Progressive Differentiable Architecture Search: Bridging the Depth Gap between Search and Evaluation

Xin Chen[1]* Lingxi Xie[2] Jun Wu[1] Qi Tian[2]

[1]Tongji University [2]Huawei Noah's Ark Lab

1410452@tongji.edu.cn 198808xc@gmail.com wujun@tongji.edu.cn tian.qi1@huawei.com

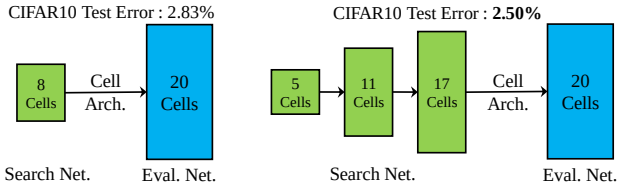
Abstract

Recently, differentiable search methods have made major progress in reducing the computational costs of neural architecture search. However, these approaches often report lower accuracy in evaluating the searched architecture or transferring it to another dataset. This is arguably due to the large gap between the architecture depths in search and evaluation scenarios. In this paper, we present an efficient algorithm which allows the depth of searched architectures to grow gradually during the training procedure. This brings two issues, namely, heavier computational overheads and weaker search stability, which we solve using search space approximation and regularization, respectively. With a significantly reduced search time (∼7 hours on a single GPU), our approach achieves state-of-the-art performance on both the proxy dataset (CIFAR10 or CIFAR100) and the target dataset (ImageNet). Code is available at `https://github.com/chenxin061/pdarts`.

1. Introduction

Image recognition is a fundamental task in the computer vision community. In the deep learning era, state-of-the-art classification performance is mostly achieved by handcrafted deep neural networks. Recently, the development of neural architecture search (NAS) has changed the convention of model design from manual to automatic, achieving remarkable success in various perceptual tasks [15, 3, 36] including image recognition [37].

Early works on NAS focused on the optimal configuration of layer type, filter size and number, activation function, *etc.*, to construct a complete network [1, 28]. Inspired by successful handcrafted architectures such as ResNet [8] and DenseNet [10], follow-up works started to explore the possibility of searching for network building blocks, or so-called cells with reinforcement learning (RL) [35, 37] and evolutionary algorithm (EA) [32, 22]. The discovered cells are then stacked orderly to construct the network for specific tasks. However, those RL-based and EA-based approaches share a common pipeline to sample and evaluate (from scratch) numerous architectures in the search space, which results in a barely affordable computational overhead, *e.g.*, hundreds or even thousands of GPU-days.



(a) DARTS (b) P-DARTS

Figure 1: Difference between DARTS and P-DARTS (our approach), with the former searching architectures in a shallow setting and evaluating them in a deep one, and the latter progressively increasing the searching depth, so as to bridge the *depth gap* between search and evaluation. Green and blue indicate search and evaluation, respectively.

Recently, Liu *et al.* proposed a differentiable scheme called DARTS [18] to get rid of the time-consuming process of architecture sampling and evaluating. It achieved comparable performance to RL-based and EA-based methods while only requiring a search cost of a few GPU-days. In DARTS, a cell is composed of multiple nodes, connected with several kinds of operations, *e.g.*, *convolution*, *pooling*. Those operations are weighted by a few architecture parameters, which are learned in the search scenario. Limited by the size of GPU memory, DARTS has to search the archi-

*This work was done when the first author was an intern at Huawei Noahs Ark Lab.

tecture in a shallow network while evaluate in a deeper one. This brings an issue named the *depth gap* (see Figure 1(a)), which means that the search stage finds some operations that work well in a shallow architecture, but the evaluation stage actually prefers other operations that fit a deep architecture better. Such gap hinders these approaches in their application to more complex visual recognition tasks.

In this work, we propose Progressive DARTS (P-DARTS), a novel and efficient algorithm to bridge the depth gap. As shown in Figure 1(b), we divide the search process into multiple stages and progressively increase the network depth at the end of each stage. While a deeper architecture requires heavier computational overhead, we propose *search space approximation* which, as the depth increases, reduces the number of candidates (operations) according to their scores in the elapsed search process. Another issue, lack of stability, emerges with searching over a deep architecture, in which the algorithm can be biased heavily towards *skip-connect* as it often leads to rapidest error decay during optimization, but, actually, a better option often resides in learnable operations such as *convolution*. To avoid this, we propose *search space regularization*, which (i) introduces operation-level Dropout [25] to alleviate the dominance of *skip-connect* during training, and (ii) controls the appearance of *skip-connect* during evaluation.

The effectiveness of P-DARTS is verified on the standard vision setting, *i.e.*, searching on CIFAR10, and evaluating on both CIFAR10 and ImageNet. We achieve state-of-the-art performance (a test error of 2.50%) on CIFAR10 with 3.4M parameters. When transferred to ImageNet, it achieves top-1/5 errors of 24.4%/7.4%, respectively, comparable to the state-of-the-art under the mobile setting. We further demonstrate the benefits of search space approximation and regularization: the former reduces the search time over CIFAR10 to 0.3 GPU-days which, to the best of our knowledge, is the fastest to date to achieve an error rate of 3% in CIFAR10, even surpassing ENAS [21], an approach specialized in efficiency; the latter makes it easy to apply P-DARTS to other proxy datasets, which we show an example on CIFAR100 (15.92% test error, 3.6M parameters).

2. Related Work

Image recognition is a fundamental task in computer vision. Recent years, with the development of deep learning, convolutional neural networks (CNNs) have been dominating image recognition [13]. A few handcrafted architectures have been proposed, including VGGNet [24], ResNet [8], DenseNet [10], *etc.*, all of which verified the importance of human experts in network design.

Our work belongs to the emerging field of neural architecture search (NAS), a process of automating architecture engineering technique [6]. Pioneer researchers started to explore the possibility of automatically generating better topology with evolutionary algorithms in the 2000's [27]. Early NAS works tried to search for a complete network topology [1, 28] while recent works focused on finding robust cells [32, 37, 22]. Lately, EA-based [22] and RL-based [37] NAS approaches achieved state-of-the-art performance in image recognition, where architectures were sampled and evaluated from the search space under the guidance of an EA-based or RL-based meta-controller. A notable drawback of the above approaches is the expensive computational overhead (3,150 GPU-days for EA-based AmoebaNet [22] and 1,800 GPU-days for RL-based NASNet [37]). PNAS proposed to learn a surrogate model to guide the search through the structure space, achieving 5× speedup than NASNet. ENAS [21] proposed to share parameters among child models to prevent evaluating candidate architectures by training them from scratch, which significantly reduced the search cost to less than one GPU-day.

DARTS [18] introduced a differentiable NAS framework, which achieved remarkable performance and efficiency improvement. Following DARTS, SNAS [33] proposed to constrain the architecture parameters to be one-hot to tackle the inconsistency in optimizing objectives between search and evaluation scenarios. ProxylessNAS [2] adopted the differentiable framework and proposed to search architectures on the target task instead of adopting the conventional proxy-based framework.

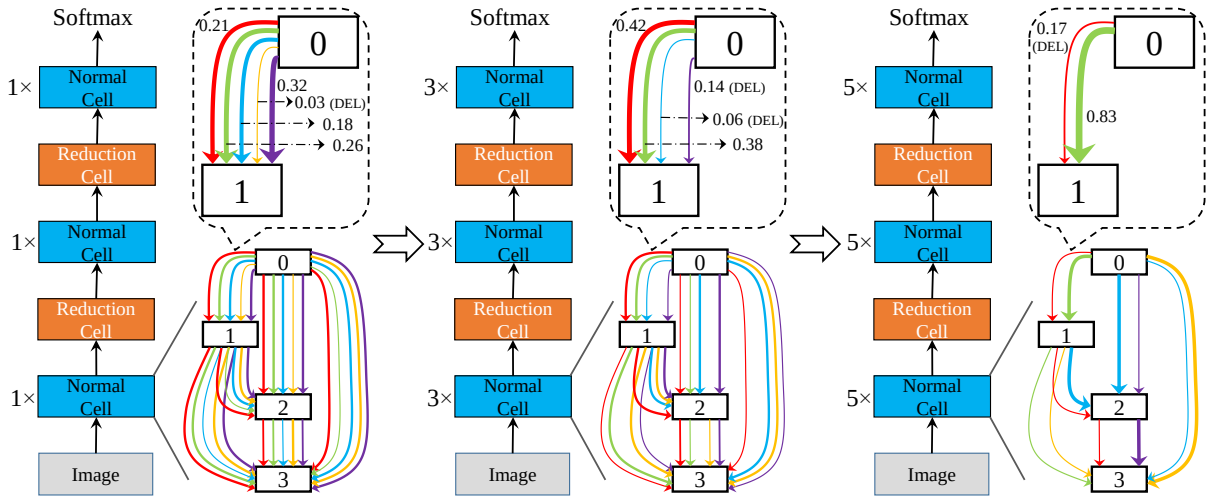
3. Method

3.1. Preliminary: DARTS

In this work, we leverage DARTS [18] as our baseline framework. Our goal is to search for a robust cell and apply it to a network of L cells. A cell is defined as a directed acyclic graph (DAG) of N nodes, $\{x_0, x_1, \cdots, x_{N-1}\}$, where each node is a network layer, *i.e.*, performing a specific mathematical function. We denote the operation space as $\mathcal{O}$, in which each element represents a candidate function $o(\cdot)$. An edge $\mathrm{E}_{(i,j)}$ represents the information flow connecting node i and node j, which consists of a set of operations weighted by the architecture parameters $\alpha^{(i,j)}$, and is thus formulated as:

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

where $i < j$ so that *skip-connect* can be applied. An intermediate node can be represented as $x_j = \sum_{i<j} f_{i,j}(x_i)$, and the output node is $x_{N-1} = \mathrm{concat}(x_2, x_3, \cdots, x_{N-2})$, where $\mathrm{concat}(\cdot)$ concatenates all input signals in the channel dimension. For more technical details, please refer to the original DARTS paper [18].



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

Figure 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 search 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 3 and 2 accordingly. 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.

3.2. Progressively Increasing the Searching Depth

In DARTS, architecture search is performed on a network of 8 cells while the discovered architecture is evaluated on a network of 20 cells. However, there is a big difference between the behaviors of shallow and deep networks [11, 26, 8], which implies that the structures we prefer in the search process are not necessarily the optimal 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. This is caused by that shallow networks often enjoy faster gradient descent during the search process, but contradicts the common sense that deeper networks tend to perform better [24, 29, 8, 10]. Therefore, we propose to bridge the depth gap, and we take the strategy that progressively increases the network depth during the search process, so that at the end of search, the depth is sufficiently close to the setting used in evaluation. Here we prefer a progressive manner, rather than directly increasing the depth to the target level, because we expect search in shallow networks to reduce the search space with respect to the candidate operations, so as to alleviate the risk of search in deep networks. We will verify the effectiveness of this progressive strategy in Section 4.4.1.

Difficulty comes from two aspects. First, the number of structures increases exponentially with the depth, which brings issues in both time and memory. In particular, in DARTS, GPU memory usage is proportional to the depth of searched networks. The limited GPU memory forms a major obstacle, and the most straightforward solution is to reduce the number of channels in each operation – DARTS [18] tried it but reported a slight performance deterioration. 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, which refers to the scores of operations in the previous stage as the criterion of selection. Details of search space approximation are presented in Section 3.2.1.

Second, we find that when searching on a deeper architecture, the differentiable approaches tend to bias towards the *skip-connect* operation, because it accelerates forward/backward propagation and often leads to the fastest way of gradient descent. However, since such an operation is parameter-free, its ability of learning visual representations is relatively weak. To this end, we propose another scheme named **search space regularization**, which adds operation-level Dropout [25] 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 Section 3.2.2.

3.2.1 Search Space Approximation

The idea of search space approximation is shown as the toy example in Figure 2. The 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 search network consists of L_k cells and the size of the operation space is O_k, *i.e.*, $|\mathcal{O}^k_{(i,j)}| = O_k$.

According to our motivation, at the initial stage, the search network 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 architecture parameters α_{k-1} are learned and the scores of the candidate operations on each connection are ranked according to α_{k-1}. We increase the depth of the searched architecture by stacking more cells, *i.e.*, $L_k > L_{k-1}$, and approximate the operation space in the meantime. This is to say, the new operation set $\mathcal{O}^k_{(i,j)}$ 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, which are defined to be those assigned with a lower weight during the previous stage, $\mathfrak{S}_{k-1}$. As shown in Table 3, this strategy is memory efficient, which makes our approach easy to be deployed on regular GPUs, *e.g.*, with a memory of 16GB.

This process of increasing architecture depth continues until it is sufficiently close to that used in evaluation. After the last search stage, we determine the final cell topology (bold lines in Figure 2(c)) according to the learned architecture parameters α_K. Following DARTS, we keep two top-weighted non-*zero* operations (at most 1 for a distinct edge) for each intermediate node.

3.2.2 Search Space Regularization

At the start of each stage, $\mathfrak{S}_k$, we train the (modified) architecture from scratch, *i.e.*, all network weights are initialized, because several candidates have been dropped[1]. However, training a deeper network is harder than training a shallow one [26]. 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 reason that *skip-connect* often leads to rapid gradient descent, especially on the proxy datasets (CIFAR10 or CIFAR100) which are relatively small and easy to fit. Consequently, the search process tends to generate architectures with many *skip-connect* operations, which limits the number of learnable parameters and thus produces unsatisfying performance at the evaluation stage. This is essentially a kind of over-fitting.

[1]We also tried to start with network parameters learned from the last stage, $\mathfrak{S}_{k-1}$, and adjust the architecture weights accordingly, *i.e.*, the weights of preserved operations should still sum to one, and each weight is proportional to the corresponding value at the end of $\mathfrak{S}_{k-1}$. This strategy reported lower accuracy, because the prior from the previous stage guided the algorithm towards an architecture suitable for a shallow network instead of a deep one.

We address this problem by search space regularization, which consists of two parts. First, we insert operation-level Dropout [25] after each *skip-connect* operation, so as 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 drop them by assigning low 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 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 search processes on CIFAR10 with exactly the same 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, as we observed before, the recognition performance at the evaluation stage is also highly correlated to this number. This motivates us to design the second regularization rule, architecture refinement, which simply controls the number of preserved *skip-connects*, after the final search stage, to be a constant M. This is done with an iterative process, which starts with constructing a cell topology using the standard algorithm 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 architecture parameters. Note that this may bring up other *skip-connects* to the topology, so we repeat this procedure until the desired number is achieved.

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 architecture weights, based on which we could not build up a powerful architecture even with a fixed number of *skip-connects*.

3.3. Relationship to Prior Work

PNAS [16] explored the search space progressively by searching for operations node-by-node within each cell. Our approach has a similar search manner but comes from a different motivation. In addition, we perform the progressive 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.

SNAS [33] aimed at eliminating the bias between the search and evaluation objectives of differentiable NAS approaches by forcing the architecture weights on each edge

Architecture	Test Err. (%)		Params	Search Cost	Search Method
	C10	C100	(M)	(GPU-days)	
DenseNet-BC [10]	3.46	17.18	25.6	-	manual
NASNet-A + cutout [37]	2.65	-	3.3	1800	RL
AmoebaNet-A + cutout [22]	3.34	-	3.2	3150	evolution
AmoebaNet-B + cutout [22]	2.55	-	2.8	3150	evolution
Hireachical Evolution [17]	3.75	-	15.7	300	evolution
PNAS [16]	3.41	-	3.2	225	SMBO
ENAS + cutout [21]	2.89	-	4.6	0.5	RL
DARTS (first order) + cutout [18]	3.00	17.76†	3.3	1.5‡	gradient-based
DARTS (second order) + cutout [18]	2.76	17.54†	3.3	4.0‡	gradient-based
SNAS + mild constraint + cutout [33]	2.98	-	2.9	1.5	gradient-based
SNAS + moderate constraint + cutout [33]	2.85	-	2.8	1.5	gradient-based
SNAS + aggressive constraint + cutout [33]	3.10	-	2.3	1.5	gradient-based
ProxylessNAS [2] + cutout	2.08	-	5.7	4.0	gradient-based
P-DARTS CIFAR10 + cutout	2.50	16.55	3.4	0.3	gradient-based
P-DARTS CIFAR100 + cutout	2.62	15.92	3.6	0.3	gradient-based
P-DARTS CIFAR10 (large) + cutout	2.25	15.27	10.5	0.3	gradient-based
P-DARTS CIFAR100 (large) + cutout	2.43	14.64	11.0	0.3	gradient-based

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

to be one-hot. Our work is also able to get rid of the bias, which we investigate from enlarging the architecture depth.

ProxylessNAS [2] 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 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 GPU-days on CIFAR10).

4. Experiments

4.1. Datasets

We conduct experiments on three popular image classification datasets, including CIFAR10, CIFAR100 [12] and ImageNet [4]. Architecture search is performed on CIFAR10 and CIFAR100, and the discovered architectures are evaluated on all three datasets.

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

We use ILSVRC2012 [23] to test the transferability of the architectures discovered on CIFAR10 and CIFAR100. ILSVRC2012 is a subset of ImageNet [4] which contains 1,000 object categories and 1.28M training and 50K validation images. Following the conventions [37, 18], we apply the mobile setting where the input image size is 224×224 and the number of multi-add operations is restricted to be less than 600M.

4.2. Architecture Search

4.2.1 Implementation Details

The whole search process consists of 3 stages. Since we adopt DARTS as the backbone framework, the search space and network configuration are the same as DARTS at the initial stage (stage 1) except that the number of cells is set to be 5 (this is for acceleration – we tried the original setting and obtained similar results). The number of cells increases from 5 to 11 for the intermediate stage (stage 2) and 17 for the final stage (stage 3). Meanwhile, the size of operation space is set to be 8, 5 and 3 at stage 1, 2 and 3, respectively.

The initial Dropout probability on *skip-connect* for the reported results is set to be 0.0, 0.4, 0.7 on CIFAR10 for

Architecture	Test Err. (%)		Params	×+	Search Cost	Search Method
	top-1	top-5	(M)	(M)	(GPU-days)	
Inception-v1 [29]	30.2	10.1	6.6	1448	-	manual
MobileNet [9]	29.4	10.5	4.2	569	-	manual
ShuffleNet 2× (v1) [34]	26.4	10.2	∼5	524	-	manual
ShuffleNet 2× (v2) [19]	25.1	-	∼5	591	-	manual
NASNet-A [37]	26.0	8.4	5.3	564	1800	RL
NASNet-B [37]	27.2	8.7	5.3	488	1800	RL
NASNet-C [37]	27.5	9.0	4.9	558	1800	RL
AmoebaNet-A [22]	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B [22]	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C [22]	24.3	7.6	6.4	570	3150	evolution
PNAS [16]	25.8	8.1	5.1	588	225	SMBO
MnasNet-92 [31]	25.2	8.0	4.4	388	-	RL
DARTS (second order) [18]	26.7	8.7	4.7	574	4.0	gradient-based
SNAS (mild constraint) [33]	27.3	9.2	4.3	522	1.5	gradient-based
ProxylessNAS (GPU) [2]	24.9	7.5	7.1	465	8.3	gradient-based
P-DARTS (searched on CIFAR10)	24.4	7.4	4.9	557	0.3	gradient-based
P-DARTS (searched on CIFAR100)	24.7	7.5	5.1	577	0.3	gradient-based

Table 2: Comparison with state-of-the-art architectures on ImageNet (mobile setting).

stage 1, 2 and 3, respectively, and 0.1, 0.2, 0.3 for CIFAR100. Considering the tradeoff between classification accuracy and computational overhead, the final discovered cells are restricted to keep at most 2 *skip-connect* operations. Such a setting also guarantees a fair comparison with DARTS and other state-of-the-art approaches. For each stage, we train the network using a batch size of 96 for 25 epochs, where in the first 10 epochs only network parameters are tuned while network parameters and architecture parameters are learned 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 architecture parameters. GPU memory related hyper-parameters are selected depending on the memory size of the GPU used in the experiments. For acceleration, we leverage the first order optimization scheme of DARTS to learn the architecture parameters.

4.2.2 Search Results

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, connections in the architecture discovered by P-DARTS cascade for more levels than DARTS, in other words, there are more layers in the cell, making the evaluation network further deeper and achieving better classification performance.

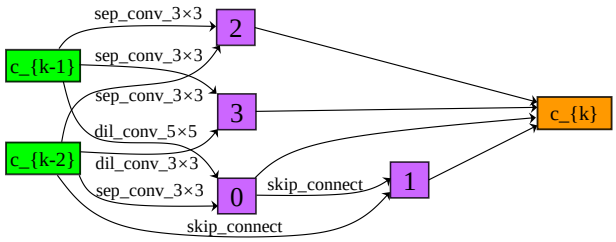
Notably, our method also allows architecture search on CIFAR100 while prior approaches mostly failed. The evaluation results in Table 1 show that the discovered architecture outperforms those transferred ones. We also perform architecture search on CIFAR100 with DARTS using the publicly available code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

4.3. Architecture Evaluation

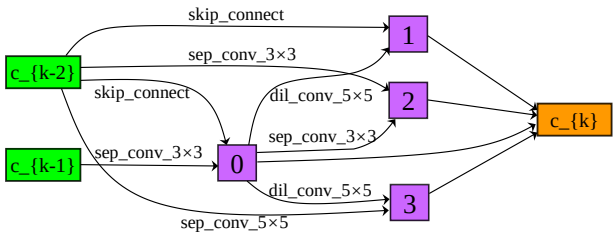
4.3.1 Evaluation on CIFAR10 and CIFAR100

An evaluation network of 20 cells and 36 initial channels is trained from scratch for 600 epochs with batch size 128. Cutout regularization [5] of length 16, drop-path [14] of probability 0.3 and auxiliary towers [29] of weight 0.4 are applied. A standard SGD optimizer with a weight decay of 0.0003 for CIFAR10 and 0.0005 for CIFAR100 and a momentum of 0.9 is used. The initial learning rate is 0.025, which is decayed to 0 following the cosine rule. We further increase the number of initial channels from 36 to 64 to explore the performance limitation of our searched architecture, which is denoted as the large setting.

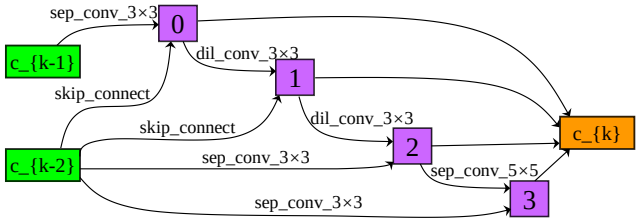
Evaluation results and comparison with state-of-the-art approaches are summarized in Table 1. As demonstrated in Table 1, P-DARTS achieves a 2.50% test error on CIFAR10 with a search cost of only 0.3 GPU-days. To obtain the same performance, AmoebaNet [22] spent four orders of magnitude more computational resources (0.3 GPU-day vs 3150 GPU-days). Our P-DARTS also outperforms DARTS and SNAS by a large margin. Notably, architectures discov-



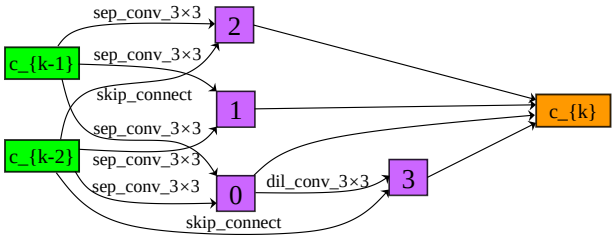
(a) Stage 1, CIFAR10 Test Err. 2.90%



(b) Stage 2, CIFAR10 Test Err. 2.82%



(c) Stage 3, CIFAR10 Test Err. 2.58%



(d) DARTS_V2, CIFAR10 Test Err. 2.83%

Figure 3: Normal cells discovered by different search stages of P-DARTS and second order DARTS (DARTS_V2). 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 $\mathrm{E}_{(0,1)}$ of stage 1 is a parameter-free *skip_connect*, thus it is strictly not a deep connection.

ered by P-DARTS outperform ENAS, the previously most efficient approach, in both classification performance and search cost, while with fewer parameters.

We transfer architectures discovered on CIFAR10 to CIFAR100 for both DARTS and P-DARTS. The evaluation results show the superiority of P-DARTS. Furthermore, we also conduct architecture search on CIFAR100. The discovered architecture outperforms DARTS on both CIFAR10 and CIFAR100. For a fair comparison, architecture search is also performed on CIFAR100 for DARTS with the publicly released code but get much higher evaluation test error. An interesting point is that architecture discovered on CIFAR10 outperforms that discovered on CIFAR100 when evaluated on CIFAR10, and vice versa. Such a phenomenon provides evidence to the existence of dataset bias in NAS.

4.3.2 Evaluation on ImageNet

The ILSVRC 2012 [23] is used to test the transferability of architectures discovered on CIFAR10 and CIFAR100. We adopt the same network configuration as DARTS, *i.e.*, an evaluation network of 14 cells and 48 initial channels. Each network is trained from scratch for 250 epochs with batch size 1024 on 8 Nvidia Tesla V100 GPUs, which takes 3 days with our PyTorch [20] 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 [30] and auxiliary loss tower are applied during training. Learning rate warmup [7] is applied for the first 5 epochs since large batch size and learning rate are adopted.

Evaluation results and comparison with state-of-the-art approaches are summarized in Table 2. Architectures discovered on CIFAR10 and CIFAR100 by P-DARTS outperform DARTS by a large margin in terms of classification performance, which demonstrates the transfer capability of the discovered architectures. Notably, P-DARTS achieves lower test error than MnasNet [31] and ProxylessNAS [2], whose search space is carefully designed for ImageNet.

4.4. Diagnostic Experiments

4.4.1 Comparison on the Depth of Search Networks

Since the search process of P-DARTS is split into multiple stages, we extract architectures from each search stage with the same rule. Architectures of different stages are evaluated to demonstrate their capability for image classification. The topology of discovered architectures (only normal cells are shown) and their corresponding performance are summarized in Figure 3. Additionally, we add the architecture discovered by second order DARTS (DARTS_V2, 8 cells in the search network) for comparison.

The architecture generated by stage 3 achieves the lowest test error among others, which validates the effectiveness of our scheme. From Figure 3 we can observe that

these architectures share some common edges, for example *sep_conv_3×3* at edge $\mathrm{E}_{(c_{k-2},2)}$ for stage 1, 2 and 3 and at edge $\mathrm{E}_{(c_{k-1},0)}$ for stage 2, 3 and DARTS_V2. Meanwhile, there are also differences between them, which may be the key factor that affects the capability of these architectures. Architectures generated by shallow search networks prefer to keep shallow connections, while with deeper search networks, the discovered architectures start to pick intermediate nodes as input for rear nodes, 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.

4.4.2 Effectiveness of Search Space Approximation

The search process takes ∼7 hours (0.3 days) on a single Nvidia Tesla P100 GPU with 16GB memory to produce the final architectures. We collect GPU memory usage data of architecture search process for 3 individual runs, which is shown in Table 3. The memory usage is stable and out of memory error barely occurs, showing the validity of the search space approximation scheme on memory efficiency.

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

Table 3: Peak GPU memory usage at different stages during three individual runs. The memory limit is 16GB.

We perform experiments to demonstrate the effectiveness on improving classification accuracy. We only perform the final stage of the search process on two different search spaces with the same setting. The first search space is approximated by the previous search stages and the other is randomly sampled from the full search space. To obtain a better result for the randomly sampled one, we repeat the whole process for 3 times with different seeds and pick the best one. The best performance for the randomly sampled search space is 3.43% test error, which is much worse than 2.58%, the one obtained by the approximated search space. Such results reveal the necessity of the search space approximation scheme.

4.4.3 Effectiveness of Search Space Regularization

We perform experiments to validate the effectiveness of search space regularization, *i.e.*, operation-level Dropout and architecture refinement. Firstly, experiments are conducted to test the effect of the operation-level Dropout scheme. Two sets of initial Dropout rates are tested, *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 searching with Dropout, the corresponding test errors are 2.69%, 2.84% and 2.97%, significantly outperforming those without Dropout. According to the experimental results, all 8 operations in the normal cell of the architecture discovered without Dropout are *skip-connects* before architecture refinement, while it is 4 for architecture discovered with Dropout. The reduction of *skip-connect* operations verifies the effectiveness of search space regularization on stabilizing the search process.

During experiments, we observe strong coincidence between the classification performance of an architecture and the number of *skip-connect* operations in it. We conduct a quantitative experiment to verify it. Architecture refinement is applied to one search process to produce multiple architectures where the number of preserved *skip-connect* operations 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 it, *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-connect* operations are replaced by operations with learnable parameters, *e.g.*, *convolution*, resulting in a more complex and powerful architecture.

The above observation offers an inspiration for the second kind of search space regularization, architecture refinement, whose capability is validated by the following experiments. We run another 3 architecture search experiments 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 ± 0.16% and the evaluated architectures are with 2, 3 and 4 *skip-connect* operations in normal cells. After architecture refinement, all three architectures are with 2 *skip-connect* operations in normal cells, resulting in a diminished test error of 2.65 ± 0.05%. The reduction of standard deviation reveals the improvement on the stability for the search process.

5. Conclusions

In this work, we propose a progressive version of differentiable architecture search to bridge the depth gap between search and evaluation scenarios. The core idea is to gradually increase the depth of candidate architectures 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 the fastest NAS speed to achieve an error rate of 3% on CIFAR10, meanwhile achieving superior performance in both the proxy dataset and the target dataset.

Our research defends the importance of depth in differentiable architecture search, paves a new way of approximation by sacrificing width, *i.e.*, the number of operations. We expect that in the future with more powerful computational resources, depth is still the dominant factor in exploring the architecture space.

References

[1] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

[2] H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[3] X. Chu, B. Zhang, H. Ma, R. Xu, J. Li, and Q. Li. Fast, accurate and lightweight super-resolution with neural architecture search. *arXiv preprint arXiv:1901.07261*, 2019.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[5] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[6] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[7] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

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

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[10] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[12] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[14] G. Larsson, M. Maire, and G. Shakhnarovich. FractalNet: Ultra-deep neural networks without residuals. In *ICLR*, 2017.

[15] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei. Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation. *arXiv preprint arXiv:1901.02985*, 2019.

[16] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[17] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018.

[18] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[19] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. ShuffleNet V2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in Pytorch. In *NIPS-W*, 2017.

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

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

[26] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *NIPS*, 2015.

[27] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[28] M. Suganuma, S. Shirakawa, and T. Nagao. A genetic programming approach to designing convolutional neural network architectures. In *GECCO*, 2017.

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[31] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. MnasNet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

[32] L. Xie and A. Yuille. Genetic CNN. In *ICCV*, 2017.

[33] S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: Stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[34] X. Zhang, X. Zhou, M. Lin, and J. Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

[35] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018.

[36] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[37] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.