STACNAS: TOWARDS STABLE AND CONSISTENT OPTIMIZATION FOR DIFFERENTIABLE NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Earlier methods for Neural Architecture Search were computationally expensive. Recently proposed Differentiable Neural Architecture Search algorithms such as DARTS can effectively speed up the computation. However, the current formulation relies on a relaxation of the original problem that leads to unstable and suboptimal solutions. We argue that these problems are caused by three fundamental reasons: (1) The difficulty of bi-level optimization; (2) Multicollinearity of correlated operations such as max pooling and average pooling; (3) The discrepancy between the optimization complexity of the search stage and the final training. In this paper, we propose a grouped variable pruning algorithm based on one-level optimization, which leads to a more stable and consistent optimization solution for differentiable NAS. Extensive experiments verify the superiority of the proposed method regarding both accuracy and stability. Our new approach obtains state-of-the-art accuracy on CIFAR-10, CIFAR-100 and ImageNet.

1 INTRODUCTION

Using 3150 GPU days, Google researchers showed in (Real et al. (2018)) that it is possible to automatically find neural network architectures that are superior to the ones designed by human experts. However, this method needs to fully train each candidate architecture until convergence. Two factors are known to contribute to the difficulty of the *NAS* problem. First, the search space is huge: the most popular *NASNet search space* (Zoph et al. (2018)) contains approximately 10^{15} models. Second, the basic method to evaluate the performance of a given architecture is to fully train it from scratch, and this is very time-consuming. To improve the efficiency of *NAS*, Pham et al. (2018) proposed ENAS where all child models are forced to share weights to avoid training each model separately from scratch. DARTS (Liu et al. (2018)) relaxes the discrete search space to be continuous so that one can use stochastic gradient descent to simultaneously learn the architectures and model parameters on a given dataset. Unlike randomized, evolutionary, and reinforcement learning based discrete architecture search, DARTS can score every architecture in the search space by training a single architecture model, and the total computational cost is roughly the same as the training time of using the final architecture. To reduce the search complexity, DARTS employs the idea of computation cell as the building block of the final architecture, and this is the same as Zoph et al. (2018); Real et al. (2018). The learned cell is then stacked together to form a wider and deeper convolutional neural network, which becomes the final architecture.

However, the current differentiable approach relies on a relaxation of the original problem that leads to unstable and suboptimal solutions. In our extensive experiments with DARTS, we found that DARTS suffers from the following problems. First, the performance variation of the architectures found by DARTS with different random initial seeds is very large. In some cases, DARTS may find architectures with worst performance than the average of randomly picked architectures in the search space. This is also the reason why DARTS needs to use 4 GPU days to perform the search four times and then use another 1 GPU day to pick the best architecture. Second, although DARTS works fine on CIFAR-10, it performs poorly on many other datasets. For example, on CIFAR-100, when we run DARTS until convergence, it ends up with architectures with many skip connections and poor performance. Third, our experiments indicate that the correlation between the accuracy of the fully trained stand-alone model vs the score obtained from DARTS during the search phase is

only 0.2. This means that DARTS relaxation is not a good approximation in terms of predicting the performance of the stand-alone final architecture.

The above mentioned problems are caused by approximation errors between discrete architectures and their continuous relaxations. In addition, we found that the performance of DARTS is negatively affected by the following three factors, which we will remedy in this work: (1) The difficulty of bi-level optimization: the optimization algorithm relies on a 2nd order approximation, which is slow and may not find the optimal solution. (2) Multicollinearity of correlated operations such as max pooling and average pooling. When there are two operations corresponding to feature maps that are highly correlated, the weights assigned to their architectures may no longer represent their true importance. Appropriate operations could be pruned as a result. This is similar to the problem caused by multicollinearity in regression. (3) The discrepancy between the optimization complexity of the search stage architecture and the final architecture. As well studied (He et al. (2016); Sankararaman et al. (2019)), including skip connections would make the gradient flows easier and optimization of deep neural network more stable. In our experiment with multiple datasets, when the parent architecture in the proxy search space is too shallow, the algorithms would tend to select fewer skip connections and when the parent architecture is too deep, the algorithm would select more than the optimal number of skip connections. Therefore matching the search stage and final stage optimization complexity is needed to allow the system to select the proper amount of skip connections. However, since DARTS can only handle relatively shallow models under the same computational resource constraints, this leads to suboptimality.

To address these problems, we propose grouped backward pruning solution for *NAS* called StacNAS (STAble and Consistant differentiable Neural Architecture Search) based on one-level optimization, which allows the matching of the optimization complexity for search and final training. These improvements allow StacNAS to achieve the state-of-the-art performance using the *NASNet search space* (Zoph et al. (2018)).

In our experiments in Section (3), we show that StacNAS is able to find a convolutional cell that achieves a test error of 2.33 (with same training code as DARTS for fair comparison with DARTS) and 1.85 with additional training tricks used in other works (this is for fair comparison with performance reported in some recent papers employing such tricks) on CIFAR-10 for image classification using around 3.6M parameters. This is the current state-of-the-art result among all *NAS* methods. Since we employ the simple one-level optimization, we are also able to directly search over IMAGENET (mobile setting), and achieve a top-1 error of 24.3 (with DARTS code) and 23.48 (with additional training tricks). This result beats the current state-of-the-art $EfficientNetB0$ (Tan & Le (2019)) at the same model complexity.

Our contributions can be summarized as follows:

1. We introduce the idea of grouping similar operations to compensate for the effect of multicollinearity. This allows our method to select good operators more accurately.
2. We use a progressive backward variable pruning approach to solve the discrepancy between the optimization complexity of the search stage proxy architecture and training architecture. We invite the unit gradient confusion (Sankararaman et al. (2019)) to measure the optimization complexity of the search and training architectures.
3. We show that coupled with the grouped backward variable pruning solution, the simpler one-level joint optimization of both architecture and model parameters is sufficient for NAS with differentiable relaxation. Because it is easier to ensure convergence in one-level optimization, our method converges significantly faster, and it can scale up to complex models and large data sets, enabling direct search on ImageNet.

2 METHODOLOGY

In this section, we present our joint optimization approach with grouped variable pruning as an improved solution to the difficulties observed in differentiable relaxation of NAS. Section 2.1 describes the search space used in this work. Section 2.2 summarizes the differentiable relaxation of NAS proposed in Liu et al. (2018), which is the basic model we try to improve. Section 2.3 introduces our approach of joint optimization with grouped variable pruning.

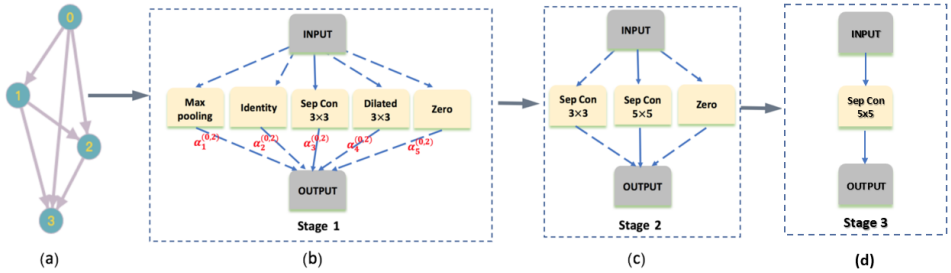

Figure 1: An overview of StacNAS: (a) A cell is a DAG with nodes connected by unknown operations. (b) Improved continuous relaxation by placing a by-group mixture of candidate operations on each edge. (c) When a certain type of operation is activated, include all candidate operations in this type to form a new mixture operation. (d) Deriving the final architecture.

2.1 Search Space

We first use similar search space as that of DARTS, where a convolutional neural network is formed by stacking a series of building blocks called cells (Zoph et al. (2018); Real et al. (2018); Liu et al. (2018)), and only two types of cells are learned: the *normal cell* and the *reduction cell*. The only difference between them is that every reduction cell reduces the image size by a factor of 2 through the use of stride 2. Then we enlarge the search space to allow all cells in the final architecture to be different.

A cell can be seen as a directed acyclic graph (DAG) as shown in Figure 1, where an ordered sequence of N nodes $\{x_1, \cdots, x_N\}$ are connected by directed edges (i, j). Each node x_i is a latent representation (i.e. a feature map) and each directed edge (i, j) represents some operation $o(\cdot) \in O$ that transforms x_i. Within a cell, each internal node is computed as the sum of all its predecessors:

$$x_j = \sum_{i<j} o^{(i,j)}(x_i). \tag{1}$$

As in Liu et al. (2018), we include the following 7 candidate operations: identity; 3×3 average pooling; 3×3 max pooling; 3×3 separable convolutions; 5×5 separable convolutions; 3×3 dilated separable convolutions; 5×5 dilated separable convolutions. A special *zero* operation is also included to serve as a scaling factor for each edge (i, j) . According to Liu et al. (2018), for a cell with $N = 4$ nodes, this corresponds to 10^{18} possible choices of architectures.

In this framework, the task of designing the cell structure is to determine the most important two preceding edges for all internal nodes and the best choice of operations for these selected edges.

2.2 Differentiable Relaxation of NAS

DARTS (Liu et al. (2018)) proposed a continuous relaxation of the categorical choice of operations and edges so that the relative importance of them can be learned through stochastic gradient descent (SGD). Specifically, to make the search space continuous, DARTS replaces the discrete choice of operation $o(\cdot) \in O$ with a weighted sum over all candidate operations (Figure 1):

$$\bar{o}^{(i,j)}(x) = \sum_{o \in O} \alpha_o^{(i,j)} o^{(i,j)}(x_i) \quad \text{where} \quad \alpha_o^{(i,j)} = \frac{\exp(\beta_o^{(i,j)})}{\sum_{o' \in O} \exp(\beta_{o'}^{(i,j)})} \tag{2}$$

which is called the architecture parameters.

Intuitively, a well learned $\alpha = \{\alpha_o^{(i,j)}\}$ could represent the relative importance/contribution of the operation $o^{(i,j)}$ for transforming the feature map x_i.

After the relaxation, the task of architecture search reduces to learning a set of continuous architecture parameters $\alpha = \{\alpha^{(i,j)}\}$. Specifically, one first trains a parent/proxy network that contains all candidate operations and connections/edges through Equation 1 and Equation 2. During training,

one would be able to learn which operations and which edges are redundant. After the training algorithm converges, a compact architecture is obtained by pruning the unimportant operations and edges.

In DARTS, the authors formulated the optimization of the architecture parameters α and the weights w associated with the architecture as a bi-level optimization problem (Anandalingam & Friesz (1992)), where α is treated as the upper-level variable and w as the lower-level variable:

$$\min_{\alpha} \; L_{val}(w^*(\alpha), \alpha) \tag{3}$$

$$\text{s.t. } w^*(\alpha) = \text{argmin}_w \; L_{train}(w, \alpha) \tag{4}$$

where L_{train} and L_{val} represent for the training and validation loss respectively.

Evaluating Equations 4 can be prohibitive due to the expensive inner optimization. Therefore, DARTS proposed to approximate $w^*(\alpha)$ by adapting w using only a single gradient update step, and update w and α alternatively using the training and validation set respectively.

After obtaining the continuous architecture parameters α, the redundant operations and edges are pruned to obtain a compact architecture by:

1. Retaining $k = 2$ strongest predecessors for each internal node, where the importance of an edge is defined as $max_{o \in O, o \neq zero} \alpha_o^{(i,j)}$;
2. Replacing every mixed operation with the most important operation: $\bar{o}^{(i,j)} = \text{argmax}_{o \in O} \alpha_o^{(i,j)}$.

2.3 A Stable and Consistant Optimization Algorithm

2.3.1 One-level Optimization

Although DARTS employed a bi-level optimization approach to avoid overfitting, it suffers from the following problems:

- Bi-level optimization is very difficult, leading to unstable convergence and poor results. In fact, if we run the original DARTS algorithm for a long time on CIFAR-10, then we obtain a solution with α_{zero} to be almost 1. Moreover, a direct application of DARTS on CIFAR-100 fails to find a meaningful architecture.
- The memory cost of the second-order approximation of bi-level optimization is more than twice of that of the one-level optimization [1], and hence the search stage has to use a much smaller proxy parent model for searching cell structures. The searched cells are then stacked into a much deeper network. This would cause the optimization discrepancy mentioned before, which we will describe in more details in Section 2.3.3.
- The original training data has to be split into training data and validation data. With the current 1:1 split ratio, only half of the data could be used to learn w. The learning of model parameter w can be negatively affected due to the lack of data. For example, on CIFAR-100, which contains more classes with fewer images in each class than CIFAR-10, the original DARTS algorithm selects cells dominated with identity and a large scaling α_{zero}.

To solve these difficulties, we employ one-level optimization, where we jointly optimize both the architecture parameter α and the model parameter w. In our experiment, we use all of the original training data to update α and w together by descending on w and α using

$$w_t = w_{t-1} - \eta_t \partial_w L_{train}(w_{t-1}, \alpha_{t-1}) \tag{5}$$

$$\alpha_t = \alpha_{t-1} - \delta_t \partial_\alpha L_{train}(w_{t-1}, \alpha_{t-1}) \tag{6}$$

simultaneously, where η_t and δ_t are learning rates.

Therefore, in this work, we advocate that the architecture parameter α and architecture weights w should be optimized simultaneously using one-level optimization. Coupled with the solutions proposed in Section 2.3.2 and Section 2.3.3, we show that the proposed integrated method based on one-level optimization can achieve the state-of-the-art performance.

[1] For two-level optimization, the gradients of w on both training and validation mini-batches have to be loaded in the memory to calculate the gradient of α

2.3.2 GROUPED VARIABLE PRUNING

In classical statistical literature, there is a phenomenon called multicollinearity, in which one predictor variable in a multiple regression model can be linearly predicted by other variables with a high degree of accuracy. In this situation, the coefficient estimates may change erratically with respect to small changes in the model or the training data. Therefore, for variable selection in a multivariate regression model with collinear predictors, the values of coefficients may no longer give valid results about which predictors are redundant.

The construction of the continuous search space as described in Section 2.1 is very similar to a multiple regression model: feature maps transformed by all candidate operations are summed together and weighted by coefficients α. The relative importance of operation o is learned through the value of α_o. When there are operations producing highly correlated feature maps (such as max pooling and average pooling), their weights α may no longer be representative of their real importance. To measure the operation correlation, we adopt the same procedure as Gastaldi (2017): we calculated the correlation between the flattened feature map of two operations for the first edge of the last cell. And the result suggests that max pooling and average pooling are in one group (their correlation is 0.52), 3×3 and 5×5 separable convolutions are in the same group (correlation 0.38), 3×3 and 5×5 dilated separable convolutions are in the same group (correlation 0.27). The details of the correlation calculation procedure and the full correlation matrix are provided in the Appendix B.

To verify the conjectures about multicollinearity would cause the system to prune important operations, we conducted an experiment with CIFAR-10, including an unbalanced number of operations from different groups in the search space, such as max pooling, 3×3 separable convolutions, 5×5 separable convolutions and *zero*. This ends up with a normal cell dominated with max pooling, although the architecture would have higher validation accuracy if separable convolutions are selected. The cause of this problem is due to the multicollinearity of the two separable convolutions so that they produce feature maps linearly correlated, which splits the contribution in the mixed operation. In the extreme condition, suppose they produce exactly the same feature maps, there is only one degree of freedom for them as long as $\alpha_{sep3\times3} + \alpha_{sep5\times5} = \alpha_{sep}$.

To remedy this problem, we propose a two-stage grouped backward variable pruning strategy for selecting the best operations based on one-level optimization as shown in Figure 1 and Algorithm 1. For the 7 candidate operations mentioned in Section 2.1, they are divided into 4 groups according to their linear correlation estimation. The correlation measures are obtained automatically from a training data (in our case, CIFAR-10) as detailed in the Appendix B. The grouping can also be done automatically based on the data.

- Group 1: identity (skip_connect);
- Group 2: 3×3 max pooling; 3×3 average pooling;
- Group 3: 3×3 separable convolutions; 5×5 separable convolutions;
- Group 4: 3×3 dilated separable convolutions; 5×5 dilated separable convolutions.

At stage 1, we only include one operation from each group to prevent the multicollinearity phenomenon. For example, we let the following four operations compete first to decide which types of operation to use at edge (i, j): identity; 3×3 max pooling; 3×3 separable convolutions; 3×3 dilated separable convolutions.

At stage 2, suppose that the operator 3×3 separable convolutions is selected at edge (i, j), we then replace the mixed operation $\widehat{o}^{(i,j)}$ with a weighted sum of all operations in the same group (Group 3). Again, at every stage, a special *zero* operation is also included to serve as a scaling factor for each edge (i, j).

2.3.3 OPTIMIZATION COMPLEXITY MATCHING

In our experiment with multiple datasets, when the search stage architecture is too shallow, the algorithms would tend to select fewer skip connections and when the search stage architecture is too deep, the algorithm would select more than the optimal number of skip connections. Both these two scenarios lead to poor performance of the final architecture.

Algorithm 1 StacNAS- Stable and Consistent differerentiable NAS

procedure STAGE 1 (Group search)
    Create a by-group operation $\tilde{o}^{(i,j)}(x)$ parametrized by $\tilde{\alpha}_o^{(i,j)}$ for each edge (i,j)
    if $t < T$ **then**
        1. Update weights w_t by descending $\partial_w L_{train}(w_{t-1}, \tilde{\alpha}_{t-1})$
        2. Update architecture parameter $\tilde{\alpha_t}$ by descending $\partial_{\tilde{\alpha}} L_{train}(w_{t-1}, \tilde{\alpha}_{t-1})$
    Set optimal $\tilde{\alpha}^* = \alpha^T$
    Activate the optimal operation group based on $\tilde{\alpha}^*$
procedure STAGE 2 (Backward)
    Use all the group member of $o^{*(i,j)}$ to create mixed operation $\hat{o}^{(i,j)}(x)$ parametrized by $\hat{\alpha}_o^{(i,j)}$
    if $t < T$ **then**
        1. Update weights w_t by descending $\partial_w L_{train}(w_{t-1}, \hat{\alpha}_{t-1})$
        2. Update architecture parameter α_t by descending $\partial_{\hat{\alpha}} L_{train}(w_{t-1}, \hat{\alpha}_{t-1})$
    Set optimal $\hat{\alpha}^* = \alpha_T$
    Select the best operation $o^{*(i,j)} \leftarrow \text{argmax}_{o \in O} \hat{\alpha}^{(i,j)}$
    Prune the redundant operations and edges.
procedure STAGE 3 (Deformable Architecture Training)
    Create $\widehat{o}^{(i,j)}(x)$ parametrized by $\widehat{\alpha}_o^{(i,j)}$ for each edge (i,j) by sum of $o^{*(i,j)}$ and $o_{zero}^{(i,j)}$.
    if $t < T$ **then**
        1. Update weights w_t by descending $\partial_w L_{train}(w_{t-1}, \alpha_{t-1})$
        2. Update architecture parameter α_t by descending $\partial_{\alpha} L_{train}(w_{t-1}, \alpha_{t-1})$

In DARTS, a stack of 8 cells are used to serve as the proxy parent network during the search stage (due to the high memory cost of bi-level optimization), and then a stack of 20 cells are used to build the final architecture. This introduces a significant gap between the optimization complexity of the proxy search architecture and the final architecture.

As well studied (He et al. (2016); Sankararaman et al. (2019)), including skip connections would make the gradient flows easier and optimization of the deep neural network more stable. Therefore, whether a NAS algorithm can select the proper amount of skip connections and their location would significantly affect the performance of the designed architecture. Hence, properly matching the search stage and final stage optimization complexity is vital towards a consistent optimization of the system.

To quantify the optimization difficulty more concretely, we use a measurement called gradient confusion, introduced in (Sankararaman et al. (2019)). The authors empirically show that when gradient confusion is high, stochastic gradients produced by different mini-batches may be negatively correlated, slowing down convergence. But when gradient confusion is low, SGD has better convergence properties. Deeper networks usually correspond to higher gradient confusion, and skip connection can significantly reduce gradient confusion.

Formally, gradient confusion is defined to be an upper bound of the negative inner product between gradients evaluated at all mini-batches. Formally, let $\{L_i\}_{i=1}^m$ be losses for mini-batches, and let W be trainable parameters. Then a gradient confusion $\zeta \geq 0$ at W could be estimated by the following formula

$$\zeta = \max\{0, \max_{i,j=1,\ldots,m} \{- < \nabla L_i(W), \nabla L_j(W) >\}\} \tag{7}$$

By calculating the gradient confusion (details are provided in the Appendix D), we can determine the desired depth (number of cells stacked) for the two search stages, to make their optimization complexity compatible with that of the final architecture. This allows the proposed algorithm to be able to optimize the search architecture with the same level of difficulty with the final architecture. We find that by matching the optimization complexity, the system would automatically select the optimal number of skip connections.

2.4 RELATIONSHIP TO PRIOR WORK

The authors of ProxylessNAS (Cai et al. (2018)) argued that the original DARTS algorithm cannot search architectures directly on ImageNet due to its large memory footprint, and proposed to solve the memory explosion problem by sampling only one path/operation at each edge to be activated at training time of the search stage. This solution reduces the memory requirement to the same level of training the final model, and thus it is possible to directly search on large datasets such as ImageNet without using a proxy dataset. In this work, we make it possible to directly search over large dataset by the combination of one-level optimization and the design of a hierarchical (two-stage) search space, as both schemes reduce the memory requirement of our algorithm compared to the original DARTS.

Similar to ProxylessNAS, SNAS (Xie et al. (2018)) also samples only one operation at each edge for each training step, where their purpose is to solve the inconsistency problem between the performance of derived child networks and converged parent networks instead of memory concerns. The authors proposed to use Gumbel random variable Maddison et al. (2016) to assign the architecture parameter credit through gradient-based methods. Our work tackles the inconsistency problem from a different perspective through the grouped backward procedure and the matching of the optimization complexity. Empirical results (see Table 1 and 3) suggest that the proposed methods are superior to the strategy proposed in SNAS.

3 EXPERIMENTS

3.1 NAS BENCHMARKS

In this section, we validate the effectiveness of our proposed method for the image classification task.

We conduct our first set of experiments on CIFAR-10 and CIFAR-100 (Krizhevsky et al. (2009)), each containing 50,000 training RGB images and 10,000 test images.

In the first stage of the search algorithm, we include the following operations in O: identity; 3×3 max pooling; 3×3 separable convolutions; 3×3 dilated separable convolutions and *zero*.

By calculating the gradient confusion for the two search stages and the final architecture, we empirically found that using 14 cells for the first stage and 20 cells for the second stage matches the optimization complexity of the final architecture. This probably because for the first stage, all the mixed operations contain non-identity operation, which can be seen as architecture with no pure skip connection. This results in higher gradient confusion, and shallower structure has to compensate it by using less stacking cells of 14. For the second stage, proper amount of skip connection has been activated, so using 20 cells would match the final architecture.

Then this proxy parent network by stacking 14 cells is trained using one-level optimization for 100 epochs. After convergence, the optimal operation group is activated based on the learned α.

In the second stage, we replace the mixed operation $\widehat{o}^{(i,j)}$ with a weighted sum of all operations in the activated group from stage 1. Then a proxy parent network by stacking 20 cells is trained using one-level optimization for 100 epochs.

To investigate the transferability of the cell [1] searched on CIFAR-10 and CIFAR-100, we further evaluate it on ImageNet (Russakovsky et al. (2015)) (mobile setting) (Figure 3).

Besides those declared[2], all other parameters and settings are identical to the ones used in Liu et al. (2018).

[1] This cell is provided in the supplementary material.

[2] All of our experiments for CIFAR10 and CIFAR100 were performed using NVIDIA Tesla V100 GPUs, and NVIDIA Tesla 8*V100 GPUs for IMAGENET.

Table 1: CIFAR-10 Benchmark. ***We emphasis that to obtain an unbiased result, we reported the mean and standard deviation over 8 single runs (i.e. search once and evaluate once, repeats the procedure 8 times with different seeds). For many other NAS methods like DARTS, 4 searchs are conducted to pick the best one.** *base* represents for that the final architecture training procedure is exactly the same with DARTS for a fair comparison; *fancy* represents for that the final training adopts training tricks for comparison with other NAS methods. Training details of base and fancy are provided in the Appendix C

		Test Error (%)		Params	Search	Search
Architecture	Source	Best	Average	(M)	Cost	Method
AmoebaNet-B	Real et al. (2018)	2.55	N/A	2.8	3150	evolution
ENAS	Pham et al. (2018)	2.89	N/A	4.6	0.5	RL
DARTS (2nd order)	Liu et al. (2018)	N/A	2.76±0.09	3.3	4+1*	SGD
ProxylessNAS	Cai et al. (2018)	2.08	N/A	**5.7**	4	SGD
SNAS	Xie et al. (2018)	N/A	2.85 ±0.02	2.9	1.5	SGD
PDARTS	Chen et al. (2019)	2.5	N/A	3.4	0.3	SGD
Base	StacNAS(ours)	2.33	2.48±0.08	3.9	0.8	SGD
Fancy	StacNAS(ours)	**1.85**	2.02 ±0.06	3.9	0.8	SGD

Table 2: CIFAR-100 Benchmark

		Test Error (%)		Params	Search	Search
Architecture	Source	Best	Average	(M)	Cost	Method
DARTS (2nd order)	Liu et al. (2018)	N/A	17.54 ± 0.27	3.8	4+1	SGD
PDARTS	Chen et al. (2019)	15.92	N/A	3.6	0.3	SGD
Base	StacNAS (ours)	15.90	16.11±0.2	4.3	0.8	SGD
Fancy	ours	**12.9**	14.3± 0.18	4	0.8	SGD

Table 3: ImageNet Benchmark (mobile setting)

		Test Error (%)		Params	Search	Search
Architecture	Source	top-1	top-5	(M)	Cost	Method
AmoebaNet-A	Real et al. (2018)	25.5	8	5.1	3150	evoluation
AmoebaNet-B	Real et al. (2018)	26	8.5	5.3	3150	evoluation
AmoebaNet-C	Real et al. (2018)	24.3	7.6	**6.4**	3150	evoluation
DARTS	Liu et al. (2018)	26.7	8.7	4.7	4	SGD
ProxylessNAS	Cai et al. (2018)	24.9	N/A	N/A	8.3	SGD
SNAS	Xie et al. (2018)	27.3	9.2	4.3	1.5	SGD
PDARTS	Chen et al. (2019)	24.4	7.4	4.9	0.3	SGD
EfficientNet-B0	Tan & Le (2019)	23.7	6.8	5.3	N/A	RL
C10 (base)	StacNAS (ours)	24.4	7.3	5.3	1	SGD
C10 (fancy)	StacNAS(ours)	**23.48**	6.4	5.3	1	SGD
C100 (base)	StacNAS (ours)	24.34	7.1	4.9	1	SGD
IMAGENET(base)	StacNAS (ours)	24.31	6.4	5.7	20	SGD

3.2 PREDICTED PERFORMANCE CORRELATION (CIFAR-10)

In differentiable NAS, the learned architecture parameter α is supposed to represent the relative importance of one candidate operation verse the others. To check the correlation between the accuracy of a stand-alone architecture with different candidate operations and the corresponding α, we replace the selected operation in the first edge of the first cell for the final architecture with all the other candidate operations in the first stage of StacNAS, and fully train them until converge. The obtained stand-alone accuracy is compared with the corresponding α learned for each candidate operation. Their correlation is plotted in Figure 2.

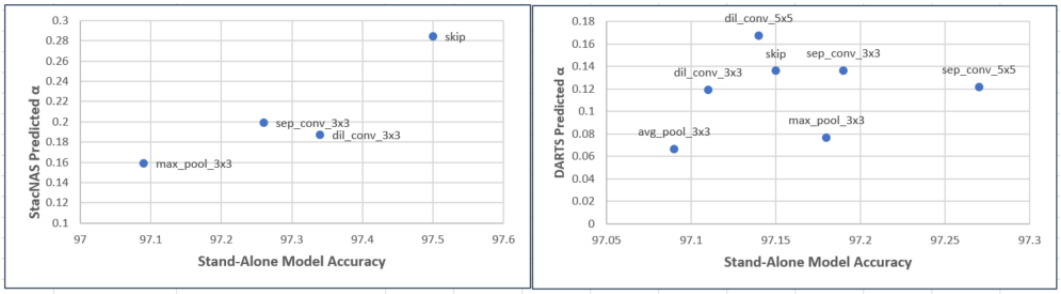

Figure 2: Correlation between standa-lone model and learned α: (a) Correlation coefficient of StacNAS: 0.91; (b) Correlation coefficient of DARTS: 0.2.

3.3 Ablation Experiments

Overall algorithm In Table 4 we reported the ablation studies. Results show that bi-level optimization with operation grouping and complexity matching can improve the original darts, but underperforms the proposed StacNAS.

Table 4: Ablation Experiments

	CIFAR10		CIFAR100	
	two-level	one-level	two-level	one-level
baseline	2.97±0.32	2.74±0.12	19.8 ± 1.33	16.93±0.89
grouped backward (GB)	2.82±0.26	2.68±0.10	18.25±1.19	16.68±0.65
GB+complexity match	2.73±0.23	2.48±0.08	18.13±0.61	16.11±0.2

Stacking layers The validation error on CIFAR10 of 8 repeated experiments for 8cells+8cells is 2.74 ± 0.12, for 14cells+17cells is 2.58 ± 0.07, for 17cells+20cells is 2.53 ± 0.05, which are all inferior to the result of 14cells+20cells.

Selection of representative operations We also experiment on CIFAR10 with randomly selected operations from each group for stage 1. The results for CIFAR10 are 2.53 ± 0.08, which means the selection of the operations for the first stage is not sensitive.

4 Conclusion

This paper introduced a stable and consistent optimization solution to differentiable NAS, which we call StacNAS. Our method addresses some difficulties encountered in the original DARTS algorithm such as bi-level optimization, multicollinearity of correlated operations, and the fundamental challenges of matching neural network optimization complexity in NAS and in the final training. It was shown that our method leads to the state-of-the-art image classification results on multiple benchmark datasets.

References

G Anandalingam and Terry L Friesz. Hierarchical optimization: An introduction. *Annals of Operations Research*, 34(1):1–11, 1992.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *arXiv preprint arXiv:1904.12760*, 2019.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6, 2009.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

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.

Karthik A Sankararaman, Soham De, Zheng Xu, W Ronny Huang, and Tom Goldstein. The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. *arXiv preprint arXiv:1904.06963*, 2019.

Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

	avg_3x3	max_3x3	sep_3x3	sep_5x5	dil_3x3	dil_5x5
avg_pool_3x3	**1.00**	**0.52**	-0.03	-0.03	0.00	-0.04
max_pool_3x3	**0.52**	**1.00**	0.00	0.02	-0.13	-0.08
sep_conv_3x3	-0.03	0.00	**1.00**	**0.38**	0.15	0.08
sep_conv_5x5	-0.03	0.02	**0.38**	**1.00**	0.11	0.06
dil_conv_3x3	0.00	-0.13	0.15	0.11	**1.00**	**0.27**
dil_conv_5x5	-0.04	-0.08	0.08	0.06	**0.27**	**1.00**

Table 5: Operator Correlation Matrix

A Appendix

B Operator Correlation Estimation

We Compute the correlations among different operators using CIFAR-10. This is for the purpose of grouping similar operators in our procedure. The correlation measures are obtained as follows.

1. A network of 14 cells is trained 100 epochs using parameters in the original DARTS, where the cell DAG consists 4 intermediate nodes and 14 learnable edges, and each edge is a weighted summation of all the 8 operators in the space.
2. With the trained model, for a given training image, we may pass it through the network, and store the output of the six operators (exclude "none" and "skip_connect") on all edges. On each edge, flatten the six output tensors into six vectors. Repeat until more than 10000 images in the training set have been processed.
3. Use the resulting data to calculate the correlations among the operators.

The resulting correlation matrix for the operators are presented in Table 5. The table clearly shows three distinct pairs of highly correlated operators, which form the three groups (plus a separate group for the skip_connect operator) in our method.

C Train Details

CIFAR-10 and CIFAR-100 The final architecture is a stack of 20 cells: 18 normal cells and 2 reduction cells, posioned at the 1/3 and 2/3 of the network respectively. For the *base* training, we trained the network exactly the same with DARTS. For the *fancy* training, we add AutoAugment (CIFAR10) (Cubuk et al. (2018)) policy and trained the network for 1200 epochs, other than these, all the other settings are same as DARTS.

ImageNet The final architecture is a stack of 14 cells: 12 normal cells and 2 reduction cells, posioned at the 1/3 and 2/3 of the network respectively. For the *base* training, we trained the network exactly the same with DARTS. For the *fancy* training, we add AutoAugment (IMAGENET) (Cubuk et al. (2018)) policy and trained the network for 600 epochs, other than these, all the other settings are same as DARTS.

D Esimation of Gradient Confusion

We compared gradient confusion for several different settings, including different numbers of cells, different stages, with/without an auxiliary head. The result are provided in the following table.

For each setting, we first train the architecture for 100 epochs and then ran 10 more iterations with randomly selected mini-batch using the trained parameters and compute the gradient confusion using formula (7) and obtain ζ_i for each iteration i. Then, we compute the mean of the 10 ζ_i we got and divided them by the corresponding model parameter size M. Formally,

$$\bar{\zeta} = \sum_{i=11}^{10} \zeta_i / M \tag{8}$$

We treat it as the measurement of the optimization complexity for the corresponding setting.

	mean	param(M)	confusion
augment100	2.15	3.8	0.56
stage220cell	1.2	2.1	0.57
stage1_14cell	0.83	1.4	0.59
stage1_17cell	1.56	1.7	0.91
stage1_20cell	2.06	2.14	0.96

Table 6: Gradient Confusion Estimation

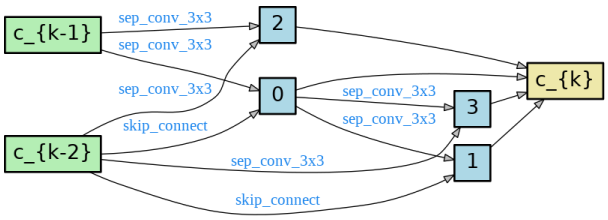

Figure 3: Normal cell learned on CIFAR-10

E CELLS OF BENCHMARK DATASET

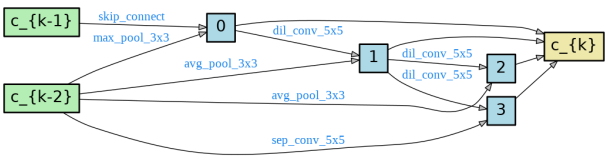

Figure 4: Reduction cell learned on CIFAR-10

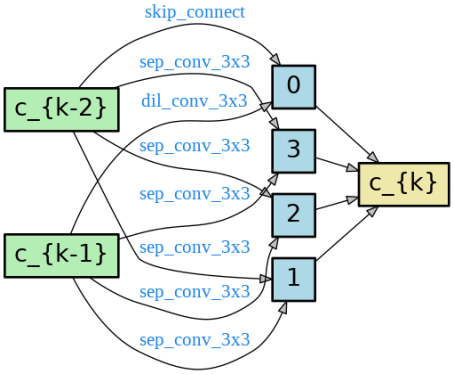

Figure 5: Normal cell learned on CIFAR-100

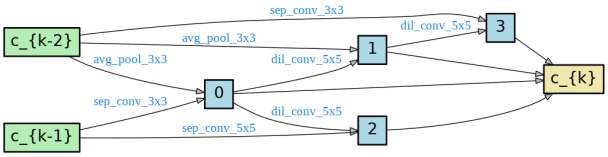

Figure 6: Reduction cell learned on CIFAR-100

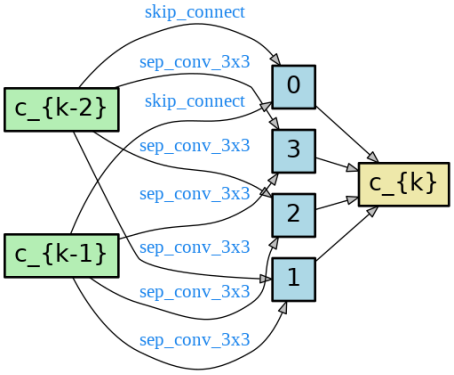

Figure 7: Normal cell learned on IMAGENET(mobile setting)

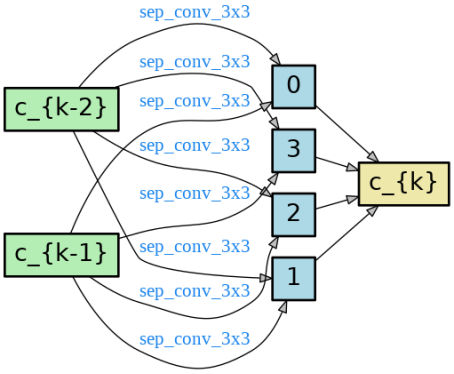

Figure 8: Reduction cell learned on IMAGENET(mobile setting)