gradients to optimize neural architecture [27], [28], [29], [30]. Because of the discreteness of architectures, however, it is impossible to propagate gradients to optimize architects directly. To eliminate this problem, various methods are proposed to estimate gradients for searching architectures. Typically, NAO [28] utilizes RNN networks as encoder and decoder to map architectures into a continuous network embedding space and conduct optimization in this space with gradient-based method. Although NAO achieves remarkable results, 200 GPU days are required to obtain a suitable network, this search cost make it incompetent for wide application. To relax the discrete search space to be continuous, DARTS [27] builds a superent with architecture parameters and optimize architecture parameters with back-propagated gradients. Another stream of researchers formulate NAS as a pruning process. They build a fully connected network and pruning redundant paths with compression methods such as sparse regulation [49] and binarization [50]. Although gradient-based NAS methods demonstrate as efficient as 0.5 GPU days searching cost [27] on the CIFAR-10 dataset, final architectures are obtained by discretizing a continuous supernet, which will naturally lead to the estimation bias of every operation and completeness destruction of supernet. Furthermore, the number of operation for every node is fixed strictly, leading to a limited search space. To prevent the impact of completeness destruction, several methods optimize discrete architecture directly. Technically, ProxylessNAS [50] casts NAS as a path-level selecting process and optimize individual paths with binary optimization [51]. Contemporary to this work, SNAS [30] samples and optimizes candidate architectures directly with concrete optimization [52]. Unfortunately, they only focus on the affect of discretization and still suffer from the limitation of search space, in spite of their remarkable efficiency and results.

Different from the sampling-based NAS and gradient-based NAS, we focus on bridging the gap between searching and validating while keeping the efficiency of gradient-based NAS methods. For effectiveness, an architecture ensemble method is proposed to enlarge search space by allowing more complicated combinations of operations. As for efficiency, operations for different paths are sampled according to the corresponding architecture probabilities where efficient gradient optimization methods are applied to update architecture probabilities.

3 Differentiable Architecture Search

In practice, any architecture in the search space can be parameterized with a binary code. Before introducing our approach, we first briefly review the objective of NAS. Without loss of generality, the architecture search space $\mathcal{A}$ can be naturally represented by directed acyclic graphs (DAG) each consisting of an ordered sequence of nodes. For a specific architecture, it always corresponds to a graph $\alpha \in \mathcal{A}$, represented as $\mathcal{N}(\alpha, w)$ with network weights w. Intrinsically, the goal of NAS is to find a graph $\alpha^* \in \mathcal{A}$ that minimizes the validation loss, where the network weights w^* associated with the architecture α are obtained by minimizing the training loss, *i.e.*,

$$\begin{aligned} &\min_{\alpha \in \mathcal{A}} \ \mathcal{L}_{val}(\mathcal{N}(\alpha, w^*)), \\ &s.t. \ w^* = \arg\min_{w} \mathcal{L}_{train}(\mathcal{N}(\alpha, w)). \end{aligned} \tag{1}$$

This implies that the essence of NAS is to solve a bi-level optimization problem, which is hard to optimize due to the nested relationship between architecture parameters α and network weights w. To handle this issue, we parameterize architectures with binary codes and devote to jointly learning α and w in a differentiable way.

3.1 Parameterizing Architectures With Binary Codes

For simplicity, we denote all DAGs with n ordered nodes as $\mathcal{A} = \{e^{(i,j)} | 1 \le i < j \le n\}$, where $e^{(i,j)}$ indicates a directed edge from the i-th node to the j-th node. Corresponding to each directed edge $e^{(i,j)}$, there are a set of candidate primitive operations $\mathcal{O} = \{o_1, \cdots, o_K\}$, such as convolution, pooling, identity, and zero. With these operations, the output at the j-th node can be formulated as

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}), \tag{2}$$

where $x^{(i)}$ denotes the input from the i-th node, and $o^{(i,j)}(\cdot)$ is a function applied to $x^{(i)}$ which can be decomposed into a superposition of primitive operations in $\mathcal{O}$, *i.e.*,

$$\begin{aligned} &o^{(i,j)}(x^{(i)}) = \sum_{k=1}^{K} A_k^{(i,j)} \cdot o_k(x^{(i)}), \\ &s.t. \ A_k^{(i,j)} \in \{0, 1\}, \ 1 \le k \le K, \end{aligned} \tag{3}$$

where $o_k(\cdot)$ is the k-th candidate primitive operation in $\mathcal{O}$, and $A_k^{(i,j)}$ signifies a binary weight to indicate whether the operation $o_k(\cdot)$ is utilized on the edge $e^{(i,j)}$. For a network, by such definition, there is one and only one architecture code $\mathbf{A} \in \{0,1\}^{n \times n \times K}$ that corresponds to it, which implies that we can learn the code $\mathbf{A}$ to approximate the optimal architecture in $\mathcal{A}$.

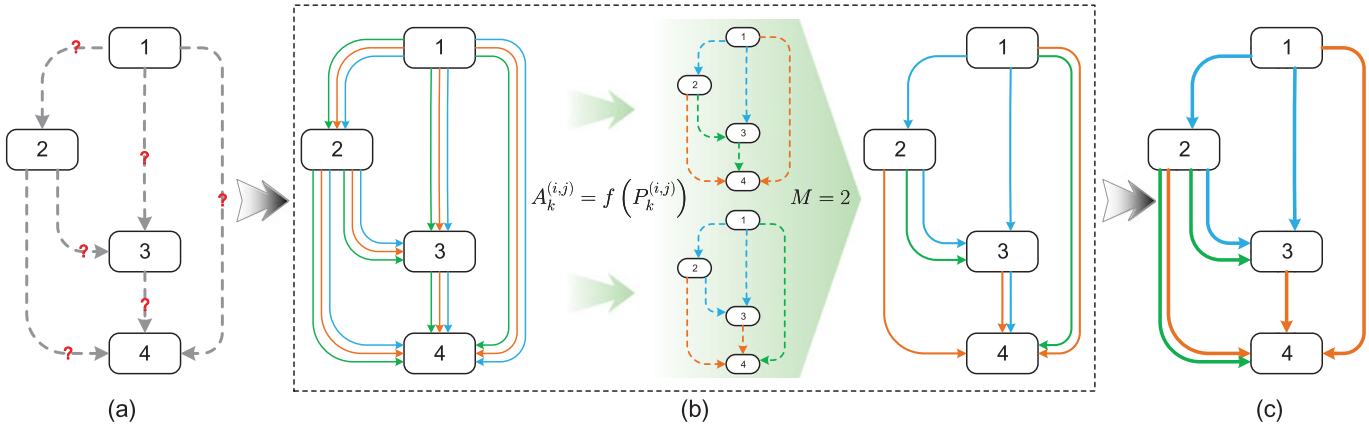


Fig. 1. A conceptual visualization for the searching process with $M = 2$. (a) First, an architecture (i.e., directed acyclic graph) consisting of four ordered nodes is predefined. (b) In the searching process, with three candidate primitive operations (*i.e.*, green, orange and cyan lines), the binary function $f(\cdot)$ is employed to generate a path according to corresponding probabilities in a differentiable manner for M times. The sampled paths are ensembled to generate a new architecture. (c) Finally, the details of the cell can be generated according to the learned architecture probabilities.

3.2 From Probability Vectors to Binary Codes

Benefiting from the uniqueness property of our architecture code $\mathbf{A}$, the task of learning an architecture can therefore be converted to learning the optimal binary code $\mathbf{A}$. However, it is a fussy NP-hard problem, and is difficult to solve directly. To overcome the obstacle, we introduce a binary function $f(\cdot)$ to approach the optimal binary codes with probability vectors, which can be easily obtained in deep models. Formally, the categorical choice in Eq. (3) can be rewritten as

$$\begin{aligned} &\tilde{o}^{(i,j)}(x^{(i)}) = \sum_{k=1}^{K} f\left(P_k^{(i,j)}\right) \cdot o_k(x^{(i)}), \\ &s.t. \ \sum_{k=1}^{K} P_k^{(i,j)} = 1, \quad P_k^{(i,j)} \geq 0, \\ &f\left(P_k^{(i,j)}\right) \in \{0,1\}, \ 1 \leq k \leq K, \end{aligned} \tag{4}$$

where $P_k^{(i,j)}$ is the k-th element in the probability vector $\mathbf{P}^{(i,j)} \in \mathbb{R}^K$ and denotes the probability of choosing the k-th operation on the edge $e^{(i,j)}$, and $f(\cdot)$ represents a binary function that suffices to map a probability vector to a binary code and pass gradients in a continuous manner. Specifically, $f(\cdot)$ is chosen to be a monotonically increasing function in our method, *i.e.*,

$$f\left(P_{k_1}^{(i,j)}\right) \leq f\left(P_{k_2}^{(i,j)}\right), \text{ if } \ P_{k_1}^{(i,j)} \leq P_{k_2}^{(i,j)}, \tag{5}$$

where $1 \leq k_1, k_2 \leq K$, $P_k^{(i,j)}$ is the k-th element of $P^{(i,j)}$. By substituting $A_k^{(i,j)}$ with $f(P_k^{(i,j)})$ and considering $P_k^{(i,j)}$ instead as the variable to be optimized, we have successfully achieved a continuous relaxation. Benefiting from the flexibility of our formulation, furthermore, the optimization of NAS in Eq. (1) can be seamlessly jointed together, *i.e.*,

$$\begin{aligned} &\min_{\alpha} \ \mathbb{E}_{\alpha \sim \mathbf{P}}[\mathcal{L}_{val}(\mathcal{N}(\alpha, w^*))], \\ &s.t. \ w^* = \arg\min_{w} \mathcal{L}_{train}(\mathcal{N}(\alpha, w)). \end{aligned} \tag{6}$$

where $\alpha \sim \mathbf{P}$ signifies that an architecture α is sampled from the architecture distribution $\mathbf{P} \in \mathbb{R}^{n \times n \times K}$.

With the objective in Eq. (6), the main process of optimizing it is to minimize the expected performance of architectures associated with K probability vectors $\mathbf{P} \in \mathbb{R}^{n \times n \times K}$. Unfortunately, Eq. (6) is difficult to solve directly as higher order derivatives are required. To solve this problem, an alternative optimization strategy is applied to optimize $\mathbf{P}$ and w in two individual datasets iteratively. That is, the network α is first generated from the binary function $f(\cdot)$ and architecture probability $\mathbf{P}$. Afterward, gradients of $\mathbf{P}$ and w calculated on two divided datasets respectively are yielded to modify these parameters better. Because of the differentiability, both architecture probabilities and weights can be optimized end-to-end by the standard back-propagation algorithm. In the end, the network architecture α is identified by $\mathbf{P}$, and the network weights are estimated by retraining on the whole training set. A conceptual visualization of such a process is illustrated in Fig. 1.

4 DATA: Differentiable Architecture Approximation

Although the reformulation presented in Section 3.2 makes the search space continuous, how to define the binary function $f(\cdot)$ as desired to map each probability to a binary code needs to be sorted out. For a coarse $f(\cdot)$, it may aggravate the gap between architectures during searching and validating, such as DARTS [27] and SNAS [30] that strictly limit the binary codes as one-hot vectors. As for a refined $f(\cdot)$, we introduce an Ensemble Gumbel-Softmax (EGS) estimator to optimize the NAS problem with a principled approximation. As such, our model can be directly optimized with the back-propagation algorithm in an end-to-end way, bridge the gap between architectures during searching and validating as much as possible, yielding an efficient and effective searching mechanism.

4.1 Gumbel-Softmax (GS)

A natural formulation for representing discrete variable is to use the categorical distribution. However, partially due to the inability to back-propagate information through samples, it seems rarely applied in deep learning. In this work, we resort to the Gumbel-Max trick [53] for enabling back-propagation and representing the process of taking decision as sampling from a categorical distribution, in order to

perform NAS in a principled way. Specifically, given a probability vector $\mathbf{p} = [p_1, \ldots, p_K]$ and a discrete random variable with $P(L = k) \propto p_k$, we sample from the discrete variable L by introducing the Gumbel random variables. To be more specific, we let

$$L = \arg \max_{k \in \{1, \cdots, K\}} \hat{L}_k, \tag{7}$$

where $\hat{L}_k$ indicates the probability p_k is the maximal entry in $\mathbf{p}$, which is estimated by the Gumbel-Softmax. Formally, the Gumbel-Softmax (GS) estimator can be expressed as:

$$\hat{L}_k = \frac{\exp((\log p_k + G_k)/\tau)}{\sum_{k=1}^{K} \exp((\log p_k + G_k)/\tau)}, \ 1 \leq k \leq K, \tag{8}$$

where τ is a temperature. When $\tau \to 0$, $[\hat{L}_1, \ldots, \hat{L}_K]$ converges to an one-hot vector, and in the other extreme it will become a discrete uniform distribution with $\tau \to +\infty$. $\{G_k\}_{k \leq K}$ is a sequence of the standard Gumbel random variables, and they are typically sampled from the Gumbel distribution $G = -\log(-\log(X))$ with $X \sim U[0, 1]$. Benefiting from the Gumbel random variables, the expectation of selecting path k is equal to probability p_k, as shown by [53]:

$$\mathbb{E}\big(Pr\big(argmax(\hat{L}) = k\big)\big) = p_k. \tag{9}$$

An obstacle to directly applying such an approach is that the argmax operation is not continuous and cannot pass gradient. One straightforward way of dealing with this problem is to replace the argmax operation in Eq. (7) with a softmax [54]. Unfortunately, the distribution neural architecture is intrinsically discrete and forcing it to be continuous may introduce bias between searching and validating. To solve this problem, we estimate the gradient of $\hat{L}_k$ by the gradient of the one-hot vector of L.

From the expression in Eq. (8), we see that GS estimator pertains solely to deal with the problems that only one category requires to be determined, *i.e.*, the outputs are one-hot vectors instead of any binary code. In NAS, however, an optimal architecture may require multiple operations on one edge, considering the practical significance [3], [56]. For instance, the residual module $\mathbf{y} = F(\mathbf{x}) + I(\mathbf{x})$ in ResNets [3] consists of two operations with a learnable mapping $F(\cdot)$ and the identity $I(\cdot)$. That is, choosing different operations in $\mathcal{O}$ may not be mutually exclusive but compatible. One direct way of handling this limitation is to map all possible operation combinations to 2^K-dimensional vectors, where K is the number of candidate operations in $\mathcal{O}$. However, it seems difficult to search architectures efficiently when there are many candidate operations, *i.e.*, K is really large.

4.2 Ensemble Gumbel-Softmax (EGS) for Any Binary Code

To address the aforementioned limitation in the traditional GS estimator, Ensemble Gumbel-Softmax (EGS) estimator is proposed to model the binary function $f(\cdot)$ formulated in Eq. (4), which is capable of choosing diversiform numbers of operations on different edges. To this end, architectures are equally recoded into a group of one-hot vectors that can be sampled from probability vectors with the GS estimator.

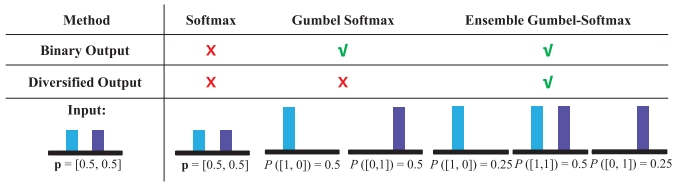


Fig. 2. A visualized comparison between Softmax, Gumbel-Softmax and ensemble Gumbel-Softmax ($M = 2$). For a probability vector $\mathbf{p} = [0.5, 0.5]$, Gumbel-Softmax solely pertains to sample only two binary codes with the same probability, *i.e.*, $P([1, 0]) = P([0, 1]) = 0.5$. In contrast, our ensemble Gumbel-Softmax is capable of sampling more diversified binary codes, *i.e.*, [1,0], [1,1] and [0,1]. Furthermore, the probabilities of sampling these binary codes are logical. Typically, it is conceptually intuitive that the probability of sampling [1,1] is larger than the probabilities of sampling the others since the probabilities in $\mathbf{p} = [0.5, 0.5]$ are equal to each other.

Because of the equivalency, in turn, any architecture is sampled by compositing the results from the GS estimator.

For clarity of exposition, the recoding of $\mathbf{A}^{(i,j)} \in \{0, 1\}^K$ is described only, where $\mathbf{A}^{(i,j)}$ implies the chosen operations on an edge $e^{(i,j)}$. Naturally, such a K-dimensional vector $\mathbf{A}^{(i,j)} \in \{0, 1\}^K$ can be recoded into a superposition of K one-hot vectors, *i.e.*,

$$\mathbf{A}^{(i,j)} = \sum_{k=1}^{K} v_k \cdot \mathbf{a}_k^{(i,j)}, \quad v_k \in \{0, 1\}, \ 1 \leq k \leq K, \tag{10}$$

where $\mathbf{a}_k^{(i,j)} \in \mathbb{R}^K$ is a K-dimensional one-hot vector that uniquely corresponds to the operation $o_k \in \mathcal{O}$, $v_k = 1$ implies that the operation o_k is chosen on edge $e^{(i,j)}$, and $v_k = 0$ otherwise. Benefiting from the equivalence, any architecture code can be represented with a group of one-hot vectors, and one-hot vectors can also be sampled from probability vectors with the GS estimator. Intrinsically, such straightforward process can be considered as the inverse operation of the binary function $f(\cdot)$. That is, the problem of modeling the binary function $f(\cdot)$ can be recast as to find the inversion of such process.

Inspired from the above relationship between architecture codes, one-hot vectors and probability vectors, we model the binary function $f(\cdot)$ by introducing the inversion of this relationship. In Fig. 2, a visualized comparison between the GS and EGS estimators intuitively shows that our EGS estimator is better than the GS estimator, in terms of both sampling capability and rationality. Given a probability vector, the EGS estimator, an ensemble of multiple GS estimators, is profound for sampling any binary code, *i.e.*,

Ensemble Gumbel-Softmax. For a K-dimensional probability vector $\mathbf{p} = [p_1, \ldots, p_K] \in \mathbb{R}^K$ and M one-hot vectors $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(M)}\}$ sampled from $\mathbf{p}$ with the GS estimator, the K-dimensional binary code $\mathbf{b} = [b_1, \ldots, b_K] \in \{0, 1\}^K$ sampled with EGS is

$$b_k = \max_{1 \leq i \leq M} \left(z_k^{(i)}\right), 1 \leq k \leq K,$$

where M is sampling times, b_k is the k-th element in $\mathbf{b}$, and $z_k^{(i)}$ indicates the k-th element in $\mathbf{z}^{(i)}$.

4.3 Minimizing Distribution Gap With Architecture Distribution Constraint (ADC)

Although EGS is in a position to obtain arbitrary binary codes in a differentiable manner, it still suffers from high

variance in the validating process. The basic idea of DATA is sampling a suitable child network from supernet under the hypothesis that the sampled child architecture and supernet have the same representation capability and performance. Formally, to keep the consistent performance of sampled architectures in both searching and validating stage, the original distribution $\mathbf{P}$ of the supernet and the binary distribution $f(\mathbf{P})$ of sampled architectures, are supposed to obey the following approximation:

$$\mathbf{P}^{(i,j)} \approx f\Big(\mathbf{P}^{(i,j)}\Big), \tag{11}$$

where $f\big(\mathbf{P}^{(i,j)}\big)$ is the binary vector sampled by the EGS estimator. Unfortunately, the gap between the original distribution $\mathbf{P}^{(i,j)}$ and the binary decision $f\big(\mathbf{P}^{(i,j)}\big)$ always exists. Since EGS is intrinsically a sampling function that may cause the discrepancy between continuous distribution of supernet and binary distribution of child architectures, the discrepancy will in turn reduce the stability of sampling and inconsistent performance of child architectures. Based on the experiments conducted on the CIFAR-10 dataset, we illustrate the obtained architecture distribution and the derived distribution by $f(\cdot)$ in Fig. 4a and 4b, respectively. As the illustration shows, the approximation Eq. (11) is hardly satisfied, the difference between these two distributions will lead to the unignorable representative gap between obtained architectures and the supernet. Beside, the probability difference between operations is too narrow to identity necessary operation, which may make the operation selection more sensitive to noise. As a consequence, sampled child architectures are of high variety, damaging the stability of sampling process and consistency between supernet and child architectures.

To solve this problem, a constraint on the architecture distribution $\mathcal{G}(\cdot)$ is introduced to help the original distribution $\mathbf{P}$ converge to sampled binary distributions $f\big(\mathbf{P}^{(i,j)}\big)$ in the searching process. Specifically, the regulation in this work is generally formulated as

$$\mathcal{G}(\mathbf{P}) = \sum_{i,j}\Big|\mathbf{P}^{(i,j)} - f\Big(\mathbf{P}^{(i,j)}\Big)\Big|^t, \tag{12}$$

where t is a hyper-parameter in $\mathcal{G}(\cdot)$. With different t, various losses can be derived. In our experiment, the L1 distance loss $t = 1$ is employed. By restricting the learned architecture distribution to approach the natural discrete one, ADC is capable of reducing the discrepancy between these two distributions and the noise of sampling. The objective function in Eq. (1) can therefore be modified as:

$$\begin{aligned} &\min_{\alpha}\ \mathbb{E}_{\alpha\sim\mathbf{P}}[\mathcal{L}_{val}(\mathcal{N}(\alpha, w^*))] + \gamma\mathcal{G}(\mathbf{P}),\\ &s.t.\ w^* = \arg\min_{w}\mathcal{L}_{train}(\mathcal{N}(\alpha, w)), \end{aligned} \tag{13}$$

where γ represents the regulation weight. With the guidance of this regulation, the gap between architecture distribution can be bridge effectively and efficiently.

4.4 Understanding DATA

To reveal the serviceability and sampling capability of the developed EGS estimator, according to the definition *Ensemble Gumbel-Softmax*, three basic propositions are given in the following.

Interpretability. For arbitrary probability vector $\mathbf{p} = [p_1, \ldots, p_K]$ *and sampling times* M*, the* K*-dimensional binary code* $\mathbf{b} \in \{0,1\}^K$ *sampled with EGS always meets*

$$\begin{aligned} P(b_{k_1} = 1) \le P(b_{k_2} = 1) \Leftrightarrow p_{k_1} \le p_{k_2},\\ 1 \le k_1, k_2 \le K, \end{aligned} \tag{14}$$

where $P(b_k = 1)$ is the probability of $b_k = 1$, and $P(b_{k_1} = 1) = P(b_{k_2} = 1) \Leftrightarrow p_{k_1} = p_{k_2}$.

Proposition *Interpretability* means that the binary codes sampled with the EGS estimator strictly depend on the probabilities at the corresponding locations. EGS satisfies Eq. (5) and always tends to be a monotonically increasing function in terms of probability. That is, EGS suffices to act as the binary function $f(\cdot)$.

Expressivity: *For arbitrary probability vector* $\mathbf{p} = [p_1, \ldots, p_K]$ *and number of sampling times* M*, the EGS estimator is capable of sampling* $\binom{K}{M} \times \left(2^M - 1\right)$ *different binary codes, which includes the whole binary codes with up to* M *ones and at least 1 one.*

Proposition *Expressivity* indicates that the sampling capability of the EGS estimator increases exponentially with sampling times M. In practice, larger M is always employed to deal with more complex tasks for effect, and smaller one can be utilized to search more lightweight networks for efficiency. Benefiting from the high expressivity of our search space, DATA suffices to explore the distribution of architectures in a giant search space.

Diversity: *For arbitrary probability distribution vector* $\mathbf{p} = [p_1, \ldots, p_K]$ *and number of sampling times* M*, the probability for coded architecture* $\mathbf{a} = [a_1, \ldots, a_i, \ldots, a_K], a_i \in \{0,1\}$ *can be calculated by:*

$$\begin{aligned} P(\mathbf{a}) = \sum_{t_i}\frac{M!}{t_1!\cdots t_K!}\prod_{j=0}^{K}(a_j p_j)^{t_j},\\ t_1 + t_2 + \cdots + t_K = M, t_i \ge a_i, t_i \in Z. \end{aligned} \tag{15}$$

Proposition *Diversity* demonstrates that the posterior for any architecture in the search space can be calculated according to the learned probability distribution. Different from most existing NAS methods that search for proper architecture for specific task, DATA makes a further step to explore the distribution for neural architecture and learn the intrinsic probability of architectures.

5 Experiments

In this section, we evaluate the performance of our method on several tasks and benchmark datasets. Furthermore, numerous ablation experiments are also conducted to systematically and comprehensively analyze the proposed method. Our search space is totally based on the one proposed by DARTS [27] except for unlimited number of operations for every edge. For convolution cells, every cell consists of $n = 7$ nodes, among which the output node is defined as the depthwise concatenation of all the intermediate nodes. The larger networks are always built by stacking multiple cells together. In the k-th cell, the first and the second nodes are set equally to the outputs in the $(k-2)$-th

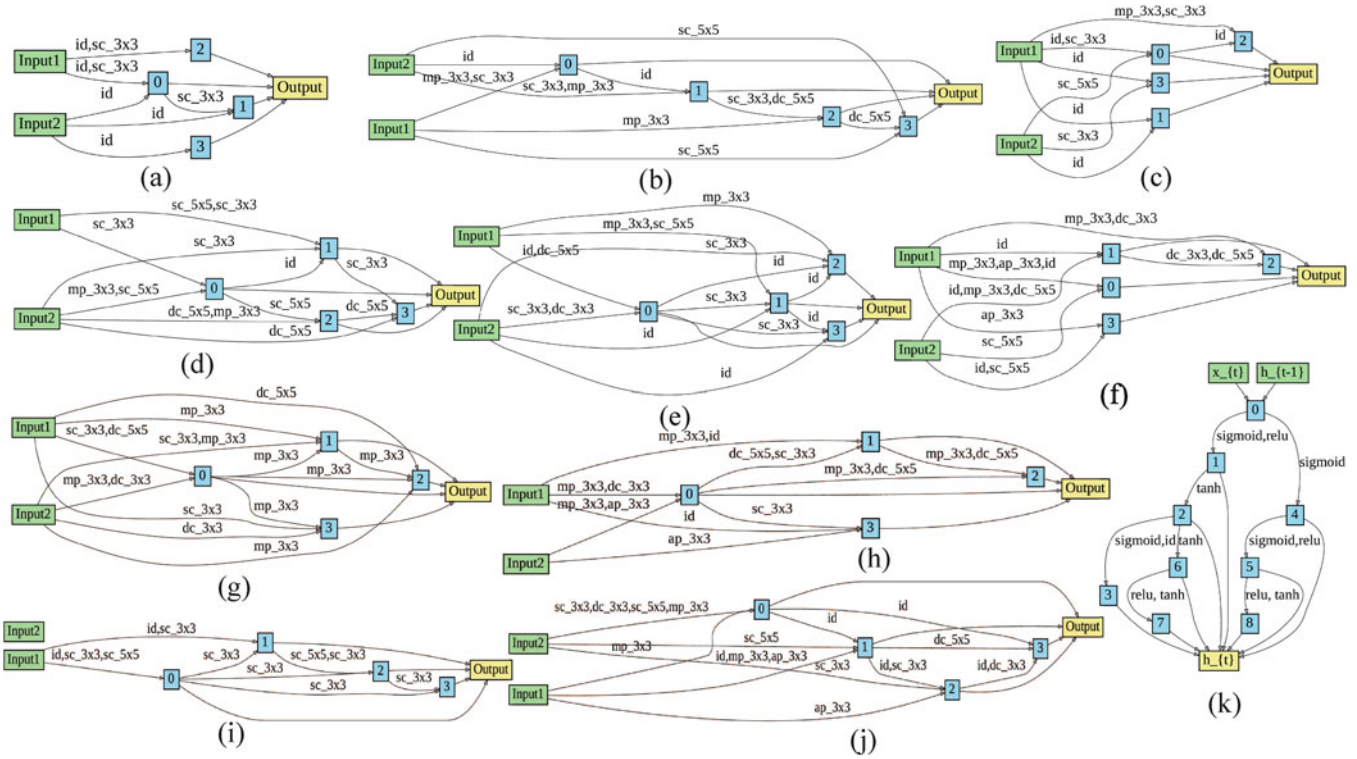


Fig. 3. Architectures learned on different tasks with DATA, where the five operations: separable convolution, dilated convolution, max pooling, average pooling and identity mapping are represented by 'sc', 'dc', 'mp', 'ap' and 'id' in short. Normal cells and reduction cells learned on classification task with (a) (b) M = 4 and (c) (d) M = 7. (e) (f) Normal cells and reduction cells learned on one-shot learning task, (g) (h) five-shot learning task and (i) (j) unsuperwised clustering task. (k) Recurrent cell learned on language modeling task.

and the $(k-1)$-th cells, respectively, with 1×1 convolution as necessary. The candidate primitive set $\mathcal{O}$ for convolutional cell includes eight typical operations, *i.e.*, 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and zero. In order to preserve their spatial resolutions, all operations are of stride one and convolutional feature maps are padded if necessary. The ReLU-Conv-BN order is utilized in the whole convolution operations, and every separable convolution is always applied twice. As for recurrent architecture, there are $n = 12$ nodes in a recurrent cell. Similar to ENAS [22] and DARTS [27], the very first intermediate node is obtained by linearly transforming the two input nodes, adding up the results and then passing through the tanh function, and the rest of activation functions are learned with DATA and enhanced with the highway [57]. The batch normalization [58] is applied in each node to prevent gradient explosion in searching, and disable it during validation. In addition, the recurrent network consists of only a single cell, *i.e.*, any repetitive pattern is not assumed in the recurrent architecture. Available operations include five popular functions: sigmoid, tanh, relu, identity and zero, following the setting in [22], [25], [27]. As a greatly improved work of DARTS, specifically, the experimental settings always inherit from it, if not particularly stated.

5.1 Architecture Learning on Image Classification

A good starting point for neural architecture search is the classical and widely-applied computer vision task, image classification. We apply DATA on the standard image classification dataset, *i.e.*, the CIFAR-10 dataset, with sampling time M set to 4 and 7. A supernet consisting of 8 cells is applied following previous works [16], [25], [27], [34]. The reduction cell where all the operations adjacent to the input nodes are of stride two is utilized at the $1/3$ and the $2/3$ of the total depth of the network, and the remaining cells are normal cells whose operations are of stride one.

To evaluate the searched architecture, we train a larger network of 20 cells from scratch for 600 epochs with batch size 96 and report its test error, following [27]. For fair comparison, we apply cutout with size 16, path dropout of probability 0.2 and auxiliary towers with weight 0.4 following exiting works [16], [25], [27], [34], [61]. We report the mean of five independent runs with different initializations. The obtained normal cells and reduction cells with $M = 4$ and $M = 7$ are illustrated in Fig. 3a, 3b, 3c, 3d, and the FLOPs of the obtained architectures are 443 M and 528 M respectively.

Table 1 gives the classification results of DATA and other NAS methods, which shows that DATA yields comparable or better results comparing with other state-of-the-art

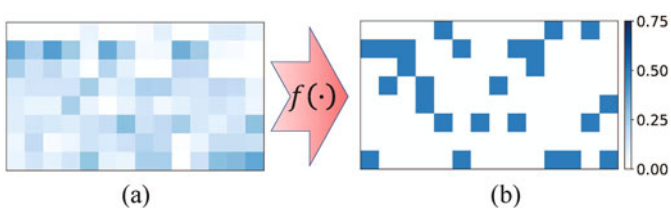


Fig. 4. The sampling process of DATA is actually transforming the learned distribution into binary distribution, inducting the distribution discrepancy between derived architecture and the supernet.

TABLE 1
Comparison of Image Classification Architectures on CIFAR-10

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)
ResNet-110 [3]	6.43	1.70	-
DenseNet-BC [59]	3.46	25.6	-
NAS v1 [24]	5.50	4.2	22400
NAS v2 [24]	6.01	2.5	22400
NAS v3 [24]	4.47	7.1	22400
NASNet-A+cutout [25]	2.65	3.3	2000
NASNet-B [25]	3.73	2.6	2000
PNAS [34]	3.41	3.2	225
Hierarchical evolution [60]	3.75	15.7	300
AmoebaNet-A [16]	3.34	3.2	3150
AmoebaNet-B+cutout [16]	**2.55**	2.8	3150
ENAS+cutout [22]	2.89	4.6	0.5
NAONet [28]	3.18	10.6	200
NAONet-WS [28]	3.53	3.1	0.4
DARTS(1-th order)+cutout [27]	3.00	3.3	0.5
DARTS(2-th order)+cutout [27]	**2.76**	3.3	1
SNAS+mild+cutout [30]	2.98	2.9	1.5
SNAS+moderate+cutout [30]	2.85	2.8	1.5
SNAS+aggressive+cutout [30]	3.10	2.3	1.5
GDAS [26]	2.93	3.4	0.21
DSO-NAS [49]	2.74	3.0	1
Random baseline+cutout	3.29	3.2	-
DATA ($M = 4$)+cutout	2.64	2.8	1
DATA ($M = 7$)+cutout	**2.53**	3.4	1

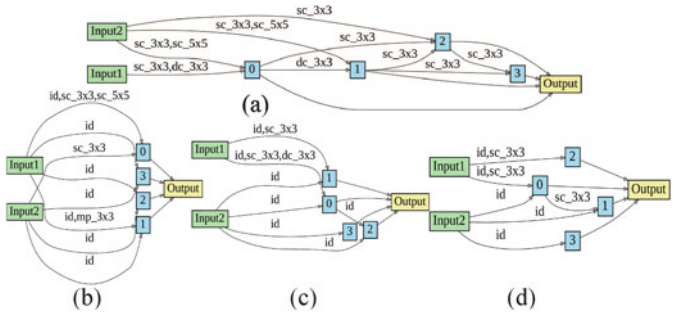


Fig. 5. Evolution of normal cells during searching. The children architectures obtained at the (a) 25-th, (b) 50-th, (c) 75-th, (d) 100-th epoches.

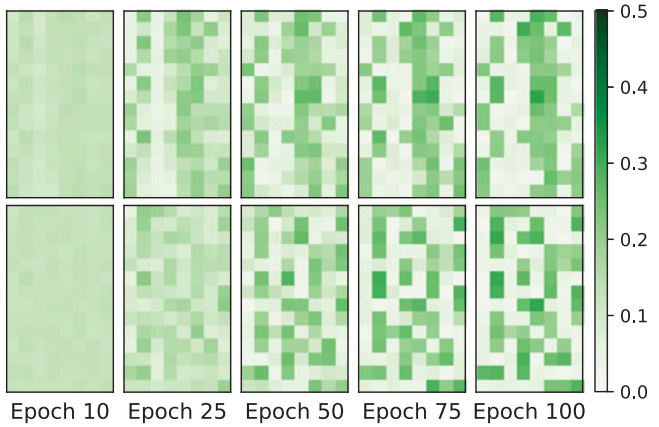


Fig. 6. Obtained architecture distributions in the searching process. The distributions of norm cell and reduction cell are shown in the first row and the second row respectively.

methods with significantly less computation resources. Our experiments verify that DATA can effectively and efficiently search worthy architectures for classification. In DATA, furthermore, higher accuracy is yielded when $M = 7$ compared with $M = 4$. This scenario is in accordance with our motivation that richer search space is beneficial for obtaining better architecture.

5.2 Visualization of Searching Process

To better understand the searching process of DATA, we sample the architecture structures generated in different epochs to visualize the searching process of DATA with $M = 4$. As illustrated in Fig. 5. The searching process of DATA is similar to the process of neural network pruning. DATA starts from highly connected architecture and gradually deletes unless connections, obtaining difficult patterns. Finally, DATA converges for the specific task. Interestingly, the skip connection operation is usually combined with different convolution operations such as separate convolution and dilated convolution, which confirm one of the basic ideas of DATA that the cooperation between different operations contribute to better performance of architectures. The obtained patterns may also help understand architecture and inspire architecture design.

We also visualize the probability for every operation in the searching process, shown in Fig. 6. Our experiments show the probability distribution starts from uniform distribution and gradually indicates the importance of every operation. By taking advantage of ADC, the distribution architecture gradually converges to binary distribution.

Apart from the evolution of architecture distribution, it is also notable that the convergence speed of DATA is comparable or better than other neural architecture methods, shown in Fig. 7a. Compared with ENAS and SNAS, DATA takes fewer epochs to converge to higher validation accuracy even with larger search space, indicating the efficiency of DATA. At the end of searching, the validation accuracy of our method is also higher than other methods. Besides, performance of obtained architectures also improves steadily in the searching process, as illustrated in Fig. 7b.

To prove the validity of DATA to bridge the gap between searching and validating, we compare performance of different gradient-based methods in these two phases, specifically, the accuracy of supernet and child networks on the validation set. As shown in Table 2, although DARTS converges faster than DATA, the child architectures of it suffer from unignorable accuracy drop due to the gap between searching and validating. As for DATA, the performance of child architectures is consistent with the supernet, indicating DATA can bridge this gap effectively. It is also notable that the architecture distribution constraint is capable of improving the performance of both supernet and child network, indicating the effectiveness of it.

5.3 Transferability Validation on ImageNet

To prove the transferability of our method, we transform the architecture learned on CIFAR-10 to a large and standard dataset, ImageNet datasets. On ImageNet, the mobile setting where the input image size is 224×224 is applied and the number of multiply-add operations is restricted to be under 600M. Following [27], we evaluate the obtained

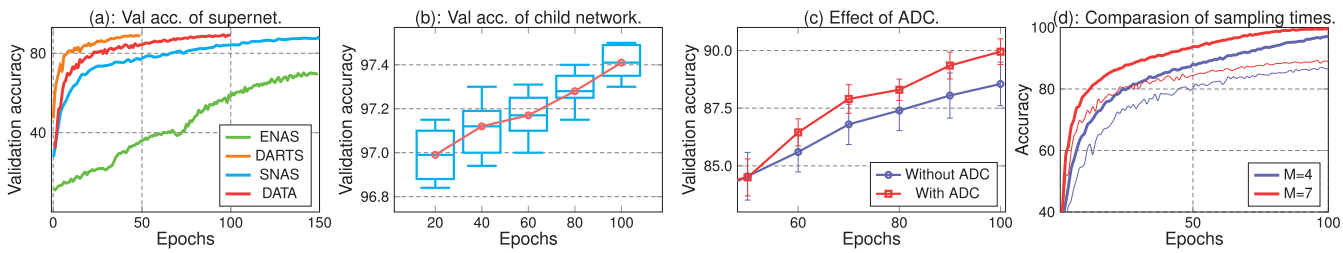


Fig. 7. Searching process of DATA for convolutional architecture on CIFAR-10. (a) The validation accuracy of supernet in the searching process. (b) The visualized comparison of searched architectures between different search epochs. (c) The visualized effect of architecture distribution constrain (ADC) on validation accuracy in the searching process. (d) The visualized effect of sampling times on supernet in during searching, bold curves denote training accuracy and thin curves denote validation accuracy.

structures in an architecture consisting of 14 cells. The architecture is trained for 250 epoches with batch size 128, weight decay 3×10^{-5} and poly learning rate scheduler with initial learning rate 0.1. Other regulations including label smoothing [55] and auxiliary loss [61] are used for fair comparison following [27].

In Table 3, we report the quantitative results on ImageNet. Note that the cell searched on CIFAR-10 can be smoothly employed to deal with the large-scale classification task. Compared with other gradient-based NAS methods, furthermore, greater margins are yielded on ImageNet. A possible reason is that more complex architectures can be searched in DATA because of the larger search space. Consequently, such more complex architectures handle more complex task on ImageNet with better performance. Our experiments also indicate that DATA achieves better performance than hand-engineered architecture ShuffleNet V2 [64]. As one of the state-of-the-art architecture, the designing cost of ShuffleNet V2 requires years of expertise and experience with large amount of hyperparameters tuning. On the contrary, DATA takes only one GPU-day to learn a comparable architecture from scratch, indicating the effectiveness of our method and the power of NAS, or broadly AutoML.

5.4 Transferability Validation on PASCAL VOC

To further investigate the transferability of the convolution cells searched on CIFAR-10, we validate the capability of them on a more complex task, *i.e.*, the semantic segmentation task on PASCAL VOC 2012. In the experiments, we apply the network structure searched on CIFAR-10 as feature extractor and combine it with the head adopted in Deeplab v3 [65]. All architectures are pretrained for 120 epochs on ImageNet and then trained on the semantic segmentation task for 350 epochs using SGD with batch size 64.

TABLE 2
Validation Accuracy Gap on CIFAR-10
(Lower Error Rate is Better)

Architecture	Search Valid. Error	Child net valid. Error	Search Cost (GPU days)
DARTS(1-th) [27]	12.33	45.34	0.5
DARTS(2-th) [27]	11.23	45.21	1
DATA ($M=4$)	11.67	9.43	1
DATA ($M=4$) + ADC	10.94	9.32	1
DATA ($M=7$)	11.08	9.21	1
DATA ($M=7$) + ADC	10.82	9.33	1

Other hyperparameters and data augmentation methods are set following [65] while we multiply the original learning rate by four time to 0.028 as a quadruple batch size is applied.

Compared with other NAS methods, DATA achieves better performance by a large margin while less computation resources. It is also notable that DATA may achieve even better results with larger sampling times. The results in Table 4 demonstrate the transferability of architectures searched on the CIFAR-10 dataset and verify that DATA has more prominent superiority on more complex tasks, not just toy tasks on the tiny datasets, because of a large search space that is proportional to the sampling times M.

5.5 Architecture Learning on Few Shot Learning

In the few shot learning, architecture or specifically, feature extractor, is extremely important as the quality of feature is highly dependent on it, shown by [66]. However, the importance of architectures received little attention and almost all few shot learning methods are equipped with the backbones designed for normal classification task. In this paper, we combine our method with a popular few shot learning method, prototype network [67], to automatic the architecture designing process for few shot learning. All experiments are conducted on a popular few-shot learning benchmark, *mini*ImageNet, with the splits proposed by [68].

The search space of cells for few shot learning task is similar to that for the CIFAR-10 classification task introduced in Section 5.1, merely a network with three normal cells and two reduction cells are applied. As for the architecture structure, considering that the images in *mini*ImageNet is larger then the images in CIFAR-10, we modify the stride-one-convolution in the stem of architecture to a convolution with stride 2 and applied a max pooling to the final feature map of network, followed by a convolution layer with kernel 1×1 to adjust the number of channels to 64, resulting in a 1600-dimensions output following [67]. Meanwhile, we follow their procedure by searching or training network on the 64 training classes and using the 16 validation classes for monitoring generalization performance only, while reporting performance on the independent 16 test classes.

In the searching process, we split the training set into two parts according to classes, 32 classes for training weights and 32 classes for architecture, sampling times is set to $M=7$. As for architecture search, we combine our method with 1-shot learning and 5-shot learning tasks and apply 10-way episodes. We match train shot to test shot and the number of query points per episode for every class is set to 15. The

TABLE 3
Comparison With Classifiers on ImageNet in the Mobile Setting (Lower Test Error is Better)

Architecture	Test Error (%)		Params	FLOPs	Search Cost	Number	Search
	Top 1	Top 5	(M)	(M)	(GPU days)	of Ops	Method
Inception-v1 [63]	30.2	10.1	6.6	1448	-	-	manual
MobileNet [64]	29.4	10.5	4.2	569	-	-	manual
ShuffleNet-v2 2× [65]	25.1	-	∼5	591	-	-	manual
PNAS [34]	25.8	8.1	5.1	588	∼225	8	SMBO
AmoebaNet-A [16]	25.5	8.0	5.1	555	3150	19	evolution
AmoebaNet-B [16]	26.0	8.5	5.3	555	3150	19	evolution
AmoebaNet-C [16]	**24.3**	**7.6**	6.4	570	3150	19	evolution
NASNet-A [25]	26.0	8.4	5.3	564	2000	13	RL
NASNet-B [25]	27.2	8.7	5.3	488	2000	13	RL
NASNet-C [25]	27.5	9.0	4.9	558	2000	13	RL
Mnas-Net-92 [39]	25.2	8.0	4.4	388	-	6	RL
DARTS [27]	26.7	8.7	4.7	574	1	7	gradient-based
SNAS (mild constraint) [30]	27.3	9.2	4.3	522	1.5	7	gradient-based
ProxylessNAS(GPU) [50]	24.9	7.5	7.1	388	8.3	6	gradient-based
GDAS [26]	26.0	8.5	5.3	581	0.21	7	gradient-based
DSO-NAS-share [49]	25.4	8.3	4.7	567	6	4	gradient-based
DATA ($M = 4$)	25.6	8.5	4.2	521	1	7	gradient-based
DATA ($M = 7$)	**24.9**	**8.0**	5.0	584	1	7	gradient-based

hypernetwork is trained for 300 epochs, we set initial learning rate to 0.1 and multiply it by 0.2 for every 100 episodes. The architectures obtained by searching on the one-shot learning and five-shot learning task are represented by DATA-OS and DATA-FS respectively, the normal cells and reduction cells of them are shown in Fig. 3e, 3f, 3g, 3h.

As for architecture evaluating, the same network utilized in the searching process is trained from scratch for 1-shot classification and 5-shot classification task. Following [67], the architecture is trained with 30-way episodes for 1-shot classification and 20-way episodes for 5-shot classification. We match train shot to test shot and each class contains 15 query points per episode. We train the architecture for 300 epochs, learning rate is set to 0.1 and is multiplied by 0.2 every 100 episodes.

Other hyperparameters and data augmentation methods including horizontal flip, random crop are set following [67].

The results of our experiments are shown in Table 5, where DATA-CLS represents the architecture obtained by searching on the classical classification task (M=7). From the table, NAS improves the performance of few-shot learning by a large margin while our method achieves comparable or even better performance than other NAS or few-shot methods. Besides, the architectures searched on the few-shot learning task achieve better performance than the architecture transformed from classical classification task, indicating DATA is capable of searching suitable architecture based on specific tasks.

TABLE 4
Semantic Segmentation on the PASCAL VOC-2012

Architecture	mIOU (%)	Params (M)	Search Cost (GPU days)
NASNet-A [25]	74.7	12.6	2000
AmoebaNet-A [16]	75.2	12.2	3150
DARTS [27]	73.8	12.4	1
DATA ($M = 4$)	74.1	**11.7**	1
DATA ($M = 7$)	**75.6**	12.7	1

5.6 Architecture Learning on Unsupervised Clustering

As a totally data-driven task that attempts to explore knowledge from unlabeled data, clustering is an essential data analysis tool in pattern analysis and machine learning. A heavy investment direction of clustering is developing unsupervised features extracting techniques as the quality of feature is extremely important for improving the performance of clustering. In this paper, we incorporate our method with unsupervised cluster task to search for the suitable architectures for it. Experiments are conducted on the CIFAR-10 dataset.

The search space we applied for unsupervised cluster task is similar to that for image classification task, expect that a shallow network with only three normal cells and two

TABLE 5
Comparison of Few Shot Learning Algorithms on *mini*ImageNet (Architectures With * are Obtained on CIFAR-10 Classification Task)

Method	*mini*ImageNet 5-way Acc.		Search Cost
	1-shot	5-shot	(GPU days)
ML LSTM [70]	43.44 ± 0.77	60.60 ± 0.71	-
MatchingNet [69]	43.56 ± 0.84	55.31 ± 0.73	-
ProtoNet [68]	49.42 ± 0.78	68.20 ± 0.66	-
RelationNet [71]	50.44 ± 0.82	65.32 ± 0.70	-
MAML [72]	48.70 ± 1.84	63.11 ± 0.92	-
NASNet-A* [25]	52.94 ± 0.96	70.86 ± 0.84	2000
AmoebaNet-A* [16]	53.86 ± 0.93	71.42 ± 0.78	3150
DARTS* [27]	52.02 ± 0.93	70.41 ± 0.91	1
DATA-CLS*	52.86 ± 0.87	70.53 ± 0.89	1
DATA-OS	$\mathbf{54.77 \pm 0.89}$	70.88 ± 0.74	1
DATA-FS	53.82 ± 0.82	$\mathbf{72.17 \pm 0.88}$	1

TABLE 6
Comparison of Unsupervised Clustering Methods on CIFAR-10 (Architectures With * are Obtained on CIFAR-10 Classification Task)

Method	Clustering Results.			Search Cost
	NMI	ARI	ACC	(GPU days)
K-means [25]	0.0871	0.0487	0.2289	-
DAE [74]	0.2506	0.1627	0.2971	-
DEC [75]	0.2568	0.1607	0.3010	-
CatGAN [76]	0.2646	0.1757	0.3152	-
DAC [73]	0.4379	0.3399	0.4778	-
NASNet-A* [25]	0.4656	0.3642	0.4901	2000
AmoebaNet-A* [16]	0.4732	0.3794	0.5134	3150
DARTS* [27]	0.4563	0.3611	0.4962	1
DATA-CLS*	0.4703	0.3788	0.5122	1
DATA-CLU	**0.4763**	**0.3811**	**0.5162**	**0.5**

reduction cells is applied. Similar to the classification experiments, we split the training set into two parts in the searching process, half for training weights and half for architecture. As for architecture search, we combine our method with a state-of-the-art unsupervised learning method DAC [72]. The hypernetwork is trained for 50 epochs, we used an initial learning rate of 0.1 and multiply it by 0.1 for every 10 epochs. In the evaluation stage, the obtained architectures are combined with DAC and trained from scratch in the unsupervised clustering task. Following [72], all architectures are trained with learning rate 0.1 for 100 epochs for fair comparison. Other hyperparameters are also set following [72]. The normal cell and reduction cell of the obtained architecture are shown in Fig. 3j and 3k respectively.

The results are shown in Table 6, where DATA-CLS signifies the architecture obtained by searching on the classification task (M=7) and DATA-CLU represents the architecture obtained by searching on the clustering task. Our experiments demonstrate that the architectures obtained by DATA achieve better results on the unsupervised clustering task than other NAS methods with less search cost. It is also notable that the architectures obtained by our method have high transferability while the architecture searched on specific task achieves even better performance.

5.7 Architecture Learning on Language Modeling

Another important application of deep learning is recurrent network. Instead of extracting feature based on spatial neighborhood like convolution network, recurrent network model data according to time series. We conduct our method on the language modeling task to explore architecture search for the recurrent network.

Following [22], [27], we adopt a RNN search space consisting of 12 nodes and conduct searching process on the PTB dataset. Sampling time $M = 2$ and $M = 3$ is applied to search suitable topology structure of recurrent network as well as the activation functions between nodes.

The searching process is conducted on the PTB dataset. A single-layer recurrent network consisting of searched cells is trained with 1600 epochs, and batch size 64 using averaged SGD. Both of the embedding and the hidden sizes are set to 850 to ensure our model size is comparable with other baselines. Other hyper-parameters are set following [27].

TABLE 7
Comparison With State-of-the-Art Language Models on PTB

Architecture	Perplexity		Params	Search Cost
	valid	test	(M)	(GPU days)
Variational RHN [57]	67.9	65.4	23	-
LSTM [77]	60.7	58.8	24	-
LSTM+SC [78]	60.9	58.3	24	-
LSTM+SE [79]	58.1	56.0	22	-
DARTS (1-th order) [27]	60.2	57.6	23	0.5
DARTS (2-th order) [27]	58.1	55.7	23	1
ENAS [22]	68.3	63.1	24	0.5
GDAS [26]	59.8	57.5	23	0.4
DATA ($M = 2$)	58.4	56.3	22	0.5
DATA ($M = 3$)	**57.2**	**55.2**	23	0.5

We illustrate the obtained recurrent architecture in Fig. 3k.

Table 7 lists the results of this experiment. From the table, DATA is in a position to search recurrent architectures effectively. It demonstrates empirically that the back-propagation algorithm can guide DATA to hit a preferable recurrent architecture, while maintaining the required efficiency. Similar to the conclusion in the experiments on CIFAR-10, lower perplexity is achieved when a larger M is used, which verifies that a large search space is also valuable for searching recurrent architectures.

5.8 Transferability Validation on WT2

To demonstrate the generalizability of our method, we transfer the block structure learned on PTB to the WT2 dataset. Different from the settings on PTB, we apply embedding hidden sizes 700, weight decay 5×10^{-7}, and hidden-node variational dropout 0.15 following [27]. Other hyperparameters remain the same in the experiments on PTB. In Table 8, the results on the WT2 dataset indicate that the transferability is also retentive on recurrent architectures. Conclusively, the consistent results of the above experiments on ImageNet, PASCAL VOC and WT2 strongly demonstrate the transferability on both convolutional and recurrent architectures.

5.9 Ablation Study

In this section, extensive ablation studies are conducted to analyze the DATA algorithm synthetically. Specifically, the

TABLE 8
Comparison With State-of-the-Art Language Models on WT2

Architecture	Perplexity		Params	Search Cost
	valid	test	(M)	(GPU days)
LSTM+AL [80]	91.5	87.0	28	-
LSTM+CP [81]	-	68.9	-	-
LSTM [77]	69.1	66.0	33	-
LSTM+SC [78]	69.1	65.9	24	-
LSTM+SE [79]	66.0	63.3	33	-
DARTS (2-th order) [27]	69.5	66.9	33	1
ENAS [22]	72.4	70.4	33	0.5
GDAS [26]	71.0	69.4	33	0.4
DATA ($M = 2$)	67.4	64.5	33	0.5
DATA ($M = 3$)	**66.7**	**64.4**	33	0.5

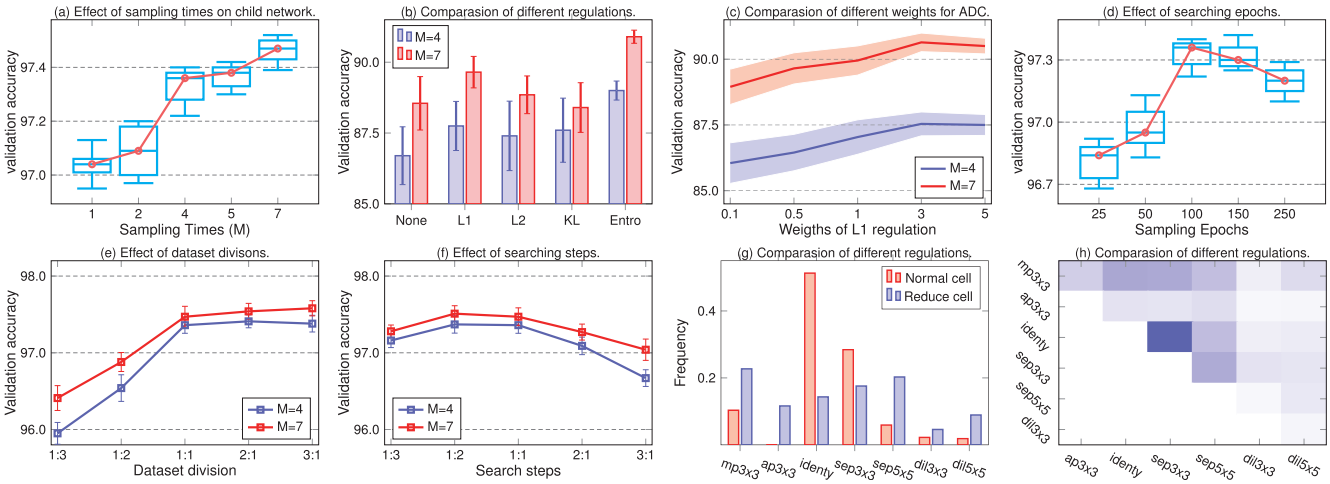


Fig. 8. (a) Performance of child architectures obtained with different sampling times. (b) Effect of different regulations on mean accuracy and sampling variance of supernet. (c) Validation accuracy and sampling variance of supernet under different L1 regulation weights during searching. (d) Performance of child architectures under different number of searching epochs. (e) Performance of child networks obtained with different data divisions, where $m:n$ represents the ratio of data division for optimizing weights and architecture probabilities. (f) Effect of different searching steps for network weights and probabilities parameters, where $m:n$ represents optimizing network weights for m steps followed by n steps for architecture probabilities for every $m+n$ steps. (g) Frequency of the operations selected by the obtained architectures. (h) Frequency of the incorporation of every two operations applied by the obtained architectures.

experiments are performed on the CIFAR-10 dataset and all hyperparameters inherit directly from the statements in Section 5.1, if not stated particularly.

5.9.1 Sensitivity to Number of Sampling Times

In order to empirically analyze the sensitivities of the number of sampling times M. We conduct searching process with different M. The performance of obtained architectures is visualized in Fig. 8a. Our experiments show that larger M indicates higher and more stable performance. This is in accordance with the statement in Proposition 2, i.e., more capable networks might be found with larger M.

For fair comparison, two experiments are conducted. First, we apply DARTS in an enlarged search space where more operations are allowed in a single path. Besides, we conduct DATA in the same search space of DARTS and fix the number of operations to two. The results are shown in Table 9, where selecting number N indicates selecting the top-N operations with the highest architecture parameters for every path. From the table, DARTS is not capable of managing combinations of different operations due to the estimation gap introduced before, it tends to select superfluous while inefficient operations that may dramatically introduce more parameters. On the contrary, DATA achieves better results with the same search space, indicating the effectiveness of DATA. Furthermore, better results are demonstrated by DATA in a richer search space, which verifying that richer search space is beneficial for obtaining better architectures.

5.9.2 Effectiveness of ADC

To evaluate the effectiveness of distribution constraint introduced in Section 4.3, we compare the effect of different distribution constraints and visualize the corresponding architecture distributions. Four different constraint functions are compared in our experiments: L1 loss function Eq. (16), L2 loss function Eq. (17), KL-divergence loss function Eq. (18) and Entropy loss function Eq. (19):

$$\mathcal{G}_{L1}(\mathbf{P}) = \sum_{i,j} \left| \mathbf{P}^{(i,j)} - f\left(\mathbf{P}^{(i,j)}\right) \right| \tag{16}$$

$$\mathcal{G}_{L2}(\mathbf{P}) = \sum_{i,j} \left(\mathbf{P}^{(i,j)} - f\left(\mathbf{P}^{(i,j)}\right) \right)^2 \tag{17}$$

$$\mathcal{G}_{KL}(\mathbf{P}) = \sum_{i,j} \mathbf{P}^{(i,j)} \log \left(\frac{\mathbf{P}^{(i,j)}}{f\left(\mathbf{P}^{(i,j)}\right)} \right) \tag{18}$$

$$\mathcal{G}_{Entropy}(\mathbf{P}) = \sum_{i,j} -\mathbf{P}^{(i,j)} \log \left(\mathbf{P}^{(i,j)} \right). \tag{19}$$

First, we visualize their influence on the supernet during searching in Fig. 8b. From the illustration, specific regulations like L1 loss and Entropy loss are conducive to achieve better performance where L2 loss and KL loss affect negatively. Our experiments demonstrate the L1 loss and Entropy loss are capable of reducing the

TABLE 9
Sensitivity to the Number of Operations for DARTS [27] on CIFAR-10

Architecture	Selecting Numbers(N)	Test Error	params (M)	Search Cost (GPU days)
DARTS (1th)	1	3.00	3.30	0.5
DARTS (1th)	2	3.05	4.00	0.5
DARTS (1th)	3	2.96	5.20	0.5
DARTS (2th)	1	2.76	3.30	1
DARTS (2th)	2	2.75	4.12	1
DARTS (2th)	3	2.77	5.40	1
DATA ($M=4$)	2	2.72	3.24	1
DATA ($M=7$)	2	2.62	3.52	1
DATA ($M=4$)	-	2.64	**2.84**	1
DATA ($M=7$)	-	**2.53**	3.48	1

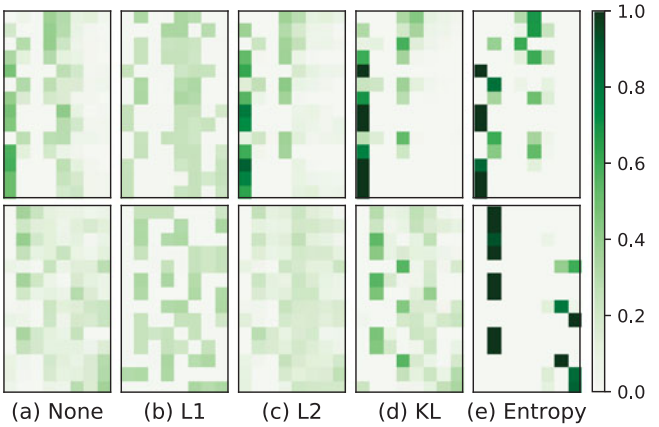


Fig. 9. Architecture distributions obtained under the effects of different kinds of regulations on the CIFAR-10 dataset. We visualize the distributions of norm cells in the first row while the reduction cells in the second row. (a) No regulation. (b) L1 loss function. (c) L2 loss function. (d) KL-divergence loss function. (e) Entropy loss function.

sampling variance from 1.9 to 1.1 and 0.4 respectively, improving the steadily of searching process. Besides, we also conduct experiments to explore the influence of different weights of regulation. As shown by Fig. 8c, a suitable regulation weight is capable of reducing the variance while a too large weight will affect the performance of the supernet.

To better understand how distribution constraint affect, we visualize the obtained distributions under different distribution constraint, shown in Fig. 9. Obviously, distribution constraints like L1 loss and Entropy loss guide distribution of architecture converge to discrete form, improving the stability and robustness of architecture sampling. Beside, the difference between the probability of operations is more distinguishable, reducing the influence of noise when selecting operations.

Specially, distribution of architecture tends to be one-hot under the guidance of Entropy loss, which is similar to the effect of annealing Softmax. Other constraints like L2 loss and KL loss reduce the individuality of different architecture probabilities and induce higher variance.

5.9.3 Sensitivity to Number of Searching Epochs

We conduct experiments to evaluate the sensitivity of our method to the number of epochs in the searching process. On the CIFAR-10 dataset, we vary searching epochs from 25 to 250 and adjust corresponding learning rate schedulers. The results are reported in Fig. 8d. From the illustration, suitable number of searching epochs is essential for searching process. Inadequate searching epochs may cause underfitting of architecture distribution while excessive searching epochs may leading to over-fitting and damage the generalize ability of child architectures.

5.9.4 Sensitivity to Dataset Divisions

To observe the effect of different divisions of dataset to the neural architecture search method, we vary the ratio of training data for weights and architecture probabilities under different sampling times.

TABLE 10
Sensitivity to Initialization on CIFAR-10

Acc of pretrained model	Sampling Times(M)	Params (M)	Test Error(%)	Search Cost (GPU days)
None	4	2.84	2.64	1.0
	7	3.37	2.53	1.0
76.05	4	3.25	2.57	1.5
	7	3.86	2.51	1.5
83.21	4	3.86	2.53	2.0
	7	5.12	2.48	2.0
90.45	4	4.12	2.50	2.5
	7	5.32	2.43	2.5

Specifically, the number of total steps for optimizing architecture distribution is remained to prevent the effect of searching steps.

The results are illustrated in Fig. 8e. As weights outnumber the architecture probabilities by a large margin, optimizing weights with more data helps reinforce the generalization ability of supernet, improving the quality of child networks.

5.9.5 Sensitivity to Optimizing Steps for Weights and Architecture Probabilities

To explore the influence of the optimizing steps for weights and architecture probabilities in the neural architecture search method, we perform experiments with different optimizing steps for weights and architecture probabilities and compare their effects.

The numbers of optimizing steps for weights and architecture probabilities are kept unchanged during searching. The results shown in Fig. 8f imply that the balance of optimizing speed of weights and architecture probabilities is essential for obtaining high-performance architectures, just as shown by Eq. (1). Updating weights or architecture probabilities too rapidly may induce bad local optimum, affecting the searching process.

5.9.6 Effectiveness of Pretrained Model

Shown by Eq. (1), better initialization of w that is closer to ideal w^* is essential for the nested optimization problem. To study the effect of initialization of w, the supernet is first pretrained on the dataset for weights for different epochs to obtain initializations with different accuracies. Then, we load the weights of the pretrained supernet and conduct the original searching process. In the pretrain stage, we sample operations for different edges randomly to ensure the robustness of weights. The results shown in Table 10 indicate that DATA obtains architectures with better performance while larger in size. The reason of this phenomenon may be that better initialization of weights makes the combination between different operations stronger and DATA obtains more architecture patterns with good performance. Further analysis, our experiments demonstrate that the pretraining technique can be applied to make trade-off between performance and efficiency.

TABLE 11
Hardware Performance of DATA

Architecture	Params (M)	FLOPs (M)	Memory (G)	Latency (ms)
DARTS [27]	4.7	574	6.3	50.1
NASNet-A [25]	5.3	564	6.5	56.0
AmoebaNet-A [16]	5.1	555	5.7	44.7
DGAS [26]	5.3	581	6.1	48.5
SNAS [30]	4.8	522	4.9	75.5
DATA($M = 4$)	4.3	521	5.1	40.3
DATA($M = 7$)	5.0	584	6.2	53.7

5.9.7 Statistics of Obtained Architecture Patterns

As NAS has achieved impressed performance in many fields, the obtained components of architectures may also provide meaning insight for manual network designing. To analysis the relationship between operations, we investigate the popularity of operations and the incorporation between different operations. Based on the architectures obtained by DATA, a statistics on the frequency of single operation and the incorporation between every two operations is carried out. The results are shown in Fig. 8g and 8h, respectively. Our experiments highlight that operations like identity mapping and separable convolution 3×3 are helpful for feature extracting as they are popular for normal cells. On the other hand, reduction cells prefer operations with large receptive filed, indicating these operations are essential for feature reducing. As for the incorporation of operations, the combination of identity mapping and separable convolution with kernel 3×3 is wildly applied, implying this kind of cooperation may be helpful for feature extraction and gradient backpropagation.

5.9.8 Hardware Performance of DATA

To investigate the hardware performance of DATA, we evaluate the obtained architectures based on two hardware metric, *i.e.*, GPU memory cost and latency. In the experiments, all architectures are evaluated with a network with 14 cells, batch size is set to 32 and input size 224×224 is applied. Our experiments are conducted on one TITAN Xp GPU. The results shown in Table 11 indicate that performance of DATA on the memory cost and latency are comparable or better than other NAS methods, although a larger and more complicated search space is applied. As shown by Fig. 3, DATA tends to select efficient operations like identity-mapping. Besides, although we break the limitation on the number of operations for node, many nodes keep less than two operations, which is also beneficial for reducing the hardware complexity of architectures.

6 Conclusions and Future Work

We present DATA to bridge the gap of architectures during searching and validating in a differentiable manner. For this purpose, the EGS estimator that consists of an ensemble of a group of Gumbel-Softmax estimators is developed, which is in a position to sample an architecture that approaches to the one during searching as close as possible, while guaranteeing the required efficiency. Besides, a general regulation for architecture distribution ADC is introduced to further reduce the discrepancy by pushing the continuous architecture distribution to discrete one. By searching with the standard back-propagation, DATA is able to outperform the state-of-the-art architecture search methods on various tasks, with remarkably better efficiency.

Future work may include searching architectures of the whole networks with the proposed method and injecting the EGS estimator into deep models to handle other machine learning tasks. Besides, how to incorporate EGS with hardware budgets such as inference speed, FLOPs also desire further study. For the first work, the sampling capability of the EGS estimator guarantees the practicability of searching any networks, but how to further improve search efficiency and design specific ADC for entire architecture remain to be investigated. For the second work, the differentiability of the EGS estimator indicates that it can be utilized anywhere in networks, *e.g.*, an interesting direction is to recast the clustering process into our ensemble Gumbel-Softmax. By aggregating inputs in each cluster, conclusively, a general pooling for both deep networks and deep graph networks [81], [82] can be developed to deal with euclidean and non-euclidean structured data uniformly. As for the last work, hardware metrics may be taken into account in the evaluation of child networks and help the EGS estimator generate efficient architectures.

Acknowledgments

This work was supported by the Major Project for New Generation of AI under Grant No. 2018AAA0100400, the National Natural Science Foundation of China under Grants 91646207 and 61976208. The work of Z. Lin was supported by National Key R&D Program of China (2019AAA0105200), NSF China (grant no.s 61625301 and 61731018), Major Scientific Research Project of Zhejiang Lab (grant no.s 2019KB0AC01 and 2019KB0AB02), Beijing Academy of Artificial Intelligence, and Qualcomm. The authors would like to thank Lele Yu, Jie Gu, Cheng Da, Yukang Chen, and Jiemin Fang for their invaluable contributions in shaping the early stage of this work.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
[2] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[4] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jul. 2017.
[5] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015.
[6] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017.
[7] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected *CRFs*," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2018.
[8] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[9] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, pp. 55:1–55:21, 2019.

[10] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.

[11] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *Proc. Int. Conf. Learn. Representations*, 2019.

[12] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," *CoRR*, vol. abs/1904.00420, 2019.

[13] P. Kamath, A. Singh, and D. Dutta, "Neural architecture construction using envelopenets," *CoRR*, vol. abs/1803.06744, 2018.

[14] R. Miikkulainen *et al.*, "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, 2017.

[15] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[16] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI*, 2018, pp. 4780–4789.

[17] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *Proc. Int. Conf. Learn. Representations*, 2020.

[18] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Mach. Intell.*, vol. 1, no. 1, pp. 24–35, 2019.

[19] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *Proc. Int. Conf. Learn. Representations Workshop Track*, 2017.

[20] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 459–468.

[21] C. Hsu *et al.*, "MONAS: multi-objective neural architecture search using reinforcement learning," *CoRR*, vol. abs/1806.10332, 2018.

[22] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4092–4101.

[23] Z. Zhong, J. Yan, W. Wu, J. Shao, and C. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2423–2432.

[24] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2017.

[25] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.

[26] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 1761–1770.

[27] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019.

[28] R. Luo, F. Tian, T. Qin, E. Chen, and T. Liu, "Neural architecture optimization," in *Proc. Conf. Neural Inf. Process. Syst.*, 2018, pp. 7827–7838.

[29] R. Shin, C. Packer, and D. Song, "Differentiable neural network architecture search," in *Proc. Int. Conf. Learn. Representations Workshop Track*, 2018.

[30] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019.

[31] F. P. Casale, J. Gordon, and N. Fusi, "Probabilistic neural architecture search," *CoRR*, vol. abs/1902.05116, 2019.

[32] T. Elsken, J. H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," in *Proc. Int. Conf. Learn. Representations Workshop Track*, 2018.

[33] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. UAI*, 2019, Art. no. 129.

[34] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 19–35.

[35] L. Chen *et al.*, "Searching for efficient multi-scale architectures for dense image prediction," in *Proc. Conf. Neural Inf. Process. Syst.*, 2018, pp. 8713–8724.

[36] C. Liu *et al.*, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 82–92.

[37] Y. Chen, T. Yang, X. Zhang, G. Meng, C. Pan, and J. Sun, "DetNAS: Neural architecture search on object detection," in *Proc. Conf. Neural Inf. Process. Syst.*, 2019, pp. 6638–6648.

[38] G. Ghiasi, T. Lin, R. Pang, and Q. V. Le, "NAS-FPN: learning scalable feature pyramid architecture for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 7036–7045.

[39] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2820–2828.

[40] D. Tran, J. Ray, Z. Shou, S. Chang, and M. Paluri, "ConvNet architecture search for spatiotemporal feature learning," *CoRR*, vol. abs/1708.05038, 2017.

[41] J. Chang, X. Zhang, Y. Guo, G. Meng, S. Xiang, and C. Pan, "DATA: differentiable architecture approximation," in *Proc. Conf. Neural Inf. Process. Syst.*, 2019, pp. 874–884.

[42] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," in *Proc. Int. Conf. Learn. Representations Workshop Track*, 2018.

[43] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: one-shot model architecture search through hypernetworks," in *Proc. Int. Conf. Learn. Representations*, 2018.

[44] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. 27th Int. Joint Conf. Artif. Intell. Best Sister Conf.*, 2018, pp. 5369–5373.

[45] A. Veit and S. J. Belongie, "Convolutional networks with adaptive inference graphs," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 3–18.

[46] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, Jan. 1994.

[47] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, 2008.

[48] R. Józefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, 2015, pp. 2342–2350.

[49] X. Zhang, Z. Huang, and N. Wang, "You only search once: Single shot neural architecture search via direct sparse optimization," *CoRR*, vol. abs/1811.01567, 2018.

[50] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Representations*, 2019.

[51] M. Courbariaux, Y. Bengio, and J. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Conf. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[52] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. Int. Conf. Learn. Representations*, 2017.

[53] E. J. Gumbel, *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*. US Govt. Print. Office, 1954, vol. 33.

[54] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. Int. Conf. Learn. Representations*, 2017.

[55] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[56] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, "Recurrent highway networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 4189–4198.

[57] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[58] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.

[59] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Int. Conf. Learn. Representations*, 2018.

[60] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1294–1303.

[61] C. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Proc. 18th Int. Conf. Artif. Itell. Statist.*, 2015, pp. 562–570.

[62] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

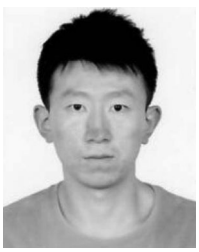
[63] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[64] N. Ma, X. Zhang, H. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 122–138.

[65] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *CoRR*, vol. abs/1706.05587, 2017.

[66] W. Chen, Y. Liu, Z. Kira, Y. F. Wang, and J. Huang, "A closer look at few-shot classification," in *Proc. Int. Conf. Learn. Representations*, 2019.
[67] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," in *Proc. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4077–4087.
[68] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proc. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3630–3638.
[69] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Representations*, 2017.
[70] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1199–1208.
[71] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.
[72] J. Chang, L. Wang, G. Meng, S. Xiang, and C. Pan, "Deep adaptive image clustering," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 5880–5888.
[73] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, 2010.
[74] J. Xie, R. B. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 478–487.
[75] X. Yi, E. Walia, and P. Babyn, "Unsupervised and semi-supervised learning with categorical generative adversarial networks assisted by wasserstein distance for dermoscopy image classification," *CoRR*, vol. abs/1804.03700, 2018.
[76] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," in *Proc. Int. Conf. Learn. Representations*, 2018.
[77] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," in *Proc. Int. Conf. Learn. Representations*, 2018.
[78] Z. Yang, Z. Dai, R. Salakhutdinov, and W. W. Cohen, "Breaking the softmax bottleneck: A high-rank RNN language model," in *Proc. Int. Conf. Learn. Representations*, 2018.
[79] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," in *Proc. Int. Conf. Learn. Representations*, 2017.
[80] E. Grave, A. Joulin, and N. Usunier, "Improving neural language models with a continuous cache," in *Proc. Int. Conf. Learn. Representations*, 2017.
[81] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018.
[82] J. Chang, J. Gu, L. Wang, G. Meng, S. Xiang, and C. Pan, "Structure-aware convolutional neural networks," in *Proc. Conf. Neural Inf. Process. Syst.*, 2018, pp. 11–20.

Xinbang Zhang received the BS degree in automation from the University of Northeastern University, Shenyang, China, in 2017. He is currently working toward the PhD degree from the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China. His research interests include network pruning and Auto ML.



Jianlong Chang received the BS degree from the School of Mathematical Sciences, University of Electronic Science and Technology of China, Chengdu, China, in 2015, and the PhD degree from the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2020. He is currently a research scientist with Huawei Cloud & AI. His current research interests include relation-based DL, auto ML, graph networks, and unsupervised learning.

Yiwen Guo received the BE degree from Wuhan University, Wuhan, China, in 2011, and the PhD degree in electrical engineering from Tsinghua University, Beijing, China, in 2016. He is currently a research scientist with Bytedance AI Lab. His current research interests include computer vision, pattern recognition, and machine learning.

Gaofeng Meng (Senior Member, IEEE) received the BS degree in applied mathematics from Northwestern Polytechnical University, in 2002, the MS degree in applied mathematics from Tianjing University, in 2005, and the PhD degree in control science and engineering from Xi'an Jiaotong University, in 2009. He is currently an associate professor at the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences. He also serves as an associate editor of *Neurocomputing*. His current research interests include document image processing, computer vision, and pattern recognition.

Shiming Xiang received the BS degree in mathematics from Chongqing Normal University, Chongqing, China, in 1993, the MS degree from Chongqing University, Chongqing, China, in 1996, and the PhD degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2004. From 1996 to 2001, he was a lecturer with the Huazhong University of Science and Technology, Wuhan, China. He was a postdoctorate candidate with the Department of Automation, Tsinghua University, Beijing, China, until 2006. He is currently a professor at the National Lab of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China. His research interests include pattern recognition, machine learning, and computer vision.

Zhouchen Lin (Fellow, IEEE) received the PhD degree in applied mathematics from Peking University, in 2000. He is currently a professor with the Key Laboratory of Machine Perception, School of Electronics Engineering and Computer Science, Peking University. His research interests include computer vision, image processing, machine learning, pattern recognition, and numerical optimization. He is an area chair of CVPR 2014/16/19/20/21, ICCV 2015, NIPS/NeurIPS 2015/18/19/20, AAAI 2019/20, IJCAI 2020/21, ICML 2020, and ICLR 2021. He was an associate editor of the *IEEE Transactions on Pattern Analysis and Machine Intelligence* and is currently an associate editor of the *International Journal of Computer Vision*.

Chunhong Pan received the BS degree in automatic control from Tsinghua University, Beijing, China, in 1987, the MS degree from the Shanghai Institute of Optics and Fine Mechanics, Chinese Academy of Sciences, China, in 1990, and the PhD degree in pattern recognition and intelligent system from the Institute of Automation, Chinese Academy of Sciences, Beijing, in 2000. He is currently a professor with the National Laboratory of Pattern Recognition of Institute of Automation, Chinese Academy of Sciences. His research interests include computer vision, image processing, computer graphics, and remote sensing.