International Journal of Computer Vision (2021) 129:638–655
https://doi.org/10.1007/s11263-020-01396-x

Progressive DARTS: Bridging the Optimization Gap for NAS in the Wild

Xin Chen[1] · Lingxi Xie[2] · Jun Wu[3,4] · Qi Tian[2]

Received: 20 December 2019 / Accepted: 20 October 2020 / Published online: 3 November 2020

Abstract
With the rapid development of neural architecture search (NAS), researchers found powerful network architectures for a wide range of vision tasks. Like the manually designed counterparts, we desire the automatically searched architectures to have the ability of being freely transferred to different scenarios. This paper formally puts forward this problem, referred to as NAS in the wild, which explores the possibility of finding the optimal architecture in a proxy dataset and then deploying it to mostly unseen scenarios. We instantiate this setting using a currently popular algorithm named differentiable architecture search (DARTS), which often suffers unsatisfying performance while being transferred across different tasks. We argue that the accuracy drop originates from the formulation that uses a super-network for search but a sub-network for re-training. The different properties of these stages have resulted in a significant optimization gap, and consequently, the architectural parameters "over-fit" the super-network. To alleviate the gap, we present a progressive method that gradually increases the network depth during the search stage, which leads to the Progressive DARTS (P-DARTS) algorithm. With a reduced search cost (7 hours on a single GPU), P-DARTS achieves improved performance on both the proxy dataset (CIFAR10) and a few target problems (ImageNet classification, COCO detection and three ReID benchmarks). Our code is available at https://github.com/chenxin061/pdarts.

Keywords Neural architecture search · Optimization gap · Progressive DARTS

1 Introduction

Recently, the research progress of computer vision has been largely boosted by deep learning (LeCun et al. 2015). The core part of deep learning is to design and optimize deep neural networks, for which a few popular models have been manually designed and achieved state-of-the-art performance at that time (Krizhevsky et al. 2012; Szegedy et al. 2015; He et al. 2016; Zhang et al. 2018; Howard et al. 2017). However, designing neural network architectures requires both expertise and heavy computational resources. The appearance of neural architecture search (NAS) has changed this situation, which aims to discover powerful network architectures automatically and has achieved remarkable success in image recognition (Zoph and Le 2017; Zoph et al. 2018; Liu et al. 2018a; Tan and Le 2019).

In the early age of NAS, researchers focused on heuristic search methods, which sample architectures from a large search space and perform individual evaluations. Such approaches, while being safe in finding powerful architectures, require massive computational overheads (Zoph and Le 2017; Real et al. 2018; Zoph et al. 2018). To alleviate this burden, researchers have designed efficient approaches to reuse computation in the searched architectures (Cai et al. 2018), which was later developed into constructing a super-network to cover the entire search space (Pham et al. 2018). Among them, DARTS (Liu et al. 2019a) is an elegant solution that relaxes the discrete search space into a continuous, differen-

Communicated by Mei Chen.

✉ Jun Wu
wujun@fudan.edu.cn

Xin Chen
1410452@tongji.edu.cn

Lingxi Xie
198808xc@gmail.com

Qi Tian
tian.qi1@huawei.com

[1] Tongji University, Shanghai, People's Republic of China

[2] Huawei Inc., Shenzhen, People's Republic of China

[3] School of Computer Science, Fudan University, Shanghai, People's Republic of China

[4] Shanghai Key Lab of Intelligent Information Processing, Fudan University, Shanghai, People's Republic of China

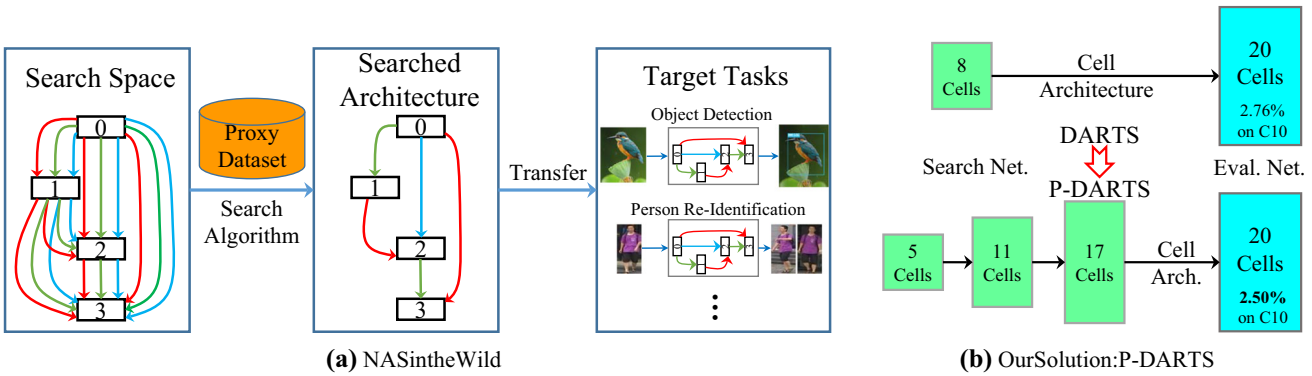


Fig. 1 Left: the setting of **NAS in the wild**, which aims to transfer the optimal architecture found in the proxy dataset to unknown scenarios. Right: our solution, P-DARTS (bottom), bridges the **optimization gap** between architecture search and evaluation by gradually increasing the depth of the super-network. DARTS (top) is also placed here for comparison. Green and blue indicate search and evaluation, respectively

tiable function. Thus, the search process requires optimizing the super-network and can be finished within GPU-hours.

Despite the efficiency of super-network-based search methods, most of them suffer from the issue of instability, which indicates that (i) the accuracy can be sensitive to random initialization, and (ii) the searched architecture sometimes incurs unsatisfying performance in other datasets or tasks. While directly searching over the target problem is always a solution, we argue that studying this topic may unleash the potentials of NAS. To this end, we formalize a setting named **NAS in the wild**, illustrated in Figure 1, which advocates for the searched architecture on any proxy dataset to be easily deployed to different application scenarios.

We argue that the instability issue originates from that the search stage fits the *super-network* on the proxy dataset, but the re-training stage actually applies the optimal *sub-network* to either the same dataset or a different task. Even if the proxy dataset and the target dataset are the same, one cannot expect the best super-network, after being pruned, produces the best sub-network. This is called the **optimization gap**. In this work, we explore a practical method to alleviate the gap, which involves gradually adjusting the super-network so that its properties converge to the sub-network by the end of the search process.

Our approach, named Progressive DARTS (P-DARTS), is built on DARTS, a recently published method for differentiable NAS. As shown in Figure 1(b), the search process of P-DARTS is divided into multiple stages, and the depth of the super-network gets increased at the end of each stage. This brings two technical issues, and we provide solutions accordingly. First, since heavier computational overheads are required when searching with a deeper super-network, we propose **search space approximation**, which reduces the number of candidates (operations) when the network depth is increased. Second, optimizing a deep super-network may cause unstable gradients, and thus the search algorithm is biased heavily towards *skip-connect*, a learning-free operator that often falls on a rapid direction of gradient decay. Consequently, it reduces the learning ability of the found architecture, for which we propose *search space regularization*, which (i) introduces operation-level Dropout (Srivastava et al. 2014) to alleviate the dominance of *skip-connect* during training, and (ii) regularizes the appearance of *skip-connect* when determining the final sub-network.

The effectiveness of P-DARTS is firstly verified on the standard vision setting, i.e., searching and evaluating the architecture on the CIFAR10 dataset. We achieve state-of-the-art performance (a test error of 2.50%) on CIFAR10 with 3.4M parameters. In addition, we demonstrate the benefits of search space approximation and regularization: the former reduces the search cost to 0.3 GPU-days on CIFAR10, surpassing ENAS (Pham et al. 2018), an approach known for search efficiency; the latter largely reduces the fluctuation of individual search trials and thus improving its reliability. Next, we investigate the application in the wild, in which the searched architecture on CIFAR10 transfers well to CIFAR100 classification, ImageNet classification, COCO detection, and three person re-identification (ReID) tasks, e.g., on ImageNet, it achieves top-1/5 errors of 24.4%/7.4%, respectively, comparable to the state-of-the-art under the mobile setting. Furthermore, architecture search is also performed on ImageNet, and the discovered architecture shows superior performance.

The preliminary version of this work appeared as (Chen et al. 2019a). In this journal version, we extend the original work by several aspects. First, we advocate for a new setting named *NAS in the wild*, which provides a benchmark for evaluating the generalization ability of NAS approaches. Second, to obtain good performance on this new setting, we extend the depth gap raised in (Chen et al. 2019a) into a more general optimization gap that exists in differentiable architecture search and investigate the width gap, another

critical factor of optimization gap aside from the depth gap. As a side benefit, the improved approach can directly search on ImageNet and thus produces more powerful architectures for high-resolution input images. Third, we complement a few diagnostic experiments to further reveal that bridging the optimization gap is helpful to accomplish the goal of NAS in the wild.

The remaining part of this paper is organized as follows. Section 2 briefly introduces related work to our research. Then, Sect. 3 illustrates the problem, NAS in the wild, and Sect. 4 elaborates the optimization gap and the P-DARTS approach. After extensive experiments are shown in Sect. 5, we conclude this work in Sect. 6.

2 Related Work

Image recognition is a fundamental task in computer vision. In recent years, with the development of deep learning, CNNs have been dominating image recognition (Krizhevsky et al. 2012; Simonyan and Zisserman 2015; He et al. 2016). A few elaborately designed handcrafted architectures have been proposed, including VGGNet (Simonyan and Zisserman 2015), ResNet (He et al. 2016), DenseNet (Huang et al. 2017), etc., all of which highlighted the importance of human expertise in network design.

In the era of hand-designed architectures, the main roadmap of architecture design resided in how to enlarge the depth of CNNs efficiently. AlexNet (Krizhevsky et al. 2012) proposed to use the ReLU activation function and Local Response Normalization (LRN) to alleviate the gradient diffusion and achieved the state-of-the-art performance on ImageNet classification at that time. VGGNet (Simonyan and Zisserman 2015) proposed to stack convolutions with identical small kernel size and initialize deeper networks with previously learned weights of a shallow work, which resulted in a network of 19 layers. GoogLenet (Szegedy et al. 2015) introduced to connect convolutions with different kernel sizes in parallel, which led to a reduction of network parameters, an increase of network depth, and a promotion on parameter utilization. In ResNet (He et al. 2016), the depth of networks was further increased to 152 layers for ImageNet and even 1202 layers for CIFAR10, with the help of the newly proposed skip connection and residual block. After that, DenseNet (Huang et al. 2017) inserted skip connection between all layers in the building block to formulate a densely connected CNN, which largely strengthened information propagation and feature reutilization. Apart from this depth route, network width was also a critical aspect of performance promotion. WRN (Zagoruyko and Komodakis 2016) explored the possibility of scaling up the network width of ResNet and achieved brilliant results. PyramidNet (Han et al. 2017) extended this idea to design a pyramid-like ResNet, which further promoted the network capability.

This work lies in the category of the emerging field of neural architecture search, a process of automating architecture engineering technique (Elsken et al. 2018). In the early 2000s, pioneer researchers attempted to generate better topology automatically with evolutionary algorithms (Stanley and Miikkulainen 2002). Early NAS works tried to search for basic components and topology of neural networks to construct a complete network (Baker et al. 2017; Suganuma et al. 2017; Xie and Yuille 2017), while recent works focused on finding robust cells (Zoph et al. 2018; Real et al. 2018; Dong and Yang 2019b). Among these works, heuristic algorithms were widely adopted in the NAS pipeline. Baker *et al.* (Baker et al. 2017) firstly applied reinforcement learning (RL) to neural architecture search and adopted an RNN-based controller to guide the sampling process for the network configuration. (Xie and Yuille 2017) encoded the architecture of a CNN into binary codes and used a general evolutionary algorithm to evolve for a better global network topology. Considering the weakness of the scalability of a global network architecture, (Zoph et al. 2018) adopted RL to search for the configuration of building blocks, which are also referred to as cells. (Real et al. 2018) proposed to regularize the standard evolutionary algorithm in the NAS pipeline with aging evolution and, for the first time, surpassed the best manually designed architectures on image recognition.

A critical drawback of the above approaches is the expensive search cost (e.g., 3150 GPU-days for EA-based AmoebaNet (Real et al. 2018) and 20, 000 GPU-days for RL-based NASNet (Zoph and Le 2017)), because their methods require to sample and evaluate numerous architectures by training them from scratch. There were two lines of solutions. The first one involved reducing the search space (Zoph et al. 2018), and the second one optimized the exploration policy (e.g., learning a surrogate model (Liu et al. 2018a)) in the search space so that the search process becomes more efficient.

Recently, search efficiency has become one of the main concerns on NAS, and the search cost was reduced to a few GPU-days with the help of weight sharing technique (Pham et al. 2018; Liu et al. 2019a). In this pipeline, a super-network that contains all candidate architectures in the search space is trained, and sub-architectures are evaluated with shared weights from the super-network. ENAS (Pham et al. 2018) proposed to adopt a parameter sharing scheme among child models to bypass the time-consuming process of candidate architecture evaluation by training them from scratch, which dramatically reduced the search cost to less than one GPU-day. DARTS (Liu et al. 2019a) introduced a differentiable NAS framework to relax the discrete search space into a continuous one by weighting candidate operations with architectural parameters, which achieved comparable perfor-

mance and remarkable efficiency improvement compared to previous approaches. Following DARTS, GDAS (Dong and Yang 2019b) proposed to use the Gumbel-softmax sampling trick to guide the sub-graph selection process. With the BinaryConnect scheme, ProxylessNAS (Cai et al. 2019) adopted the differentiable framework and proposed to search architectures on the target task instead of adopting the conventional proxy-based framework. A main drawback of DARTS-based approaches is the instability issue caused by the optimization gap depicted in Sect. 1. SNAS (Xie et al. 2019b) proposed to constrain the architecture parameters to be one-hot to tackle the inconsistency in optimizing objectives between search and evaluation scenarios, which can be regarded as an attempt of reducing the optimization gap. However, SNAS reported only comparable classification performance to DARTS on both proxy and target datasets.

3 Problem: *NAS in the Wild*

We investigate the setting of *NAS in the Wild*, which seeks for a NAS algorithm that can search in a *proxy* dataset and freely transfer to a wide range of target datasets or even other types of recognition tasks. This is important for real-world scenarios, as there may not be sufficient resources, in terms of either data or computation, for a complete NAS process to be executed.

Note that the community has witnessed a few recent works, sometimes referred to as *proxyless* NAS (Cai et al. 2019), in searching neural architectures on the target dataset directly. Our setting does not contradict these efforts, and we argue that both settings have their own advantages. On the one hand, searching on the target dataset directly enables more accurate properties of the specified dataset to be captured and, most often, leads to improved performance on the target dataset. On the other hand, we desire the ability of directly transferring the searched architecture to other scenarios. This task not only makes it easier in application, but also raises new challenges which we believe beneficial for the research field of NAS.

The most significant difficulty brought by this setting is the enlarged gap between the search stage and the evaluation stage, which we will elaborate in detail in Sect. 4.2. In this paper, we present a practical solution that largely shrinks this gap and thus improves the ability of model transfer.

4 Method: Progressive DARTS

4.1 Preliminary: Differentiable Architecture Search

Our work is based on DARTS (Liu et al. 2019a), which adopts a cell-based search framework that searches for robust architectures of building blocks, i.e., cells, and then stacks searched cells orderly for L times to construct the target network. Thus, the search space is represented in the form of cells. A cell is denoted as a directed acyclic graph (DAG) $\mathcal{G}$ and composed of N nodes (vertexes) and their corresponding edges. A node x_i represents a feature layer, i.e., the output of a specific operation. The first two nodes of a cell are the input nodes, which come from the outputs of previous cells or stem convolutions located at the beginning of the network. We denote the operation space as $\mathcal{O}$, in which each element represents a candidate operation (mathematical function) $o(\cdot)$. An intermediate node x_j is connected to all of its preceding nodes $\{x_0, x_1, x_2, ..., x_{j-1}\}$ with edge $\mathrm{E}_{(i,j)}(i < j)$, where operations from the operation space are used to link the information flow between node x_i and x_j. To relax the discrete search space to be continuous, operations on each edge are weighted with a set of architectural parameters $\alpha^{(i,j)}$, which is normalized with the Softmax function and is thus formulated as:

$$f_{i,j}(x_i) = \sum_{o \in \mathcal{O}_{i,j}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x_i). \tag{1}$$

All feature maps passed into the intermediate node x_j are integrated together by summation, denoted as $x_j = \sum_{i<j} f_{i,j}(x_i)$. The output node is defined as $x_{N-1} = \mathrm{concat}(x_2, x_3, \cdots, x_{N-2})$, where $\mathrm{concat}(\cdot)$ concatenates all input signals in the channel dimension.

The architectural parameters in DARTS are jointly optimized with the network parameters, i.e., the convolutional weights. The output architecture is generated by operation pruning according to the learned architectural parameters, with at most one non-*zero* operation on a specific edge and two preserved edges for each intermediate node. For more technical details, please refer to the original DARTS paper (Liu et al. 2019a).

4.2 The Optimization Gap

The most significant drawback of DARTS, especially when discussed in the scenario of *NAS in the wild*, lies in the gap between search and evaluation. To be specific, the best architecture found in the search stage is not necessarily the best one in the evaluation stage. This subsection aims to analyze the reason of this phenomenon.

We use $\mathcal{D}_\mathrm{S}$ and $\mathcal{D}_\mathrm{E}$ to denote the proxy dataset and the target dataset, respectively, i.e., the datasets that the NAS approach is searched and evaluated on. We also use $\mathcal{A}$ to denote the space of architecture search, namely, each element in $\mathcal{A}$ corresponds to an architecture that is possible to be chosen. Given $\mathcal{A}$ and the architectural parameter, $\boldsymbol{\alpha}$, one can determine the final architecture, e.g., in DARTS, this is

done by a greedy algorithm and the preserved architecture is written as $\mathcal{A}(\boldsymbol{\alpha})$. Note that $\mathcal{A}$ can be different during the search and evaluation stage, e.g., in DARTS, the search stage has 8 cells and the evaluation stage has much more. We use $\mathcal{A}_{\mathrm{S}}$ and $\mathcal{A}_{\mathrm{E}}$ to avoid ambiguity.

With the above notation system, the overall objective of the search stage can be written as:

$$\alpha^{\star} = \arg\min_{\alpha} \left\{ \min_{\omega_{\mathrm{S}}} \mathcal{L}_{\mathrm{S}}(\omega_{\mathrm{S}}; \mathcal{A}_{\mathrm{S}}(\alpha), \mathcal{D}_{\mathrm{S}}) \right\}. \tag{2}$$

This optimization determines the best architecture, $\mathcal{A}_{\mathrm{S}}(\boldsymbol{\alpha}^{\star})$ to be used afterwards. However, at the evaluation stage, it is possible that the configuration is changed towards better performance, and so the best architecture is changed into $\mathcal{A}_{\mathrm{E}}(\boldsymbol{\alpha}^{\star})$ and thus we have:

$$\omega_{\mathrm{E}}^{\star} = \arg\min_{\omega_{\mathrm{E}}} \mathcal{L}_{\mathrm{E}}\left(\omega_{\mathrm{E}}; \mathcal{A}_{\mathrm{E}}\left(\boldsymbol{\alpha}^{\star}\right), \mathcal{D}_{\mathrm{E}}\right), \tag{3}$$

where ω_{E} denotes the network parameters at the evaluation time. The above flowchart makes an assumption that if an individual search process is performed under the same configuration of evaluation, namely,

$$\alpha^{\star} = \arg\min_{\alpha} \left\{ \min_{\omega_{\mathrm{E}}} \mathcal{L}_{\mathrm{E}}(\omega_{\mathrm{E}}; \mathcal{A}_{\mathrm{E}}(\alpha), \mathcal{D}_{\mathrm{E}}) \right\}, \tag{4}$$

the best architecture obtained will be the same, or nearly the same, as that found in Eqn (2). In other words, Eqn (2) is an inevitable approximation of Eqn (4), since (i) $\mathcal{A}_{\mathrm{E}}$ can be more complex than $\mathcal{A}_{\mathrm{S}}$ so that the search stage cannot support multiple operators to be considered under this larger space, and (ii) in our setting, *NAS in the wild*, $\mathcal{D}_{\mathrm{E}}$ is not assumed to be known beforehand.

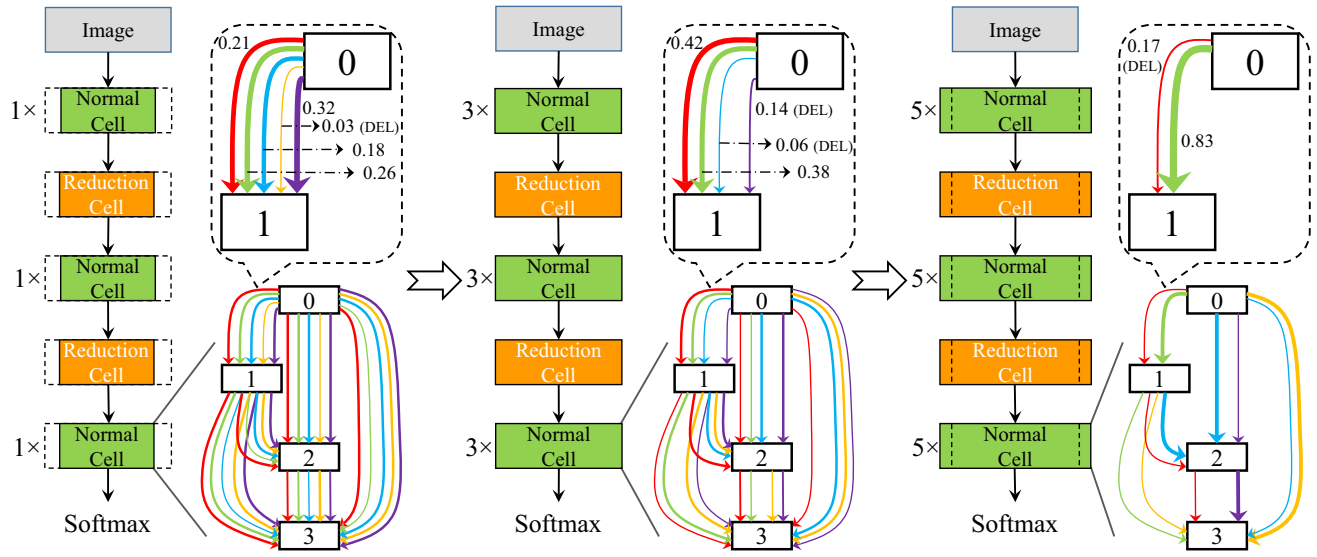
Hence, there are possibly a significant mismatch between search and evaluation scenarios, e.g., in network shape, hyper-parameters, training policies, etc. We summarize these problems into the **optimization gap** between training the super-network and applying the sub-network to the target network. For example, a typical optimization gap comes from the inconsistency of the operation pruning process, since the objective of the super-network is to jointly optimize network weights ω_{S} of all candidate operations and the architectural parameters α, while the objective of training the target network is only to optimize the network weights ω_{E} of a few selected operations. These mismatches can result in dramatic performance deterioration when the discovered architectures are applied to real-world applications.

Our solution is to reduce the optimization gap as we can. As the gap between the proxy and target datasets cannot be eliminated (we will see examples in Sect. 5.4 that how the domain gap affects the performance of NAS), we mainly focus on the gap between $\mathcal{A}_{\mathrm{S}}(\boldsymbol{\alpha}^{\star})$ and $\mathcal{A}_{\mathrm{E}}(\boldsymbol{\alpha}^{\star})$. Two strategies are introduced. **First**, we gradually shrink the gap between $\mathcal{A}_{\mathrm{S}}$ and $\mathcal{A}_{\mathrm{E}}$ during the search process, meanwhile we apply approximation to keep the algorithm efficient in both time and memory – we will introduce it as *search space approximation* in Sect. 4.3.1. **Second**, we reduce the variation of the architectural parameter, $\boldsymbol{\alpha}^{\star}$, by applying some reasonable priors to it. This effectively avoids the search algorithm to "over-fit" on $\mathcal{A}_{\mathrm{S}}$ – we will introduce it as *search space regularization* in Sect. 4.3.2.

4.3 Progressive Search to Bridge the Depth Gap

Among the optimization gaps, that caused by different network depths is one of the main sources of performance deterioration. We propose to alleviate it by progressively increasing the search depth, which is built upon our search space approximation scheme. Besides, the mismatch on network width, i.e., the number of channels of feature maps, is also an essential factor associated with performance when searching architectures on large and complex datasets, and we tackle it by progressively increasing search width. Here we mainly discuss the depth gap and leave the discussion on the width gap to Sect. 5.2.5.

To be specific, the architecture search process of DARTS is performed on a super-network with 8 cells, and the discovered architecture is evaluated on a network with either 20 cells (on CIFAR10) or 14 cells (on ImageNet). There is a considerable difference between the behavior of shallow and deep networks (Ioffe and Szegedy 2015; Srivastava et al. 2015; He et al. 2016), which implies that the architectures we discovered in the search process are not necessarily the optimal one for evaluation. We name this the **depth gap** between search and evaluation. To verify it, we executed the search process of DARTS for multiple times and found that the normal cells of discovered architectures tend to keep shallow connections instead of deep ones, i.e., the search algorithm prefers to preserve those edges connected to the input nodes instead of cascading between intermediate nodes. This is because shallow networks often enjoy faster gradient descent during the search process. However, such property contradicts the common sense that deeper networks tend to perform better (Simonyan and Zisserman 2015; Szegedy et al. 2015; He et al. 2016; Huang et al. 2017). Therefore, we propose to bridge the depth gap with the strategy that progressively increases the network depth during the search process, so that at the end of the search, the depth of the super-network is sufficiently close to the network configuration used in the evaluation. Here we adopt a progressive manner, instead of directly increasing the search depth to the target level, because we expect to search in shallow networks to reduce the search space with respect to candidate operations, so as to alleviate the risk of search in deep networks. The effectiveness of this progressive strategy will be verified in Sect. 5.2.1.



(a) Initial Stage **(b)** Intermediate Stages **(c)** Final Stage

Fig. 2 The overall pipeline of P-DARTS (best viewed in color). For simplicity, only one intermediate stage is shown, and only the normal cells are displayed. The depth of the super-network increases from 5 at the initial stage to 11 and 17 at the intermediate and final stages, while the number of candidate operations (shown in connections with different colors) is shrunk from 5 to 4 and 2 accordingly. The dashed rectangles in (a) and (c) are identical to those in (b), indicating a progressive width growth during the search process. The lowest-scored ones at the previous stage are dropped (the scores are shown next to each connection). We obtain the final architecture by considering the final scores and possibly additional rules

The difficulty comes from two aspects. First, the computational overhead increases linearly with the search depth, which brings issues in both time expenses and computational overheads. In particular, in DARTS, GPU memory usage is proportional to the depth of the super-network. The limited GPU memory forms a major obstacle, and the most straightforward solution is to trim the channel number in each operation – DARTS (Liu et al. 2019a) tried it but reported a slight performance deterioration, because it enlarged the mismatch on network width, another aspect of the optimization gap. To address this problem, we propose a **search space approximation** scheme to progressively reduce the number of candidate operations at the end of each stage according to the architectural parameters, the scores of operations in the previous stage as the criterion of selection. Details of search space approximation are presented in Sect. 4.3.1.

Second, we find that when searching on a deeper super-network, the differentiable approaches tend to bias towards the *skip-connect* operation, because it accelerates forward and backward propagation and often leads to the fastest route of gradient descent. However, this operation is parameter-free, which implies a relatively weak ability to learn visual representations. To this end, we propose another scheme named **search space regularization**, which adds an operation-level Dropout (Srivastava et al. 2014) to prevent the architecture from "over-fitting" and restricts the number of preserved *skip-connects* for further stability. Details of search space regularization are presented in Sect. 4.3.2.

4.3.1 Search Space Approximation

As demonstrated previously, a straightforward way to tackle the optimization gap is to directly apply the configuration of $\mathcal{A}_{\mathrm{E}}$ during the search process. However, this will cause severe computational overhead, which impels us to consider more on the efficiency in computation. A toy example is presented in Figure 2 to demonstrate the idea of search space approximation. The entire search process is split into multiple stages, including an initial stage, one or a few intermediate stages, and a final stage. For each stage, $\mathfrak{S}_k$, the super-network is constructed with L_k cells and the operation space consists of O_k candidate operations, i.e., $|\mathcal{O}^k_{(i,j)}| = O_k$. Note that in Figure 2, both the depth gap and the width gap are presented but in this section we focus on the depth gap and leave the investigation of the width gap to Sect. 5.2.5.

According to our motivation, the super-network of the initial stage is relatively shallow, but the operation space is large ($\mathcal{O}^1_{(i,j)} \equiv \mathcal{O}$). After each stage, $\mathfrak{S}_{k-1}$, the architectural parameters α_{k-1} are optimized and the scores of the candidate operations on each edge are ranked according to the learned α_{k-1}. We increase the depth of the super-network by stacking more cells, i.e., $L_k > L_{k-1}$, and approximate the operation space according to the ranking scores in the meantime. As a consequence, the new operation set on each edge $\mathcal{O}^k_{(i,j)}$ has a smaller size than $\mathcal{O}^{k-1}_{(i,j)}$, or equivalently, $O_k < O_{k-1}$. The criterion of approximation is to drop a part of less important operations on each edge, which are

specified to be those assigned with a lower weight during the previous stage, $\mathfrak{S}_{k-1}$. As shown in Table 2, this strategy is memory-efficient, which enables the deployment of our approach on regular GPUs, e.g., with a memory of 16GB.

The growth of architecture depth continues until it is sufficiently close to that used in the evaluation. After the last search stage, the final cell topology (bold lines in Figure 2(c)) is derived according to the learned architecture parameters α_K. Following DARTS, for each intermediate node, we keep two individual edges whose largest non-*zero* weights are top-ranked and preserve the most important operation on each retained edge.

4.3.2 Search Space Regularization

At the start of each stage, $\mathfrak{S}_k$, we train the (modified) architecture from scratch, i.e., all network weights and architectural parameters are re-initialized randomly, because several candidates have been abandoned on each edge[1]. However, training a deeper network is harder than training a shallow one (Srivastava et al. 2015). In our particular setting, we observe that information prefers to flow through *skip-connect* instead of *convolution* or *pooling*, which is arguably due to the fact that *skip-connect* often leads to rapid gradient descent, especially on small proxy datasets (CIFAR10 or CIFAR100) which are relatively easy to fit. The gradient of a *skip-connect* operation with respect to the input is always 1.0, while that of *convolutions* is much smaller ($\left[10^{-3}, 10^{-2}\right]$ according to our statistics). Another important reason is that, during the start of training, weights in *convolutions* are less meaningful, which results in unstable outputs compared to *skip-connect* which is weight-free, and such outputs are not likely to have high weights in classification. Both reasons make *skip-connect* accumulate weights much more rapidly than other operations. Consequently, the search process tends to generate architectures with many *skip-connect* operations, which limits the number of learnable parameters and thus produces an unsatisfying performance at the evaluation stage. This is essentially a kind of over-fitting.

We address this problem by search space regularization, which consists of two parts. First, we insert operation-level Dropout (Srivastava et al. 2014) after each *skip-connect* operation to partially "cut off" the straightforward path through *skip-connect*, and facilitate the algorithm to explore other operations. However, if we constantly block the path through *skip-connect*, the algorithm will unfairly drop them by assigning lower weights to them, which is harmful to the final performance. To address this contradiction, we gradually decay the Dropout rate during the training process in each search stage. Thus, the straightforward path through *skip-connect* is blocked at the beginning and treated equally afterward when parameters of other operations are well learned, leaving the algorithm itself to make the decision.

Despite the use of operation-level Dropout, we still observe that *skip-connect*, as a special kind of operation, has a significant impact on recognition accuracy at the evaluation stage. Empirically, we perform 3 individual search processes on CIFAR10 with identical search setting, but find that the number of preserved *skip-connects* in the normal cell, after the final stage, varies from 2 to 4. In the meantime, the recognition performance at the evaluation stage is also highly correlated to this number, as we observed before. This motivates us to design the second regularization rule, architecture refinement, which simply restricts the number of preserved *skip-connect* operations of the final architecture to be a constant M. This is done with an iterative process, which starts with constructing a cell topology using the standard rule described by DARTS. If the number of *skip-connects* is not exactly M, we search for the M *skip-connect* operations with the largest architecture weights in this cell topology and set the weights of others to 0, then redo cell construction with modified architectural parameters.

We emphasize that the second regularization technique must be applied on top of the first one, otherwise, in the situations without operation-level Dropout, the search process is producing low-quality architectural weights, based on which we could not build up a powerful architecture even with a fixed number of *skip-connects*.

4.4 Relationship to Prior Work

Though having a similar name, Progressive NAS or PNAS (Liu et al. 2018a) was driven by a different motivation. PNAS explored the search space progressively by searching for operations node-by-node within each cell. Our approach shares a similar progressive search manner – we perform the search at the cell level to enlarge the architecture depth, while PNAS did it at the operation level (within a cell) to reduce the number of architectures to evaluate.

There exist other efforts in alleviating the optimization gap between search and evaluation. For example, SNAS (Xie et al. 2019b) aimed at eliminating the bias between the search and evaluation objectives of differentiable NAS approaches by forcing the architecture weights on each edge to be one-hot. Our work is also able to get rid of the bias, which we achieve by enlarging the architecture depth during the search stage.

[1] We also tried to start with architectural parameters learned from the previous stage, $\mathfrak{S}_{k-1}$, and adjust them according to Eq. 1 to ensure that the weights of preserved operations should still sum to one. This strategy reported slightly lower accuracy. Actually, we find that only an average of 5.3 (out of 14 normal edges) most significant operations in $\mathfrak{S}_1$ continue to have the largest weight in $\mathfrak{S}_2$, and the number is only slightly increased to 6.7 from $\mathfrak{S}_2$ to $\mathfrak{S}_3$ – this is to say, deeper architectures may have altered preferences.

Another example of bridging the optimization gap is ProxylessNAS (Cai et al. 2019), which introduced a differentiable NAS scheme to directly learn architectures on the target task (and hardware) without a proxy dataset. It achieved high memory efficiency by applying binary masks to candidate operations and forcing only one path in the over-parameterized network to be activated and loaded into GPU. Different from it, our approach tackles the memory overhead by search space approximation. Besides, ProxylessNAS searched for global topology instead of cell topology, which requires strong priors on the target task as well as the search space, while P-DARTS does not need such priors. Our approach is much faster than ProxylessNAS (0.3 GPU-days vs. 4.0 GPU-days on CIFAR10 and 2.0 GPU-days vs. 8.3 GPU-days on ImageNet).

Some recent work (Shu et al. 2020; Zela et al. 2020) pointed out some similar phenomena as ours. (Shu et al. 2020) systematically investigated the convergence condition of some existing NAS methods built upon cell-based search space, e.g., DARTS, and revealed that the cells discovered by these methods tend to keep shallow and wide cell structures because such topologies leads to smoother loss landscape and smaller gradient variance than others, which coincides with the observation in our work. (Zela et al. 2020) studied the relationship between the eigenspectrum of the Hessian of the validation loss with respect to the architectural parameters and the generalization error of discovered architectures and proposed two schemes to stabilize the search process, i.e., early stopping and inner objective regularization. Our search space regularization schemes can also make the search process stabilized, while our regularization is different from theirs.

EfficientNet (Tan and Le 2019) also investigated the problem that the searched and deployed architectures have different sizes (depth, width, and resolution). However, it applied the method that magnifies the searched network directly which is the same as the DARTS baseline. Our method allows the searched architecture to grow gradually to the target size instead of directly scaling. Besides, two different search spaces are used by these two algorithms. It was widely accepted in the NAS community that the search space used by EfficientNet (also used by MnasNet (Tan et al. 2019), ProxylessNAS (Cai et al. 2019), etc.) is more friendly to ImageNet and the search space used by DARTS is more friendly to CIFAR10.

Last but not least, we believe that the phenomenon that the *skip-connect* operation emerges may be caused by the mathematical mechanism that DARTS was built upon. Some recent work (Bi et al. 2019) pointed out issues in optimization, and we look forward to exploring the relationship between these issues and the optimization gap.

5 Experiments

5.1 The CIFAR10 and CIFAR100 Datasets

Following standard vision setting, we search and evaluate architectures on the CIFAR10 (Krizhevsky and Hinton 2009) dataset. To further demonstrate the capability of our proposed method, we also execute architecture search on CIFAR100.

Each of CIFAR10 and CIFAR100 has 50K/10K training/testing images with a fixed spatial resolution of 32×32, which are distributed over 10/100 classes. In the architecture search scenario, the training set is randomly split into two equal subsets, one for learning network parameters (e.g., convolutional weights) and the other for tuning the architectural parameters (i.e., operation weights). In the evaluation scenario, standard training/testing split is applied.

5.1.1 Architecture Search

The whole search process is split into 3 stages. The search space and network configuration are identical to DARTS at the initial stage (stage 1) except that only 5 cells are stacked in the search network for acceleration (we tried the original setting with 8 cells and obtained similar results). The number of stacked cells increases from 5 to 11 for the intermediate stage (stage 2) and 17 for the final stage (stage 3). The numbers of operations preserved on each edge of the super-network are 8, 5, and 3 for stage 1, 2, and 3, respectively.

The Dropout probability on *skip-connect* is decayed exponentially and the initial values for the reported results are set to be 0.0, 0.4, 0.7 on CIFAR10 for stage 1, 2 and 3, respectively, and 0.1, 0.2, 0.3 for CIFAR100. For a proper tradeoff between classification accuracy and computational overhead, the final discovered cells are restricted to keep at most 2 *skip-connects*, which guarantees a fair comparison with DARTS and other state-of-the-art approaches. For each stage, the super-network is trained for 25 epochs with a batch size of 96, where only network parameters are tuned in the first 10 epochs while both network and architectural parameters are jointly optimized in the rest 15 epochs. An Adam optimizer with learning rate $\eta = 0.0006$, weight decay 0.001 and momentum $\beta = (0.5, 0.999)$ is adopted for architectural parameters. The limitation of GPU memory is the main concern when we determine hyper-parameters related to GPU memory size, e.g., the batch size. The first-order optimization scheme of DARTS is leveraged to learn the architectural parameters in consideration of acceleration, thus the architectural parameters and network parameters are optimized in an alternative manner.

The architecture search process on CIFAR10 and CIFAR100 is performed on a single Nvidia Tesla P100, which takes around 7 hours, resulting in a search cost of 0.3 GPU-days. When we change the GPU device to an Nvidia Tesla V100

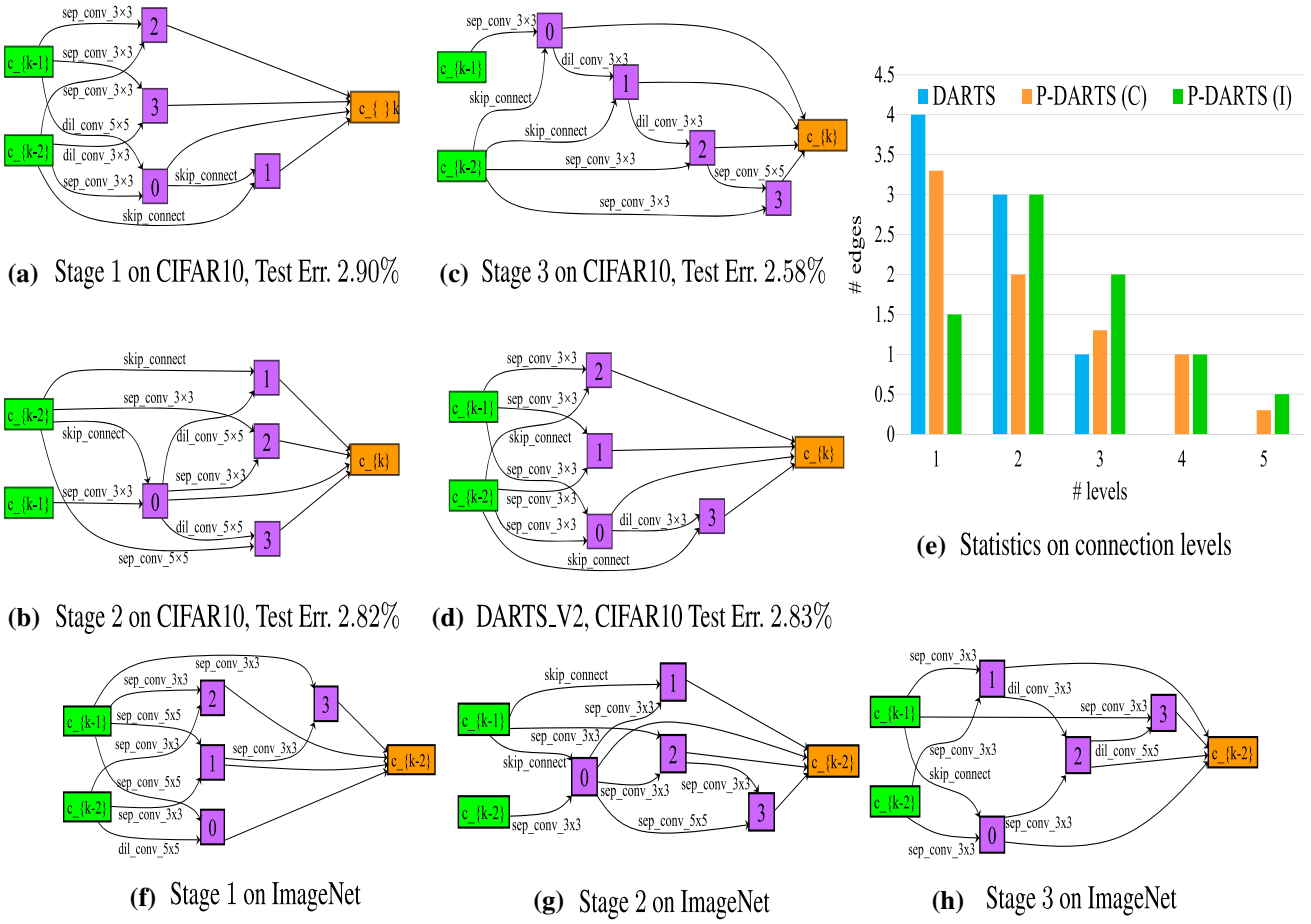


Fig. 3 (a)–(d): normal cells discovered by different search stages of P-DARTS and the second-order DARTS (DARTS_V2) on CIFAR10. The depths of search networks are 5, 11 and 17 cells for stage 1, 2 and 3 of P-DARTS and 8 for DARTS_V2. When the depth of the search network increases, more deep connections are preserved. Note that the operation on edge $E_{(0,1)}$ of stage 1 is a parameter-free *skip_connect*, thus it is strictly not a deep connection. (e): number of edges with different connection levels in the discovered architectures of DARTS and P-DARTS. More deep connections with higher connection levels are preserved in architectures discovered by P-DARTS, while only one exists in the architecture searched by DARTS. The letter C and I in the legend denote CIFAR10 and ImageNet, respectively. (f)–(h): normal cells discovered by different search stages of P-DARTS on ImageNet. The depths of search networks are 5, 8 and 11 cells for stage 1, 2 and 3, respectively

(16GB), the search cost is reduced to 0.2 GPU-days (around 4.5 hours).

Architectures discovered by P-DARTS on CIFAR10 tend to preserve more deep connections than the one discovered by DARTS, as shown in Figure 3(c) and Figure 3(d). Moreover, the deep connections in the architecture discovered by P-DARTS are deeper than that in DARTS, which means that the longest path in the cell cascades more levels in depth. In other words, there are more serial layers in the cell instead of parallel ones, making the target network further deeper and achieving better classification performance.

Notably, our method also allows architecture search on CIFAR100 while prior approaches mostly failed. The evaluation results in Table 1 show that the discovered architecture on CIFAR100 outperforms those architectures transferred from CIFAR10. We tried to perform architecture search on CIFAR100 with DARTS using the code released by the authors but get architectures full of *skip-connects*, which results in much worse classification performance.

5.1.2 Architecture Evaluation

Following the convention(Liu et al. 2019a), an evaluation network stacked with 20 cells and 36 initial channels is trained from scratch for 600 epochs with a batch size of 128. Additional regularization methods are also applied including Cutout regularization (DeVries and Taylor 2017; Zhong et al. 2017) of length 16, drop-path (Larsson et al. 2017) of probability 0.3 and auxiliary towers (Szegedy et al. 2015) of weight 0.4. A standard SGD optimizer with a momentum of

Table 1 Comparison with state-of-the-art architectures on CIFAR10 and CIFAR100. † indicates that this result is obtained by transferring the corresponding architecture to CIFAR100. ‡ We ran the publicly available code with necessary modifications to fit PyTorch 1.0, and a single run took about 0.5 GPU-days for the first order and 2 GPU-days for the second order, respectively

Architecture	Test Err. (%)		Params (M)	Search cost (GPU-days)	Search method
	C10	C100			
DenseNet-BC (Huang et al. 2017)	3.46	17.18	25.6	–	Manual
NASNet-A + cutout (Zoph et al. 2018)	2.65	–	3.3	1800	RL
AmoebaNet-A + cutout (Real et al. 2018)	3.34	–	3.2	3150	Evolution
AmoebaNet-B + cutout (Real et al. 2018)	2.55±0.05	–	2.8	3150	Evolution
Hireachical Evolution (Liu et al. 2018b)	3.75±0.12	–	15.7	300	Evolution
PNAS (Liu et al. 2018a)	3.41±0.09	–	3.2	225	SMBO
ENAS + cutout (Pham et al. 2018)	2.89	–	4.6	0.5	RL
DARTS (first order) + cutout (Liu et al. 2019a)	3.00±0.14	17.76†	3.3	1.5‡	Gradient-based
DARTS (second order) + cutout (Liu et al. 2019a)	2.76±0.09	17.54†	3.3	4.0‡	Gradient-based
SNAS + mild constraint + cutout (Xie et al. 2019b)	2.98	–	2.9	1.5	Gradient-based
SNAS + moderate constraint + cutout (Xie et al. 2019b)	2.85±0.02	–	2.8	1.5	Gradient-based
SNAS + aggressive constraint + cutout (Xie et al. 2019b)	3.10±0.04	–	2.3	1.5	Gradient-based
GDAS + cutout (Dong and Yang 2019b)	2.82	18.13	2.5	0.2	Gradient-based
SETN (T=1K) + cutout (Dong and Yang 2019a)	2.69	17.25	4.6	1.8	Gradient-based
PC-DARTS + cutout (Xu et al. 2020)	2.57±0.07	–	3.6	0.1	Gradient-based
ProxylessNAS (Cai et al. 2019) + cutout	2.08	–	5.7	4.0	Gradient-based
P-DARTS (searched on CIFAR10) + cutout	2.50±0.06	17.20	3.4	0.3	Gradient-based
P-DARTS (searched on CIFAR100) + cutout	2.62	15.92±0.18	3.6	0.3	Gradient-based
P-DARTS (searched on CIFAR10, large) + cutout	2.25	15.27	10.5	0.3	Gradient-based
P-DARTS (searched on CIFAR100, large) + cutout	2.43	14.64	11.0	0.3	Gradient-based

0.9, a weight decay of 0.0003 for CIFAR10 and 0.0005 for CIFAR100 is adopted to optimize the network parameters. The cosine annealing scheme is applied to decay the learning rate from 0.025 to 0. To explore the potential of the searched architectures, we further increase the number of initial channels from 36 to 64, which is denoted as the large setting.

Evaluation results and comparison with state-of-the-art approaches are summarized in Table 1. As demonstrated in Table 1, P-DARTS achieves a 2.50% test error on CIFAR10 with a search cost of only 0.3 GPU-days. To obtain a similar performance, AmoebaNet (Real et al. 2018) spent thousands of GPU-hours, which is four orders of magnitude more computational resources. Our P-DARTS also outperforms DARTS and SNAS by a large margin with comparable parameter count. Notably, architectures discovered by P-DARTS outperform ENAS, one of the previously most efficient approaches, in both classification performance and search cost, with fewer parameters.

The architectures discovered both DARTS and P-DARTS on CIFAR10 are transferred to CIFAR100 to test the transferability between similar datasets. Obvious superiority of P-DARTS is observed in terms of classification accuracy. As mentioned previously, P-DARTS is able to support architecture search on other proxy datasets such as CIFAR100. For a fair comparison, we tried to perform architecture search on CIFAR100 with the publicly available code of DARTS but resulted in architectures full of *skip-connect* operations. The discovered architecture on CIFAR100 significantly outperforms those architectures transferred from CIFAR10. An interesting point is that the directly searched architectures perform better when evaluated on the search dataset than those transferred ones for both CIFAR10 and CIFAR100. Such a phenomenon provides a proof of the existence of dataset bias in NAS.

5.2 Diagnostic Experiments

Before continuing to ImageNet search and in-the-wild evaluation experiments, we conduct diagnostic studies to better understand the properties of P-DARTS.

5.2.1 Comparison on the Depth of Search Networks

Since the search process of P-DARTS is divided into multiple stages, we perform a case study to extract architectures from each search stage with the same rule to validate the usefulness of bridging the depth gap. Architectures from each stage are evaluated to demonstrate their capability for image classifica-

tion. The topology of discovered architectures (only normal cells are shown) and their corresponding classification performance are summarized in Figure 3 ((a)–(d)). To show the difference in the topology of cells searched with different depth, we add the architecture discovered by second-order DARTS (DARTS_V2, 8 cells in the search network) for comparison.

The lowest test error is achieved by the architecture obtained from the final search stage (stage 3), which validates the effectiveness of shrinking the depth gap. From Figure 3 ((a)–(d)) we can observe that these discovered architectures share some common edges, for example $sep\_conv\_3 \times 3$ at edge $\mathrm{E}_{(c_{k-2},2)}$ for all stage of P-DARTS and at edge $\mathrm{E}_{(c_{k-1},0)}$ for stage 2, 3 of P-DARTS and DARTS_V2. These common edges serve as a solid proof that operations with high importance are preserved by the search space approximation scheme. Differences also exist between these discovered architectures, which we believe is the key factor that affects the capability of these architectures. Architectures generated by shallow search networks tend to keep shallow connections, while with deeper search networks, the discovered architectures prefer to pick some preceding intermediate nodes as input, resulting in cells with deep connections. This is because it is harder to optimize a deep search network, so the algorithm has to explore more paths to find the optimum, which results in more complex and powerful architectures.

Additionally, we perform quantitative analysis on the discovered architectures by P-DARTS of three individual runs and summarize the average levels of their connections in Figure 3(e). For comparison, we also add the architecture found by the second-order DARTS into this analysis. While the preserved edges of DARTS are almost all shallow (7 over 8 of level 1 and level 2), P-DARTS tends to keep more deep edges.

5.2.2 Effectiveness of Search Space Approximation

The search process takes $\sim$ 7 hours on a single Nvidia Tesla P100 GPU with 16GB memory to produce the final architectures. We monitor the GPU memory usage of the architecture search process for 3 individual runs and collect the peak value to verify the effectiveness of the search space approximation scheme[2], which is shown in Table 2. The memory usage is stably under the limit of the adopted GPU, and out of memory error barely occurs, showing the validity of the search space approximation scheme on memory efficiency.

[2] Here, we do not change the batch size to fit into the GPU memory because even under a fixed batch size, the usage of GPU memory can vary since the set of preserved candidates can differ, for example, a convolutional operator occupies more memory than a pooling operator. This is why we need to discuss the stability of GPU memory usage.

Table 2 Peak GPU memory usage at different stages during three individual runs

Run No.	Mem. usage (GB)		
	Stage 1	Stage 2	Stage 3
1	9.8	14.0	14.2
2	9.8	14.4	14.5
3	9.8	14.2	14.3

The memory limit is 16GB

We perform experiments to demonstrate the effectiveness of the search space approximation scheme on improving classification accuracy. Only the final stage of the search process is executed on two different search spaces with identical settings. The first search space is progressively approximated from previous search stages, and the other is randomly sampled from the full search space. To eliminate the influence of randomness, we repeat the whole process for the randomly sampled one for 3 times with different seeds and pick the best one. The lowest test error for the randomly sampled search space is 3.43%[3], which is much higher than 2.58%, the one obtained with the approximated search space. Moreover, we performed an additional experiment with a fixed depth (8 cells) and shrunk sets of operations ($8 \rightarrow 5 \rightarrow 3$, as used in the paper), which results in a test error of 2.70%, significantly lower than the 3.00% test error obtained by the first-order DARTS. These results reveal the necessity of the search space approximation scheme.

In our pipeline, one or a few intermediate stages are allowed so that the total stages of the search process can be more than 3. We have tested another setting with 3 intermediate stages (5 stages in total, the number of cells in each stage is 5, 8, 11, 14, 17 and the corresponding number of candidates preserved in each stage is 8, 6, 5, 4, 3). Without deliberate hyper-parameter fine-tuning, we obtained a test error of 2.63% on CIFAR10, which is close to the number 2.50% reported in Table 1. This experiment suggests that our approach is not sensitive to the configuration of progressive search. Actually, the number of stages and the corresponding numbers of cells and preserved candidates in each stage can be adjusted according to the configuration of search space, character of the target task, and hardware (GPU memory).

5.2.3 Effectiveness of Search Space Regularization

We perform experiments to validate the effectiveness of search space regularization, i.e., operation-level Dropout, and architecture refinement.

Effectiveness of operation-level Dropout. Firstly, experiments are conducted to test the influence of the operation-

[3] The mean test error of these three trials is 3.61% $\pm$ 0.21% (the corresponding errors are 3.43%, 3.51% and 3.89%, respectively).

level Dropout scheme. Two sets of initial Dropout rates are adopted, i.e., 0.0, 0.0, 0.0 (without Dropout) and 0.0, 0.3, 0.6 (with Dropout) for stage 1, 2 and 3, respectively. To eliminate the potential influence of the number of *skip-connects*, the comparison is made across multiple values of M.

Test errors for architectures discovered without Dropout are 2.93%, 3.28% and 3.51% for $M = 2$, 3 and 4, respectively. When operation-level Dropout is applied, the corresponding test errors are 2.69%, 2.84% and 2.97%, significantly outperforming those without Dropout. According to the experimental results, all 8 preserved operations in the normal cell of the architecture discovered without Dropout are *skip-connects* before architecture refinement, while the number is 4 for the architecture discovered with Dropout. The diminishing on the number of *skip-connect* operations verifies the effectiveness of search space regularization on stabilizing the search process.

Effectiveness of architecture refinement. During experiments, we observe strong coincidence between the classification accuracy of architectures and the number of *skip-connect* operations in them. We perform a quantitative experiment to analyze it. Architecture refinement is applied to the same search process to produce multiple architectures where the number of preserved *skip-connect* operations in the normal cell varies from 0 to 4.

The test errors are positively correlated to the number of *skip-connects* except for $M = 0$, i.e, 2.78%, 2.68%, 2.69%, 2.84% and 2.97% for $M = 0$ to 4, while the parameters count is inversely proportional to the *skip-connect* count, i.e., 4.1M, 3.7M, 3.3M, 3.0M and 2.7M, respectively. The reason lies in that, with a fixed number of operations in a cell, the eliminated parameter-free *skip-connects* are replaced by operations with trainable parameters, e.g., *convolution*, resulting in more complex and powerful architectures.

The above observation inspired us to propose the second search space regularization scheme, architecture refinement, whose capability is validated by the following experiments. We run another 3 architecture search experiments, all with initial Dropout rates of 0.0, 0.3, and 0.6 for stage 1, 2, and 3, respectively. Before architecture refinement, the test error is $2.79 \pm 0.16\%$ and the discovered architectures are with 2, 3 and 4 *skip-connects* in normal cells. After architecture refinement, all three searched architectures are with 2 *skip-connects* in normal cells, resulting in a diminished test error of $2.65 \pm 0.05\%$. The reduction of the average test error and standard deviation reveals the improvement of the stability for the search process.

Applying search space regularization to DARTS. We apply our proposed search space regularization scheme to the original first-order DARTS, and the test error on CIFAR10 is reduced to 2.78%, significantly lower than the original 3.00% but still considerably higher than P-DARTS (2.50%). This reveals that the proposed regularization scheme is also effective in searching for relatively shallower architectures, yet another source of improvement comes from increasing search depth. The positive results indicate that the proposed search space regularization can also be plugged into other DARTS-based approaches.

5.2.4 Stability of Progressive DARTS

To verify the stability brought by P-DARTS, we execute two additional search processes of P-DARTS on CIFAR10 and the test errors of them are 2.60% and 2.43%, respectively. Adding the one in Table 1, 2.50%, this results in a mean test error of $2.51\% \pm 0.07\%$. We have also executed three individual runs for DARTS and the mean test error is $2.85\% \pm 0.23\%$ (three individual runs reported 3.11%, 2.68% and 2.77%, respectively). That being said, P-DARTS not only enjoys a lower mean test error, but also the standard deviation is much smaller, indicating that the stability of search is improved. Notably, all discovered architectures appear to have deeper connections than those for DARTS, which is believed to be one of the key factors to achieve high performance. Aside from the one in Table 3, two additional search process are also performed on ImageNet and reports a test error of 24.2% and 24.1%, respectively, very close to the one in Table 3, which also verifies the stability of our approach. The above results demonstrates the effectiveness of the alleviation of the optimization gap on the stability of the search process.

5.2.5 Discussion: Other Optimization Gaps

Apart from depth gap that we have addressed in this paper, other aspects of the optimization gap can also affect the search process of super-network-based NAS approaches. Here, we briefly discuss two aspects of them.

The width gap. One of the straightforward option comes from the width of the network. Note that during the search stage on CIFAR, the base channel number is set to be 16, while that is enlarged as 48 when the searched architecture is transferred to ImageNet (see the experimental settings in the following section). This also claims a significant optimization gap.

Therefore, it is natural to progressively increase the network width during the search stage, just like what we have done for the network depth. However, we find that the performance gain brought by this strategy is limited. Digging into the searched architecture, we find that when an increased network width is used, the search algorithm tends to find architectures with small (3×3) convolutional kernels, while the original version tends to find architectures with a considerable portion of big (5×5) kernels. Consequently, the comparison between these two options is not fair on CIFAR10, as the original (not progressively widened) version often has a larger number of parameters. This also delivers an

Table 3 Comparison with state-of-the-art architectures on ImageNet (mobile setting)

Architecture	Test Err. (%)		Params	×+	Search cost	Search method
	Top-1	Top-5	(M)	(M)	(GPU-days)	
Inception V1 (Szegedy et al. 2015)	30.2	10.1	6.6	1448	–	Manual
MobileNet (Howard et al. 2017)	29.4	10.5	4.2	569	–	Manual
ShuffleNet V1 2× (Zhang et al. 2018)	26.4	10.2	∼5	524	–	Manual
ShuffleNet V2 2× (Ma et al. 2018)	25.1	-	∼5	591	–	Manual
NASNet-A (Zoph et al. 2018)	26.0	8.4	5.3	564	1800	RL
NASNet-B (Zoph et al. 2018)	27.2	8.7	5.3	488	1800	RL
NASNet-C (Zoph et al. 2018)	27.5	9.0	4.9	558	1800	RL
AmoebaNet-A (Real et al. 2018)	25.5	8.0	5.1	555	3150	Evolution
AmoebaNet-B (Real et al. 2018)	26.0	8.5	5.3	555	3150	Evolution
AmoebaNet-C (Real et al. 2018)	24.3	7.6	6.4	570	3150	Evolution
PNAS (Liu et al. 2018a)	25.8	8.1	5.1	588	225	SMBO
RandWire-WS (Xie et al. 2019a)	25.3	7.8	5.6	583	–	Randomly wiring
MnasNet-92 (Tan et al. 2019)	25.2	8.0	4.4	388	–	RL
ProxylessNAS (GPU) (Cai et al. 2019)	24.9	7.5	7.1	465	8.3	Gradient-based
MobileNetV3 (Howard et al. 2019)	24.8	–	5.4	219	–	NetAdapt
EfficientNet-B0 (Tan and Le 2019)‡	22.7	6.5	5.3	390	–	RL
DARTS (second order) (Liu et al. 2019a)	26.7†	8.7	4.7	574	4.0	Gradient-based
SNAS (mild constraint) (Xie et al. 2019b)	27.3	9.2	4.3	522	1.5	Gradient-based
GDAS (Dong and Yang 2019b)	26.0	8.5	4.4	581	0.2	Gradient-based
SETN (Dong and Yang 2019a)	25.3	8.0	5.4	599	1.8	Gradient-based
MdeNAS (Zheng et al. 2019)	24.5	7.9	6.1	–	0.2	MDL
PC-DARTS (searched on CIFAR10) (Xu et al. 2020)	25.1	7.8	5.3	586	0.1	Gradient-based
PC-DARTS (searched on ImageNet) (Xu et al. 2020)	24.2	7.3	5.3	597	3.8	Gradient-based
P-DARTS (searched on CIFAR10)	24.4	7.4	4.9	557	0.3	Gradient-based
P-DARTS (searched on ImageNet)	24.1	7.3	5.4	597	2.0	Gradient-based

†: the top-1 test error is 25.4% when the learning rate decay schedule is cosine annealing; ‡: EfficientNet-B0 is equipped with additional network components like swish activation function and SE module, and trained with extra techniques such as AutoAugment (Cubuk et al. 2018) and stochastic depth (Huang et al. 2016)

important message: the value of shrinking the optimization gap will be enlightened in a relatively "fair" (Chu et al. 2019) search environment.

While there is no evident improvement on CIFAR10 when addressing the width gap, significant accuracy gain is obtained when architecture search is performed on ImageNet with both depth gap and width gap addressed (please refer to Sect. 5.3). On the one hand, the performance gain on ImageNet owes to the shrinkage of the width gap, which enables a successful search process. On the other hand, directly searching architectures on ImageNet eliminated the dataset gap, constituting another critical factor for performance improvement.

The gap brought by other hyper-parameters. In the search setting of DARTS, all the affine parameters of batch normalization are discarded because the architectural parameters are dynamically learned across the whole search process, and the affine parameters will rescale the output of each operation according to incorrect statistics. On the contrary, the affine option of batch normalization is switched on to recover the data distribution during the evaluation scenario, which forms another aspect of the optimization gap. However, this gap is hard to address because a bunch of additional issues may arise if we switch it on.

Furthermore, the data augmentation gap, including the inconsistent settings of Cutout, is another inconsistency between search and evaluation. There also may exist other aspects of the optimization gap, e.g., Dropout, auxiliary loss tower, etc. In (Bi et al. 2019), the authors briefly discussed some aspects of the above-mentioned ones, while the influence of these options was not clearly stated. In fact, a different setting on these aspects may cause other additional problems to disrupt qualitative and quantitative analysis on them. Additionally, the fluctuation on small scale datasets like CIFAR10 may also cause dramatic impacts on the analysis, while the

computational burden obstructs the analysis on large-scale datasets.

5.3 The ImageNet Dataset

We also search architectures directly on ImageNet to validate the applicability of our search algorithm to large-scale datasets. Experiments are performed on ILSVRC2012 (Russakovsky et al. 2015), a subset of ImageNet (Deng et al. 2009) which contains 1000 object categories and 1.28M training and 50K validation images. Following the conventions (Zoph et al. 2018; Liu et al. 2019a; Wu et al. 2019), we randomly sample a 100-class subset of the training images for architecture search. Similar to CIFAR10, all images and standard dataset partition are adopted during architecture evaluation.

5.3.1 Architecture Search

We use a similar configuration to the one used on CIFAR10 except for some minor changes. We set the numbers of cells to be 5, 8 and 11 and adjust the dropout rate to 0.0, 0.3, 0.6. Meanwhile, we increase the number of initial channels from 16 to 28, and 40 for stage 1, 2, and 3, respectively.

Architecture search on ImageNet is executed with 8 Nvidia Tesla V100, which takes around 6 hours, thus a search cost of 2 GPU-days. The search cost of P-DARTS on ImageNet is even smaller than PC-DARTS(Xu et al. 2020), a memory-efficient differentiable approach proposed recently, which demonstrates the efficiency of our proposed search space approximation scheme.

During the search process, the "over-fitting" phenomenon is largely alleviated and the number of *skip-connect* operation is well controlled. This comes from two aspects. On the one hand, gradients assigned to *skip-connects* is successfully suppressed by the first search space regularization method, i.e., adding operation level dropout on *skip-connect* operations. On the other hand, the variety and complexity of ImageNet make it more difficult to fit with those parameter-free operations than CIFAR10 and CIFAR100, forcing the network to train those operations with learnable parameters. Moreover, the discovered architecture is also with plenty of deep connections, as demonstrated in Figure 3. Such a character guarantees a favorable classification performance. We have also attempted to search architectures without progressively increasing the network width, but the discovered architectures resulted in worse classification performance, which demonstrates the usefulness of the progressive width scheme.

5.3.2 Architecture Evaluation

The transferability to large-scale recognition datasets of architecture discovered on CIFAR10 is firstly tested on the ILSVRC2012. Concurrently, the capability of the architecture directly searched on ImageNet is also evaluated. We apply the mobile setting for the evaluation scenario where the input image size is 224×224, and the number of multi-add operations is restricted to be less than 600M. A network configuration identical to DARTS is adopted, i.e., an evaluation network of 14 cells and 48 initial channels. We train each network from scratch for 250 epochs with batch size 1024 on 8 Nvidia Tesla V100 GPUs, which takes about 3 days with our PyTorch (Paszke et al. 2019) implementation. The network parameters are optimized using an SGD optimizer with an initial learning rate of 0.5 (decayed linearly after each epoch), a momentum of 0.9, and a weight decay of 3×10^{-5}. Additional enhancements, including label smoothing (Szegedy et al. 2016) and auxiliary loss tower, are applied during training. Since large batch size and learning rate are adopted, we apply learning rate warmup (Goyal et al. 2017) for the first 5 epochs.

Our evaluation results and the comparison with state-of-the-art approaches are summarized in Table 3. The architecture transferred from CIFAR10 outperforms DARTS, PC-DARTS and SNAS by a large margin in terms of classification performance, which demonstrates superior transfer capability of the discovered architectures. Notably, architectures discovered by P-DARTS on CIFAR10 achieve lower test error than MnasNet (Tan et al. 2019) and Proxyless-NAS (Cai et al. 2019), whose search space is modified from the MobileNet-v2 architecture which was originally designed for ImageNet and, as far as we know, very rarely used for CIFAR experiments. The architecture directly searched on ImageNet achieves superior performance compared to those transferred ones (+0.3% top-1 accuracy gain) and is comparable to the state-of-the-art directly-searched models in the DARTS-based search space (Xu et al. 2020).

For better reference, we also put some recent work, MobileNet-v3 (Howard et al. 2019), RandWire-WS (Xie et al. 2019a) and EfficientNet-B0 (Tan and Le 2019), into Table 3. However, due to large difference in network configuration (MobileNet-v3 applied multiple additional components like the H-swish activation function to improve the capability, RandWire-WS adopted a quite different search space, and EfficientNet-B0 adopted multiple techniques including swish activation function, SE module, AutoAugment (Cubuk et al. 2018) data augmentation, etc. to enhance its performance[4]), the comparison between these two methods and others is less informative. Moreover, EfficientNet

[4] Individually, swish activation function reduced the top-1 test error of NASNet-A from 26.4% to 25.0%(Ramachandran et al. 2017), SE module brought an performance gain of 0.7% (from 25.5% to 24.8%) on MnasNet (Tan et al. 2019), and AutoAugment achieved an accuracy gain of 1.3% on ResNet-50 (Cubuk et al. 2018). With swish activation function, SE module and AutoAugment, the compound gain is 2.5% (from 25.2% of MnasNet-92 to 22.7% of EfficientNet-B0.)

Table 4 Detection results on the MS-COCO dataset (test-dev). The displayed FLOPs only includes the computations in the network backbone. † denotes the results in this line are from (Duan et al. 2019); ‡: this model is equipped with BiFPN

Network	Input Size	Backbone	×+	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
SSD300 (Liu et al. 2016)	300 × 300	VGG-16	31.4B	23.2	41.2	23.4	5.3	23.2	39.6
SSD512 (Liu et al. 2016)	512 × 512	VGG-16	80.4B	26.8	46.5	27.8	9.0	28.9	41.9
SSD513 (Liu et al. 2016)†	513 × 513	ResNet-101	43.4B	31.2	50.4	33.3	10.2	34.5	49.8
SSDLiteV1 (Howard et al. 2017)	320 × 320	MobileNetV1	1.2B	22.2	–	–	–	–	–
SSDLiteV2 (Sandler et al. 2018)	320 × 320	MobileNetV2	0.7B	22.1	–	–	–	–	–
SSDLiteV3 (Tan et al. 2019)	320 × 320	MnasNet-A1	0.6B	23.0	–	–	3.6	20.5	43.2
SSD320 (Liu et al. 2016)	320 × 320	PC-DARTS (ImageNet)	1.2B	28.9	46.9	30.0	7.9	32.0	48.3
EfficientDet-D0 (Tan et al. 2020)‡	512 × 512	EfficientNet-B0	2.5B	34.6	53.0	37.1	-	-	-
SSD320 (Liu et al. 2016)	320 × 320	DARTS	1.1B	27.3	45.0	28.3	7.6	30.2	46.0
SSD320 (Liu et al. 2016)	320 × 320	P-DARTS (CIFAR10)	1.1B	28.9	46.8	30.2	7.3	32.2	48.2
SSD320 (Liu et al. 2016)	320 × 320	P-DARTS (ImageNet)	1.2B	29.9	47.8	31.5	9.0	33.2	50.0
SSD512 (Liu et al. 2016)	512 × 512	DARTS	2.9B	31.8	50.3	33.8	11.7	37.1	49.7
SSD512 (Liu et al. 2016)	512 × 512	P-DARTS (CIFAR10)	2.9B	33.6	52.8	35.6	13.3	39.7	51.1
SSD512 (Liu et al. 2016)	512 × 512	P-DARTS (ImageNet)	3.1B	34.1	52.9	36.3	14.3	40.0	52.1

offers a way of building large models with compound scaling. As our work has addressed the depth aspect to build a deep one, there is little option in the search space we adopt, while building a larger model may require to adjust multiple options simultaneously as EfficientNet did. While some recent works showed that a large model did bring higher performance (Real et al. 2018; Liu et al. 2018a; Zoph et al. 2018), we are not able to try it since massive computations are required.

5.4 Evaluation in the Wild

To further test the transferability of the discovered architectures to scenarios in the wild, we embed our discovered architectures as backbones into two other vision tasks, i.e., object detection and person re-identification. On both tasks, we have observed superior performance compared to both baseline methods and the DARTS backbone, which reveals that the desirable characters obtained on image recognition by P-DARTS can be well transferred to scenarios in the wild.

5.4.1 Transferring to Object Detection

Object detection is also a fundamental task in the vision community and also an important task of the scenario in the wild (Liu et al. 2019b). We plug the discovered architectures and corresponding weights pre-trained on ImageNet into Single-Shot Detectors (SSD) (Liu et al. 2016), a popular light-weight object detection framework. The capability of our backbones is tested on the benchmark dataset MS-COCO (Lin et al. 2014), which contains 80 object categories and more than 1.5M object instances. We train the pipeline with the "trainval35K" set, i.e., a combination of the 80k training and 35k validation images. The performance is tested on the *test-dev* set.

Results are summarized in Table 4. Equipped with the P-DARTS backbone searched on CIFAR10, the P-DARTS-SSD320 achieves a superior AP of 28.9% with only 1.1B FLOPs, which is 5.7% higher in AP with 29× fewer FLOPs than SSD300, and even 2.1% higher in AP with 73× fewer FLOPS than the SSD512. With similar FLOPs, P-DARTS-SSD320 outperforms the DARTS-SSD320 by 1.6% in AP. Compared to those light-weight backbones, i.e., backbones belong to the MobileNet family, our P-DARTS-SSD320 enjoys a superior AP by a large margin, while with an acceptable amount of extra FLOPs than these light-weight backbones. With larger input image size, the P-DARTS-SSD512 surpasses the SSD513 by an AP of 2.4%, while the FLOPs count of the P-DARTS backbone is 14× smaller than its ResNet-101 backbone. The results with the backbone searched by P-DARTS on ImageNet are further impressive, which achieves an AP of 29.9% with the backbone searched on CIFAR10 for P-DARTS-SSD320, and 34.1% for P-DARTS-SSD512. With a comparable amount of FLOPs, our P-DARTS-SSD320 (ImageNet) outperforms the PC-DARTS-SSD320 (ImageNet), the previous most powerful one with DARTS-based backbone, by a significant margin of 1.0% on AP. These results indicate that the discovered architectures by P-DARTS are well transferred to object detection and produce superior performance.

There also exist works that applied NAS approaches to improving two-stage object detection, e.g., based on FPN (Lin et al. 2017). For example, DetNAS (Chen et al. 2019b) achieved an AP of 36.6%, outperforming 33.9%

Table 5 Results of person re-identification on Market-1501, DukeMCMT-reID and MSMT17

Backbone	# Parts	Feat. Dim	×+ (M)	Market-1501		DukeMCMT-reID		MSMT17	
				Rank-1	mAP	Rank-1	mAP	Rank-1	mAP
Auto-ReID (Quan et al. 2019)	–	–	–	94.5	**85.1**	–	–	78.2	52.5
ResNet-50	1	2,048	4,120	87.9	72.8	72.0	57.2	48.3	24.6
DARTS	1	768	573	91.9	79.3	82.1	66.7	61.5	37.9
P-DARTS (CIFAR10)	1	768	556	93.0	81.4	83.8	68.7	67.0	42.0
P-DARTS (ImageNet)	1	768	596	92.0	78.4	83.6	67.8	66.7	39.6
ResNet-50	3	2,048	4,120	92.8	80.3	84.6	71.5	71.6	46.2
DARTS	3	768	573	94.2	83.6	86.2	74.7	77.4	53.0
P-DARTS (CIFAR10)	3	768	556	**94.6**	84.8	87.3	**75.5**	**79.5**	56.0
P-DARTS (ImageNet)	3	768	596	93.7	83.9	**90.0**	75.3	77.2	53.4
ResNet-50	6	2,048	4,120	93.1	81.0	86.1	74.0	71.1	46.0
DARTS	6	768	573	93.4	83.2	86.4	74.2	77.1	53.8
P-DARTS (CIFAR10)	6	768	556	93.6	83.4	87.3	74.6	79.2	**56.4**
P-DARTS (ImageNet)	6	768	596	93.0	83.0	86.7	74.0	76.8	53.6

Bold indicates the best one
The displayed FLOPs only includes the computations in the network backbone

reported by ResNet50 that has 10× more FLOPs than the backbone of DetNAS. NAS-FPN (Ghiasi et al. 2019) used a more powerful backbone, AmoebaNet (Real et al. 2018), and reported an even higher detection accuracy. EfficientDet (Tan et al. 2020) adopted state-of-the-art backbones, the EfficientNet series, and equipped a more competitive feature fusion network, BiFPN (which is reported to improve the detection AP by a margin of up to 2% compared to a normal FPN (Tian et al. 2020)), result with sate-of-the-art detection performance. Deploying DARTS-based backbones onto FPN or BiFPN requires non-trivial efforts, since the number of stages for feature extraction does not match. We will explore this possibility with a newly designed search space.

5.4.2 Transferring to Person Re-Identification

Person re-identification is an important practical vision task and has been attracting more attention from both academia and industry (Wang et al. 2018; Li et al. 2018) because of its broad applications in surveillance and security. Apart from those task-specific modules, the backbone architecture is a critical factor for performance promotion. We replace the previous backbones with our P-DARTS architectures (searched on both CIFAR10 and ImageNet) and test the transferability on three benchmark datasets, i.e., *Market-1501* (Zheng et al. 2015), *DukeMTMC-reID* (Zheng et al. 2017) and *MSMT17* (Wei et al. 2018). Experiments are executed with the pipeline of Part-based Convolutional Baseline (PCB) (Sun et al. 2018), and all backbones are pre-trained on ImageNet. We set the numbers of parts to be 1, 3, and 6 to make an exhaustive comparison.

(a) ImageNet **(b)** MS-COCO **(c)** CIFAR10 **(d)** Market-1501

Fig. 4 Samples from ImageNet, MS-COCO, CIFAR10 and Market-1501. Please zoom in to see clearer

Results are summarized in Table 5. The P-DARTS backbones outperform the ResNet-50 backbone by a large margin with fewer FLOPs and a smaller feature dimensionality. With a similar backbone size, P-DARTS (CIFAR10) surpasses DARTS on all three datasets with different part numbers, suggesting a superior transferability of our searched architecture. However, with the P-DARTS (ImageNet) backbone, the per-

formance is only comparable to the DARTS backbone and worse than the P-DARTS (CIFAR10) backbone. Additionally, we add the NAS-based Auto-ReID (Quan et al. 2019) into comparison. Compared to the DARTS and P-DARTS models, the models of Auto-ReID have different network components, model size, etc., making it unfair to compare these two set of models. Thus, we only list it here as reference.

It is worth noting that the preferences to CIFAR-searched and ImageNet-searched backbones are different between object detection and person re-identification tasks. This is due to the domain gap between the architecture search task and the target tasks. While the original images used in ImageNet classification and COCO object detection are similarly with high resolution and data distribution, images used in ReID are in worse condition, which is more similar to the situation in CIFAR10. We showcase in Figure 4 samples from ImageNet, COCO, CIFAR10, and Market-1501, where the domain gap between them can be visually observed. A notable phenomenon is that with the ResNet-50 backbone, performance keeps rising when increasing the part number, while the best performance reaches to the peak when the part number is 3 with both DARTS and P-DARTS backbones. This is arguably because of the larger feature dimensionality adopted in ResNet-50 backbone, which also implies the potential of further performance promotion on P-DARTS backbones with a larger feature dimensionality and part number. The domain gap seems to be the critical difficulty in the future of *NAS in the wild*.

6 Conclusions

In this work, we propose a progressive version of differentiable architecture search to bridge the optimization gap between search and evaluation scenarios for NAS in the wild. The core idea, based on that optimization gap is caused by the difference between the policies of search and evaluation, is to gradually increase the depth of the super-network during the search process. To alleviate the issues of computational overhead and instability, we design two practical techniques to approximate and regularize the search process, respectively. Our approach reports superior performance in both proxy datasets (CIFAR and ImageNet) and target datasets (object detection and person re-identification added) with significantly reduced search overheads.

Our research puts forward the optimization gap in super-network-based NAS and highlights the significance of the consistency between search and evaluation scenarios. To solve it in terms of network depth and width, the P-DARTS algorithm paves a new way by approximating the search space. We expect that our initial work serves as a modest spur to induce more researchers to contribute their ideas to further alleviate the optimization gap and design effective and generalized NAS algorithms.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China under Grant Nos. 61831018, and 61631017, and Guangdong Province Key Research and Development Program Major Science and Technology Projects under Grant 2018B010115002.

References

Baker, B., Gupta, O., Naik, N., & Raskar, R. (2017). Designing neural network architectures using reinforcement learning. In *ICLR*.

Bi, K., Hu, C., Xie, L., Chen, X., Wei, L., & Tian, Q. (2019). Stabilizing darts with amended gradient estimation on architectural parameters. arXiv:1910.11831.

Cai, H., Chen, T., Zhang, W., Yu, Y., & Wang, J. (2018). Efficient architecture search by network transformation. In *AAAI*.

Cai, H., Zhu, L., & Han, S. (2019). ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*.

Chen, X., Xie, L., Wu, J., & Tian, Q. (2019a). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*.

Chen, Y., Yang, T., Zhang, X., Meng, G., Xiao, X., & Sun, J. (2019b). Detnas: Backbone search for object detection. In *NeurIPS*.

Chu, X., Zhang, B., Xu, R., & Li, J. (2019). Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. arXiv:1907.01845.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data. arXiv:1805.09501.

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *CVPR*.

DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv:1708.04552.

Dong, X., & Yang, Y. (2019a). One-shot neural architecture search via self-evaluated template network. In *ICCV*.

Dong, X., & Yang, Y. (2019b). Searching for a robust neural architecture in four gpu hours. In *CVPR*.

Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., & Tian, Q. (2019). Centernet: Keypoint triplets for object detection. In *ICCV*.

Elsken, T., Metzen, J. H., & Hutter, F. (2018). Neural architecture search: A survey. arXiv:1808.05377.

Ghiasi, G., Lin, T. Y., & Le, Q. V. (2019). Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., & He, K. (2017). Accurate, large minibatch SGD: Training ImageNet in 1 hour. arXiv:1706.02677.

Han, D., Kim, J., & Kim, J. (2017). Deep pyramidal residual networks. In *CVPR*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. (2019) Searching for mobilenetv3. In *ICCV*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.

Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *ECCV*, Springer.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *CVPR*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*.
Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. Tech. rep., Citeseer.
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *NIPS*.
Larsson, G., Maire, M., & Shakhnarovich, G. (2017). FractalNet: Ultra-deep neural networks without residuals. In *ICLR*.
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.
Li, J., Ma, A. J., & Yuen, P. C. (2018). Semi-supervised region metric learning for person re-identification. *IJCV*, *126*(8), 855–874.
Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *ECCV*.
Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In *CVPR*.
Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L. J., Fei-Fei, L., Yuille, A., Huang, J., & Murphy, K. (2018a). Progressive neural architecture search. In *ECCV*.
Liu, H., Simonyan, K., Vinyals, O., Fernando, C., & Kavukcuoglu, K. (2018b). Hierarchical representations for efficient architecture search. In *ICLR*.
Liu, H., Simonyan, K., & Yang, Y. (2019a). DARTS: Differentiable architecture search. In *ICLR*.
Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2019b). Deep learning for generic object detection: A survey. In *IJCV*.
Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *ECCV*.
Ma, N., Zhang, X., Zheng, H. T., & Sun, J. (2018). ShuffleNet V2: Practical guidelines for efficient cnn architecture design. In *ECCV*.
Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*.
Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., & Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *ICML*.
Quan, R., Dong, X., Wu, Y., Zhu, L., & Yang, Y. (2019). Auto-reid: Searching for a part-aware convnet for person re-identification. In *ICCV*.
Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. arXiv:1710.05941.
Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2018). Regularized evolution for image classifier architecture search. arXiv:1802.01548.
Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). ImageNet large scale visual recognition challenge. *IJCV*, *115*(3), 211–252.
Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*.
Shu, Y., Wang, W., & Cai, S. (2020). Understanding architectures learnt by cell-based neural architecture search. In *ICLR*.
Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR*.
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, *15*(1), 1929–1958.
Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Training very deep networks. In *NIPS*.
Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*(2), 99–127.
Suganuma, M., Shirakawa, S., & Nagao, T. (2017). A genetic programming approach to designing convolutional neural network architectures. In *GECCO*.
Sun, Y., Zheng, L., Yang, Y., Tian, Q., & Wang, S. (2018). Beyond part models: Person retrieval with refined part pooling (and a strong convolutional baseline). In *ECCV*.
Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In *CVPR*.
Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *CVPR*.
Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*.
Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*.
Tan, M., Pang, R., & Le, Q. V. (2020). Efficientdet: Scalable and efficient object detection. In *CVPR*.
Tian, Z., Shen, C., Chen, H., & He, T. (2020). Fcos: A simple and strong anchor-free object detector. arXiv:2006.09214.
Wang, H., Zhu, X., Gong, S., & Xiang, T. (2018). Person re-identification in identity regression space. *IJCV*, *126*(12), 1288–1310.
Wei, L., Zhang, S., Gao, W., & Tian, Q. (2018). Person transfer gan to bridge domain gap for person re-identification. In *CVPR*.
Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., & Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*.
Xie, L., & Yuille, A. (2017). Genetic CNN. In *ICCV*.
Xie, S., Kirillov, A., Girshick, R., & He, K. (2019a). Exploring randomly wired neural networks for image recognition. In *ICCV*.
Xie, S., Zheng, H., Liu, C., & Lin, L. (2019b). SNAS: Stochastic neural architecture search. In *ICLR*.
Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G. J., Tian, Q., & Xiong, H. (2020). PC-DARTS: Partial channel connections for memory-efficient architecture search. In *ICLR*.
Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. arXiv:1605.07146.
Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., & Hutter, F. (2020). Understanding and robustifying differentiable architecture search. In *ICLR*.
Zhang, X., Zhou, X., Lin, M., Sun, J. (2018). ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*.
Zheng, L., Shen, L., Tian, L., Wang, S., Wang, J., Tian, Q. (2015). Scalable person re-identification: A benchmark. In *ICCV*.
Zheng, X., Ji, R., Tang, L., Zhang, B., Liu, J., & Tian, Q. (2019). Multinomial distribution learning for effective neural architecture search. In *ICCV*.
Zheng, Z., Zheng, L., & Yang, Y. (2017). Unlabeled samples generated by gan improve the person re-identification baseline in vitro. In *ICCV*.
Zhong, Z., Zheng, L., Kang, G., Li, S., & Yang, Y. (2017) Random erasing data augmentation. arXiv:1708.04896.
Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *ICLR*.
Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *CVPR*.