STABILIZING DARTS WITH AMENDED GRADIENT ESTIMATION ON ARCHITECTURAL PARAMETERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite the great advantage in search efficiency, DARTS often suffers **weak stability**, which reflects in the large variation among individual trials as well as the sensitivity to the hyper-parameters of the search process. This paper owes such instability to an **optimization gap** between the super-network and its sub-networks, namely, improving the validation accuracy of the super-network does not necessarily lead to a higher expectation on the performance of the sampled sub-networks. Then, we point out that the gap is due to the inaccurate estimation of the architectural gradients, based on which we propose an amended estimation method. Mathematically, our method guarantees **a bounded error** from the true gradients while the original estimation does not. Our approach bridges the gap from two aspects, namely, amending the estimation on the architectural gradients, and unifying the hyper-parameter settings in the search and re-training stages. Experiments on CIFAR10, ImageNet, and Penn Treebank demonstrate that our approach largely improves search stability and, more importantly, enables DARTS-based approaches to explore much larger search spaces that have not been investigated before.

1 INTRODUCTION

With the development of deep learning (LeCun et al., 2015), deep neural networks (Krizhevsky et al., 2012) have become the standard tool for learning representations in a complicated feature space. Recent years have witnessed the trend of using deeper (He et al., 2016) and denser (Huang et al., 2017) networks, while there is no justification that whether these manually designed architectures are optimal. Recently, researchers started considering the possibility of learning network architectures automatically from data, which led to the appearance of neural architecture search (NAS) (Zoph & Le, 2017), a sub research field in automated machine learning (AutoML).

The idea of NAS is to replace the manual network design with an automatic algorithm. The early age of NAS mainly involves using heuristic search methods (*e.g.*, reinforcement learning (Zoph & Le, 2017; Zoph et al., 2018; Liu et al., 2018) and evolutionary algorithms (Real et al., 2017; Xie & Yuille, 2017; Real et al., 2019)) to sample architectures from a large search space and evaluating them individually. Despite the notable success, these methods often requires a vast amount of computation. To alleviate the burden, researchers proposed to reuse computation from the previously optimized networks (Cai et al., 2018) or build a super-network (Pham et al., 2018) to share computation among all possible architectures. These efforts combined into the so-called 'one-shot' search methods (Howard et al., 2019; Guo et al., 2019) which, as the super-network is trained only once, are typically 2–3 orders of magnitude faster than the individual search methods.

This paper focuses on DARTS (Liu et al., 2019), a specific class of one-shot search in which the search space is slacked so that the choices among operators are formulated using some continuous 'architectural' parameters. This simplifies the search procedure into an end-to-end optimization, but also raises the risk of **instability** (Li & Talwalkar, 2019; Sciuto et al., 2019; Zela et al., 2020). Researchers reported that DARTS-based algorithms can sometimes generate weird architectures that produce considerably worse accuracy than those generated in other individual runs (Zela et al., 2020). Although some practical methods (Chen et al., 2019; Nayman et al., 2019; Xu et al., 2020) have been developed to reduce search variance, the following property of DARTS persists and has not been studied thoroughly: when DARTS gets trained for sufficiently long, *e.g.*, extending the default number of 50 epochs to 200 epochs, almost **all** DARTS-based approaches converge to a dummy architecture in which all edges are occupied by skip-connect. These architectures, with few trainable parameters, are often far from producing high accuracy, in particular, on large datasets like ImageNet, although the validation accuracy in the search stage continues growing with more epochs. Existing solutions include searching for several times and choosing the best one in validation (Liu et al., 2019), using other kinds of techniques such as decoupling modules (Cai et al., 2019; Guo et al., 2019), adjusting search space during optimization (Chen et al., 2019; Nayman et al., 2019; Noy et al., 2020), regularization (Xu et al., 2020; Chen & Hsieh, 2020), early termination (Liang et al., 2019), *etc*. However, these approaches seem to develop heuristic remedies rather than analyze it from the essence, *i.e.*, how instability happens in mathematics and how to maximally avoid it.

In this paper, we offer a new perspective to this problem. The key observation is that the instability issue is caused by the **optimization gap** between the search and re-training stages. That is to say, a high validation accuracy of the super-network does not guarantee the high quality of the final architecture. To solve this issue, we investigate it from a perspective which is less studied before. We reveal that DARTS has been using an inaccurate approximation to calculate the gradients with respect to the architectural parameters ($\boldsymbol{\alpha}$ as in the literature), and we amend the error by slightly modifying the second-order term in gradient computation. Mathematically, we prove that the amended term has a bounded error, *i.e.*, the angle between the true and estimated gradients is smaller than 90°, while the original DARTS did not guarantee so. With this modification, the search performance is stabilized and the dummy all-skip-connect architecture does not appear even after a very long search process. Consequently, one can freely allow the search process to arrive at convergence, and hence the sensitivity to hyper-parameters is alleviated.

Practically, our approach involves using an amended second-order gradient, so that the computational overhead is comparable to the *second-order* version of DARTS. Experiments are performed on two image classification datasets, CIFAR10 and ImageNet, and a language modeling dataset, Penn Treebank. In all experiments, the search process, after arriving at convergence, produces competitive architectures and classification accuracy comparable to the state-of-the-arts.

2 AN AMENDED ARCHITECTURAL GRADIENT ESTIMATION FOR DARTS

2.1 PRELIMINARIES: DIFFERENTIABLE ARCHITECTURE SEARCH

Differentiable NAS approaches start with defining a **super-network**, which is constrained in a search space with a pre-defined number of layers and a limited set of neural operators. The core idea is to introduce a 'soft' way operator selection (*i.e.*, using a weighted sum over the outputs of a few operators instead of taking the output of only one), so that optimization can be done in an end-to-end manner. Mathematically, the super-network is a function $\mathbf{f}(\mathbf{x};\boldsymbol{\omega},\boldsymbol{\alpha})$, with $\mathbf{x}$ being input, and parameterized by network parameters $\boldsymbol{\omega}$ (*e.g.*, convolutional kernels) and architectural parameters $\boldsymbol{\alpha}$ (*e.g.*, indicating the importance of each operator between each pair of layers). $\mathbf{f}(\mathbf{x};\boldsymbol{\omega},\boldsymbol{\alpha})$ is differentiable to both $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$, so that gradient-based approaches (*e.g.*, SGD) can be applied for optimization.

In the example of DARTS, $\mathbf{f}(\mathbf{x};\boldsymbol{\omega},\boldsymbol{\alpha})$ is composed of a few cells, each of which contains N nodes, and there is a pre-defined set, $\mathcal{E}$, denoting which pairs of nodes are connected. For each connected node pair (i,j), $i<j$, node j takes $\mathbf{x}_i$ as input and propagates it through a pre-defined operator set, $\mathcal{O}$, and sums up all outputs: $\mathbf{y}^{(i,j)}\left(\mathbf{x}_i\right)=\sum_{o\in\mathcal{O}}\frac{\exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathcal{O}}\exp(\alpha_{o'}^{(i,j)})}\cdot o(\mathbf{x}_i)$.

Here, normalization is performed by computing softmax on the architectural weights. Within each unit of the search process, $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ get optimized alternately. After that, the operator o with the maximal value of $\alpha_o^{(i,j)}$ is preserved for each edge (i,j). All network parameters $\boldsymbol{\omega}$ are discarded and the obtained architecture is re-trained from scratch.

2.2 THE OPTIMIZATION GAP OF DARTS

Our research is motivated by an observation that DARTS, at the end of a regular training process with, say, 50 epochs (Liu et al., 2019), has not yet arrived at convergence. To verify this, we increase the length of each training stage from 50 to 200 epochs, and observe two weird facts shown in Figure 1. **First**, the weight of the *none* operator monotonically goes up – at 200 epochs, the weight has achieved 0.95 on most edges of the normal cells, however, the *none* operator is not considered in the final

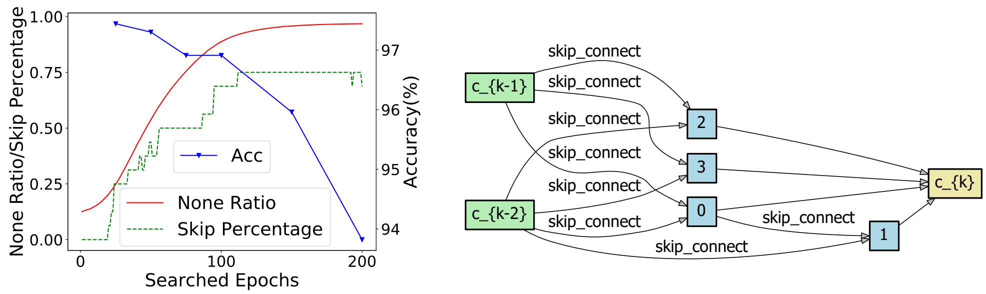


Figure 1: **Left**: a typical search process of the *first-order* DARTS, in which 200 search epochs are used. Red, green and blue lines indicate the average weight of *none*, the ratio of preserved skip-connect operators, and the re-training accuracy, respectively, with respect to the number of search epochs. **Right**: the normal cell after 200 search epochs, in which all operators are skip-connect. **Such failure consistently happens for both *first-order* and *second-order* DARTS.**

architecture. **Second**, almost all preserved operators are the skip-connect (*a.k.a.*, *identity*) operator, a parameter-free operator that contributes little to feature learning – and surprisingly, it occupies 30% to 70% of the weight remained by the *none* operator. Such a network has very few trainable parameters, and thus usually reports unsatisfying performance at the re-training stage, in particular, lower than a randomly sampled network (please refer to the experiments in Section 3.1.1).

Despite dramatically bad sub-networks are produced, the validation accuracy of the super-network keeps growing as the search process continues. In a typical run of the *first-order* DARTS on CIFAR10, from 50 to 200 search epochs, the validation accuracy of the super-network is boosted from 88.82% to 91.06%, while the re-training accuracy of the final architecture reduced from 97.00% to 93.82%. This implies an **optimization gap** between the super-network and its sub-networks. Specifically, the search process aims to improve the validation accuracy of the super-network, but this does not necessarily result in high accuracy of the optimal sub-network determined by the architectural parameters[1], $\boldsymbol{\alpha}$. A practical solution is to terminate the search process early (Liang et al., 2019), however, despite its effectiveness, early termination makes the search result sensitive to the initialization ($\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$), the hyper-parameters of search (*e.g.*, learning rate), and the time of termination. Consequently, the stability of DARTS is inevitably weakened.

2.3 Amending the Architectural Gradients

We point out that the failure owes to inaccurate estimation of $\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}),\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$, the gradient with respect to $\boldsymbol{\alpha}$. Following the chain rule of gradients, this quantity equals to

$$\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}+\nabla_{\boldsymbol{\alpha}}\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}. \tag{1}$$

For the simplicity of notations, we denote $\mathbf{g}_1=\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$ and $\mathbf{g}_2=\nabla_{\boldsymbol{\alpha}}\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$. $\mathbf{g}_1$ is easy to compute as is done in the *first-order* DARTS, while $\mathbf{g}_2$, in particular $\nabla_{\boldsymbol{\alpha}}\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$, is not (see Appendix A.2). To estimate $\nabla_{\boldsymbol{\alpha}}\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$, we note that $\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha})$ has arrived at the optimality on the training set, hence $\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}^{\dagger}),\boldsymbol{\alpha}=\boldsymbol{\alpha}^{\dagger}}\equiv\mathbf{0}$ holds for any $\boldsymbol{\alpha}^{\dagger}$. Differentiating with respect to any $\boldsymbol{\alpha}^{\dagger}$ on both sides, we have $\nabla_{\boldsymbol{\alpha}^{\dagger}}\left(\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}^{\dagger}),\boldsymbol{\alpha}=\boldsymbol{\alpha}^{\dagger}}\right)\equiv\mathbf{0}$. When $\boldsymbol{\alpha}^{\dagger}=\boldsymbol{\alpha}_t$, it becomes:

$$\nabla_{\boldsymbol{\alpha}^{\dagger}}\left(\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}^{\dagger}),\boldsymbol{\alpha}=\boldsymbol{\alpha}^{\dagger}}\right)\Big|_{\boldsymbol{\alpha}^{\dagger}=\boldsymbol{\alpha}_t}=\mathbf{0}. \tag{2}$$

Again, applying the chain rule to the left-hand side gives:

$$\nabla^2_{\boldsymbol{\alpha},\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}+\nabla_{\boldsymbol{\alpha}}\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}\cdot\nabla^2_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}=\mathbf{0}, \tag{3}$$

[1]This opinion is different from that of (Zela et al., 2020), which believed a super-network with higher validation accuracy must be better, and owed unsatisfying sub-network performance to the final discretization step of DARTS. Please refer to Appendix A.1 for the detailed elaboration.

where we use the notation of $\nabla^2_{\boldsymbol{\alpha},\boldsymbol{\omega}}(\cdot) \doteq \nabla_{\boldsymbol{\alpha}}(\nabla_{\boldsymbol{\omega}}(\cdot))$ throughout the remaining part of this paper. We denote $\mathbf{H} \doteq \nabla^2_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$, the Hesse matrix corresponding to the optimum, $\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t)$. $\mathbf{H}$ is symmetric and positive-definite, and thus invertible, which gives that $\nabla_{\boldsymbol{\alpha}}\boldsymbol{\omega}^\star(\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t} = -\,\nabla^2_{\boldsymbol{\alpha},\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}\cdot\mathbf{H}^{-1}$. Substituting it into $\mathbf{g}_2$ gives

$$\mathbf{g}_2 = -\,\nabla^2_{\boldsymbol{\alpha},\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}\cdot\mathbf{H}^{-1}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}. \tag{4}$$

Note that no approximation has been made till now. To compute $\mathbf{g}_2$, the main difficulty lies in $\mathbf{H}^{-1}$ which, due to the high dimensionality of $\mathbf{H}$ (over one million in DARTS), is computationally intractable. We directly replace $\mathbf{H}^{-1}$ with $\mathbf{H}$, which leads to an approximated term:

$$\mathbf{g}_2' = -\eta\cdot\,\nabla^2_{\boldsymbol{\alpha},\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}\cdot\mathbf{H}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}, \tag{5}$$

where $\eta > 0$ is named the **amending coefficient**, the only hyper-parameter of our approach, and its effect will be discussed in the experimental section. A nice property of $\mathbf{g}_2'$ is that the angle between $\mathbf{g}_2$ and $\mathbf{g}_2'$ does not exceed $90°$(*i.e.*, $\langle\mathbf{g}_2',\mathbf{g}_2\rangle \geqslant 0$, as proved in Appendix A.3).

As the final step, we compute $\mathbf{g}_1$ and $\mathbf{g}_2'$ following the *second-order* DARTS (see Appendix A.4). Overall, the computation of Eqn equation 5 requires similar computational costs of the *second-order* DARTS. On an NVIDIA Tesla-V100 GPU, each search epoch requires around 0.02 GPU-days on the standard 8-cell search space on CIFAR10.

2.4 What Makes Our Approach Better Than DARTS?

DARTS fails mostly due to inaccurate estimation of $\mathbf{g}_2$. The *first-order* DARTS directly discarded this term, *i.e.*, setting $\mathbf{g}_2^{\mathrm{1st}} = \mathbf{0}$, and the *second-order* DARTS used $\mathbf{I}$ (the identity matrix) to replace $\mathbf{H}^{-1}$ in Eqn equation 4. Let us denote the approximated term as $\mathbf{g}_2^{\mathrm{2nd}}$, then the property that $\left\langle\mathbf{g}_2^{\mathrm{2nd}},\mathbf{g}_2\right\rangle \geqslant 0$ does not hold. Consequently, there can be a significant gap between the true and estimated values of $\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega}^\star(\boldsymbol{\alpha}),\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$. Such inaccuracy accumulates with every update on $\boldsymbol{\alpha}$, and eventually causes $\boldsymbol{\alpha}$ to converge to the degenerated solutions. Although all (up to 100) our trials of the original DARTS search converge to architectures with all skip-connect operators. We do not find a theoretical explanation why this is the only ending point and leave it as an open problem.

On the contrary, with an amended approximation, $\mathbf{g}_2'$, our approach can survive after a sufficiently long search process. The longest search process in our experiments has 500 epochs, after which the architecture remains mostly the same as that after 100 epochs. In addition, the final architecture seems converged (not changing with more search epochs) as (i) the *none* operator does not dominate any edge; and (ii) the weight of the dominating operator in each edge is still gradually increasing.

The rationality of our approach is also verified by the validation accuracy of the search stage. In a typical search process on CIFAR10, the *first-order* DARTS, by directly discarding $\mathbf{g}_2$, reports an average validation accuracy of 90.5%. The *second-order* DARTS adds $\mathbf{g}_2^{\mathrm{2nd}}$ but reports a reduced validation accuracy. Our approach, by adding $\mathbf{g}_2'$, achieves an improved validation accuracy of 91.5%, higher than both versions of DARTS. This indicates that our approach indeed provides more accurate approximation so that the super-network is better optimized.

Hyper-Parameter Consistency Our goal is to bridge the optimization gap between the search and re-training phases. Besides amending the architectural gradients to avoid ‘over-fitting’ the super-network, another important factor is to make the hyper-parameters used in search and re-training consistent. In the contexts of DARTS, examples include using different depths (*e.g.*, DARTS used 8 cells in search and 20 cells in re-training) and widths (*e.g.*, DARTS used a basic channel number of 16 in search and 36 in re-training), as well as using different training strategies (*e.g.*, during re-training, a few regularization techniques including Cutout (DeVries & Taylor, 2017), Dropout (Srivastava et al., 2014) and auxiliary loss (Szegedy et al., 2015) were used, but none of them appeared in search). More importantly, the final step of search (removing 6 out of 14 edges from the structure) can cause another significant gap. In Section 3.1.1, we will discuss some practical ways to bridge these gaps towards higher stability. For more analysis on this point, please refer to Appendix B.1.

2.5 Discussions and Relationship to Prior Work

A few prior DARTS-based approaches noticed the issue of instability and alleviated it in different ways. For example, P-DARTS (Chen et al., 2019) fixed the number of preserved skip-connect operators, PC-DARTS (Xu et al., 2020) used edge normalization to eliminate the *none* operator, while XNAS (Nayman et al., 2019) and DARTS+ (Liang et al., 2019) introduced a few human expertise to stabilize search. However, we point out that (i) either P-DARTS or PC-DARTS, with carefully designed methods or tricks, can also fail in a long enough search process (more than 200 epochs); and that (ii) XNAS and DARTS+, by adding human expertise, somewhat violated the design principle of AutoML, in which one is expected to avoid introducing too many hand-designed rules.

A recent work (Zela et al., 2020) also tried to robustify DARTS in a simplified search space, and using various optimization strategies, including ℓ_2-regularization, adding Dropout and using early termination. We point out that these methods are still sensitive to initialization and hyper-parameters, while our approach provides a mathematical explanation and enjoys a better convergence property (see Section 3.1.1 and Appendix C.2).

We also draw the connection between our approach and heuristic search methods including using reinforcement learning or an evolutionary algorithm as a controller. From the viewpoint of optimization, these methods are more stable because the optimization of $\boldsymbol{\alpha}$ is decoupled from that of $\boldsymbol{\omega}$, so that it does not require $\boldsymbol{\omega}$ to arrive at $\boldsymbol{\omega}^{*}$, but only need a reasonable approximation of $\boldsymbol{\omega}^{*}$ to predict model performance. Our approach sheds light on introducing a similar property, *i.e.*, robustness to approximated $\boldsymbol{\omega}^{\star}$, which helps in stabilizing differentiable search approaches.

3 EXPERIMENTS

3.1 RESULTS ON CIFAR10

The CIFAR10 dataset (Krizhevsky & Hinton, 2009) has $50{,}000$ training and $10{,}000$ testing images, equally distributed over 10 classes. We mainly use this dataset to evaluate the stability of our approach, as well as analyze the impacts of different search options and parameters.

We search and re-train similarly as DARTS. During the search, all operators are assigned equal weights on each edge. The batch size is set to be 96. An Adam optimizer is used to update architectural parameters, with a learning rate of 0.0003, a weight decay of 0.001 and a momentum of $(0.5, 0.999)$. The number of epochs is to be discussed later. During re-training, the base channel number is increased to 36. An SGD optimizer is used with an initial learning rate starting from 0.025, decaying with cosine annealing, and arriving at 0 after 600 epochs. The weight decay is set to be 0.0003, and the momentum is 0.9. The amending coefficient, η, is set to be 0.1 throughout the experiments. For a detailed analysis on η, please refer to Appendix C.1.

3.1.1 STABILIZED SEARCH RESULTS

First, we investigate the stability of our approach by comparing it (with $\eta = 0.1$) to DARTS (Liu et al., 2019), P-DARTS (Chen et al., 2019), and PC-DARTS (Xu et al., 2020). We run all the competitors for 200 search epochs[2] to guarantee convergence in the final architecture. Results are summarized in Table 1. One can see that all others, except PC-DARTS, produce lower accuracy than that of random search (some of them, DARTS and P-DARTS, even degenerate to all-skip-connect architectures), but our approach survives, indicating the amended approximation effectively boosts search robustness. Moreover, we verify the search stability by claiming a 0.58% advantage over random search.

Architecture	Test Err. (%)	Params (M)	#P
Random Search[†]	3.29	3.2	-
DARTS (*first-order*)	6.18	1.4	0
DARTS (*second-order*)	5.15	1.5	0
P-DARTS	5.38	1.5	0
PC-DARTS	3.15	2.4	3
Our Approach	2.71	3.3	7
w/o amending term	3.15	3.9	6
w/o consistency	3.08	3.3	5

Table 1: The performance of DARTS, P-DARTS, PC-DARTS and our approach after 200 search epochs on CIFAR10. '#P' means the number of parametric operators (any convolution) in the final normal cell. [†]: we borrow the results from DARTS (Liu et al., 2019).

Table 1 also provides an ablation study by switching off the amending term or the consistency of training hyper-parameters (*i.e.*, adding Cutout, Dropout, and the auxiliary loss tower to the search stage with the same parameters, *e.g.*, the Dropout ratio, as they are used in re-training). Without any one of

[2]For P-DARTS, we run each of its three search stages for 100 epochs, and do not use the heuristic rule that preserves exactly two skip-connect operators.

them, the error rate significantly increases to more than 3%, still better than random search but the advantage becomes much weaker. These results verify our motivation, *i.e.*, shrinking the optimization gap from any aspects can lead to better search performance. The architectures with and without unified hyper-parameters are shown in Figure 2 (top) and Figure 5 (middle), respectively.

We also perform experiments in the simplified search space defined by (Zela et al., 2020). Without bells and whistles, we obtain an error rate of 2.55% which is significantly better than random search (3.05%) reported in the paper. Please refer to Appendix C.2 for more results under this setting.

3.1.2 Exploring More Complex Search Spaces

Driven by the benefit of shrinking the optimization gap, we further apply two modifications. **First**, to avoid edge removal, we partition the search stage into two sub-stages: the former chooses 8 active edges from the 14 candidates, and the latter, restarting from scratch, determines the operators on each preserved edge. Technical details are provided in Appendix B.2. Another option that can save computational costs is to fix the edges in each cell, *e.g.*, each node i is connected to node $i-1$ and the least indexed node (denoted by c_{k-2} in most conventions). Note that our approach also works well with all 14 edges preserved, but we have used 8 edges to be computationally fair to DARTS. **Second**, we use the same width (*i.e.*, the number of basic channels, 36) and depth (*i.e.*, the number of cells, 20) in both search and re-training. The modification on depth reminds us of the *depth gap* (Chen et al., 2019) between search and re-training (the network has 8 cells in search, but 20 cells in re-training). Instead of using a progressive search method, we directly search in an augmented space (see the next paragraph), thanks to the improved stability of our approach.

We denote the original search space used in DARTS as $\mathcal{S}_1$, which has six normal cells and two reduction cells, and all cells of the same type share the architectural parameters. This standard space contains 1.1×10^{18} distinct architectures. We also explore a more complex search space, denoted by $\mathcal{S}_2$, in which we relax the constraint of sharing architectural parameters, meanwhile the number of cells increases from 8 to 20, *i.e.*, the same as in the re-training stage. Here, since the GPU memory is limited, we cannot search with all seven operators, so we only choose two, namely skip-connect and sep-conv-3x3, which have very different properties. This setting allows a total of 1.9×10^{93} architectures to appear, significantly surpassing the capacity of most existing cell-based search spaces.

The searched results in $\mathcal{S}_1$ and $\mathcal{S}_2$ with fixed or searched edges are shown in Figure 2, and their performance summarized in Table 2. By directly searching in the target space, $\mathcal{S}_2$, the error is further reduced from 2.71% to 2.60% and 2.63% with fixed and searched edges, respectively. The improvement seems small on CIFAR10, but when we transfer these architectures to ImageNet, the corresponding advantages become more significant (0.4%, see Table 3).

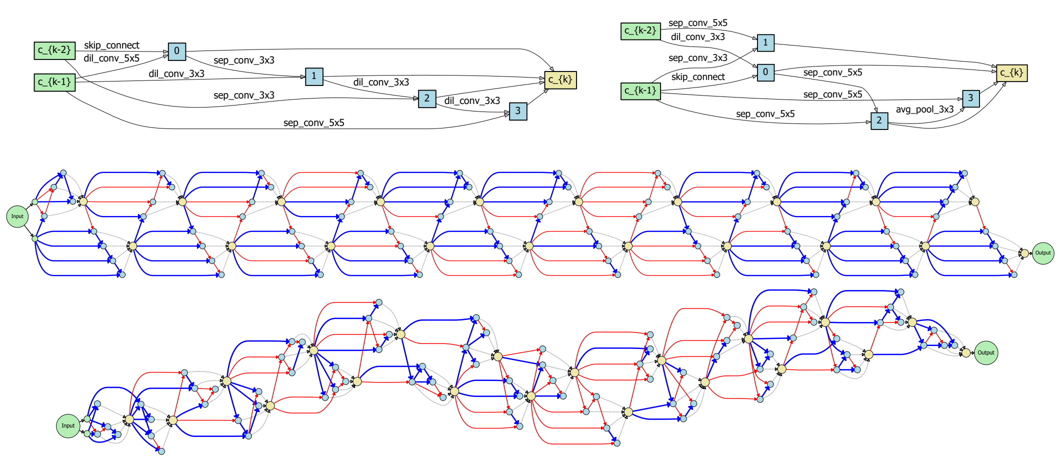


Figure 2: **Top**: the normal and reduction cells found in $\mathcal{S}_1$ with edge pruning, *i.e.*, following the original DARTS to search on full (14) edges. **Middle** & **Bottom**: the architecture found in $\mathcal{S}_2$ with fixed and searched edges, in which red thin, blue bold, and black dashed arrows indicate skip-connect, sep-conv-3x3, and concatenation, respectively. *This figure is best viewed in color.*

Architecture	Test Err. (%)	Params (M)	Search Cost (GPU-days)	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	-	manual
ENAS (Pham et al., 2018) w/ Cutout	2.89	4.6	0.5	RL
NAONet-WS (Luo et al., 2018)	3.53	3.1	0.4	NAO
AmoebaNet-B (Real et al., 2019) w/ Cutout	2.55±0.05	2.8	3150	evolution
SNAS (moderate) (Xie et al., 2018) w/ Cutout	2.85±0.02	2.8	1.5	gradient-based
DARTS (*first-order*) (Liu et al., 2019) w/ Cutout	3.00±0.14	3.3	0.4	gradient-based
DARTS (*second-order*) (Liu et al., 2019) w/ Cutout	2.76±0.09	3.3	1.0	gradient-based
P-DARTS (Chen et al., 2019) w/ Cutout	2.50	3.4	0.3	gradient-based
PC-DARTS (Xu et al., 2020) w/ Cutout	2.57±0.07	3.6	0.1	gradient-based
Amended-DARTS, $\mathcal{S}_1$, pruned edges, w/ Cutout	2.71±0.09	3.3	1.7	gradient-based
Amended-DARTS, $\mathcal{S}_2$, fixed edges, w/ Cutout	2.60±0.15	3.6	1.1	gradient-based
Amended-DARTS, $\mathcal{S}_2$, searched edges, w/ Cutout	2.63±0.09	3.0	3.1*	gradient-based

Table 2: Comparison with the state-of-the-arts on CIFAR10. *: Edge search, the (optional) first step of our approach, takes 2.1 out of 3.1 GPU-days (please refer to Appendix B.3 for details).

We also execute DARTS (with early termination, otherwise it fails dramatically) and random search on $\mathcal{S}_2$ with the fixed-edge setting, and they report 0.25% and 0.29% deficits compared to our approach (DARTS is slightly better than random search). When we transfer these architectures to ImageNet, the deficits become more significant (1.7% and 0.8%, respectively, and DARTS performs even worse). This provides a side evidence to randomly-wired search (Xie et al., 2019), advocating for the importance of designing stabilized approach on large search spaces.

3.1.3 Comparison to the State-of-the-Arts

Finally, we compare our approach with recent approaches, in particular, differentiable ones. Result are shown in Table 2. Our approach produces competitive results among state-of-the-arts, although it does not seem to beat others. We note that existing approaches often used additional tricks, *e.g.*, P-DARTS assumed a fixed number of skip-connect operators, which shrinks the search space (so as to guarantee stability). More importantly, all these differentiable search approaches must be terminated in an early stage, which makes them less convincing as search has not arrived at convergence. These tricks somewhat violate the ideology of neural architecture search; in comparison, our approach, though not producing the best performance, promises a more theoretically convinced direction.

3.2 Results on ImageNet

We use ILSVRC2012 (Russakovsky et al., 2015), a subset of ImageNet (Deng et al., 2009), for experiments. It has $1{,}000$ classes, 1.3M training images, and 50K testing images. We directly use the architectures searched on CIFAR10 and compute a proper number of basic channels, so that the FLOPs of each architecture does not exceed 600M, *i.e.*, the mobile setting. During re-training, there are a total of 250 epochs. We use an SGD optimizer with an initial learning rate of 0.5 (decaying linearly after each epoch), a momentum of 0.9 and a weight decay of 3×10^{-5}.

Architecture	Test Err. (%)		Params	$\times+$	Search Cost	Search Method
	top-1	top-5	(M)	(M)	(GPU-days)	
ShuffleNet 2× (v2) (Ma et al., 2018)	25.1	-	∼5	591	-	manual
MnasNet-92 (Tan et al., 2019)	25.2	8.0	4.4	388	-	RL
PNAS (Liu et al., 2018)	25.8	8.1	5.1	588	225	SMBO
AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	570	3150	evolution
SNAS (mild) (Xie et al., 2018)	27.3	9.2	4.3	522	1.5	gradient-based
DARTS (*second-order*) (Liu et al., 2019)	26.7	8.7	4.7	574	4.0	gradient-based
P-DARTS (CIFAR10) (Chen et al., 2019)	24.4	7.4	4.9	557	0.3	gradient-based
PC-DARTS (Xu et al., 2020)‡	24.2	7.3	5.3	597	3.8	gradient-based
Amended-DARTS, $\mathcal{S}_1$, pruned edges	24.7	7.6	5.2	586	1.7	gradient-based
Amended-DARTS, $\mathcal{S}_2$, fixed edges	24.3	7.4	5.5	590	1.1	gradient-based
Amended-DARTS, $\mathcal{S}_2$, searched edges	24.3	7.3	5.2	596	3.1	gradient-based

Table 3: Comparison with the state-of-the-arts on ILSVRC2012, under the *mobile setting*. ‡: these architectures are searched on ImageNet.

The comparison of our approach to existing work is shown in Table 3. In the augmented search space, $\mathcal{S}_2$, our approach reports a top-1 error rate of 24.3% without either AutoAugment (Cubuk et al., 2019) or Squeeze-and-Excitation (Hu et al., 2018). This result, obtained after search convergence, is competitive among state-of-the-arts. In comparison, without the amending term, DARTS converges to a weird architecture in which some cells are mostly occupied by skip-connect and some others by sep-conv-3x3. This architecture reports a top-1 error of 26.0%, which is even inferior to random search (25.1%). Last but not least, the deficit of $\mathcal{S}_1$, compared to $\mathcal{S}_2$, becomes more significant on ImageNet. This verifies the usefulness of shrinking the optimization gap in challenging datasets.

Architecture	Perplexity		(GPU-days)
	val	test	
NAS (Zoph & Le, 2017)	-	64.0	$\sim$10000
ENAS (Pham et al., 2018)[†]	60.8	58.6	0.5
Random search baseline[†]	61.8	59.4	2.0
DARTS (first order) (Liu et al., 2019)	60.2	57.6	0.5
DARTS (second order) (Liu et al., 2019)	58.1	55.7	1.0
GDAS (Dong & Yang, 2019)	59.8	57.5	0.4
NASP (Yao et al., 2020)	59.9	57.3	0.1
R-DARTS (Zela et al., 2020)	-	57.6	-
Amended-DARTS	**57.1**	**54.8**	1.0

Table 4: Comparison among NAS approaches on Penn Treebank. [†] Results borrowed from (Liu et al., 2019).

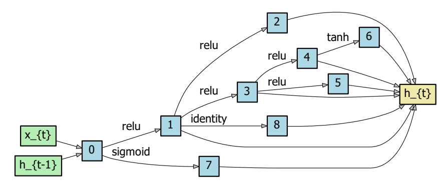


Figure 3: The recurrent cell searched by our approach on the Penn Treebank dataset. The random seed is 3.

3.3 Results on Penn Treebank

We also evaluate our approach on the Penn Treebank dataset, a popular language modeling task on which the recurrent cell connecting LSTM units are being searched (Zoph & Le, 2017; Pham et al., 2018; Liu et al., 2019). We follow the implementation of DARTS to build the search pipeline and make two modifications, namely, (i) amending the second-order term according to Eqn equation 5, (ii) unify the weight decay and variational Dropout ratio between search and evaluation. The search process continues till the architecture does not change for sufficiently long, and the evaluation stage is executed for $8{,}000$ epochs (*i.e.*, until convergence, following the released code of DARTS), on which the best snapshot on the validation set is transferred to the test set.

Results are shown in Table 4, and the searched recurrent cell shown in Figure 3. Note that the DARTS paper reported 58.1/55.7 validation/test perplexity (ppl), but our reproduction obtains 58.5/56.3, slightly lower than the original implementation. Our approach with amended gradients reports 57.1/54.8, showing a significant gain over the baseline. As far as we know, this is the **best** results ever reported in the DARTS space. Prior DARTS-based approaches mostly reported worse results than the original DARTS (see Table 4), or restricted to image-level operations (*e.g.*, PC-DARTS (Xu et al., 2020)), but our approach is a fundamental improvement over DARTS that boosts both computer vision and language modeling tasks. More importantly, we emphasize that after gradient fixation, the search process becomes more robust: we have achieved similar performance, in terms of ppl, using a few different random seeds, while the original DARTS seems quite sensitive to the random seed.

4 Conclusions

In this paper, we present an effective approach for stabilizing DARTS, the state-of-the-art differentiable search method. Our motivation comes from that DARTS-based approaches mostly converge to all-skip-connect architectures when they are executed for a sufficient number of epochs. We analyze this weird phenomenon mathematically and find the reason to be in the dramatic inaccuracy in gradient computation of the architectural parameters. With an alternative approximation based on the optimality of the network parameters, for the first time we prove a bounded estimation error. In both image classification and language modeling, our approach shows improved stability, with which we are able to explore much larger search spaces and obtain better performance.

Our research sheds light on NAS research in several aspects. **First**, we reveal the importance of proper approximation in differentiable architecture search. **Second**, we put forward the usefulness of hyper-parameter consistency in improving search results. **Third**, thanks to improved stability, our algorithm can explore larger search spaces, which we believe is the future trend of NAS.

References

H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. In *AAAI Conference on Artificial Intelligence*, 2018.

H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. 2019.

X. Chen, L. Xie, J. Wu, and Q. Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *International Conference on Computer Vision*, 2019.

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In *International Conference on Machine Learning*, 2020.

E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. In *Computer Vision and Pattern Recognition*, 2019.

J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, 2009.

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

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Computer Vision and Pattern Recognition*, 2019.

Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, 2016.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *International Conference on Computer Vision*, 2019.

J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Computer Vision and Pattern Recognition*, 2018.

G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Computer Vision and Pattern Recognition*, 2017.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

H. Liang, S. Zhang, J. Sun, X. He, W. Huang, K. Zhuang, and Z. Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. J. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *European Conference on Computer Vision*, 2018.

H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

R. Luo, F. Tian, T. Qin, E. Chen, and T. Y. Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, 2018.

N. Ma, X. Zhang, H. T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *European Conference on Computer Vision*, 2018.

N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik-Manor. Xnas: Neural architecture search with expert advice. *arXiv preprint arXiv:1906.08031*, 2019.

Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. Asap: Architecture search, anneal and prune. In *International Conference on Artificial Intelligence and Statistics*, 2020.

H. Pham, M. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, 2018.

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 *International Conference on Machine Learning*, 2017.

E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition*, 2015.

M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Computer Vision and Pattern Recognition*, 2019.

L. Xie and A. L. Yuille. Genetic CNN. In *International Conference on Computer Vision*, 2017.

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

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *International Conference on Computer Vision*, 2019.

Y. Xu, L. Xie, X. Zhang, X. Chen, G. J. Qi, Q. Tian, and H. Xiong. Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. In *International Conference on Learning Representations*, 2020.

Q. Yao, J. Xu, W. W. Tu, and Z. Zhu. Efficient neural architecture search via proximal iterations. In *AAAI Conference on Artificial Intelligence*, 2020.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Computer Vision and Pattern Recognition*, 2018.

A MATHEMATICAL PROOFS AND ANALYSES

In this section, we provide some details to complement the theoretical part of the main article.

Recall that the main goal of this paper is to compute $\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}),\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$, the gradient with respect to the architectural parameters, $\boldsymbol{\alpha}$. It is composed into the sum of $\mathbf{g}_1+\mathbf{g}_2$, and we use $\mathbf{g}_2'$ to approximate $\mathbf{g}_2$. The form of $\mathbf{g}_1$, $\mathbf{g}_2$ and $\mathbf{g}_2'$ is:

$$\mathbf{g}_1 = \nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}. \tag{6}$$

$$\begin{aligned}\mathbf{g}_2 &= \nabla_{\boldsymbol{\alpha}}\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t} \\ &= -\nabla^2_{\boldsymbol{\alpha},\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}\cdot\mathbf{H}^{-1}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}.\end{aligned} \tag{7}$$

$$\mathbf{g}_2' = -\eta\cdot\nabla^2_{\boldsymbol{\alpha},\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega},\boldsymbol{\alpha})\big|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}\cdot\mathbf{H}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}. \tag{8}$$

A.1 OUR OPINIONS ON THE OPTIMIZATION GAP

This part corresponds to Section 2.2 in the main article.

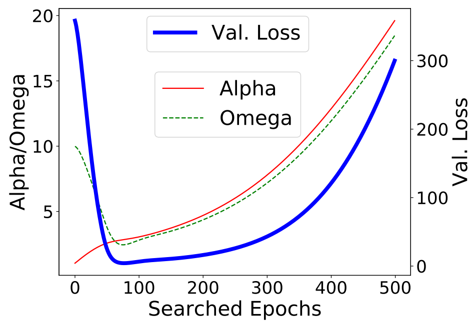
In (Zela et al., 2020), the authors believed that a super-network with a higher validation accuracy always has a higher probability of generating strong sub-networks, and they owed the weird behavior of DARTS to the discretization step (preserving the operator with the highest weight on each edge and discarding others) after the differentiable search phase. Here, we provide a different opinion, detailed as follows:

1. **A high validation accuracy does not necessarily indicate a better super-network.** In training a weight-sharing super-network, *e.g.*, in DARTS, the improvement of validation accuracy can be brought by two factors, namely, the super-network configuration (corresponding to $\boldsymbol{\alpha}$) becomes better or the network weights (corresponding to $\boldsymbol{\omega}$) are better optimized. We perform an intuitive experiment in which we fix the initialized $\boldsymbol{\alpha}$ and only optimize $\boldsymbol{\omega}$. After 50 epochs, the validation accuracy of the super-network is boosted from 10% (random guess) to 88.28%. While the performance seems competitive among a regular training process, the sampled sub-network is totally random.
2. **The advantage of our approach persists even without discretization-and-pruning.** From another perspective, we try to skip the discretization-and-pruning step at the end of the search stage and re-train the super-network (with $\boldsymbol{\alpha}$ fixed) directly. To reduce computational costs, we use a shallower super-network, and perform both search and re-training for 600 epochs on CIFAR10 to guarantee convergence. Without the amending term, the validation error in the search stage and the testing error in the re-training stage are 12.8% and 7.4%, respectively, and these numbers become 10.5% and 5.4% after the amending term is added. This indicates that amending the gradient estimation indeed improves the super-network, and the advantage persists even when discretization-and-pruning is not used.
3. **Inaccuracy of gradient estimation also contributes to poor performance.** As detailed in Section A.2, we construct a toy example that applies bi-level optimization to the loss function of $\mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha};\boldsymbol{x})=(\boldsymbol{\omega}\boldsymbol{x}-\boldsymbol{\alpha})^2$. We show that even when $\boldsymbol{\omega}^{\star}$ is achieved at every iteration, the gradient of $\boldsymbol{\alpha}$, $\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}),\boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$, can be totally incorrect. When this happens in architecture search, the super-network can be pushed towards sub-optimal or even random architectures, while the validation accuracy can continue growing.

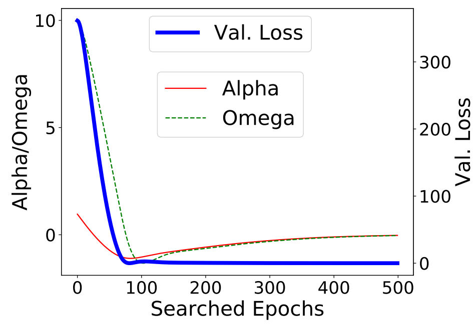
Therefore, our opinion is that the 'optimization gap' is brought by the inconsistency between search and re-training – edge pruning after search is one aspect, and the inaccuracy of gradient estimation is another one which seems more important. We alleviate the pruning issue by first selecting (or fixing) a few edges and then determining the operator on each preserved edge. However, such a two-stage search process can be very unstable if the gradient estimation remains inaccurate. In other words, amending errors in gradient computation lays the foundation of hyper-parameter consistency.

A.2 IMPORTANCE OF THE SECOND-ORDER GRADIENT TERM, $\mathbf{g}_2$

This part corresponds to Section 2.3 in the main article.



(a) A search process without the amending term.



(b) A search process with the amending term.

Figure 4: Two search processes with a simple loss function $\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\alpha}; \mathbf{x}) = (\boldsymbol{\omega} \cdot \mathbf{x} - \boldsymbol{\alpha})^2$, with all of $\boldsymbol{\omega}$, $\boldsymbol{\alpha}$ and $\mathbf{x}$ being 1-dimensional scalars. Red, green and blue curves indicate the value of $\boldsymbol{\alpha}$, $\boldsymbol{\omega}$ and the validation loss, respectively.

DARTS (Liu et al., 2019) owed the inaccuracy in computing $\mathbf{g}_2$ to that $\boldsymbol{\omega}^\star(\boldsymbol{\alpha})$ is difficult to arrive at (*e.g.*, requiring a lot of computation), and believed that $\mathbf{g}_2$ goes to $\mathbf{0}$ when the optimum is achieved. We point out that this is not correct even in a very simple example of convex optimization, detailed as follows.

Let the loss function be $\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\alpha}; \mathbf{x}) = (\boldsymbol{\omega} \cdot \mathbf{x} - \boldsymbol{\alpha})^2$. Then, the only difference between $\mathcal{L}_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha}) = \mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\alpha}; \mathbf{x}_{\text{train}})$ and $\mathcal{L}_{\text{val}}(\boldsymbol{\omega}, \boldsymbol{\alpha}) = \mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\alpha}; \mathbf{x}_{\text{val}})$ lies in the input, $\mathbf{x}$. Assume that the training dataset contains a sample, $\mathbf{x}_{\text{train}} = \mathbf{1}$ and validation dataset contains another sample, $\mathbf{x}_{\text{val}} = \mathbf{2}$. It is easy to derive that the local optimum of $\mathcal{L}_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha})$ is $\boldsymbol{\omega}^\star(\boldsymbol{\alpha}) = \boldsymbol{\alpha}$. Substituting $\mathbf{x}_{\text{val}} = \mathbf{2}$ into the loss function yields $\mathcal{L}_{\text{val}}(\boldsymbol{\omega}, \boldsymbol{\alpha}) = (\boldsymbol{\omega} \cdot \mathbf{2} - \boldsymbol{\alpha})^2$. When $\boldsymbol{\alpha} = \boldsymbol{\alpha}_t$ ($\boldsymbol{\alpha}_t$ denotes the current status of $\boldsymbol{\alpha}$), $\boldsymbol{\omega}$ arrives at $\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t)$, $\mathbf{g}_1 = 2(\boldsymbol{\alpha}_t - 2\boldsymbol{\alpha}_t) = -2\boldsymbol{\alpha}_t$ and $\mathbf{g}_2 = 4\boldsymbol{\alpha}_t \neq \mathbf{0}$. In other words, when $\boldsymbol{\omega}$ arrives at $\boldsymbol{\omega}^\star(\boldsymbol{\alpha})$, $\mathcal{L}_{\text{val}}(\boldsymbol{\omega}^\star(\boldsymbol{\alpha}), \boldsymbol{\alpha}) = \boldsymbol{\alpha}^2$, we have $\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\text{val}}(\boldsymbol{\omega}^\star(\boldsymbol{\alpha}), \boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t} = 2\boldsymbol{\alpha}_t \neq \mathbf{g}_1$.

In summary, $\mathbf{g}_2$ will not be zero even when $\boldsymbol{\omega}^\star$ is achieved, which implies that ignoring $\mathbf{g}_2$ may lead to a wrong direction of optimization – in the toy example above, the loss will increase monotonically towards infinity if $\mathbf{g}_2$ is ignored. We believe that this inaccuracy also contributed to the weird phenomena in real-world data (*i.e.*, when DARTS was run on CIFAR10, the optimization process collapses to a dummy network architecture which produces poor classification results).

To provide better understanding of the search process, we plot the statistics from two search processes, without and with the amending term, in Figure 4. Without the amending term, *i.e.*, the computation of $\mathbf{g}_2$ suffers considerable inaccuracy, $\boldsymbol{\alpha}$ (shown as the red curve) is always going towards an incorrect direction. The validation loss is first reduced (because the initialized $\boldsymbol{\omega}$ is poor and thus in the first half of the search process, the accuracy improvement brought by $\boldsymbol{\omega}$ surpasses the accuracy drop brought by $\boldsymbol{\alpha}$), but, as the search process continues, the first-order gradient is reduced and the inaccuracy of $\mathbf{g}_2$ dominates $\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\text{val}}(\boldsymbol{\omega}^\star(\boldsymbol{\alpha}), \boldsymbol{\alpha})|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$. Consequently, both parameters are quickly pushed away from the optimum and the search process 'goes wild' (*i.e.*, the loss 'converges' to infinity). This problem is perfectly solved after the amending term is applied.

A.3 Proof of $\langle \mathbf{g}_2', \mathbf{g}_2 \rangle \geqslant 0$

This part corresponds to Section 2.3 in the main article.

Substituting $\mathbf{g}_2$ and $\mathbf{g}_2'$ into $\langle \mathbf{g}_2', \mathbf{g}_2 \rangle$ gives:

$$\langle \mathbf{g}_2', \mathbf{g}_2 \rangle = \eta \cdot \nabla_{\boldsymbol{\omega}} \mathcal{L}_{\text{val}}(\boldsymbol{\omega}, \boldsymbol{\alpha})\,|^\top_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t} \cdot \mathbf{H}^{-1} \cdot \mathbf{A} \cdot \mathbf{H} \cdot \nabla_{\boldsymbol{\omega}} \mathcal{L}_{\text{val}}(\boldsymbol{\omega}, \boldsymbol{\alpha})\,|_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}, \quad (9)$$

where $\mathbf{A} = \nabla^2_{\boldsymbol{\omega},\boldsymbol{\alpha}} \mathcal{L}_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha})\,|^\top_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t} \cdot \nabla^2_{\boldsymbol{\alpha},\boldsymbol{\omega}} \mathcal{L}_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha})\,|_{\boldsymbol{\omega}=\boldsymbol{\omega}^\star(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$, is a semi-positive-definite matrix. According to the definition of Hesse matrix, $\mathbf{H}$ is a real symmetric positive-definite matrix.

Let $\{\boldsymbol{\psi}_k\}$ be an orthogonal set of eigenvectors with respect to $\mathbf{A}$ (*i.e.*, $\boldsymbol{\psi}_k^\top \cdot \boldsymbol{\psi}_k = 1$), and $\{\lambda_k\}$ be the corresponding eigenvalues satisfying $\lambda_k \geqslant \mathbf{0}$. Let $\boldsymbol{\phi}_l$ be an eigenvector of $\mathbf{H} \cdot \mathbf{A} \cdot \mathbf{H}^{-1} + \mathbf{H}^{-1} \cdot \mathbf{A} \cdot \mathbf{H}$, and expanding $\mathbf{H}\cdot\boldsymbol{\phi}_l$ and $\mathbf{H}^{-1}\cdot\boldsymbol{\phi}_l$ based on $\{\boldsymbol{\psi}_k\}$ derives the following equation for any l:

$$\boldsymbol{\phi}_l=\sum_{k=1}^{K}a_k\cdot(\mathbf{H}\cdot\boldsymbol{\psi}_k)=\sum_{k=1}^{K}b_k\cdot\left(\mathbf{H}^{-1}\cdot\boldsymbol{\psi}_k\right), \tag{10}$$

where $\{a_k\}$ and $\{b_k\}$ are the corresponding expansion coefficients. According to the definition of eigenvalues, we have:

$$\left(\mathbf{H}\cdot\mathbf{A}\cdot\mathbf{H}^{-1}+\mathbf{H}^{-1}\cdot\mathbf{A}\cdot\mathbf{H}\right)\cdot\boldsymbol{\phi}_l=\mu_l\cdot\boldsymbol{\phi}_l, \tag{11}$$

where μ_l is the corresponding eigenvalue of $\boldsymbol{\phi}_l$. Substituting Eqn equation 10 into Eqn equation 11, we obtain:

$$\sum_{k=1}^{K}a_k\cdot\lambda_k\left(\mathbf{H}\cdot\boldsymbol{\psi}_k\right)+\sum_{k=1}^{K}b_k\cdot(\lambda_k-\mu_l)\cdot\left(\mathbf{H}^{-1}\cdot\boldsymbol{\psi}_k\right)=\mathbf{0}. \tag{12}$$

Left-multiplying Eqn equation 12 by $\left[\sum_{k=1}^{K}b_k\cdot(\lambda_k-\mu_l)\cdot\left(\mathbf{H}^{-1}\cdot\boldsymbol{\psi}_k\right)\right]^{\top}$ obtains:

$$\sum_{k=1}^{K}(\lambda_k-\mu_l)\cdot\lambda_k\cdot a_k\cdot b_k=-\left\|\sum_{k=1}^{K}b_k\cdot(\lambda_k-\mu_l)\cdot\left(\mathbf{H}^{-1}\cdot\boldsymbol{\psi}_k\right)\right\|_2^2\leqslant 0. \tag{13}$$

$\mathbf{A}$ is a real symmetric matrix sized $M_1\times M_1$, and it is obtained by multiplying a $M_1\times M_2$ matrix to a $M_2\times M_1$. Here, M_1 and M_2 are the dimensionality of $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$, respectively, and M_1 is often much larger than M_2 (*e.g.*, millions vs. hundreds in DARTS). Hence, $\mathbf{A}$ has much fewer different eigenvalues compared to its dimensionality, and so we can choose many sets of $\{\boldsymbol{\alpha}_k\}$ which are orthogonal to each other, and generate many sets of $\{a_k\}$ and $\{b_k\}$ satisfying $\sum_{k=1}^{K}\mathbf{a}_k\mathbf{b}_k=\boldsymbol{\beta}^{\top}\cdot\boldsymbol{\beta}\geqslant 0$. That being said, in most of time, we can choose one set of eigenvectors from the subspace, $\{\boldsymbol{\alpha}_k\}$, and the elements within are orthogonal to each other, which makes $\mathbf{a}_k\mathbf{b}_k\geqslant 0$ in most cases, and hence $\mu_l\geqslant 0$ for any l.

Since all of the eigenvalues with respect to the real symmetric matrix $\mathbf{H}\cdot\mathbf{A}\cdot\mathbf{H}^{-1}+\mathbf{H}^{-1}\cdot\mathbf{A}\cdot\mathbf{H}$ is not smaller than zero, so it is semi-positive-definite. This derives that $\mathbf{H}^{-1}\cdot\mathbf{A}\cdot\mathbf{H}$ is semi-positive-definite, and thus $\langle\mathbf{g}_2',\mathbf{g}_2\rangle\geqslant 0$.

A.4 Computing $\mathbf{g}_2'$ Using Finite Difference Approximation

This part corresponds to Section 2.3 in the main article.

Although we avoid the computation of $\mathbf{H}^{-1}$, the inverse matrix of $\mathbf{H}$, computing $\mathbf{H}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})|_{\boldsymbol{\omega}=\boldsymbol{\omega}^{\star}(\boldsymbol{\alpha}_t),\boldsymbol{\alpha}=\boldsymbol{\alpha}_t}$ in $\mathbf{g}_2'$ is still a problem since the size of $\mathbf{H}$ is often very large, *e.g.*, over one million. We use finite difference approximation just as used in the *second-order* DARTS (Liu et al., 2019).

Let $\boldsymbol{\epsilon}$ be a small scalar, which is set to be $\mathbf{0.01}/\left\|\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})\right\|_2$ as in the original DARTS. Using finite difference approximation to compute the gradient around $\boldsymbol{\omega}_1=\boldsymbol{\omega}+\boldsymbol{\epsilon}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})$ and $\boldsymbol{\omega}_2=\boldsymbol{\omega}-\boldsymbol{\epsilon}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})$ gives:

$$\mathbf{H}\cdot\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{val}}(\boldsymbol{\omega},\boldsymbol{\alpha})\approx\frac{\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega}_1,\boldsymbol{\alpha})-\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega}_2,\boldsymbol{\alpha})}{2\boldsymbol{\epsilon}}. \tag{14}$$

Let $\boldsymbol{\omega}_3=\boldsymbol{\omega}+\frac{1}{2}\left[\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega}_1,\boldsymbol{\alpha})-\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega}_2,\boldsymbol{\alpha})\right]$ and $\boldsymbol{\omega}_4=\boldsymbol{\omega}-\frac{1}{2}\left[\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega}_1,\boldsymbol{\alpha})-\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega}_2,\boldsymbol{\alpha})\right]$, we have:

$$\mathbf{g}_2'\approx-\boldsymbol{\eta}\times\frac{\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega}_3,\boldsymbol{\alpha})-\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathrm{train}}(\boldsymbol{\omega}_4,\boldsymbol{\alpha})}{2\boldsymbol{\epsilon}}. \tag{15}$$

B Technical Details and Search Costs

B.1 More about Hyper-Parameter Consistency

An important contribution of this paper is to reveal the need of hyper-parameter consistency, *i.e.*, using the same set of hyper-parameters, including the depth and width of the super-network, the

Dropout ratio, whether to use the auxiliary loss term, **not** to prune edges at the end of search, *etc*. This is a natural idea in both theory and practice, but most existing work ignored it, possibly because its importance was hidden behind the large optimization gap brought by the inaccurate gradient estimation in the bi-level optimization problem.

We point out that even when bi-level optimization works accurately, we may miss the optimal architecture(s) without hyper-parameter consistency. This is because each hyper-parameter will more or less change the value of the loss function and therefore impact the optimal architecture. We provide a few examples here.

- The most sensitive change may lie in the edge pruning process, *i.e.*, preserving 8 out of 14 edges in each cell and eliminating others. This may incur significant accuracy drop even when only one edge gets pruned. For example, in the normal cell searched on the $\mathcal{S}_1$-A space (see Figure 9), removing the skip-connect operator will cause the re-training error to increase from $2.71\% \pm 0.09$ to $3.36\% \pm 0.08$. The dramatic performance drop is possibly due to the important role played by the pruned edge, *e.g.*, the pruned skip-connect may contribute to rapid information propagation in network training.
- Some hyper-parameters will potentially change the optimal architecture. The basic channel width and network depth are two typical examples. When the channel number is small, the network may expect large convolutional kernels to guarantee a reasonable amount of trainable parameters; also, a deep architecture may lean towards small convolutional kernels since the receptive field is not a major bottleneck. However, the original DARTS did not unify these quantities in search and re-training, which often results in sub-optimality as noticed in (Chen et al., 2019).
- Other hyper-parameters, in particular those related to the training configuration, are also important. For example, if we expect Dropout to be used during the re-training stage, the optimal architecture may contain a larger number of trainable parameters; if the auxiliary loss tower is used, the optimal architecture may be deeper. Without hyper-parameter consistency, these factors cannot be taken into consideration.

B.2 DETAILS OF THE TWO-STAGE SEARCH PROCESS

This part corresponds to Section 3.1 in the main article.

To avoid the optimization gap brought by edge pruning, we adopt a two-stage search process in which we perform edge search followed by operator search. In the DARTS setting, the first stage involves preserving 8 out of 14 edges to be preserved. For each node, indexed j, we use $\mathcal{E}_j$ to denote the set of all combinations of edges (in the DARTS setting, each node preserves two input edges, so $\mathcal{E}_j$ contains all (i_1, i_2) pairs with $0 \leqslant i_1 < i_2 < j$). The output of node j is therefore computed as:

$$\mathbf{y}_j = \sum_{(i_1,i_2)\in\mathcal{E}_j} \frac{\exp\left(\beta^{(i_1,i_2)}\right)}{\sum_{(i_1',i_2')\in\mathcal{E}_j} \exp\left(\beta^{(i_1',i_2')}\right)} \cdot (\mathbf{y}_{i_1} + \mathbf{y}_{i_2}). \tag{16}$$

where $\beta^{(i_1,i_2)}$ denotes the edge-selection parameter of the node combination of (i_1, i_2). This mechanism is similar to the edge normalization introduced in (Xu et al., 2020) but we take the number of preserved inputs into consideration.

After edge selection is finished, a regular operator search process, starting from scratch, follows on the preserved edges to determine the final architecture.

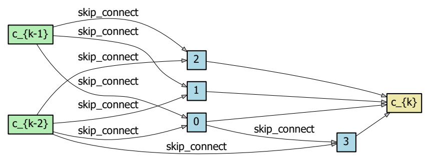
B.3 SEARCH COST ANALYSIS

This part corresponds to Section 3.1 in the main article.

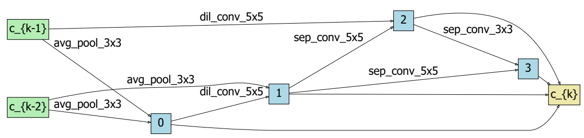
Note that the first (edge selection) stage requires around $2\times$ computational costs compared to the second (operator selection) stage. This is because edge selection works on 14 edges while operator selection on 8 edges. Fortunately, the former stage can be skipped (at no costs) if we choose to search operators on a fixed edge configuration, which also reports competitive search performance.

Search Method	Test Err. (%)					Params (M)		
	Seed #1	**Seed #2**	**Seed #3**	**average**	**best**	**Seed #1**	**Seed 2**	**Seed #3**
Random Search†	-	-	-	-	2.95	-	-	-
DARTS (early termination)†	-	-	-	3.71±1.14	3.07	-	-	-
R-DARTS (Dropout)†								
Drop Prob. = 0.0	5.30	3.73	3.34	4.12±1.04	3.34	2.21	2.43	2.85
Drop Prob. = 0.2	3.22	3.16	3.26	3.21±0.05	3.16	3.62	4.04	2.99
Drop Prob. = 0.4	2.93	3.15	3.52	3.20±0.30	2.93	4.10	3.74	3.38
Drop Prob. = 0.6	3.09	5.44	3.56	4.03±1.24	3.09	4.46	2.30	2.66
Our Approach, pruned edges	2.76±0.10	2.64±0.07	2.68±0.09	2.69±0.06	2.64±0.07	3.09	3.21	3.40
Our Approach, searched edges	2.58±0.07	2.55±0.13	2.76±0.04	2.63±0.11	2.55±0.13	3.84	3.21	2.85

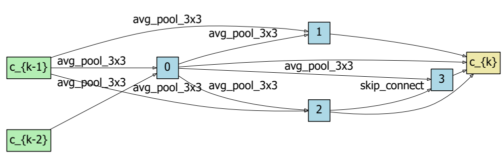
Table 5: Comparison between DARTS (early termination), R-DARTS (Dropout) and our approach in the reduced search space of $\mathcal{S}_3$ with different seeds. †: we borrow the experimental results from (Zela et al., 2020).



(a) $\eta \leqslant 0.01$, CIFAR10 Test Error: 6.18%



(b) $\eta = 0.1$, CIFAR10 Test Error: 3.08%



(c) $\eta \geqslant 1$, CIFAR10 Test Error: 7.16%

Figure 5: The normal cells (in the standard DARTS space) obtained by different amending coefficients.

C ADDITIONAL EXPERIMENTAL RESULTS

C.1 IMPACT OF THE AMENDING COEFFICIENT

This part corresponds to Section 3.1 in the main article.

We first investigate how the amending coefficient, η, defined in Eqn equation 5, impacts architecture search. To arrive at convergence, we run the search stage for 500 epochs. We evaluate different η values from 0 to 1, and the architectures corresponding to small, medium and large amending coefficients are summarized in Figure 5.

We can see that after 500 epochs, $\eta = 0.1$ produces a reasonable architecture that achieves an error rate of 3.08% on CIFAR10. Actually, even with more search epochs, this architecture is not likely to change, as the preserved operator on each edge has a weight not smaller than 0.5, and most of these weights are still growing gradually.

When η is very small, *e.g.*, $\eta = 0.001$ or $\eta = 0.01$, the change brought by this amending term to architecture search is negligible, and our approach shows almost the same behavior as the *first-order* version of DARTS, *i.e.*, $\eta = 0$. In addition, in such scenarios, although the search process eventually runs into an architecture with all skip-connect operators, the number of epochs needed increases

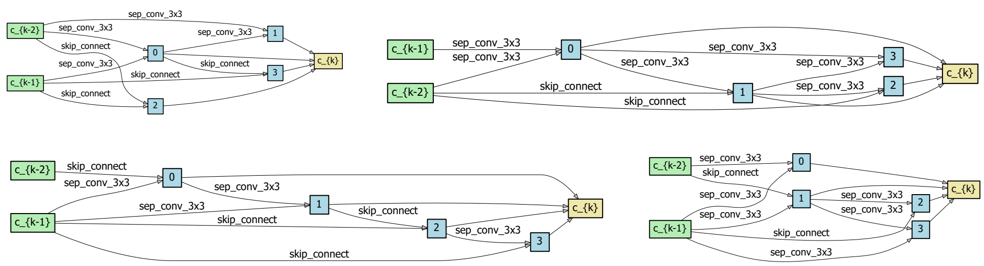


Figure 6: **Top**: the normal and reduction cells found in $\mathcal{S}_3$ with edge pruning. **Bottom**: the normal and reduction cells found in $\mathcal{S}_3$ with searched edges.

significantly, which verifies that the amending term indeed pulls architecture search away from degeneration.

On the other hand, if we use a sufficiently large η value, *e.g.*, $\eta = 1$, the amending term, $\mathbf{g}_2'$, can dominate optimization, so that the first term, *i.e.*, the gradient of architectural parameters, has limited effects in updating $\boldsymbol{\alpha}$. Note that the amending term is closely related to network regularization, therefore, in the scenarios of a large η, the network significantly prefers avg-pool-3x3 to other operators, as average pooling can smooth feature maps and avoid over-fitting. However, pooling is also a parameter-free operator, so the performance of such architectures is also below satisfaction.

Following these analyses, we simply use $\eta = 0.1$ for all later experiments. We do not tune η very carefully, though it is possible to determine η automatically using a held-out validation set. Besides, we find that the best architecture barely changes after 100 search epochs, so we fix the search length to be 100 epochs to reduce computational costs.

C.2 Comparison to (Zela et al., 2020)

(Zela et al., 2020) described various search spaces and demonstrated that the standard DARTS fails on them. They proposed various optimization strategies to robustify DARTS, including adding ℓ_2-regularization, adding Dropout, and using early termination. We search in a reduced space which we denote as $\mathcal{S}_3$, in which the candidate operators only include sep-conv-3x3, skip-connect, and *none*. Note that this is the 'safest' search space identified in R-DARTS (Zela et al., 2020), yet as shown in Table 5, R-DARTS often produced unsatisfying architectures, and choosing the best one over a few search trials can somewhat guarantee a reasonable architecture. Meanwhile, R-DARTS is sensitive to the search hyper-parameters such as the Dropout ratio.

Our approach works smoothly in this space, without the need of tuning hyper-parameters, and in either pruned or searched edges (see Figure 6). Thanks to the reduced search space, the best architecture often surpasses the numbers we have reported in Table 2.

C.3 200 Search Epochs for DARTS, P-DARTS, PC-DARTS, and Our Approach

We execute all algorithms for 200 search epochs on CIFAR10. The searched architectures by DARTS, P-DARTS, and PC-DARTS are shown in Figure 7, and that of our approach shown in Figure 2.

We find that both DARTS and P-DARTS failed completely into dummy architectures with all preserved operators to be skip-connect. PC-DARTS managed to survive after 200 epochs mainly due to two reasons: (i) the edge normalization technique is useful in avoiding skip-connect to dominate; and (ii) PC-DARTS sampled $1/4$ channels of each operator, so that the algorithm converged slower – 200 epochs may not be enough for a complete collapse.

We also visualize our approach throughout 200 epochs in $\mathcal{S}_1$ (under the edge-pruning setting) to investigate its behavior. In Figure 8, the visualized factors include the average weight of *none*, the ratio of preserved skip-connect operators, the super-network validation accuracy, and the re-training accuracy of some checkpoints during the search process.

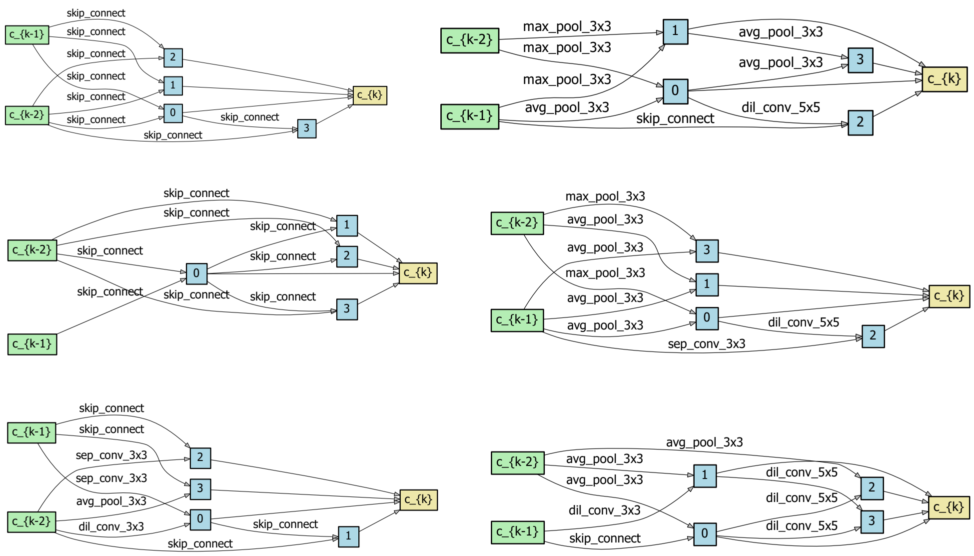


Figure 7: **Top**: the normal and reduction cells found by DARTS (test error: 6.18%). **Middle**: the normal and reduction cells found by P-DARTS (test error: 5.38%). **Bottom**: the normal and reduction cells found by PC-DARTS (test error: 3.15%).

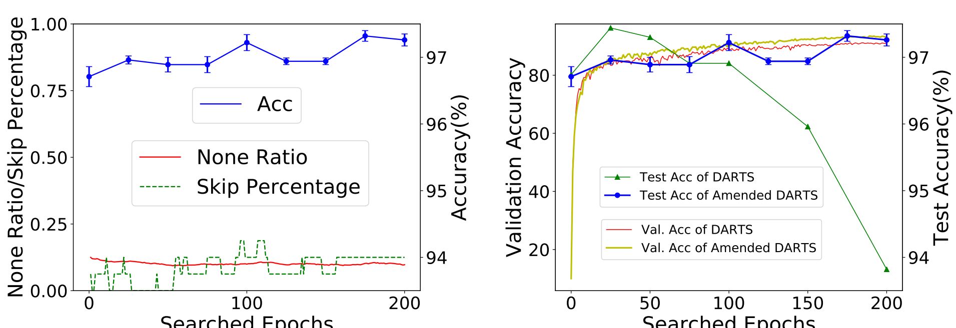


Figure 8: Details of a search process of our approach in $\mathcal{S}_1$ with 200 search epochs. **Left**: Red, green and blue curves indicate the average weight of *none*, the ratio of preserved skip-connect operators, and the re-training accuracy, respectively (corresponding to Figure 1). **Right**: Red and yellow curves indicate the validation accuracy of the super-network (in the search stage) searched by DARTS and our approach, and green and blue curves indicate the test accuracy (in the re-training stage) produced by the architectures found by DARTS and our approach, respectively.

C.4 SEARCHING WITH DIFFERENT SEEDS

On CIFAR10, we perform edge search (the first search stage) in $\mathcal{S}_2$ with different seeds (*i.e.*, random initialization), yielding three sub-architectures named $\mathcal{S}_2$-A, $\mathcal{S}_2$-B, and $\mathcal{S}_2$-C. Then, we execute the operator search process with different seeds for 3 times in $\mathcal{S}_1$ (with edge pruning), $\mathcal{S}_2$-A, $\mathcal{S}_2$-B, $\mathcal{S}_2$-C, and $\mathcal{S}_2$-F (fixed edges), and re-train each discovered architecture for 3 times. The results are summarized in Table 6. We also transfer some of the found architectures to ImageNet, and the corresponding results are listed in Table 7. The edge-searched architectures in $\mathcal{S}_2$ report inferior performance compared to the edge-fixed ones, arguably due to the inconsistency of hyper-parameters.

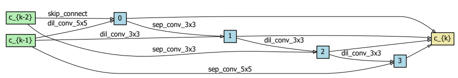
All the searched architectures are shown in Figures 9–12.

Architecture	**Test Err. (%)**				**Params**	**Search Cost**
	Run #1	**Run #2**	**Run #3**	**average**	**(M)**	**(GPU-days)**
Amended-DARTS, $\mathcal{S}_1$-A, pruned edges	2.81	2.65	2.67	2.71±0.09	3.3	1.7
Amended-DARTS, $\mathcal{S}_1$-B, pruned edges	2.82	2.73	2.90	2.82±0.09	2.9	1.7
Amended-DARTS, $\mathcal{S}_1$-C, pruned edges	2.82	2.68	2.73	2.74±0.07	3.8	1.7
Amended-DARTS, $\mathcal{S}_1$-F, fixed edges	2.69	2.68	3.05	2.81±0.21	3.5	1.0
Amended-DARTS, $\mathcal{S}_2$-A-A, searched edges	2.67	2.79	2.66	2.71±0.07	2.8	3.1
Amended-DARTS, $\mathcal{S}_2$-A-B, searched edges	2.62	2.79	2.56	2.66±0.12	3.0	3.1
Amended-DARTS, $\mathcal{S}_2$-A-C, searched edges	2.67	2.69	2.69	2.68±0.01	2.9	3.1
Amended-DARTS, $\mathcal{S}_2$-B-A, searched edges	2.59	2.73	2.57	2.63±0.09	3.0	3.1
Amended-DARTS, $\mathcal{S}_2$-B-B, searched edges	2.64	2.57	2.48	2.56±0.08	3.4	3.1
Amended-DARTS, $\mathcal{S}_2$-B-C, searched edges	2.64	2.66	2.80	2.70±0.09	3.0	3.1
Amended-DARTS, $\mathcal{S}_2$-C-A, searched edges	2.56	2.63	2.69	2.66±0.09	3.3	3.1
Amended-DARTS, $\mathcal{S}_2$-C-B, searched edges	2.61	2.70	2.77	2.69±0.08	3.2	3.1
Amended-DARTS, $\mathcal{S}_2$-C-C, searched edges	2.76	2.62	2.57	2.65±0.10	3.2	3.1
Amended-DARTS, $\mathcal{S}_2$-F-A, fixed edges	2.51	2.51	2.77	2.60±0.15	3.6	1.1
Amended-DARTS, $\mathcal{S}_2$-F-B, fixed edges	2.59	2.64	2.58	2.60±0.03	3.5	1.1
Amended-DARTS, $\mathcal{S}_2$-F-C, fixed edges	2.64	2.60	2.72	2.65±0.06	2.9	1.1
Amended-DARTS, $\mathcal{S}_3$-A, pruned edges	2.65	2.81	2.83	2.76±0.10	3.1	0.8
Amended-DARTS, $\mathcal{S}_3$-B, pruned edges	2.57	2.71	2.64	2.64±0.07	3.2	0.8
Amended-DARTS, $\mathcal{S}_3$-C, pruned edges	2.59	2.67	2.77	2.68±0.09	3.4	0.8
Amended-DARTS, $\mathcal{S}_3$-D-A, searched edges	2.52	2.43	2.69	2.55±0.13	3.2	3.0
Amended-DARTS, $\mathcal{S}_3$-D-B, searched edges	2.65	2.58	2.52	2.58±0.07	3.8	3.0
Amended-DARTS, $\mathcal{S}_3$-D-C, searched edges	2.80	2.74	2.73	2.76±0.04	2.9	3.0

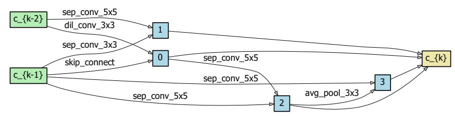
Table 6: Results of architecture search with different seeds in $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ on CIFAR10.

Architecture	**Test Err. (%)**		**Params**	$\times+$	**Search Cost**
	top-1	**top-5**	**(M)**	**(M)**	**(GPU-days)**
Amended-DARTS, $\mathcal{S}_1$-A, pruned edges	24.7	7.6	5.2	586	1.7
Amended-DARTS, $\mathcal{S}_1$-F, fixed edges	24.6	7.4	5.2	587	1.0
Amended-DARTS, $\mathcal{S}_2$-A-A, searched edges	24.7	7.5	4.9	576	3.1
Amended-DARTS, $\mathcal{S}_2$-B-A, searched edges	24.3	7.3	5.2	596	3.1
Amended-DARTS, $\mathcal{S}_2$-B-B, searched edges	24.6	7.5	5.3	585	3.1
Amended-DARTS, $\mathcal{S}_2$-B-C, searched edges	24.3	7.3	5.3	592	3.1
Amended-DARTS, $\mathcal{S}_2$-C-A, searched edges	24.3	7.5	5.2	592	3.1
Amended-DARTS, $\mathcal{S}_2$-F-A, fixed edges	24.3	7.4	5.5	590	1.1

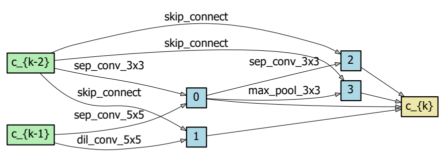
Table 7: Transferring some of the searched architectures on CIFAR10 to ILSVRC2012. The *mobile setting* is used to determine the basic channel width of each transferred architecture.



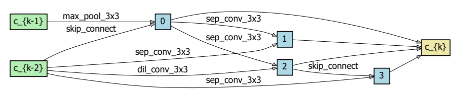
(a) the normal cell of $\mathcal{S}_1$-A



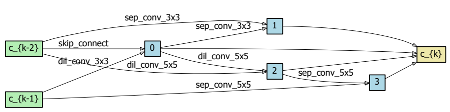
(b) the reduction cell of $\mathcal{S}_1$-A



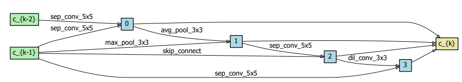
(c) the normal cell of $\mathcal{S}_1$-B



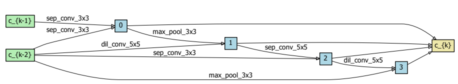
(d) the reduction cell of $\mathcal{S}_1$-B



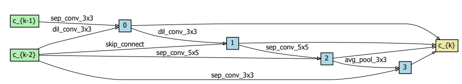
(e) the normal cell of $\mathcal{S}_1$-C



(f) the reduction cell of $\mathcal{S}_1$-C

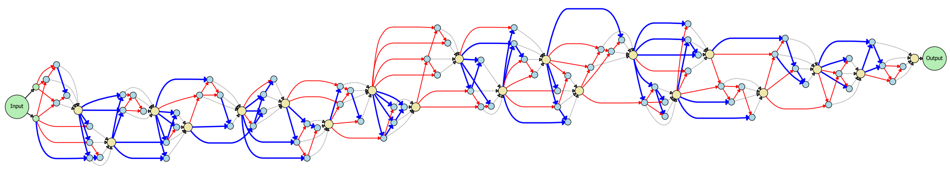


(g) the normal cell of $\mathcal{S}_1$-F

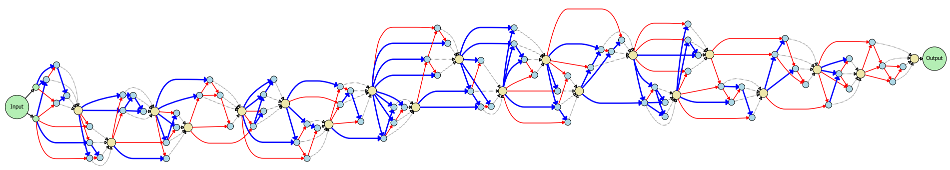


(h) the reduction cell of $\mathcal{S}_1$-F

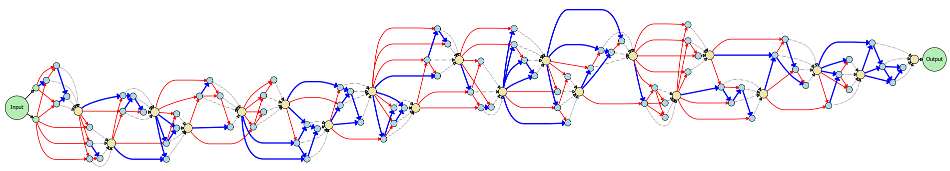
Figure 9: Architectures searched in $\mathcal{S}_1$ with pruned or fixed edges.



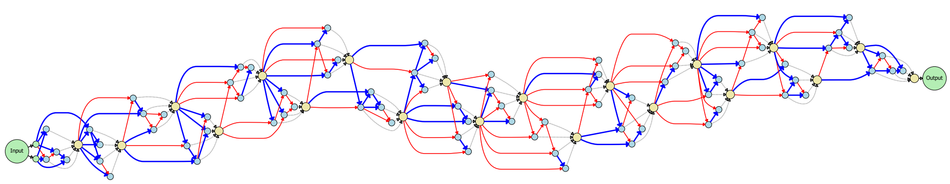
(a) the architecture of $\mathcal{S}_2$-A-A



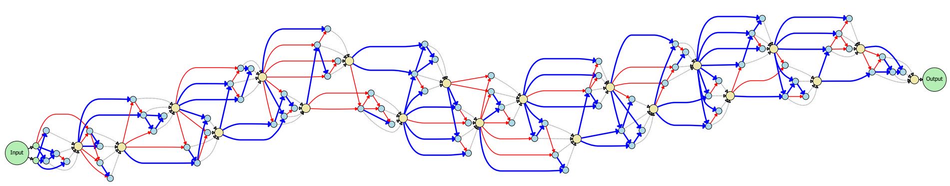
(b) the architecture of $\mathcal{S}_2$-A-B



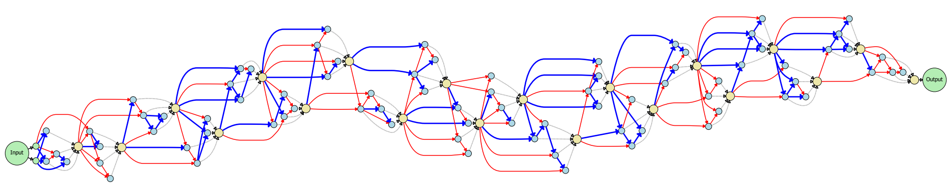
(c) the architecture of $\mathcal{S}_2$-A-C



(d) the architecture of $\mathcal{S}_2$-B-A

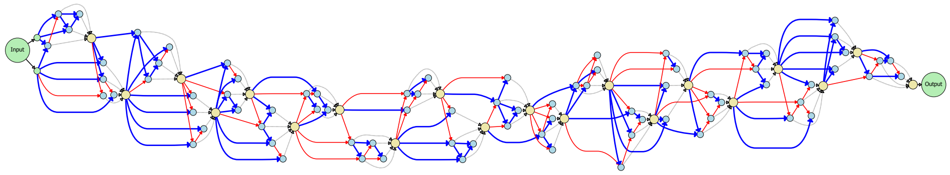


(e) the architecture of $\mathcal{S}_2$-B-B

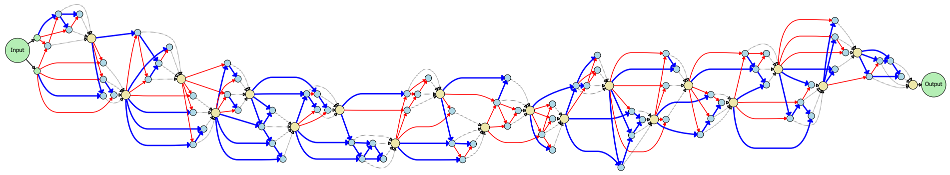


(f) the architecture of $\mathcal{S}_2$-B-C

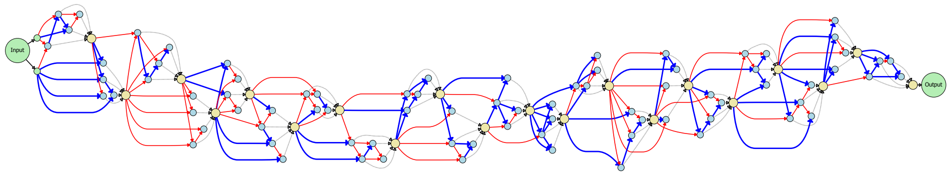
Figure 10: **(Part I)** Architectures searched in $\mathcal{S}_2$ with searched or fixed edges, in which red thin, blue bold, and black dashed arrows indicate skip-connect, sep-conv-3x3, and channel-wise concatenation, respectively. *This figure is best viewed in color.*.



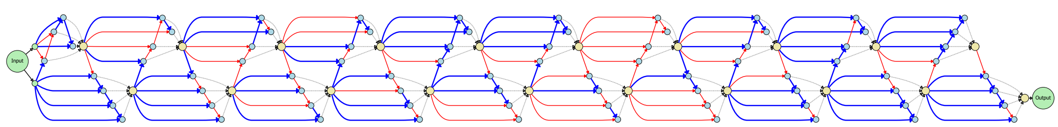
(a) the architecture of $\mathcal{S}_2$-C-A



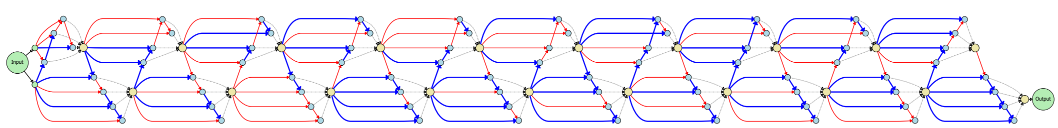
(b) the architecture of $\mathcal{S}_2$-C-B



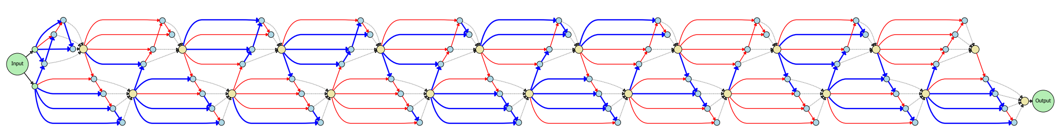
(c) the architecture of $\mathcal{S}_2$-C-C



(d) the architecture of $\mathcal{S}_2$-F-A

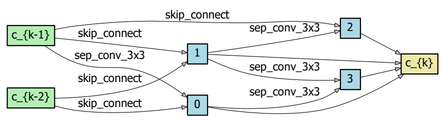


(e) the architecture of $\mathcal{S}_2$-F-B

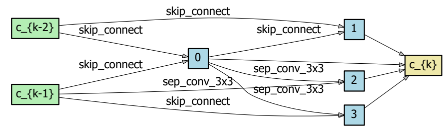


(f) the architecture of $\mathcal{S}_2$-F-C

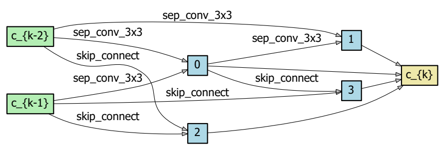
Figure 11: **(Part II)** Architectures searched in $\mathcal{S}_2$ with searched or fixed edges, in which red thin, blue bold, and black dashed arrows indicate skip-connect, sep-conv-3x3, and channel-wise concatenation, respectively. *This figure is best viewed in color.*.



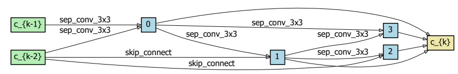
(a) the normal cell of $\mathcal{S}_3$-A



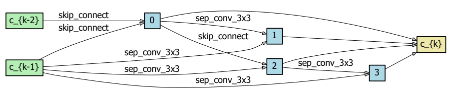
(b) the reduction cell of $\mathcal{S}_3$-A



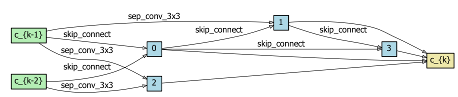
(c) the normal cell of $\mathcal{S}_3$-B



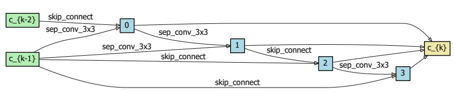
(d) the reduction cell of $\mathcal{S}_3$-B



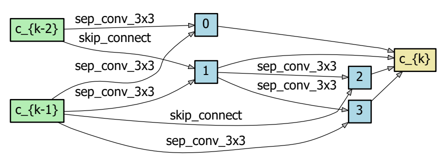
(e) the normal cell of $\mathcal{S}_3$-C



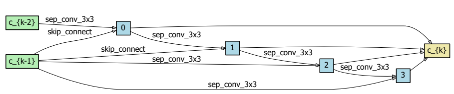
(f) the reduction cell of $\mathcal{S}_3$-C



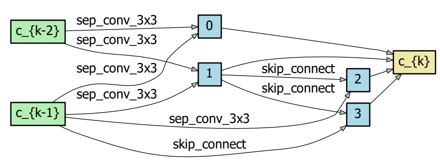
(g) the normal cell of $\mathcal{S}_3$-D-A



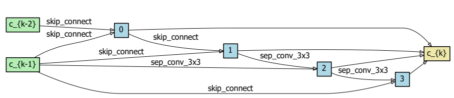
(h) the reduction cell of $\mathcal{S}_3$-D-A



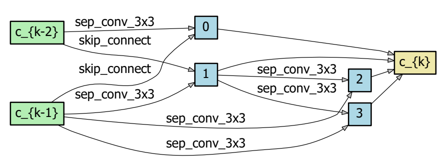
(i) the normal cell of $\mathcal{S}_3$-D-B



(j) the reduction cell of $\mathcal{S}_3$-D-B



(k) the normal cell of $\mathcal{S}_3$-D-C



(l) the reduction cell of $\mathcal{S}_3$-D-C

Figure 12: Architectures searched in $\mathcal{S}_3$.