arXiv:1907.05737v1 [cs.CV] 12 Jul 2019

PC-DARTS: Partial Channel Connections for Memory-Efficient Differentiable Architecture Search

Yuhui Xu[1*] **Lingxi Xie**[2] **Xiaopeng Zhang**[2] **Xin Chen**[3] **Guo-Jun Qi**[4]
Qi Tian[2] **Hongkai Xiong**[1]
[1]Shanghai Jiao Tong University [2]Huawei Noah's Ark Lab
[3]Tongji University [4]Huawei Cloud
{yuhuixu,xionghongkai}@sjtu.edu.cn
{198808xc,zxphistory,guojunq}@gmail.com
1410452@tongji.edu.cn tian.qi1@huawei.com

Abstract

Differentiable architecture search (DARTS) provided a fast solution in finding effective network architectures, but suffered from large memory and computing overheads in jointly training a super-net and search for an optimal architecture. In this paper, we present a novel approach, namely **Partially-Connected DARTS**, by sampling a small part of super-net to reduce the redundancy in network space, thereby performing a more efficient search without comprising the performance. In particular, we perform operation search in a subset of channels and leave the held out part unchanged. This strategy may suffer from an undesired inconsistency on selecting the edges of super-net caused by the sampling of different channels. We solve it by introducing *edge normalization*, which adds a new set of edge-level hyper-parameters during search to reduce uncertainty in search. Thanks to the reduced memory cost, PC-DARTS can be trained with a larger batch size and, consequently, enjoys both faster speed and higher training stability. Experimental results demonstrate the effectiveness of the proposed method. Specifically, we achieve an error rate of 2.57% on CIFAR10 within merely 0.1 GPU-days for architecture search, and a state-of-the-art top-1 error rate of 24.2% on ImageNet (under the mobile setting) within 3.8 GPU-days for search. *We have made our code available* `https://github.com/yuhuixu1993/PC-DARTS`.

1 Introduction

Neural architecture search (NAS) emerges as an important branch of automatic machine learning (AutoML), and has been attracting increasing attentions in both academia and industry. The key methodology of NAS is to build a large space of network architectures, resort to an efficient algorithm to explore the space, and discover the optimal structure under a specific combination of training data and constraints (*e.g.*, network size and latency). Different from early approaches that required a large amount of computations [34, 35, 24], recent one-shot approaches [23, 19] have reduced the search costs by orders of magnitudes, which advances its applications to many real-world problems. In particular, DARTS [19] model converts the operation selection into a weighted combination of a fixed set of operations. This makes the entire framework differentiable to architecture hyper-parameters and thus the search process can be accomplished in an end-to-end fashion. Despite its sophisticated design, DARTS is still subject to a large yet redundant space of network architectures, and thus suffers from heavy memory and computation overheads. This prevents the search process from using larger

[*]This work was done when the first author was an intern at Huawei Noah's Ark Lab.

Preprint. Under review.

batch sizes for either speedup or higher stability. Prior work [6] proposed to reduce the search space, which leads to an approximation that may sacrifice the optimality of the discovered architecture.

In this paper, we present a simple yet effective approach named Partially-Connected DARTS (PC-DARTS) to reduce the burdens of memory and computation. The core idea is intuitive: instead of sending all channels into the block of operation selection, we randomly sample a subset of them for operation selection, meanwhile bypassing the rest directly. We assume the computation on this subset is a surrogate approximating that on all the channels. Besides that both memory and computation costs are reduced tremendously, sampling brings another benefit that operation search is regularized and less likely to fall into local optima. However, PC-DARTS also brings a side effect, that the selection of network connectivity would become unstable as different subsets of channels are sampled across iterations. Thus, we introduce *edge normalization* to stabilize the search for network connectivity by explicitly learning an extra set of edge-selection hyper-parameters. By sharing these hyper-parameters throughout the training process, the learned network architecture is less prone to the sampled channels across iterations and thus be more stable.

Benefit from the partially connected strategy, we are able to increase the batch size accordingly. In practice, we randomly sample $1/K$ of channels for each operation selection, which reduces the memory cost by almost K times. This allows us to use a $K\times$ batch size during search, which not only accelerates search by K times, but also stabilizes search especially for large-scale datasets. Experiments on benchmark datasets well demonstrate the effectiveness of PC-DARTS. Specifically, we achieve an error rate of 2.57% with less than 0.1 GPU-days (around 1.5 hours) on a single GPU, surpassing the counterpart result of 2.76% reported by DARTS which required 1 GPU-day. Furthermore, PC-DARTS allows a direct search on ImageNet (while DARTS failed due to low stability), and achieves a state-of-the-art top-1 error of 24.2% (under the mobile setting) with only 3.8 GPU-days (11.5 hours) on 8 GPUs for search.

The remainder of this paper is organized as follows. Section 2 reviews related work, and Section 3 describes our approach. After experiments are shown in Section 4, we conclude in Section 5.

2 Related Works

Thanks to the rapid development of deep learning, significant gain in performance has been brought to a wide range of computer vision problems, most of which owed to manually desgined network architectures [16, 26, 11, 13]. Recently, a new research field named neural architecture search (NAS) has been attracting increasing attentions. The goal is to find automatic ways of designing neural architectures to replace conventional handcrafted ones. According to the heuristics to explore the large architecture space, existing NAS approaches can be roughly divided into three categories, namely, evolution-based approaches, reinforcement-learning-based approaches and one-shot approaches.

The first type of architecture search methods [18, 29, 25, 9, 24, 22] adopted evolutionary algorithms, which assumed the possibility of applying genetic operations to force a single architecture or a family evolve towards better performance. Among them, Liu *et al.* [18] introduced a hierarchical representation for describing a network architecture, and Xie *et al.* [29] decomposed each architecture into a representation of 'genes'. Real *et al.* [24] proposed aging evolution which improved upon standard tournament selection, and surpassed the best manually designed architecture since then. Another line of heuristics turns to reinforcement learning (RL) [34, 1, 35, 32, 17], which trained a meta-controller to guide the search process in the huge architecture space. Zoph *et al.* [34] first proposed using a controller-based recurrent neural network to generate hyper-parameters of neural networks. To reduce the computation cost, researchers started to search for blocks or cells [32, 35] instead of the entire network, and consequently, [35] managed to reduce the overall computational costs by a factor of 7. Other kinds of approximation, such as greedy search [17], were also applied to further accelerate search. Nevertheless, the computation costs of these approaches, based on either evolution or RL, are still beyond acceptance.

In order to accomplish architecture search within a short period of time, researchers considered to reduce the costs of evaluating each searched candidate. Early efforts include sharing weights between searched and newly generated networks [3], and later these methods were generalized into a more elegant framework named one-shot architecture search [2, 4, 19, 23, 30]. In these approaches, an over-parameterized network or super network covering all candidate operations was trained only once, and the final architecture is obtained from sampling from this super network. Among them,

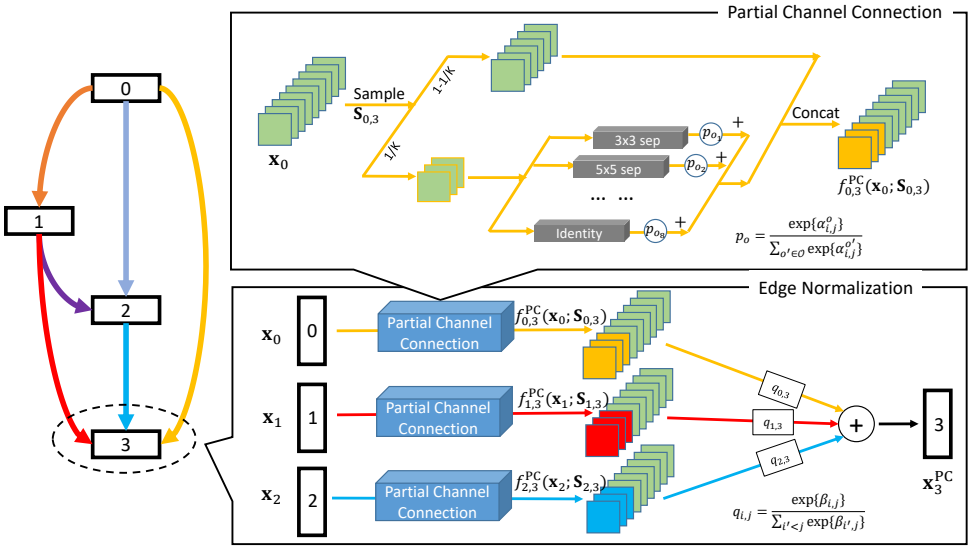


Figure 1: Illustration of the proposed approach, partially-connected DARTS (PC-DARTS). As an example, we investigate how information is propagated to node #3, *i.e.*, $j = 3$. There are two sets of hyper-parameters during search, namely, $\{\alpha^o_{i,j}\}$ and $\{\beta_{i,j}\}$, where $0 \leqslant i < j$ and $o \in \mathcal{O}$. To determine $\{\alpha^o_{i,j}\}$, we only sample a subset, $1/K$, of channels and connect them to the next stage, so that the memory consumption is reduced by K times. To minimize the uncertainty incurred by sampling, we add $\{\beta_{i,j}\}$ as *extra* edge-level parameters.

Brock *et al.* [2] trained the over-parameterized network by a HyperNet [10], and Pham *et al.* [23] proposed to share parameters among child models to avoid retraining each candidate from scratch. This paper is based on DARTS [18], which introduced a differentiable framework and thus combine the search and evaluation stages into one. Despite its simplicity, researchers detected some of its drawbacks which led to a few improved approaches beyond DARTS [4, 30, 6].

3 The Proposed Approach

3.1 Preliminaries: Differentiable Architecture Search (DARTS)

We first review the baseline DARTS [19], and define the notations for the discussion later.

Formally, DARTS decomposes the searched network into a number (L) of cells. Each cell is organized as a directed acyclic graph (DAG) with N nodes, where each node defines a network layer. There is a pre-defined space of operations denoted by $\mathcal{O}$, in which each element, $o(\cdot)$, is a fixed operation (*e.g.*, identity connection, and 3×3 convolution) performed at a network layer. Within a cell, the goal is to choose one operation from $\mathcal{O}$ to connect each pair of nodes. Let a pair of nodes be (i, j), where $0 \leqslant i < j \leqslant N - 1$, the core idea of DARTS is to formulate the information propagated from i to j as a weighted sum of $|\mathcal{O}|$ operations, namely, $f_{i,j}(\mathbf{x}_i) = \sum_{o \in \mathcal{O}} \frac{\exp\{\alpha^o_{i,j}\}}{\sum_{o' \in \mathcal{O}} \exp\{\alpha^{o'}_{i,j}\}} \cdot o(\mathbf{x}_i)$, where $\mathbf{x}_i$ is the output of the i-th node, and $\alpha^o_{i,j}$ is a hyper-parameter for weighting operation $o(\mathbf{x}_i)$. The output of a node is the sum of all input flows, *i.e.*, $\mathbf{x}_j = \sum_{i<j} f_{i,j}(\mathbf{x}_i)$, and the output of the entire cell is formed by concatenating the output of nodes $\mathbf{x}_2$–$\mathbf{x}_{N-1}$, *i.e.*, $\text{concat}(\mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{N-1})$. Note that the first two nodes, $\mathbf{x}_0$ and $\mathbf{x}_1$, are input nodes to a cell, which are fixed during architecture search.

This design makes the entire framework differentiable to both layer weights and hyper-parameters $\alpha^o_{i,j}$, so that it is possible to perform architecture search in an end-to-end fashion. After the search process is finished, on each edge (i, j), the operation o with the largest $\alpha^o_{i,j}$ value is preserved, and each node j is connected to two precedents $i < j$ with the largest $\alpha^o_{i,j}$ preserved.

3.2 Partial Channel Connections

A drawback of DARTS lies in its memory inefficiency. To accommodate $|\mathcal{O}|$ operations, in the main part of the searched architecture, $|\mathcal{O}|$ copies of operations and their outputs need to be stored at each node (*i.e.*, each network layer), leading to $|\mathcal{O}|\times$ memory to use. To fit into a GPU, one must reduce the batch size during search, which inevitably slows down search speed, and may deteriorate search stability and accuracy with a smaller batch size.

Alternatively, another solution to memory efficiency is the **partial channel connection** as depicted in Figure 1. Take the connection from $\mathbf{x}_i$ to $\mathbf{x}_j$ for example. This involves defining a channel sampling mask $\mathbf{S}_{i,j}$, which assigns 1 to selected channels and 0 to masked ones. The selected channels are sent into mixed computation of $|\mathcal{O}|$ operations, while the masked ones bypass these operations, *i.e.*, they are directly copied to the output,

$$f_{i,j}^{\mathrm{PC}}(\mathbf{x}_i;\mathbf{S}_{i,j}) = \sum_{o\in\mathcal{O}} \frac{\exp\left\{\alpha_{i,j}^{o}\right\}}{\sum_{o'\in\mathcal{O}}\exp\left\{\alpha_{i,j}^{o'}\right\}} \cdot o(\mathbf{S}_{i,j} * \mathbf{x}_i) + (1-\mathbf{S}_{i,j}) * \mathbf{x}_i. \tag{1}$$

where, $\mathbf{S}_{i,j} * \mathbf{x}_i$ and $(1-\mathbf{S}_{i,j}) * \mathbf{x}_i$ denote the selected and masked channels, respectively. In practice, we set the proportion of selected channels to $1/K$ by regarding K as a hyper-parameter. By varying K, we could trade off between architecture search accuracy (smaller K) and efficiency (larger K) to strike a balance (See Section 4.4.1 for more details).

A direct benefit brought by K is that the memory overhead of computing $f_{i,j}^{\mathrm{PC}}(\mathbf{x}_i;\mathbf{S}_{i,j})$ is reduced by K times. This allows us to use a larger batch size for architecture search. There are twofold benefits that follow instantly. First, the computing cost could also be reduced by K times. Moreover, the larger batch size implies the possibility of sampling more training data during each iteration. This is particularly important for architecture search. In most cases, the advantage of one operation over another is not significant, unless more training data are involved in a mini-batch to reduce the uncertainty in updating the parameters of network weights and architectures.

3.3 Edge Normalization

Let us look into the impact of sampling channels on neural architecture search. There are both positive and negative effects. On the **upside**, by feeding a small subset of channels for operation mixture while keeping the remainder unchanged, we make it less biased in selecting operations. In other words, for edge (i,j), given an input $\mathbf{x}_i$, the difference from using two sets of hyper-parameters $\left\{\alpha_{i,j}^{o}\right\}$ and $\left\{\alpha_{i,j}^{\prime o}\right\}$ is largely reduced, because only a small part $(1/K)$ of input channels would go through the operation mixture while the remaining channels are left unchanged. This weakens the advantage of a weight-free operation (*e.g.*, *skip-connect*, *max-pooling*, *etc.*) over a weight-equipped one (*e.g.*, various kinds of *convolution*) in $\mathcal{O}$. In the early stage, the search algorithm often prefers weight-free operations, because they do not have weights to train and thus produce more consistent outputs, *i.e.*, $o(\mathbf{x}_i)$. In contrast, the weight-equipped ones, before their weights are well optimized, would propagate inconsistent information across iterations. Consequently, weight-free operations often accumulate larger weights (namely $\alpha_{i,j}^{o}$) at the beginning, and this makes it difficult for the weight-equipped operations to beat them even after they have been well trained thereafter. *This phenomenon is especially significant when the proxy dataset (on which architecture search is performed) is difficult, and this has prevents DARTS from performing satisfactory architecture search on ImageNet.* In experiments, we will show that PC-DARTS, with partial channel connections, produces more stable and superior performance on ImageNet.

On the **downside**, in a cell, each output node $\mathbf{x}_j$ needs to pick up two input nodes from its precedents $\{\mathbf{x}_0,\mathbf{x}_1,\ldots,\mathbf{x}_{j-1}\}$, which are weighted by $\max_o \alpha_{0,j}^{o}, \max_o \alpha_{1,j}^{o}, \ldots, \max_o \alpha_{j-1,j}^{o}$, respectively, following the original DARTS. However, these architecture parameters are optimized by randomly sampled channels across iterations, and thus the optimal connectivity determined by them could be unstable as the sampled channels change over time. This could cause undesired fluctuation in the resultant network architecture. To mitigate this problem, we introduce edge normalization that weighs on each edge (i,j) explicitly, denoted by $\beta_{i,j}$, so that the computation of $\mathbf{x}_j$ becomes:

$$\mathbf{x}_j^{\mathrm{PC}} = \sum_{i<j} \frac{\exp\{\beta_{i,j}\}}{\sum_{i'<j}\exp\{\beta_{i',j}\}} \cdot f_{i,j}(\mathbf{x}_i). \tag{2}$$

Specifically, after the architecture search is done, the connectivity of edge (i, j) is determined by both $\{\alpha_{i,j}^o\}$ and $\beta_{i,j}$, for which we multiply the normalized coefficients together, *i.e.*, multiplying $\frac{\exp\{\beta_{i,j}\}}{\sum_{i'<j}\exp\{\beta_{i',j}\}}$ with $\frac{\exp\{\alpha_{i,j}^o\}}{\sum_{o'\in\mathcal{O}}\exp\{\alpha_{i,j}^{o'}\}}$. Then the edges are selected by finding the large edge weights as in DARTS. Since $\beta_{i,j}$ are shared through the training process, the learned network architecture is less prone to the sampled channels across iterations, making the architecture search more stable. In Section 4.4.2, we will show that edge normalization is also effective over the original DARTS. Finally, the extra computation required for edge normalization is negligible.

3.4 Relationship to Prior Work

Other researchers also tried to alleviate the large memory consumption of DARTS. Among prior efforts, ProxylessNAS [4] binarized the multinomial distribution $\alpha_{i,j}^o$ and samples two paths at each time, which significantly reduced memory cost and enabled direct search on ImageNet. PARSEC [5] also proposed a sampling-based optimization method to learn a probability distribution. Our solution, by preserving all operations for architecture search, achieves a higher accuracy in particular on challenging datasets like ImageNet ($+0.7\%$ over ProxylessNAS and $+1.8\%$ over PARSEC).

Another closely related work is Progressive-DARTS [6], which approximated search space in order to provide sufficient memory for deeper architecture search. In the meantime, [6] applied dropout after *skip-connect* to avoid imbalanced information between parameterized and parameter-free operations. Our solution, *i.e.*, sampling a subset of channels, shares a similar insight with path dropout yet enjoys both reduced memory consumption and stronger regularization ability. In experiments, we show that our approach allows a directly search on ImageNet, on which [6] produced less stable results.

4 Experiments

4.1 Datasets and Implementation Details

We perform experiments on CIFAR10 and ImageNet, two most popular datasets for evaluating neural architecture search. CIFAR10 [15] consists of 60K images, all of which are of a spatial resolution of 32×32. These images are equally distributed over 10 classes, with 50K training and 10K testing images. ImageNet [7] contains $1{,}000$ object categories, and 1.3M training images and 50K validation images, all of which are high-resolution and roughly equally distributed over all classes. Following the conventions [35, 19], we apply the *mobile setting* where the input image size is fixed to be 224×224 and the number of multi-add operations does not exceed 600M in the testing stage.

Following DARTS [19] as well as conventional architecture search approaches, we use an individual stage for architecture search, and after the optimal architecture is obtained, we conduct another training process from scratch. In the search stage, the goal is to determine the best sets of hyper-parameters, namely $\{\alpha_{i,j}^o\}$ and $\{\beta_{i,j}\}$ for each edge (i, j). To this end, the trainnig set is partitioned into two parts, with the first part used for optimizing network parameters, *e.g.*, convolutional weights, and the second part used for optimizing hyper-parameters. The entire search stage is accomplished in an end-to-end manner. For fair comparison, the operation space $\mathcal{O}$ remains the same as the convention, which contains 8 choices, *i.e.*, 3×3 and 5×5 *separable convolution*, 3×3 and 5×5 *dilated separable convolution*, 3×3 *max-pooling*, 3×3 *average-pooling*, *skip-connect* (*identity*), and *zero* (*none*).

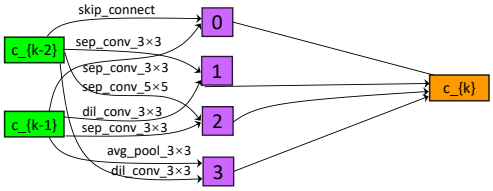
We propose an alternative and more efficient implementation for partial channel connections. For edge (i, j), we do not perform channel sampling at each time of computing $o(\mathbf{x}_i)$, but instead choose the first $1/K$ channels of $\mathbf{x}_i$ for operation mixture directly. To compensate, after $\mathbf{x}_j$ is obtained, we shuffle its channels before using it for further computations. This is the same implementation used in ShuffleNet [31], which is more GPU-friendly and thus runs faster.

4.2 Results on CIFAR10

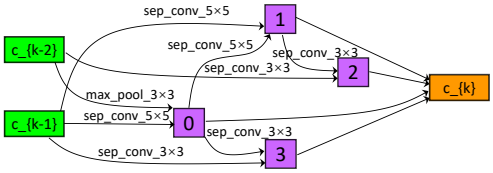
In the search scenario, the over-parameterized network is constructed by stacking 8 cells (6 normal cells and 2 reduction cells), and each cell consists of $N = 6$ nodes. We train the network for 50 epochs, with the initial number of channels being 16. The 50K training set of CIFAR10 is split into two subsets with equal size, with one subset used for training network weights and the other used for architecture hyper-parameters.

Architecture	Test Err. (%)	Params (M)	Search Cost (GPU-days)	Search Method
DenseNet-BC [13]	3.46	25.6	-	manual
NASNet-A + cutout [35]	2.65	3.3	1800	RL
AmoebaNet-A + cutout [24]	3.34±0.06	3.2	3150	evolution
AmoebaNet-B + cutout [24]	2.55±0.05	2.8	3150	evolution
Hireachical Evolution [18]	3.75±0.12	15.7	300	evolution
PNAS [17]	3.41±0.09	3.2	225	SMBO
ENAS + cutout [23]	2.89	4.6	0.5	RL
NAONet-WS [20]	3.53	3.1	0.4	NAO
DARTS (1st order) + cutout [19]	3.00±0.14	3.3	0.4	gradient-based
DARTS (2nd order) + cutout [19]	2.76±0.09	3.3	1	gradient-based
SNAS (mild) + cutout [30]	2.98	2.9	1.5	gradient-based
SNAS (moderate) + cutout [30]	2.85±0.02	2.8	1.5	gradient-based
SNAS (aggressive) + cutout [30]	3.10±0.04	2.3	1.5	gradient-based
ProxylessNAS + cutout [4]	2.08	-	4.0	gradient-based
P-DARTS + cutout [6]	2.50	3.4	0.3	gradient-based
BayesNAS + cutout [33]	2.81±0.04	3.4	0.2	gradient-based
PC-DARTS+ cutout	2.57±0.07	3.6	**0.1**†	gradient-based

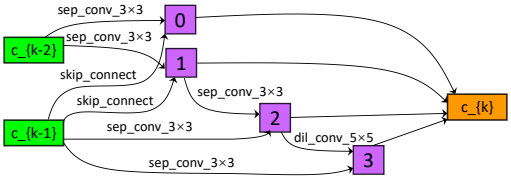
Table 1: Comparison with state-of-the-art network architectures on CIFAR10. †This time is recorded on a single GTX 1080Ti. It can be shortened into 0.06 GPU-days if we use a single Tesla V100.



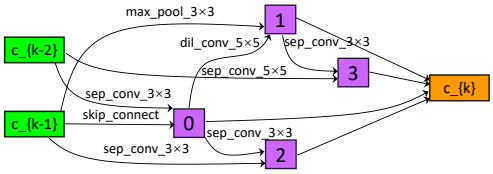
(a) the normal cell found on CIFAR10



(b) the reduction cell found on CIFAR10



(c) the normal cell found on ImageNet



(d) the reduction cell on ImageNet

Figure 2: Cells found on CIFAR10 and ImageNet. ImageNet forces the cells to be more complicated.

We set $K = 4$ for CIFAR10, *i.e.*, only $1/4$ features are sampled on each edge, so that the batch size during search is increased to 256. Besides, following [6], we freeze network hyper-parameters and only allow network parameters to be tuned in the first 15 epochs. This is to provide a warm-up for network parameters and thus alleviates the drawback of parameterized operations. The total memory cost is less than 12GB so that we can train it on most modern GPUs. The network weights are optimized by momentum SGD, with an initial learning rate of 0.1 (annealed down to zero following a cosine schedule without restart), a momentum of 0.9, and a weight decay of 3×10^{-4}. We use the Adam optimizer [14] for hyper-parameters $\{\alpha^o_{i,j}\}$ and $\{\beta_{i,j}\}$, with a fixed learning rate of 6×10^{-4}, a momentum of $(0.5, 0.999)$ and a weight decay of 10^{-3}. Owing to the increased batch size, the entire search process only requires 3 hours on a single NVIDIA GTX 1080Ti GPU, or 1.5 hours on a single Tesla V100 GPU, which is almost $4\times$ faster than the original first-order DARTS.

The evaluation stage simply follows that of DARTS. The network is composed of 20 cells (18 normal cells and 2 reduction cells), and each type of cells share the same architecture. The initial number of channels is 36. The entire 50K training set is used, and the network is trained from scratch for 600 epochs using a batch size of 128. We use the SGD optimizer with an initial learning rate of

Architecture	Test Err. (%)		Params	$\times+$	Search Cost	Search Method
	top-1	top-5	(M)	(M)	(GPU-days)	
Inception-v1 [27]	30.2	10.1	6.6	1448	-	manual
MobileNet [12]	29.4	10.5	4.2	569	-	manual
ShuffleNet 2× (v1) [31]	26.4	10.2	∼5	524	-	manual
ShuffleNet 2× (v2) [21]	25.1	-	∼5	591	-	manual
NASNet-A [35]	26.0	8.4	5.3	564	1800	RL
NASNet-B [35]	27.2	8.7	5.3	488	1800	RL
NASNet-C [35]	27.5	9.0	4.9	558	1800	RL
AmoebaNet-A [24]	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B [24]	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C [24]	24.3	7.6	6.4	570	3150	evolution
PNAS [17]	25.8	8.1	5.1	588	225	SMBO
MnasNet-92 [28]	25.2	8.0	4.4	388	-	RL
DARTS (2nd order) [19]	26.7	8.7	4.7	574	4.0	gradient-based
SNAS (mild) [30]	27.3	9.2	4.3	522	1.5	gradient-based
ProxylessNAS (GPU)‡ [4]	24.9	7.5	7.1	465	8.3	gradient-based
P-DARTS (CIFAR10) [6]	24.4	7.4	4.9	557	0.3	gradient-based
P-DARTS (CIFAR100) [6]	24.7	7.5	5.1	577	0.3	gradient-based
BayesNAS [33]	26.5	8.9	3.9	-	0.2	gradient-based
PC-DARTS (CIFAR10)	25.1	7.8	5.3	586	0.1	gradient-based
PC-DARTS (ImageNet)‡	**24.2**	**7.3**	5.3	597	3.8	gradient-based

Table 2: Comparison with state-of-the-art architectures on ImageNet (mobile setting). ‡This architecture was searched on ImageNet directly, otherwise it was searched on CIFAR10 or CIFAR100.

0.025 (annealed down to zero following a cosine schedule without restart), a momentum of 0.9, a weight decay of 3×10^{-4} and a norm gradient clipping at 5. Drop-path with a rate of 0.3 as well as cutout [8] is also used for regularization. We visualize the searched normal and reduction cells in the left-hand side of Figure 2.

Results and comparison to recent approaches are summarized in Table 1. In merely 0.1 GPU-days, PC-DARTS achieve an error rate of 2.57%, with both search time and accuracy surpassing the baseline, DARTS, significantly. To the best of our knowledge, our approach is the fastest one that achieves an error rate of less than 3%. Our number ranks among the top of recent architecture search results. ProxylessNAS used a different protocol to achieve an error rate of 2.08%, and also reported a much longer time for architecture search. P-DARTS [6] slightly outperforms our approach by searching over a deeper architecture, which we emphasize that our approach can be combined with it.

4.3 Results on ImageNet

We slightly modify the network architecture used on CIFAR10 to fit ImageNet. The over-parameterized network starts with three convolution layers of stride 2 to reduce the input image resolution from 224×224 to 28×28. 8 cells (6 normal cells and 2 reduction cells) are stacked beyond this point, and each cell consists of $N = 6$ nodes. To reduce search time, we randomly sample two subsets from the 1.3M training set of ImageNet, with 10% and 2.5% images, respectively. The former one is used for training network weights and the latter for updating hyper-parameters.

ImageNet is much more difficult than CIFAR10. To preserve more information, we use a sub-sampling rate of $1/2$, which doubles that used in CIFAR10. Still, a total of 50 epochs are trained and architecture hyper-parameters are frozen during the first 35 epochs. For network weights, we use a momentum SGD with an initial learning rate of 0.5 (annealed down to zero following a cosine schedule without restart), a momentum of 0.9, and a weight decay of 3×10^{-5}. For hyper-parameters, we use the Adam optimizer [14] with a fixed learning rate of 6×10^{-3}, a momentum $(0.5, 0.999)$ and a weight decay of 10^{-3}. We use eight Tesla V100 GPUs for search, and the total batch size is $1{,}024$. The entire search process takes around 11.5 hours. We visualize the searched normal and reduction cells in the right-hand side of Figure 2.

The evaluation stage follows that of DARTS, which also starts with three convolution layers of stride 2 that reduce the input image resolution from 224×224 to 28×28. 14 cells (12 normal cells and 2 reduction cells) are stacked beyond this point, with the initial channel number being 48. The network

is trained from scratch for 250 epochs using a batch size of 1,024. We use the SGD optimizer with a momentum of 0.9, an initial learning rate of 0.5 (decayed down to zero linearly), and a weight decay of 3×10^{-5}. Additional enhancements are adopted including label smoothing and an auxiliary loss tower during training. Learning rate warm-up is applied for the first 5 epochs.

Results are summarized in Table 2. Note that the architectures searched on CIFAR10 and ImageNet itself are both evaluated. For the former, it reports a top-1/5 error of 25.1%/7.8%, which significantly outperforms 26.7%/8.7% reported by DARTS. This is impressive given that our search time is much shorter. For the latter, we achieve a top-1/5 accuracy of 24.2%/7.3%, which is the best known performance to date. In comparison, ProxylessNAS [4], another approach that directly searched on ImageNet, used almost doubled time to produce 24.9%/7.5%, which verifies that our strategy of reducing memory consumption is more efficient yet effective.

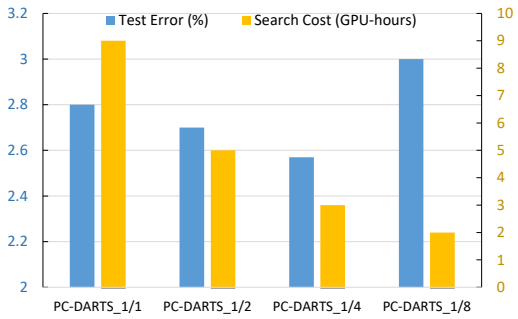


Figure 3: Search cost and accuracy comparison between our approach with different sampling rates.

CIFAR10			
PC	**EN**	**Test Error**	**Search Cost**
✗	✗	3.00%	0.4 GPU-days
✗	✓	2.82%	0.4 GPU-days
✓	✓	2.57%	0.1 GPU-days
ImageNet			
PC	**EN**	**Test Error**	**Search Cost**
✗	✗	26.8%	7.7 GPU-days
✗	✓	26.3%	7.7 GPU-days
✓	✓	25.8%	3.8 GPU-days

Table 3: Ablation study on CIFAR10 and ImageNet. PC and EN denote partial channel connections and edge normalization, respectively. All architectures on ImageNet are trained by 100 epochs (the 25.8% error corresponds to the best number, 24.2%, reported in Table 2, with 250 epochs.

4.4 Ablation Study

4.4.1 Effectiveness of Channel Proportion $1/K$

We first evaluate K, the hyper-parameter that controls the sampling rate of channels. Note that a tradeoff exists: increasing the sampling rate (*i.e.*, using a smaller K) allows more accurate information to be propagated, while sampling a smaller portion of channels casts heavier regularization and may alleviate over-fitting. To study its impacts, we evaluate the performance produced by four sampling rates, namely $1/1$, $1/2$, $1/4$ and $1/8$, on CIFAR10, and plot the results into a diagram of search time and accuracy in Figure 3. One can observe that a sampling rate of $1/4$ yields superior performance over $1/2$ and $1/1$ in terms of both time and accruacy. Using $1/8$, while being able to further reduce search time, causes a dramatic accuracy drop.

4.4.2 Contributions of Different Components of PC-DARTS

Next, we evaluate the contributions made by two components of PC-DARTS, namely, partial channel connections and edge normalization. Results are summarized in Table 3. It is clear that edge normalization brings the effect of regularization even when the channels are fully-connected. Being a component with very few extra costs, it can be freely applied to a wide range of approaches involving edge selection. In addition, edge normalization cooperates well with partial channel connections to provide further improvement. Without edge normalization, our approach can suffer low stability in both the number of network parameters and accuracy. On CIFAR10, we run search without edge normalization for several times, and the testing error ranges from 2.54% to 3.01%. On the other hand, with edge normalization, the maximal difference among five runs does not exceed 0.15%.

5 Conclusions

In this paper, we proposed a simple and effective approach named partially-connected differentiable architecture search (PC-DARTS). The core idea is to randomly sample a proportion of channels for operation search, so that the framework is more memory efficient and, consequently, a larger batch

size can be used for higher stability. Additional contribution to search stability is made by edge normalization, a light-weighted module which requires merely no extra computation. Our approach can accomplish a complete search within 0.1 GPU-days on CIFAR10, or 3.8 GPU-days on ImageNet, and report state-of-the-art classification accuracy in particular on ImageNet.

References

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

[2] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. SMASH: one-shot model architecture search through hypernetworks. 2018.

[3] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. In *AAAI*, 2018.

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

[5] F. P. Casale, J. Gordon, and N. Fusi. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116*, 2019.

[6] X. Chen, L. Xie, J. Wu, and Q. Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *arXiv preprint arXiv:1904.12760*, 2019.

[7] 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.

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

[9] T. Elsken, J. H. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.

[10] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

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

[12] 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.

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

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

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

[17] 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.

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

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

[20] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu. Neural architecture optimization. In *NeurIPS*, 2018.

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

[22] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

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

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

[25] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

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

[27] 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.

[28] 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.

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

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

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

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

[33] H. Zhou, M. Yang, J. Wang, and W. Pan. BayesNAS: A Bayesian approach for neural architecture search. 2019.

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

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