GOLD-NAS: GRADUAL, ONE-LEVEL, DIFFERENTIABLE

Anonymous authors
Paper under double-blind review

ABSTRACT

There has been a large literature on neural architecture search, but most existing work made use of heuristic rules that largely constrained the search flexibility. In this paper, we first relax these manually designed constraints and enlarge the search space to contain more than 10^{117} candidates. In the new space, most existing differentiable search methods can fail dramatically. We then propose a novel algorithm named Gradual One-Level Differentiable Neural Architecture Search (**GOLD-NAS**) which introduces a variable resource constraint to one-level optimization so that the weak operators are gradually pruned out from the super-network. In standard image classification benchmarks, GOLD-NAS can find a series of Pareto-optimal architectures within a single search procedure. Most of the discovered architectures were never studied before, yet they achieve a nice tradeoff between recognition accuracy and model complexity. GOLD-NAS also shows generalization ability in extended search spaces with different candidate operators.

1 INTRODUCTION

With the rapid development of deep learning (LeCun et al., 2015), designing powerful neural networks has been a major challenge for learning compact representations. As manually designed architectures become more and more complex (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015; Szegedy et al., 2015; He et al., 2016; Huang et al., 2017), researchers started to explore automatic design approaches. The outcome is a new methodology named neural architecture search (NAS) that has shown the potential of finding effective and/or efficient architectures that outperform human expertise.

NAS is often formulated by two aspects, namely, the search space and the search method. A good **search space** offers sufficient capacity so that there exist some high-quality architectures (either of high quality or of high efficiency) but they are difficult to discover by the manually defined rules. Currently popular search spaces (Zoph et al., 2018; Liu et al., 2019; Howard et al., 2019) are often composed of some repeatable *cells*, each of which is a relatively complex combination of basic operators (*e.g.*, convolution, pooling, *etc.*). The connectivity between these cells can be either fixed (Liu et al., 2019; Howard et al., 2019) or changeable (Real et al., 2017; Xie & Yuille, 2017). A **search method** should be able to explore the large search space efficiently, for which existing efforts are categorized into two parts. For the *individual* search methods (Zoph & Le, 2017; Real et al., 2017; Xie & Yuille, 2017) sample and evaluate architectures individually. They are often slow and difficult to generalize across datasets, so later efforts have focused on reusing computation of similar architectures (Cai et al., 2018; Luo et al., 2018). This path eventually leads to the *weight-sharing* search methods (Pham et al., 2018; Liu et al., 2019; Chu et al., 2019a; Guo et al., 2019) that trains a super-network (which contains all possible architectures) only once, after which the sampling and evaluation become much quicker, possibly producing multiple architectures (Cai et al., 2020).

In this paper, we focus on a special type of weight-sharing search methods named differentiable neural architecture search (*e.g.*, DNAS (Shin et al., 2018), DARTS (Liu et al., 2019), *etc.*). The search space is relaxed so that the architectural parameters can take continuous values and be optimized together with the network weights using gradient descent. These methods enjoy high search efficiency (*e.g.*, the typical search cost on CIFAR10 is just a few hours, making DARTS one of the most popular NAS algorithm now), but they suffers some major drawbacks, listed below.

- The search space of DARTS is *highly constrained*, *e.g.*, all normal cells share the same inner-architecture, each node receives exactly two inputs, and there is exactly one operator on each preserved edge. These constraints are helpful for the stability of NAS, but they also shrink the accuracy gain brought by powerful search methods: with some heuristic designs (*e.g.*, two skip-connect operators in each cell (Chen et al., 2019)) or search tricks (*e.g.*, early termination (Liang et al., 2019)), even random search can achieve satisfying performance.

- DARTS requires *bi-level optimization*, *i.e.*, a training phase to optimize the network weights and a validation phase to update the architecture parameters. This mechanism brings computational burden and, more importantly, considerable inaccuracy in gradient estimation that can dramatically deteriorate the search procedure (Bi et al., 2019; Zela et al., 2020).
- DARTS removes weak operators and edges all at once after the super-network has been optimized, but this step can risk a large *discretization error* especially when the weights of the pruned operators are not guaranteed to be small.

To address these problems, we advocate for an enlarged search space which borrows the cell-based design of DARTS but frees most of the heuristic constraints. In particular, all cells are allowed to have different architectures, each edge can contain more than one operators, and each node can receive input from an arbitrary number of its precedents. These modifications have increased the size of search space from about 10^{18} to more than 10^{235}. More importantly, we believe the reduction of constraints will raise new challenges to the stability and advance the research of NAS methods.

In this complex space, bi-level optimization suffers from heavy computational burden as well as the inaccuracy of gradient estimation (Bi et al., 2019). This urges us to apply one-level optimization which is easier to get rid of the computational burdens. However, as shown in (Liu et al., 2019), one-level optimization can run into dramatic failure which, according to our diagnosis, mainly owes to the discretization error caused by removing the moderate operators. Motivated by this finding, we present a novel framework which starts with a complete super-network and gradually prunes out weak operators. During the search procedure, we avoid applying heuristic rules but rely on resource constraints (*e.g.*, FLOPs) to determine which operators should be eliminated. Our algorithm is named **GOLD-NAS** which stands for Gradual One-Level Differentiable Neural Architecture Search. Compared to prior NAS approaches, GOLD-NAS requires little human expertise (Li et al., 2019; Chen & Hsieh, 2020; He et al., 2020) and is less prone of the optimization gap.

We perform experiments on CIFAR10 and ImageNet, two popular image classification benchmarks. Within a small search cost (0.4 GPU-days on CIFAR10 and 1.3 GPU-days on ImageNet), GOLD-NAS find a series of Pareto-optimal architectures that can fit into different hardware devices. In particular, the found architectures achieve a $2.99 \pm 0.05\%$ error on CIFAR10 with merely $1.58\mathrm{M}$ parameters, and a 23.9% top-1 error on ImageNet under the mobile setting. Moreover, we have tested a few extended search spaces and also obtained satisfying search results. These results pave the way of searching in a much larger space which is very challenging for most prior work.

2 Building GOLD-NAS on CIFAR10

2.1 Dataset, Settings, and Overview

The CIFAR10 dataset (Krizhevsky & Hinton, 2009) is one of the most popular benchmarks for neural architecture search. It has $50\mathrm{K}$ training images and $10\mathrm{K}$ testing images, uniformly distributed over 10 classes. Each image is RGB and has a resolution of 32×32. We follow the convention to first determine the optimal architecture and then re-train it for evaluation. The test set remains invisible in both the search and re-training phases. Detailed settings are elaborated in Appendix B.1.

We start with defining an enlarged search space (Section 2.2) that discards most manual designs and provides a better benchmark for NAS evaluation. Next, we demonstrate the need of one-level optimization (Section 2.3) and analyze the difficulty of performing discretization in the new space (Section 2.4). Finally, to solve the problem, we design a pruning algorithm (Section 2.5) that gradually eliminates weak operators and/or edges with the regularization of resource efficiency.

2.2 Breaking the Rules: Enlarging the Search Space

We first recall the cell-based super-network used in DARTS (Liu et al., 2019). It has a fixed number (L) of cells. Each cell has two input signals from the previous two cells (denoted as $\mathbf{x}_0$ and $\mathbf{x}_1$), and $N - 2$ inner nodes to store intermediate responses. For each $i < j$ except for $(i, j) = (0, 1)$, the output of the i-th node is sent to the j-th node via the edge of (i, j). Mathematically, we have $g_{i,j}(\mathbf{x}_i) = \sum_{o \in \mathcal{O}} \sigma\big(\alpha_{i,j}^{o}\big) \cdot o(\mathbf{x}_i)$ where $\mathbf{x}_i$ is the output of the i-th node, $\mathcal{O}$ is a pre-defined set of operators, and $o(\cdot)$ is an element in $\mathcal{O}$. $\sigma\big(\alpha_{i,j}^{o}\big)$ determines the weight of $o(\mathbf{x}_i)$, which is set to be $\sigma\big(\alpha_{i,j}^{o}\big) = \exp(\alpha_{i,j}^{o}) / \sum_{o' \in \mathcal{O}} \exp(\alpha_{i,j}^{o'})$. The output of the j-th cell is the sum of all information flows from the precursors, *i.e.*, $\mathbf{x}_j = \sum_{i<j} g_{i,j}(\mathbf{x}_i)$, and the final output of the cell is the concatenation of all non-input nodes, *i.e.*, $\mathrm{concat}(\mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{N-1})$. In this way, the super-network is formulated into a differentiable function, $f(\mathbf{x}) \doteq f(\mathbf{x}; \boldsymbol{\alpha}, \boldsymbol{\omega})$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ indicate the architectural parameters and network weights, respectively.

DARTS (Liu et al., 2019) and its variants (Chen et al., 2019; Nayman et al., 2019; Xu et al., 2020a) have relied on many manually designed rules to determine the final architecture. Examples include each edge can only preserve one operator, each inner node can preserve two of its precursors, and the architecture is shared by the same type (normal and reduction) of cells. These constraints are helpful for the stability of the search process, but they limit the flexibility of architecture search, *e.g.*, the low-level layers and high-level layers must have the same topological complexity which is no reason to be the optimal solution. A prior work (Bi et al., 2019) delivered an important message that the ability of NAS approaches is better evaluated in a more complex search space (in which very few heuristic rules are used). Motivated by this, we release the heuristic constraints to offer higher flexibility to the final architecture, namely, each edge can preserve an arbitrary number of operators (they are directly summed into the output), each inner node can preserve an arbitrary number of precedents, and all cell architectures are independent.

To fit the new space, we slightly modify the super-network so that $\sigma\big(\alpha_{i,j}^{o}\big)$ is changed from the softmax function to element-wise sigmoid, *i.e.*, $\sigma\big(\alpha_{i,j}^{o}\big) = \exp(\alpha_{i,j}^{o}) / \big(1 + \exp(\alpha_{i,j}^{o})\big)$. This offers a more reasonable basis to the search algorithm since the enhancement of any operator does not necessarily lead to the attenuation of all others (Chu et al., 2019b). Moreover, the independence of all cells raises the need of optimizing the complete super-network (*e.g.*, having 14 cells) during the search procedure. To fit the limited GPU memory, we preserve no more than 4 operators, *e.g.*, in the smallest search space, $\mathcal{S}_0$, only skip-connect and sep-conv-3x3 are used, same as (Bi et al., 2019; Zela et al., 2020). Note that with 2 or 4 active operators, the search space contains 3.1×10^{117} or 9.6×10^{235} architectures[1], which is far more than the capacity of the original space with either shared cells (1.1×10^{18}, (Liu et al., 2019)) or individual cells (1.9×10^{93}, (Bi et al., 2019)). Without heuristic rules, exploring this enlarged space requires more powerful search methods.

2.3 Why One-Level Optimization?

The goal of differentiable NAS is to solve the following optimization:

$$\boldsymbol{\alpha}^{\star} = \arg\min_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\omega}^{\star}(\alpha), \boldsymbol{\alpha}; \mathcal{D}_{\mathrm{train}}), \qquad \text{s.t.} \quad \boldsymbol{\omega}^{\star}(\alpha) = \arg\min_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\alpha}; \mathcal{D}_{\mathrm{train}}), \tag{1}$$

where $\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\alpha}; \mathcal{D}_{\mathrm{train}}) = \mathbb{E}_{(\mathbf{x}, \mathbf{y}^{\star}) \in \mathcal{D}_{\mathrm{train}}} [\mathrm{CE}(f(\mathbf{x}), \mathbf{y}^{\star})]$ is the loss function computed in a specified training dataset. There are mainly two methods for this purpose, known as one-level optimization and bi-level (two-level) optimization, respectively. Starting with $\boldsymbol{\alpha}_0$ and $\boldsymbol{\omega}_0$, the initialization of $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$, *one-level optimization* involves updating $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ simultaneously in each step:

$$\boldsymbol{\omega}_{t+1} \leftarrow \boldsymbol{\omega}_t - \eta_{\boldsymbol{\omega}} \cdot \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}_t, \boldsymbol{\alpha}_t; \mathcal{D}_{\mathrm{train}}), \qquad \boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\alpha}_t - \eta_{\boldsymbol{\alpha}} \cdot \nabla_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\omega}_t, \boldsymbol{\alpha}_t; \mathcal{D}_{\mathrm{train}}). \tag{2}$$

Note that, since the numbers of parameters in $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ differ significantly (from tens to millions), different learning rates ($\eta_{\boldsymbol{\alpha}}$ and $\eta_{\boldsymbol{\omega}}$) and potentially different optimizers can be used. Even in this way, the algorithm is easily biased towards optimizing $\boldsymbol{\omega}$, leading to unsatisfying performance[2]. To fix this issue, a practical way is to evaluate the performance with respect to $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ in two separate training sets, *i.e.*, $\mathcal{D}_{\mathrm{train}} = \mathcal{D}_1 \cup \mathcal{D}_2$. Hence, the goal of optimization becomes:

$$\boldsymbol{\alpha}^{\star} = \arg\min_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\omega}^{\star}(\alpha), \boldsymbol{\alpha}; \mathcal{D}_1), \qquad \text{s.t.} \quad \boldsymbol{\omega}^{\star}(\alpha) = \arg\min_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\alpha}; \mathcal{D}_2), \tag{3}$$

[1]This is the theoretical maximum. Under the resource constraints (Section 2.5), the final architecture is often in a relatively small space, but the space is still much larger than the competitors. Please refer to Appendix A.1 for the calculation of the number of possible architectures.

[2]This is because of the imbalanced effect brought by optimizing $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$, in which the latter is often more effective. An intuitive example is to fix $\boldsymbol{\alpha}$ as the status after random initialization and only optimize $\boldsymbol{\omega}$, which can still leads to a high accuracy in the training data (it is not possible to achieve this goal by only optimizing $\boldsymbol{\alpha}$). Obviously, this does not deliver any useful information to architecture design.

and, correspondingly, *bi-level optimization* is used to update $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ alternately:

$$\boldsymbol{\omega}_{t+1} \leftarrow \boldsymbol{\omega}_t - \eta_{\boldsymbol{\omega}} \cdot \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}_t, \boldsymbol{\alpha}_t; \mathcal{D}_2), \qquad \boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\alpha}_t - \eta_{\boldsymbol{\alpha}} \cdot \nabla_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\omega}_{t+1}, \boldsymbol{\alpha}_t; \mathcal{D}_1). \tag{4}$$

DARTS (Liu et al., 2019) tried both optimization methods and advocated for the superiority of bi-level optimization. However, as pointed out in (Bi et al., 2019), bi-level optimization suffers considerable inaccuracy of gradient estimation and the potential instability can increase with the complexity of the search space. This drives us back to one-level optimization. Fortunately, we find that the failure of one-level optimization can be easily prevented. Detailed analyses and experiments are provided in Appendix A.2. Here, we deliver the key message that one-level optimization is made quite stable by adding regularization (*e.g.*, Cutout (DeVries & Taylor, 2017), AutoAugment (Cubuk et al., 2019), *etc.*) to a small dataset (*e.g.*, CIFAR10) or simply using a large dataset (*e.g.*, ImageNet). So, we apply one-level optimization to the enlarged search space throughout the remaining part of this paper.

It is worth noting that prior approaches also suggested one-level optimization (Li et al., 2019) or mixed-level optimization (He et al., 2020). In comparison, our solution is more elegant by adding regularization to $\boldsymbol{\omega}$ and not involving any other requirements.

2.4 The Difficulty of Discretization

The main challenge that we encounter in the enlarged space is the difficulty of performing discretization, *i.e.*, determining the final architecture based on $\boldsymbol{\alpha}$. This is to require $\boldsymbol{\alpha}$ in Eqn equation 1 to satisfy the condition that $\sigma\big(\alpha_{i,j}^o\big) = \exp(\alpha_{i,j}^o)/\big(1+\exp(\alpha_{i,j}^o)\big)$ is very close to 0 or 1, but this constraint is difficult to be integrated into a regular optimization process like Eqn equation 2. The solution of conventional approaches (Liu et al., 2019; Chen et al., 2019; Xu et al., 2020a; Zela et al., 2020) is to perform hard pruning at the end of the search stage to eliminate weak operators from the super-network, *e.g.*, an operator is preserved if $\sigma\big(\alpha_{i,j}^o\big) > 0.5$.

This algorithm can lead to significant *discretization error*, since many of the pruned operators have moderate weights, *i.e.*, $\sigma\big(\alpha_{i,j}^o\big) = \exp(\alpha_{i,j}^o)/\big(1+\exp(\alpha_{i,j}^o)\big)$ is neither close to 0 nor 1. In this scenario, directly removing these operators can lead to dramatic accuracy drop on the super-network. Mathematically, this may push $\boldsymbol{\alpha}$ (and also $\boldsymbol{\omega}(\boldsymbol{\alpha})$) away from the current optimum, so that the algorithm may need a long training process to arrive at another optimum, or never. The reason for $\sigma\big(\alpha_{i,j}^o\big)$ being moderate is straightforward: the new space allows an arbitrary number of operators to be preserved on each edge, or more specifically, there is no internal mechanism for the operators to compete with each other. Therefore, the best strategy to fit training data is to keep **all** the operators, since almost all operators contribute more or less to the training accuracy, but this is of little use to architecture search itself.

Motivated by this, we propose to add regularization to the process of super-network training so that to penalize the architectures that use more computational resources. This mechanism is similar in a few prior work that incorporated hardware constraints to the search algorithm (Cai et al., 2019; Tan et al., 2019; Wu et al., 2019), but the goal of our method is to use the power of regularization to suppress the weight of some operators so that they can be pruned. Note that the risk of discretization error grows as the number and the strength (weights) of pruned operators. So, a safe choice is to perform pruning multiple times, in each of which only the operators with sufficiently low weights can be removed. We elaborate the details in the following subsection.

2.5 Gradual Pruning with Resource Constraints

To satisfy the condition that the weights of pruned operators are sufficiently small, we design a gradual pruning algorithm. The core idea is to start with a low regularization coefficient and increase it gradually during the search procedure. Every time the coefficient becomes larger, there will be some operators (those having higher redundancy) being suppressed to low weights. Pruning them out causes little drop in training accuracy. This process continues till the super-network size goes below a threshold. During the search process, The architectures that survive for sufficiently long are recorded, which compose the set of Pareto-optimal architectures.

Throughout the remaining part, we set the regularization term as the expected FLOPs of the super-network, and this framework can be generalized to other kinds of constraints (*e.g.*, network latency (Wu et al., 2019; Tan et al., 2019; Xu et al., 2020b)). Conceptually, adding resource constraints

Algorithm 1: Gradual One-Level Differentiable Neural Architecture Search (**GOLD-NAS**)

```
Input  : Search space $\mathcal{S}$, dataset $\mathcal{D}_{\mathrm{train}}$, balancing coefficient $\mu$, minimal FLOPs constraints
         $\mathrm{FLOPs}_{\mathrm{min}}$, learning rates $\eta_{\boldsymbol{\omega}}$, $\eta_{\boldsymbol{\alpha}}$, pruning hyper-parameters $n_0$, $\lambda_0$, $c_0$, $\xi_{\mathrm{max}}$, $\xi_{\mathrm{min}}$, $t_0$;
Output : A set of pareto-optimal architectural parameters $\mathcal{A}$;
1 Initialize $\boldsymbol{\omega}^{\mathrm{curr}}$ and $\boldsymbol{\alpha}^{\mathrm{curr}}$ as random noise, $\mathcal{A} \leftarrow \varnothing$, $\lambda \leftarrow 0$, $\Delta\lambda \leftarrow \lambda_0$, $t \leftarrow 0$;
2 repeat
3     Update $\boldsymbol{\omega}^{\mathrm{curr}}$ and $\boldsymbol{\alpha}^{\mathrm{curr}}$ using Eqn equation 2 for one epoch;
4     Let $\mathcal{E}$ be the set of active operators, and $\mathcal{E}_{\mathrm{min}}$ be the $n_0$ operators in $\mathcal{E}$ with minimal weights;
5     Prune operators in $\mathcal{E}_{\mathrm{min}}$ with weight smaller than $\xi_{\mathrm{max}}$, and operators in $\mathcal{E}$ with weight
        smaller than $\xi_{\mathrm{min}}$, let $n_{\mathrm{pruned}}$ be the number of pruned operators;
6     if $n_{\mathrm{pruned}} < n_0$ then $\Delta\lambda \leftarrow c_0\Delta\lambda$, $\lambda \leftarrow \lambda + \Delta\lambda$ else $\Delta\lambda \leftarrow \lambda_0$, $\lambda \leftarrow \lambda / c_0$;
7     if $n_{\mathrm{pruned}} = 0$ then $t \leftarrow t + 1$ else $t \leftarrow 0$;
8     if $t \geqslant t_0$ then $\mathcal{A} \leftarrow \mathcal{A} \cup \{\boldsymbol{\alpha}^{\mathrm{curr}}\}$, $t \leftarrow 0$;
9 until $\mathrm{FLOPs}(\boldsymbol{\alpha}^{\mathrm{curr}}) \leqslant \mathrm{FLOPs}_{\mathrm{min}}$;
Return : $\mathcal{A}$.
```

requires a slight modification to the objective function, Eqn equation 1. With $\mathrm{FLOPs}(\boldsymbol{\alpha})$ denoting the expected FLOPs under the architectural parameter of $\boldsymbol{\alpha}$, the overall optimization goal is to achieve a tradeoff between accuracy and efficiency. We have carefully designed the calculation of $\mathrm{FLOPs}(\boldsymbol{\alpha})$, described in Appendix A.3, so that (i) the result strictly equals to the evaluation of the `thop` library, and (ii) $\mathrm{FLOPs}(\boldsymbol{\alpha})$ is differentiable to $\boldsymbol{\alpha}$.

The top-level design of gradual pruning is to facilitate the competition between accuracy and resource efficiency. For this purpose, we modify the original objective function to incorporate the FLOPs constraint, *i.e.*, $\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\alpha}) = \mathbb{E}_{(\mathbf{x},\mathbf{y}^{\star})\in\mathcal{D}_{\mathrm{train}}}[\mathrm{CE}(f(\mathbf{x}), \mathbf{y}^{\star})] + \lambda \cdot \left(\overline{\mathrm{FLOPs}}(\boldsymbol{\alpha}) + \mu \cdot \mathrm{FLOPs}(\boldsymbol{\alpha})\right)$, where the two coefficients, λ and μ, play different roles. λ starts with 0 and vibrates during the search procedure to smooth the pruning process, resulting in a Pareto front that contains a series of optimal architectures with different computational costs. μ balances between $\mathrm{FLOPs}(\boldsymbol{\alpha})$ and $\overline{\mathrm{FLOPs}}(\boldsymbol{\alpha})$, the *expected* and *uniform* versions of FLOPs calculation (see Appendix A.3). In brief, $\mathrm{FLOPs}(\boldsymbol{\alpha})$ adds lower penalty to the operators with smaller computational costs, but $\overline{\mathrm{FLOPs}}(\boldsymbol{\alpha})$ adds a fixed weight to all operators. Hence, a larger μ favors pruning more convolutions and often pushes the architecture towards higher computational efficiency. In other words, one can tune the value of μ to achieve different Pareto fronts (see the experiments results in the next subsection).

The overall search procedure is summarized in Algorithm 1. At the beginning, λ is set to be 0 and the training procedure focuses on improving accuracy. As the optimization continues, λ gradually goes up and forces the network to reduce the weight on the operators that have fewer contribution. In each pruning round, there is an expected number of operators to be pruned. If this amount is not achieved, λ continues to increase, otherwise it is reduced. If no operators are pruned for a few consecutive epochs, the current architecture is considered Pareto-optimal and added to the output set. Our algorithm is named **GOLD-NAS**, which indicates its most important properties: **G**radual, **O**ne-**L**evel, and **D**ifferentiable. We emphasize that it is the **gradual pruning** strategy that alleviates the discretization error and enables the algorithm to benefit from the flexibility of **one-level optimization** and the efficiency of **differentiable search**.

2.6 Search Results and Comparison to Prior Work

We start with defining 6 search spaces, $\mathcal{S}_0$–$\mathcal{S}_5$. Among them, $\mathcal{S}_0$ has 2 active operators and all others have 4. The detailed configurations are shown in Table 2. We test these search spaces to validate the ability of GOLD-NAS in adjusting to various environments, *e.g.*, skip-connect and sep-conv-3x3 dominate in most DARTS-based architectures, but one of them is missing in $\mathcal{S}_2$–$\mathcal{S}_4$, raising challenges to use other operators for replacement. GOLD-NAS works well in most scenarios.

We start with $\mathcal{S}_0$. A notable benefit of GOLD-NAS is to obtain a set of Pareto-optimal architectures within one search procedure (on CIFAR10, around 10 hours on a single NVIDIA Tesla V100 card). We use two sparsity coefficients, $\mu = 1$ and $\mu = 0$, and obtain six architectures for each, with FLOPs varying from 245M to 546M. The intermediate results of the search procedure (*e.g.*, how the values of λ and $\left\{\sigma\left(\alpha_{i,j}^{o}\right)\right\}$ change with epochs) are shown in Appendix B.2. We re-train each architecture three times, and the results are shown in Table 1. One can observe the tradeoff between accuracy and efficiency. In particular, the GOLD-NAS-A architecture, with only $1.58\mathrm{M}$ parameters and $245\mathrm{M}$ FLOPs, achieves a 2.99% error on the CIFAR10 test set. To the best of our knowledge, it contains fewer parameters than any published models that beat the 3%-error mark.

Space	Active Operators	Capacity
$\mathcal{S}_0$	skip-connect, sep-conv-3x3	3.1×10^{117}
$\mathcal{S}_1$	skip-connect, max-pool-3x3, sep-conv-3x3, dil-conv-3x3	9.6×10^{235}
$\mathcal{S}_2$	avg-pool-3x3, max-pool-3x3, sep-conv-3x3, dil-conv-3x3	9.6×10^{235}
$\mathcal{S}_3$	skip-connect, max-pool-3x3, dil-conv-3x3, dil-conv-5x5	9.6×10^{235}
$\mathcal{S}_4$	skip-connect, max-pool-3x3, sep-conv-5x5, dil-conv-3x3	9.6×10^{235}
$\mathcal{S}_5$	skip-connect, avg-pool-3x3, sep-conv-3x3, dil-conv-5x5	9.6×10^{235}

Table 2: The detailed configurations of six search spaces with different sets of active operators. $\mathcal{S}_0$ is contains two operators thought to be most useful, while skip-connect is missing in $\mathcal{S}_2$ and sep-conv-3x3 missing in $\mathcal{S}_3$ and $\mathcal{S}_4$.

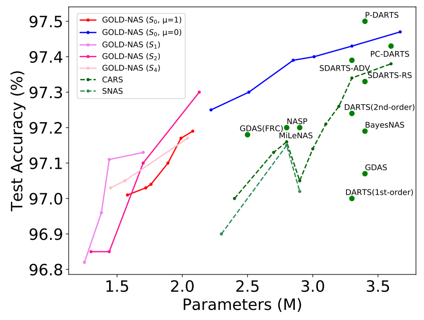


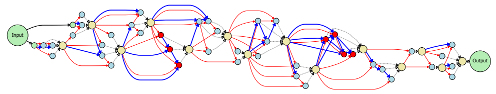
Figure 1: The accuracy-complexity tradeoff of differentiable NAS methods. GOLD-NAS achieves better Pareto-fronts compared to other methods.

We visualize the first and last architectures obtained from $\mu = 0$ and $\mu = 1$ in Figure 2, and complete results are provided in Appendix B.3. Moreover, compared to the architectures found in the original DARTS space, our algorithm allows the resource to be flexibly assigned to different stages (*e.g.*, the cells close to the output does not need much resource), and this is the reason for being more efficient. From this perspective, the enlarged search space creates more opportunities for the NAS approach, yet it is the stability of GOLD-NAS that eases the exploration in this space without heuristic rules.

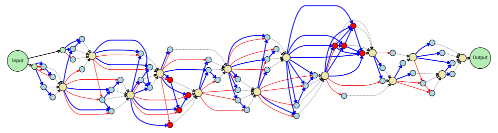
We compare the search results in $\mathcal{S}_0$ to the state-of-the-arts in Table 1. Besides the competitive performance, we claim three major advantages. **First, GOLD-NAS is faster, easier to implement, and more stable than most DARTS-based methods.** This is mainly because bi-level optimization

Architecture		**Test Err. (%)**				**Params**	**Search Cost**	**FLOPs**
		#1	**#2**	**#3**	**average**	**(M)**	**(GPU-days)**	**(M)**
DenseNet-BC (Huang et al., 2017)					3.46	25.6	-	-
ENAS (Pham et al., 2018)					2.89	4.6	0.5	626
NASNet-A (Zoph et al., 2018)					2.65	3.3	1800	605
AmoebaNet-B (Real et al., 2019)					2.55±0.05	2.8	3150	490
SNAS (moderate) (Xie et al., 2018)					2.85±0.02	2.8	1.5	441
DARTS (*1st-order*) (Liu et al., 2019)					3.00±0.14	3.3	0.4	-
DARTS (*2nd-order*) (Liu et al., 2019)					2.76±0.09	3.3	1.0	528
P-DARTS (Chen et al., 2019)					2.50	3.4	0.3	532
PC-DARTS (Xu et al., 2020a)					2.57±0.07	3.6	0.1	557
GOLD-NAS-$\mathcal{S}_0$-A		2.93	3.02	3.01	2.99±0.05	1.58	0.4	245
GOLD-NAS-$\mathcal{S}_0$-B		2.97	2.85	3.08	2.97±0.12	1.72	0.4	267
GOLD-NAS-$\mathcal{S}_0$-C		2.94	2.97	2.97	2.96±0.02	1.76	0.4	287
GOLD-NAS-$\mathcal{S}_0$-D	$\mu = 1$	2.89	2.98	2.84	2.90±0.07	1.89	0.4	308
GOLD-NAS-$\mathcal{S}_0$-E		2.75	2.86	2.89	2.83±0.07	1.99	0.4	334
GOLD-NAS-$\mathcal{S}_0$-F		2.77	2.79	2.86	2.81±0.05	2.08	0.4	355
GOLD-NAS-$\mathcal{S}_0$-G		2.73	2.84	2.67	2.75±0.09	2.22	1.1	376
GOLD-NAS-$\mathcal{S}_0$-H		2.71	2.76	2.62	2.70±0.07	2.51	1.1	402
GOLD-NAS-$\mathcal{S}_0$-I		2.52	2.72	2.60	2.61±0.10	2.85	1.1	445
GOLD-NAS-$\mathcal{S}_0$-J	$\mu = 0$	2.53	2.67	2.60	2.60±0.07	3.01	1.1	459
GOLD-NAS-$\mathcal{S}_0$-K		2.67	2.40	2.65	2.57±0.15	3.30	1.1	508
GOLD-NAS-$\mathcal{S}_0$-L		2.57	2.58	2.44	2.53±0.08	3.67	1.1	546

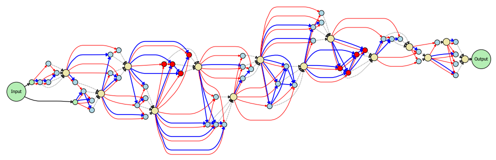
Table 1: Comparison to state-of-the-art NAS methods on CIFAR10. The architectures A–F are the Pareto-optimal obtained from a single search procedure ($\mu = 1$, better efficiency), and G–L from another procedure ($\mu = 0$). The search cost is for all six architectures sharing the same μ. Each searched architecture is re-trained three times individually.



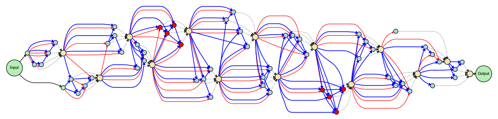
(a) GOLD-NAS-$\mathcal{S}_0$-A, 1.58M, 2.99% error



(b) GOLD-NAS-$\mathcal{S}_0$-G, 2.22M, 2.75% error



(c) GOLD-NAS-$\mathcal{S}_0$-F, 2.08M, 2.81% error



(d) GOLD-NAS-$\mathcal{S}_0$-L, 3.67M, 2.53% error

Figure 2: The first (with the highest efficiency) and last (with the highest accuracy) architectures found by two search procedures with $\mu = 1$ (left) and $\mu = 0$ (right). The red thin, blue thick, and black dashed arrows indicate skip-connect, sep-conv-3x3, and concatenation, respectively. This figure is best viewed in a colored and zoomed-in document.

requires strict mathematical conditions ($\boldsymbol{\omega}$ needs to be optimal when $\boldsymbol{\alpha}$ gets updated, which is almost impossible to guarantee (Liu et al., 2019)), yet the second-order gradient is very difficult to be accurately estimated (Bi et al., 2019). In comparison, GOLD-NAS is built on one-level optimization and avoids these burdens. Meanwhile, some useful/efficient optimization methods (*e.g.*, partial channel connection (Xu et al., 2020a)) can be incorporated into GOLD-NAS towards better search performance. **Second, GOLD-NAS achieves better tradeoff between accuracy and efficiency,** as shown in Figure 1. This mainly owes to its flexibility of assigning computational resources. **Third, GOLD-NAS finds a set of Pareto-optimal architectures within one search procedure.** This is more efficient than existing methods that achieved the same goal running individual search procedures with different constraints (Xie et al., 2018) or coefficients (Xu et al., 2020b).

GOLD-NAS also works well in other search spaces, $\mathcal{S}_1$–$\mathcal{S}_5$. We plot the Pareto fronts of $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_4$ in Figure 1, and provide all detailed numbers and searched architectures in Appendix B.4. These results indicate that GOLD-NAS easily transfers to different scenarios even when some critical operators such as skip-connect and sep-conv-3x3 are missing. In particular, in $\mathcal{S}_1$, GOLD-NAS achieves the 3%-error mark with 1.44M parameters, even lower than that in $\mathcal{S}_0$. That being said, GOLD-NAS takes advantage of the seemingly weak operators to arrive at a better accuracy-complexity tradeoff. Provided larger GPU memory, GOLD-NAS has the potential of exploring complex search spaces, possibly with little knowledge that which operators are better.

Last but not least, we investigate the performance of random search. Following prior work (Li & Talwalkar, 2019; Liu et al., 2019), we individually sample 24 valid architectures from $\mathcal{S}_0$ and evaluate the performance in a 100-epoch validation process (for technical details, please refer to Appendix B.5). The best architecture is taken into a standard re-training process. We perform random search three times and report an average error of $3.31 \pm 0.50\%$, number of parameters of 2.30 ± 0.49M, and FLOPs of 368 ± 73M. This is far behind the Pareto front by GOLD-NAS. The random search experiments in $\mathcal{S}_1$–$\mathcal{S}_5$ report the same conclusion (see Appendix B.5).

3 Generalizing GOLD-NAS to ImageNet

To reveal the generalization ability, we evaluate GOLD-NAS on the ImageNet-1K (ILSVRC2012) dataset (Deng et al., 2009; Russakovsky et al., 2015), which contains 1.3M training and 50K testing images. Following (Xu et al., 2020a), we both transfer the searched architectures from CIFAR and directly search for architectures on ImageNet. The search space remains unchanged as in CIFAR10, but three convolution layers of a stride of 2 are inserted between the input image and the first cell, down-sampling the image size from 224×224 to 28×28. Other hyper-parameter settings are mostly borrowed from (Xu et al., 2020a), as described in Appendix C.1.

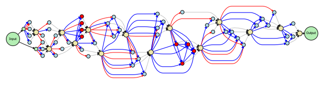
A common protocol of ImageNet-1K is to compete under the mobile setting, *i.e.*, the FLOPs of the searched architecture does not exceed 600M. We perform three individual search procedures with the basic channel number being 44, 46, and 48, respectively. We set $\mu = 0$ to achieve higher accuracy. From each Pareto front, we take the architecture that has the largest (but smaller than 600M) FLOPs for re-training. The three architectures with basic channels numbers of 44, 46 and 48 are assigned with code of X–Z, respectively. We also transplant a smaller architecture (around 500M FLOPs) found in the 44-channel search process to (590M FLOPs) by increasing the channel number from 44 to 48 – we denote this architecture as GOLD-NAS-Z-tr.

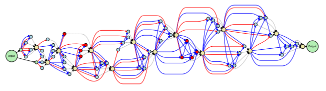
Results are summarized in Table 3. GOLD-NAS shows competitive performance among state-of-the-arts. In particular, the transferred GOLD-NAS-I reports a top-1 error of 24.7%, and the directly searched GOLD-NAS-Z reports 24.0%. Interestingly, GOLD-NAS-Z-tr reports 23.9%, showing that a longer pruning procedure often leads to higher resource efficiency. Also, GOLD-NAS enjoys smaller search costs, *e.g.*, the cost of GOLD-NAS-Z (1.7 GPU-days) is more than $2\times$ faster than prior direct search methods (Cai et al., 2019; Xu et al., 2020a).

The searched architectures on ImageNet are shown in Figure 3. GOLD-NAS tends to increase the portion of sep-conv-3x3, the parameterized operator, in the middle and late stages (close to output) of the network. This leads to an increase of parameters compared to the architectures found in the original space. This implies that GOLD-NAS assigns computational resource to different network stages more flexibly, which mainly owes to the enlarged search space and the stable search algorithm.

Architecture	**Test Err. (%)**		**Params**	$\times+$	**Search Cost**
	top-1	**top-5**	**(M)**	**(M)**	**(GPU-days)**
Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	1448	-
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	-
ShuffleNet $2\times$ (v2) (Ma et al., 2018)	25.1	-	$\sim$5	591	-
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	564	1800
MnasNet-92 (Tan et al., 2019)	25.2	8.0	4.4	388	-
AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	570	3150
SNAS (mild) (Xie et al., 2018)	27.3	9.2	4.3	522	1.5
ProxylessNAS‡ (Cai et al., 2019)	24.9	7.5	7.1	465	8.3
DARTS (Liu et al., 2019)	26.7	8.7	4.7	574	4.0
P-DARTS (Chen et al., 2019)	24.4	7.4	4.9	557	0.3
PC-DARTS (Xu et al., 2020a)‡	24.2	7.3	5.3	597	3.8
GOLD-NAS-$\mathcal{S}_0$-I	24.7	7.4	5.4	586	1.1
GOLD-NAS-$\mathcal{S}_0$-X‡	24.3	7.3	6.4	585	2.5
GOLD-NAS-$\mathcal{S}_0$-Y‡	24.3	7.5	6.4	578	2.1
GOLD-NAS-$\mathcal{S}_0$-Z‡	24.0	7.3	6.3	585	1.7
GOLD-NAS-$\mathcal{S}_0$-Z-tr‡	23.9	7.3	6.4	590	1.7

Table 3: Comparison with state-of-the-arts on ImageNet-1K, under the *mobile setting*. ‡: these architectures are searched on ImageNet, while others searched on CIFAR10.


(a) GOLD-NAS-X


(b) GOLD-NAS-Z

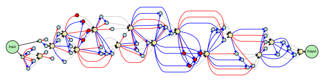

(c) GOLD-NAS-Z-tr

Figure 3: Three architectures found on ImageNet. The red thin, blue thick, and black dashed arrows indicate skip-connect, sep-conv-3x3, and concatenation, respectively. This figure is best viewed in color and by zooming in.

4 CONCLUSIONS

In this paper, we present a novel algorithm named **GOLD-NAS** (Gradual One-Level Differentiable Neural Architecture Search). Starting with the need of exploring a more challenging search space, we make use of one-level differentiable optimization and reveal the main reason for the failure lies in the discretization error. To alleviate it, we propose a gradual pruning procedure in which the resource usage plays the role of regularization that increases with time. GOLD-NAS is able to find a set of Pareto-optimal architectures with one search procedure. The search results on CIFAR10 and ImageNet demonstrate that GOLD-NAS achieves a nice tradeoff between accuracy and efficiency.

Our work delivers some new information to the NAS community. **First**, we encourage the researchers to avoid manually designed rules. This often leads to a larger search space yet very different architectures to be found – provided with stable search methods, these newly discovered architectures can be more efficient. **Second** and more importantly, reducing the optimization gap brings benefit to NAS. GOLD-NAS alleviates *discretization error*, one specific type of optimization gap, but it is imperfect as it still requires network reshape and re-training. We are looking forward to extending GOLD-NAS into a completely end-to-end search method that can incorporate various types of hardware constraints. This is an important future research direction.

References

Kaifeng Bi, Changping Hu, Lingxi Xie, Xin Chen, Longhui Wei, and Qi Tian. Stabilizing darts with amended gradient estimation on architectural parameters. *arXiv preprint arXiv:1910.11831*, 2019.

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.

Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

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.

Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019a.

Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. *arXiv preprint arXiv:1911.12126*, 2019b.

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.

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.

Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Computer Vision and Pattern Recognition*, 2020.

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

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.

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.

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.

Guilin Li, Xing Zhang, Zitong Wang, Zhenguo Li, and Tong Zhang. Stacnas: Towards stable and consistent optimization for differentiable neural architecture search. *arXiv preprint arXiv:1909.11926*, 2019.

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.

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.

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.

Richard Shin, Charles Packer, and Dawn Song. Differentiable neural network architecture search. In *International Conference on Learning Representations – Workshops*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

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.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Computer Vision and Pattern Recognition*, pp. 10734–10742, 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.

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*, 2020a.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Bowen Shi, Qi Tian, and Hongkai Xiong. Latency-aware differentiable neural architecture search. *arXiv preprint arXiv:2001.06392*, 2020b.

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 Details of the Search Algorithm

A.1 How Many Architectures Are There in the Search Spaces?

In the basic search space, $\mathcal{S}_0$, each cell (either normal or reduction) is individually determined. In each cell, there are 4 intermediate nodes, each of which can receive inputs from its precedents. In each edge, there are two possible operators, sep-conv-3x3 and skip-connect. To guarantee validity, each node must have at least one preserved operator (on any edge). This means that the n-th node ($n = 2, 3, 4, 5$) contributes $2^{2n} - 1$ possibilities because each of the $2n$ operators can be on or off, but the situation that all operators are off is invalid. Therefore, there are $\left(2^4 - 1\right)\left(2^6 - 1\right)\left(2^8 - 1\right)\left(2^{10} - 1\right) \approx 2.5 \times 10^8$ combinations for each cell. There are 14 or 20 cells, according to the definition of DARTS. If 14 cells are used, the total number of architectures in the search space is $\left(2.5 \times 10^8\right)^{14} \approx 3.1 \times 10^{117}$; if 20 cells are used, the number becomes $\left(2.5 \times 10^8\right)^{20} \approx 6.9 \times 10^{167}$.

For $\mathcal{S}_1$–$\mathcal{S}_5$, there are 4 active operators, so there are $\left(4^4 - 1\right)\left(4^6 - 1\right)\left(4^8 - 1\right)\left(4^{10} - 1\right) \approx 7.1 \times 10^16$ combinations for each cell. If 14 cells are used, the the total number of architectures in the search space is $\left(7.1 \times 10^16\right)^{14} \approx 9.6 \times 10^{235}$. Of course, under a specific FLOPs constraint, the number of architectures is much smaller than this number, but our space is still much more complex than the original one – this is a side factor that we can find a series of Pareto-optimal architectures in one search procedure.

A.2 One-Level Search in the Original Search Space

DARTS reported that one-level optimization failed dramatically in the original search space, *i.e.*, the test error is 3.56% on CIFAR10, which is even inferior to random search ($3.29 \pm 0.15\%$). We reproduced one-level optimization and reported a similar error rate of 3.54%.

We find that the failure is mostly caused by the over-fitting issue, as we have explained in the main article: the number of network weights is much larger than the number of architectural parameters, so optimizing the former is more effective but delivers no information to architecture search. To alleviate this issue, we add data augmentation to the original one-level optimization (only in the search phase, the re-training phase is unchanged at all). With merely this simple modification, the one-level searched architecture reports an error rate of $2.80 \pm 0.06\%$, which is comparable to the second-order optimization of DARTS and outperforms the first-order optimization of DARTS – note that both first-order and second-order optimization needs bi-level optimization. This verifies the potential of one-level optimization – more importantly, one-level optimization gets rid of the burden of inaccurate gradient estimation of bi-level optimization.

A.3 Calculation of the FLOPs Function

We first elaborate the ideology of designing the function. For a specific operator o, its FLOPs is easily measured by some mathematical calculation and written as a function of $\mathrm{FLOPs}(o)$. When we consider the architectural parameter $\boldsymbol{\alpha}$ in a differentiable search procedure, we should notice that the FLOPs term, $\mathrm{FLOPs}(\boldsymbol{\alpha})$, reflects the *expectation* of the FLOPs of the current architecture (parameterized by $\boldsymbol{\alpha}$). The calculation of $\mathrm{FLOPs}(\boldsymbol{\alpha})$ should consider three key points. For each individual operator o with an architectural parameter of α, (i) its expected FLOPs should increase with α, in particular, $\sigma(\alpha)$; (ii) to remove an operator from an edge, the average $\sigma(\alpha)$ value in the edge should be considered; (iii) as $\sigma(\alpha)$ goes towards 1, the penalty that it receives should increase slower – this is to facilitate a concentration of weights. Considering the above condition, we design the FLOPs function as follows:

$$\mathrm{FLOPs}(\boldsymbol{\alpha}) = \sum_{o} \ln\Big(1 + \sigma(\alpha_o) \,/\overline{\sigma(\boldsymbol{\alpha})}\Big) \cdot \mathrm{FLOPs}(o), \tag{5}$$

where the design of $\ln(1+\cdot)$ is to guarantee convexity, and we believe this form is not the optimal choice. The *uniform* version of $\mathrm{FLOPs}(\boldsymbol{\alpha})$, $\overline{\mathrm{FLOPs}}(\boldsymbol{\alpha})$, is computed via setting $\mathrm{FLOPs}(o)$ of all operators to be identical, so that the search algorithm mainly focuses on the impact of each operator on the classification error. That is to say, $\overline{\mathrm{FLOPs}}(\boldsymbol{\alpha})$ suppresses the operator with the least contribution to the classification task while $\mathrm{FLOPs}(\boldsymbol{\alpha})$ tends to suppress the most expensive operator first.

In practice, we use the `thop` library to calculate the terms of $\mathrm{FLOPs}(o)$. Let C be the number of input and output channels, and W and H be the width and height of the output. Then, the FLOPs of a skip-connect operator is 0 if the stride is 1 and $\mathrm{FLOPs}(o) = C^2HW$ if the stride is 2, and the FLOPs of a sep-conv-3x3 operator is $\mathrm{FLOPs}(o) = 2 \times \big(C^2HW + 9 \times CHW\big)$ (note that there are two cascaded convolutions in this operator).

B ADDITIONAL INFORMATION OF CIFAR10 EXPERIMENTS

B.1 HYPER-PARAMETER SETTINGS

First of all, we tried both 14-cell and 20-cell settings for the entire network, and found that they produced similar performance but the 14-cell setting is more efficient, so we keep this setting throughout the remaining part of this paper.

We use 14 cells for both search and re-train procedure, and the initial channels before the first cell is set to be 36. During the search procedure, all the architectural parameters are initialized to zero. The batch size is set to be 96. An SGD optimizer with a momentum of 0.9 is used to update the architectural parameters, $\boldsymbol{\alpha}$, and the learning rate $\eta_{\boldsymbol{\alpha}}$ is set to be 1. Another SGD optimizer is used to update the network parameters, and the only difference is that the learning rate $\eta_{\boldsymbol{\omega}}$ is set to be 0.01. The pruning pace n_0 is set to be 4, and it could be either increased to accelerate the search process (faster but less accurate) or decreased to smooth the search process (slower but more accurate). The pruning thresholds ξ_{max} and ξ_{min} are set to be 0.05 and 0.01. c_0 is set to be 2 for simplicity, and similar to n_0, it can be adjusted to change the pace of the pruning process. λ_0 is set to be 1×10^{-5}, which is chosen to make the two terms of loss function comparable to each other. t_0 is set to be 3, and it can be increased to improve the stability of the Pareto-optimal architectures or decreased to obtain a larger number of Pareto-optimal architectures. $\mathrm{FLOPs}_{\mathrm{min}}$ is set to be $240\mathrm{M}$ for $\mu = 1$ and $360\mathrm{M}$ for $\mu = 0$: this parameter is not very important because we can terminate the search process at anywhere we want. AutoAugment is applied in the search procedure to avoid over-fitting (*i.e.*, the network is easily biased towards tuning $\boldsymbol{\omega}$ than $\boldsymbol{\alpha}$, see Appendix A.2), but we do not use it during the re-training process for the fair comparison against existing approaches.

The re-training process remains the same as the convention. Each Pareto-optimal architecture is trained for 600 epochs with a batch size of 96. We use SGD optimizer with a momentum of 0.9, and the corresponding learning rate is initialized to 0.025 and annealed to zero following a cosine schedule. We use cutout, path Dropout with a probability of 0.2, and an auxiliary tower with a weight of 0.4 during the training process. The training process takes 0.3 to 1.2 days on a single NVIDIA Tesla-V100 GPU, according to the complexity (FLOPs) of each search architecture.

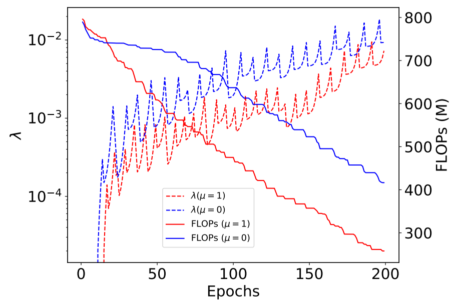
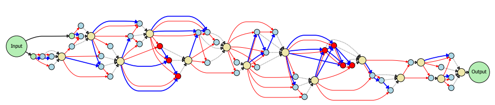


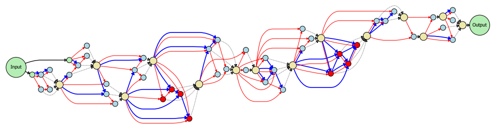
Figure 4: Visualization of two search procedures on CIFAR10 with $\eta = 0$ (blue) and $\eta = 1$ (red), respectively. λ zigzags from 0 to a large value, and the FLOPs of the super-network goes down.

B.2 ANALYSIS ON THE SEARCH PROCEDURE

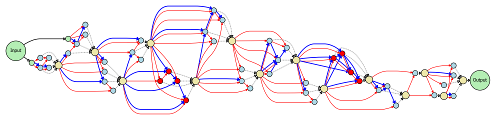
In Figure 4, we visualize the search procedures on CIFAR10 using the hyper-parameters of $\eta = 0$ and $\eta = 1$. We can observe that, as the search procedure



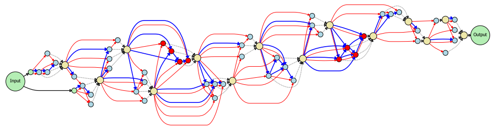
GOLD-NAS-$\mathcal{S}_0$-A, 1.58M, 2.99% error



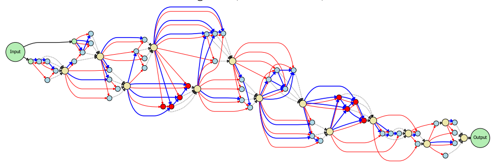
GOLD-NAS-$\mathcal{S}_0$-B, 1.72M, 2.97% error



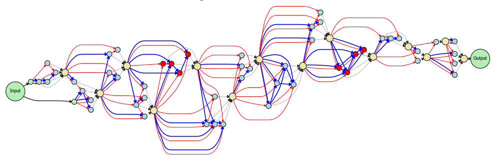
GOLD-NAS-$\mathcal{S}_0$-C, 1.76M, 2.96% error



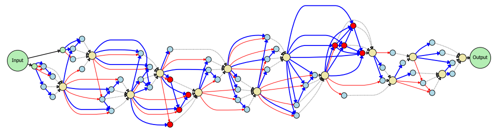
GOLD-NAS-$\mathcal{S}_0$-D, 1.89M, 2.90% error



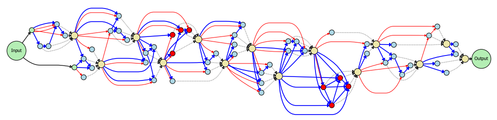
GOLD-NAS-$\mathcal{S}_0$-E, 1.99M, 2.83% error



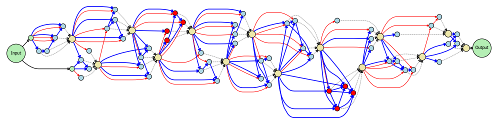
GOLD-NAS-$\mathcal{S}_0$-F, 2.08M, 2.81% error



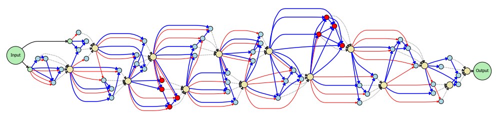
GOLD-NAS-$\mathcal{S}_0$-G, 2.22M, 2.75% error



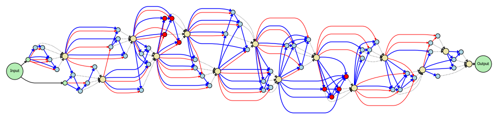
GOLD-NAS-$\mathcal{S}_0$-H, 2.51M, 2.70% error



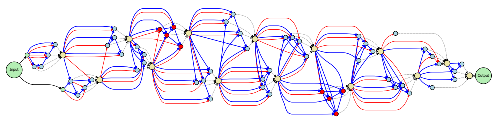
GOLD-NAS-$\mathcal{S}_0$-I, 2.85M, 2.61% error



GOLD-NAS-$\mathcal{S}_0$-J, 3.01M, 2.60% error



GOLD-NAS-$\mathcal{S}_0$-K, 3.30M, 2.57% error



GOLD-NAS-$\mathcal{S}_0$-L, 3.67M, 2.53% error

Figure 5: All architectures searched on CIFAR10 during two pruning procedures, $\eta = 1$ on the left side, $\eta = 0$ on the right side. The red thin, blue thick, and black dashed arrows indicate skip-connect, sep-conv-3x3, and concatenation, respectively. This figure is best viewed in a colored and zoomed-in document.

goes, weak operators are pruned out from the super-
network and the FLOPs of the network gradually goes down. With $\eta = 1$, the rate of pruning is much faster. More interestingly, λ, the balancing coefficient, zigzags from a small value to a large value. In each period, λ first goes up to force some operators to have lower weights (during this process, nothing is pruned and the architecture remains unchanged), and then goes down as pruning takes effect to eliminate the weak operators. Each local maximum (just before the pruning stage) corresponds to a Pareto-optimal architecture.

B.3 Visualization of the Searched Architectures

We show all searched architectures in $\mathcal{S}_0$, on CIFAR10, in Figure 5.

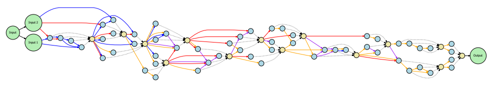
B.4 Detailed Results of Extended Search Spaces

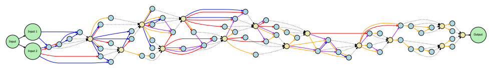
The detailed search results on $\mathcal{S}_1$–$\mathcal{S}_5$ are shown in Table 4. The searched architectures are visualized in Figure 6. Below, we briefly analyze the results in different search spaces.

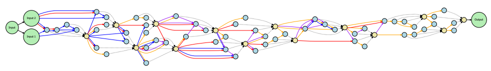
Architecture		**Test Err. (%)**				**Params**	**Search Cost**
		#1	**#2**	**#3**	**average**	**(M)**	**(GPU-days)**
Random search baseline		3.34	3.15	3.27	3.25±0.10	1.83	4.0
GOLD-NAS-$\mathcal{S}_1$-A	$\mu = 0$	3.24	3.05	3.25	3.18±0.11	1.25	1.7
GOLD-NAS-$\mathcal{S}_1$-B	$\mu = 0$	2.98	3.00	3.14	3.04±0.09	1.38	1.7
GOLD-NAS-$\mathcal{S}_1$-C	$\mu = 0$	2.85	2.80	3.03	2.89±0.12	1.44	1.7
GOLD-NAS-$\mathcal{S}_1$-D	$\mu = 0$	2.82	2.93	2.87	2.87±0.06	1.70	1.7
Random search baseline		3.22	3.44	3.42	3.36±0.12	2.02	4.0
GOLD-NAS-$\mathcal{S}_2$-A	$\mu = 0$	3.26	3.08	3.11	3.15±0.10	1.30	1.7
GOLD-NAS-$\mathcal{S}_2$-B	$\mu = 0$	3.07	3.24	3.14	3.15±0.09	1.44	1.7
GOLD-NAS-$\mathcal{S}_2$-C	$\mu = 0$	3.03	2.89	2.77	2.90±0.13	1.70	1.7
GOLD-NAS-$\mathcal{S}_2$-D	$\mu = 0$	2.69	2.61	2.79	2.70±0.09	2.13	1.7
Random search baseline		3.73	3.76	3.94	3.81±0.11	1.53	4.0
GOLD-NAS-$\mathcal{S}_3$-A	$\mu = 0$	3.28	3.42	3.56	3.42±0.14	1.43	1.7
GOLD-NAS-$\mathcal{S}_3$-B	$\mu = 0$	3.22	3.32	3.34	3.29±0.06	1.76	1.7
GOLD-NAS-$\mathcal{S}_3$-C	$\mu = 0$	3.24	3.43	3.18	3.28±0.13	1.85	1.7
Random search baseline		3.36	3.26	3.25	3.29±0.06	2.11	4.0
GOLD-NAS-$\mathcal{S}_4$-A	$\mu = 0$	3.06	2.84	3.01	2.97±0.12	1.45	1.7
GOLD-NAS-$\mathcal{S}_4$-B	$\mu = 0$	2.87	2.98	3.01	2.95±0.07	1.56	1.7
GOLD-NAS-$\mathcal{S}_4$-C	$\mu = 0$	2.86	2.78	2.84	2.83±0.04	2.04	1.7
Random search baseline		3.26	3.50	3.16	3.31±0.17	2.55	4.0
GOLD-NAS-$\mathcal{S}_5$-A	$\mu = 0$	3.07	3.05	2.99	3.04±0.04	1.50	1.7
GOLD-NAS-$\mathcal{S}_5$-B	$\mu = 0$	2.98	3.00	2.96	2.98±0.02	1.84	1.7
GOLD-NAS-$\mathcal{S}_5$-C	$\mu = 0$	2.79	2.74	2.97	2.83±0.12	2.10	1.7
GOLD-NAS-$\mathcal{S}_5$-D	$\mu = 0$	2.74	2.77	2.78	2.76±0.02	2.77	1.7
GOLD-NAS-$\mathcal{S}_5$-E	$\mu = 0$	2.72	2.71	2.66	2.70±0.03	2.88	1.7
GOLD-NAS-$\mathcal{S}_5$-F	$\mu = 0$	2.62	2.70	2.68	2.67±0.04	3.38	1.7

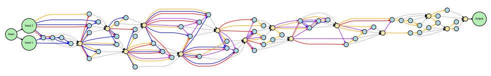
Table 4: Results of five extended search spaces. We have used a fixed rule to choose representative architectures, so the numbers of architectures may vary among different search spaces and/or procedures.

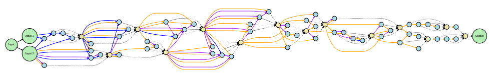
- $\mathcal{S}_0$ is the basic search space with skip-connect and sep-conv-3x3 which are believed to be the most useful operators. This space has already achieved a good accuracy-complexity tradeoff.
- $\mathcal{S}_1$ adds max-pool-3x3 and dil-conv-3x3, the two most useful operators not contained in $\mathcal{S}_0$. As expected, although the new operators do not appear very often, having these 'new functions' can arrive at a better tradeoff, *e.g.*, the architecture with $1.44\mathrm{M}$ parameters reports a 2.89% error on CIFAR10, better than the $1.58\mathrm{M}$-parameter, 2.99%-error architecture found in $\mathcal{S}_0$.
- $\mathcal{S}_5$ adds avg-pool-3x3 and dil-conv-5x5 to $\mathcal{S}_0$. These two operators rarely appear in previous search results, implying that they are less efficient. Consequently, $\mathcal{S}_5$ reports slightly worse performance than that of $\mathcal{S}_0$, indicating that GOLD-NAS may be impacted by the weak operators (*e.g.*, not able to prune them all so that the performance is at least on par with $\mathcal{S}_0$). This is important for future improvement.
- $\mathcal{S}_2$ is a search space without skip-connect, and two pooling operators max-pool-3x3 and avg-pool-3x3, are present as non-parameterized operators. The search performance is slightly worse than other search spaces, indicating that skip-connect, offering the ability of direct copy, seems the most important. The pooling operators suffers information loss and thus are not as efficient as skip-connect.
- $\mathcal{S}_3$ and $\mathcal{S}_4$ do not contain sep-conv-3x3, the critical parameterized operator that makes up most search results. The key is to use other parameterized operators for replacement. Interestingly, GOLD-NAS produces satisfying results in $\mathcal{S}_4$ (comparable to that in $\mathcal{S}_0$), implying the usefulness of sep-conv-5x5. On the other hand, the results are worse in $\mathcal{S}_3$ with two dilated convolutions, arguably because dilated convolutions are equipped with large receptive fields and thus are not friendly to the CIFAR10 dataset.

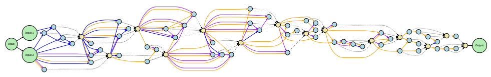

GOLD-NAS-$\mathcal{S}_1$-A, 1.25M, 3.18% error

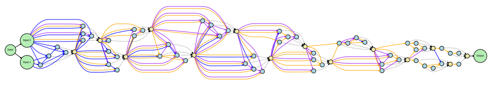

GOLD-NAS-$\mathcal{S}_1$-B, 1.38M, 3.04% error

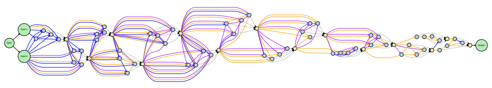

GOLD-NAS-$\mathcal{S}_1$-C, 1.44M, 2.89% error

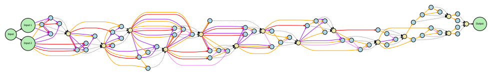

GOLD-NAS-$\mathcal{S}_1$-D, 1.70M, 2.87% error

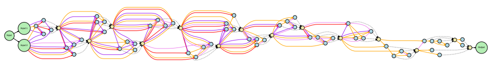

GOLD-NAS-$\mathcal{S}_2$-A, 1.30M, 3.15% error

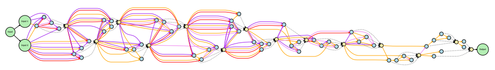

GOLD-NAS-$\mathcal{S}_2$-B, 1.44M, 3.15% error

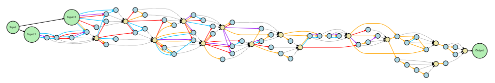

GOLD-NAS-$\mathcal{S}_2$-C, 1.70M, 2.90% error

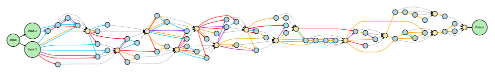

GOLD-NAS-$\mathcal{S}_2$-D, 2.13M, 2.70% error

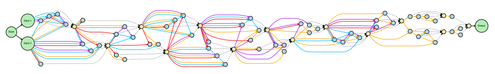

GOLD-NAS-$\mathcal{S}_3$-A, 1.43M, 3.42% error

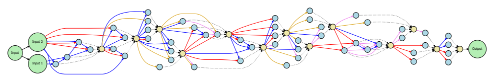

GOLD-NAS-$\mathcal{S}_3$-B, 1.76M, 3.29% error

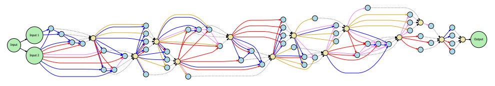

GOLD-NAS-$\mathcal{S}_3$-C, 1.85M, 3.28% error

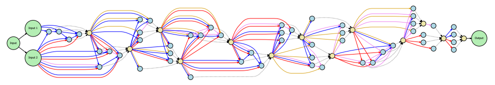

GOLD-NAS-$\mathcal{S}_4$-A, 1.45M, 2.97% error

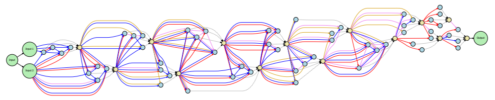

GOLD-NAS-$\mathcal{S}_4$-B, 1.56M, 2.95% error

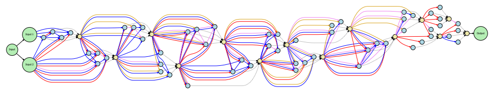

GOLD-NAS-$\mathcal{S}_4$-C, 2.04M, 2.83% error


GOLD-NAS-$\mathcal{S}_5$-A, 1.50M, 3.04% error


GOLD-NAS-$\mathcal{S}_5$-B, 1.84M, 2.98% error


GOLD-NAS-$\mathcal{S}_5$-C, 2.10M, 2.83% error


GOLD-NAS-$\mathcal{S}_5$-D, 2.77M, 2.76% error


GOLD-NAS-$\mathcal{S}_5$-E, 2.88M, 2.70% error

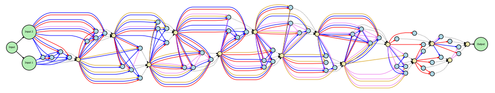
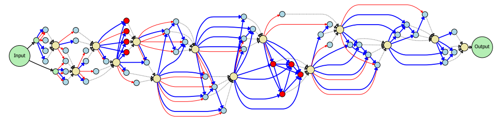

GOLD-NAS-$\mathcal{S}_5$-F, 3.38M, 2.67% error

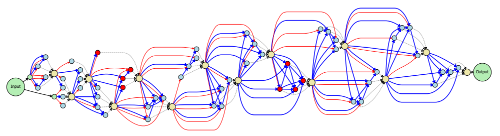
Figure 6: All architectures searched on CIFAR10 in $\mathcal{S}_1$–$\mathcal{S}_5$. The red thin, blue thick, and black dashed arrows indicate skip-connect, sep-conv-3x3, and concatenation, respectively. Other operators are presented using solid arrows: bright orange for max-pool-3x3, light blue for sep-conv-5x5, dark orange for avg-pool-3x3, bright purple for dil-conv-3x3, dark purple (magenta) for dil-conv-5x5.

B.5 Details of Random Search Experiments

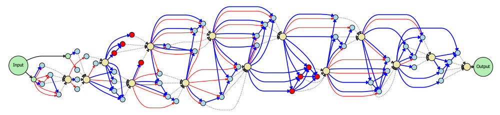
To produce the random search baseline, we randomly prune out operators from the super-network until the architecture fits the hardware constraint (*e.g.*, FLOPs). It is possible that the architecture becomes invalid during the random pruning process, and we discard such architectures. Each random search process collects 24 architectures and we train each of them for 100 epochs and pick up the best one for an entire 600-epoch re-training. As reported in the paper, we perform random search in $\mathcal{S}_0$ three times and the best architecture reports an average accuracy of $3.31 \pm 0.50\%$.



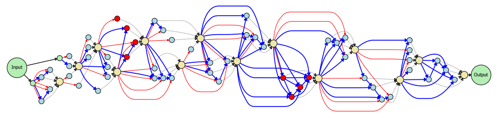
GOLD-NAS-$\mathcal{S}_0$-X, $6.4\mathrm{M}$, 24.3% top-1 error



GOLD-NAS-$\mathcal{S}_0$-Z, $6.4\mathrm{M}$, 24.3% top-1 error



GOLD-NAS-$\mathcal{S}_0$-Y, $6.3\mathrm{M}$, 24.0% error



GOLD-NAS-$\mathcal{S}_0$-Z-tr, $6.4\mathrm{M}$, 23.9% error

Figure 7: All architectures searched on ImageNet. The red thin, blue thick, and black dashed arrows indicate skip-connect, sep-conv-3x3, and concatenation, respectively. This figure is best viewed in a colored and zoomed-in document.

We have also performed random search on $\mathcal{S}_1$–$\mathcal{S}_5$ and the results are summarized in Table 4. Compared to the architectures found by GOLD-NAS with approximately the same amount of parameters, the best architectures obtained by random search suffer a $\sim 0.5\%$ deficit in all search spaces. This aligns with the conclusion in $\mathcal{S}_0$ that GOLD-NAS can find more efficient computational models.

C Additional Information of ImageNet Experiments

C.1 Hyper-parameter Settings

Following FBNet (Wu et al., 2019) and PC-DARTS (Xu et al., 2020a), we randomly sample 100 classes from the original $1{,}000$ classes of ImageNet to reduce the search cost. We do not AutoAugment during the search procedure as the training set is sufficiently large to avoid over-fitting. Other super-parameters are kept unchanged as the CIFAR10 experiments except for $\mathrm{FLOPs}_{\mathrm{min}}$, which is set to be $500\mathrm{M}$ for the ImageNet experiments.

During the re-training process, the total number of epochs is set to be 250. The batch size is set to be $1{,}024$ (eight cards). We use an SGD optimizer with an initial learning rate of 0.5 (decayed linearly after each epoch till 0), a momentum of 0.9 and a weight decay of 3×10^{-5}. The search process takes around 3 days on eight NVIDIA Telsa-V100 GPUs.

C.2 Hyper-parameter Settings

We show all searched architectures on ImageNet in Figure 7. Besides GOLD-NAS-$\mathcal{S}_0$-Y, other three architectures have been displayed in Section 3