DSNAS: Direct Neural Architecture Search without Parameter Retraining

Shoukang Hu*[1], Sirui Xie*[2], Hehui Zheng[3], Chunxiao Liu[4], Jianping Shi[4], Xunying Liu[1], Dahua Lin[1] †

Abstract

If NAS methods are solutions, what is the problem? Most existing NAS methods require two-stage parameter optimization. However, performance of the same architecture in the two stages correlates poorly. In this work, we propose a new problem definition for NAS, task-specific end-to-end, based on this observation. We argue that given a computer vision task for which a NAS method is expected, this definition can reduce the vaguely-defined NAS evaluation to i) accuracy of this task and ii) the total computation consumed to finally obtain a model with satisfying accuracy. Seeing that most existing methods do not solve this problem directly, we propose DSNAS, an efficient differentiable NAS framework that simultaneously optimizes architecture and parameters with a low-biased Monte Carlo estimate. Child networks derived from DSNAS can be deployed directly without parameter retraining. Comparing with two-stage methods, DSNAS successfully discovers networks with comparable accuracy (74.4%) on ImageNet in 420 GPU hours, reducing the total time by more than 34%.

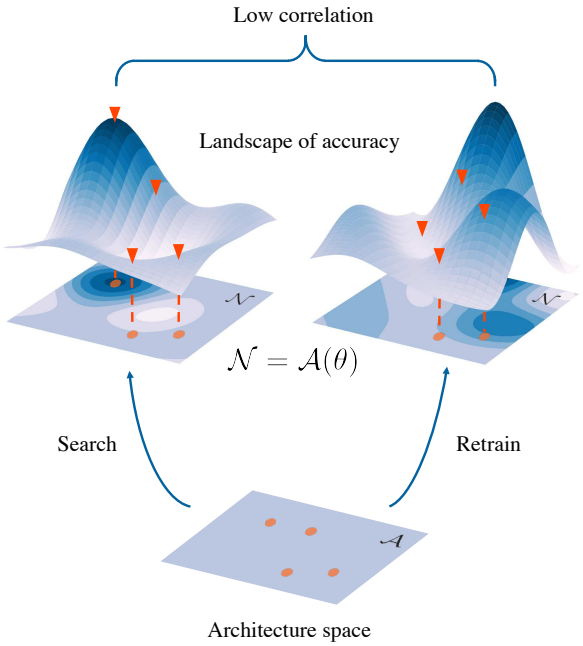

Figure 1. Projecting from the architecture space $\mathcal{A}$ to the network space $\mathcal{N}(\theta)$ with different parameter training schemes in *searching* and *retraining* results in accuracy with low correlation.

1. Introduction

The success of deep learning is partially built upon the architecture of neural networks. However, the variation of network architectures always incurs unpredictable changes in performance, causing tremendous efforts in *ad hoc* architecture design. Neural Architecture Search (NAS) is believed to be promising in alleviating this pain. Practitioners from the industry would like to see NAS techniques that automatically discover task-specific networks with reasonable performance, regardless of their generalization capability. Therefore, NAS is always formulated as a hyper-parameter optimization problem, whose algorithmic realization spans evolution algorithm [21, 7], reinforcement learning [28], Bayesian optimization [9], Monte Carlo Tree Search [25], and differentiable architecture search [14, 27, 3]. Recently, these algorithmic frameworks have exhibited pragmatic success in various challenging tasks, e.g. semantic segmentation [12] and object detection [4] *etc*.

However, even as an optimization problem, NAS is almost vaguely defined. Most of the NAS methods proposed recently are implicitly two-stage methods. These two stages are *searching* and *evaluation* (or *retraining*). While the architecture optimization process is referring to the *searching* stage, in which a co-optimization scheme is designed for parameters and architectures, there runs another round of parameter optimization in the *evaluation* stage, on the same set of training data for the same task. This is to some extent contradicting the norm in a machine learning task that no optimization is allowed in *evaluation*. A seemingly sensible argument could be that the *optimization* result of NAS is only the architecture, and the *evaluation* of an architecture is to check its performance after retraining. There is certainly no doubt that architectures that achieve high performance when retrained from scratch are reasonable choices

*Equal contribution. Work done at SenseTime Research. Correspondence skhu@se.cuhk.edu.hk, srxie@ucla.edu

†1. The Chinese Univsersity of Hong Kong; 2. University of California, Los Angeles; 3. Cambridge University; 4. SenseTime Research.

for deployment. But is this search method still valid if the searched architecture does not perform well after retraining, due to the inevitable difference of training setup in *searching* and *evaluation*?

These questions can only be answered with an assumption that the final *searching* performance can be generalized to *evaluation* stage even though the training schemes in two stages are different. Specifically, differences in training schemes may include different number of cells, different batch sizes, and different epoch numbers, *etc*. Using parameter sharing with efficiency concerns during search is also a usual cause. Unfortunately, this assumption is not a valid one. The correlation between the performance at the end of *searching* and after the retraining in *evaluation* is fairly low, as long as the parameter-sharing technique is used [20, 5].

We are thus motivated to rethink the problem definition of neural architecture search. We want to argue that as an application-driven field, there can be a diverse set of problem definitions, but every one of them should not be vague. And in this work, we put our cards on the table: we aim to tackle the *task-specific end-to-end* NAS problem. Given a task, defined by a data set and an objective (*e.g*. training loss), the expected NAS solution optimizes architecture and parameters to automatically discover a neural network with reasonable (if not optimal by principle) performance. By the term *end-to-end*, we highlight the solution only need a single-stage training to obtain a *ready-to-deploy* neural network of the given task. And the term *task-specific* highlights the boundary of this solution. The searched neural network can only handle this specific task. We are not confident whether this neural network generalizes well in other tasks. Rather, what can be expected to generalize is this NAS framework.

Under this definition, the evaluation metrics of a proposed framework become clear, namely searching efficiency and final performance. Scrutinizing most existing methods in these two metrics, we find a big niche for a brand new framework. On one side of the spectrum, gradient-based methods such as ENAS [17], DARTS [14], ProxylessNAS [3] require two-stage parameter optimization. This is because in the approximation to make them differentiable, unbounded bias or variance are introduced to their gradients. Two-stage methods always consume more computation than single-stage ones, not only because of another round of training but also the reproducibility issue [11]. On the other side of the spectrum, one-shot methods such as random search [11] and SPOS [7] can be extended to single-stage training. But since they do not optimize the architecture distribution in parameter training, the choice of prior distribution becomes crucial. A uniform sampling strategy may potentially subsume too many resources for satisfying accuracy. Lying in the middle, SNAS [27] shows a proof of concept, where the derived network maintains the performance in the *searching* stage. However, the gumbel-softmax relaxation makes it necessary to store the whole parent network in memory in both forward and backward, inducing tremendous memory and computation waste.

In this work, we confront the challenge of single-stage simultaneous optimization on architecture and parameters. Our proposal is an efficient differentiable NAS framework, Discrete Stochastic Neural Architecture Search (DSNAS). Once the search process finishes, the best-performing subnetwork is derived with optimized parameters, and no further retraining is needed. DSNAS is built upon a novel search gradient, combining the stability and robustness of differentiable NAS and the memory efficiency of discrete-sampling NAS. This search gradient is shown to be equivalent to SNAS's gradient at the discrete limit, optimizing the *task-specific end-to-end* objective with little bias. And it can be calculated in the same round of back-propagation as gradients to neural parameters. Its forward pass and back-propagation only involve the compact subnetwork, whose computational complexity can be shown to be much more friendly than DARTS, SNAS and even ProxylessNAS, enabling large-scale direct search. We instantiate this framework in a single-path setting. The experimental results show that DSNAS discovers networks with comparable performance (74.4%) in ImageNet classification task in only $\mathbf{420}$ GPU hours, reducing the total time of obtaining a *ready-to-deploy* solution by 34% from two-stage NAS.

To summarize, our main contributions are as follows:

- We propose a well-defined neural architecture search problem, *task-specific end-to-end* NAS, under the evaluation metrics of which most existing NAS methods still have room for improvement.
- We propose a *plug-and-play* NAS framework, DSNAS, as an efficient solution to this problem in large scale. DSNAS updates architecture parameters with a novel search gradient, combining the advantages of policy gradient and SNAS gradient. A simple but smart implementation is also introduced.
- We instantiate it in a single-path parent network. The empirical study shows DSNAS robustly discovers neural networks with state-of-the-art performance in ImageNet, reducing the computational resources by a big margin over two-stage NAS methods. We have made our implementation public[1].

2. Problem definition of NAS

2.1. Two-Stage NAS

Most existing NAS methods involve optimization in both *searching* stage and *evaluation* stage. In the *searching*

[1]https://github.com/SNAS-Series/SNAS-Series/

stage, there must be parameter training and architecture optimization, even though they may not run simultaneously. The ideal way is to train all possible architectures from scratch and then select the optimal one. However, it is infeasible with the combinatorial complexity of architecture. Therefore, designing the co-occurrence of parameter and architecture optimization to improve efficiency is the main challenge of any general NAS problems. This challenge has not been overcome elegantly yet. The accuracy at the end of the *searching* stage has barely been reported to be satisfying. And an *ad hoc* solution is to perform another round of parameter optimization in the *evaluation* stage.

Optimizing parameters in *evaluation* stage is not normal in traditional machine learning. Normally, the data set provided is divided into training set and validation set. Ones do learning in the *training* stage, with data from the training set. Then the learned model is tested on the withheld validation set, where no further training is conducted. With the assumption that training data and validation data are from the same distribution, the learning problem is reduced to an optimization problem. Ones can hence be confident to expect models with high training accuracy, if the assumption is correct, have high evaluation accuracy.

Allowing parameter retraining in the *evaluation* stage makes NAS a vaguely defined machine learning problem. Terming problems as *Neural Architecture Search* give people an inclined interpretation that only the architecture is the learning result, instead of parameters. But if the searched architecture is the answer, what is the problem? Most NAS methods claim they are discovering *best-performing* architecture in the designated space efficiently [3, 7, 9], but what specifically does *best-performing* mean? Given that retraining is conducted in evaluation stage, ones may naturally presume it is a meta-learning-like hyperparameter problem. Then the optimization result should exhibit some meta-level advantages, such as faster convergence, better optimum or higher transferability, etc. These are objectives that ones are supposed to state clearly in a NAS proposal. Nonetheless, objectives are only implicitly conveyed (mostly better optimum) in experiments.

Defining problem precisely is one of the milestones in scientific research, whose direct gift in a machine learning task is a clear objective and evaluation metric. Subsequent efforts can then be devoted into validating if the proposed learning loss is approximating a *necessary and sufficient* equivalence of this objective. Unfortunately, under this criterion, most existing two-stage NAS methods are reported [20, 11] failing to prove the correlation between the *searching* accuracy and the *retraining* accuracy.

2.2. Task-specific end-to-end NAS

Seeing that the aforementioned dilemma lies in the ambiguity in evaluating an architecture alone, we propose a type of problem termed *task-specific end-to-end NAS*, the solution to which should provide a *ready-to-deploy* network with optimized architecture and parameters.

Task refers to generally any machine learning tasks (in this work we discuss computer vision tasks specifically). A well-defined task should at least have a set of data representing its functioning domain, a learning objective for the task-specific motives *e.g.* classification, segmentation, etc. And the task is overwritten if there is a modification in either factor, even a trivial augmentation in the data. In other words, *task-specific* sets a boundary on what we can expect from the searched result and what cannot. This can bring tremendous operational benefits to industrial applications.

End-to-end highlights that, given a task, the expected solution can provide a *ready-to-deploy* network with satisfying accuracy, the whole process of which can be regarded as a black-box module. Theoretically, it requires a direct confrontation of the main challenge of any general NAS problem, *i.e.* co-optimizing parameter and architecture efficiently. Empirically, *task-specific end-to-end* is the best description of NAS's industrial application scenarios: i) the NAS method itself should be generalizable for any off-the-shelf tasks; and ii) when applied to a specific task, practitioners can at least have some conventional guarantees on the results. Basically, it is to reduce vaguely defined NAS problems to established tasks.

The evaluation metrics become clear under this problem definition. The performance of the final result is, by principle, the accuracy in this task. And the efficiency should be calculated based on the time from this NAS solver starts taking data to it outputs the neural network whose architecture and parameters are optimized. This efficiency metric is different from all existing works. For two-stage methods, the time for both *searching* and *evaluation* should be taken into account in this metric. Therefore, their efficiency may not be as what they claim. Moreover, two-stage methods do not optimize the objective *higher accuracy of final derived networks* in an end-to-end manner.

3. Direct NAS without retraining

3.1. Stochastic Neural Architecture Search (SNAS)

In the literature, SNAS is one of those close to a solution to the *task-specific end-to-end NAS* problem. Given any task with differentiable loss, the SNAS framework directly optimizes the expected performance over architectures in terms of this task. In this subsection, we provide a brief introduction on SNAS.

Basically, SNAS is a differentiable NAS framework that maintains the generative nature as reinforcement-learning-based methods [28]. Exploiting the deterministic nature of the Markov Decision Process (MDP) of network construction process, SNAS reformulated it as a Markov Chain. This

reformulation leads to a novel representation of the network construction process. As shown in Fig.2, nodes x_i (blue lumps) in the DAG represent feature maps. Edges (i, j) (arrow lines) represent information flows between nodes x_i and x_j, on which n possible operations $\boldsymbol{O}_{i,j}$ (orange lumps) are attached. Different from DARTS, which avoids sampling subnetwork with an attention mechanism, SNAS instantiates this Directed Acyclic Graph (DAG) with a stochastic computational graph. Forwarding a SNAS parent network is to first sample random variables $\boldsymbol{Z}_{i,j}$ and multiplying it to edges (i, j) in the DAG:

$$\tilde{\boldsymbol{O}}_{i,j}(\cdot) = \boldsymbol{Z}_{i,j}^T \boldsymbol{O}_{i,j}(\cdot). \tag{1}$$

Ones can thus obtain a Monte Carlo estimate of the expectation of task objective $L_{\boldsymbol{\theta}}(\boldsymbol{Z})$ over possible architectures:

$$\mathbb{E}_{\boldsymbol{Z}\sim p_{\boldsymbol{\alpha}}(\boldsymbol{Z})}[L_{\boldsymbol{\theta}}(\boldsymbol{Z})], \tag{2}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ are parameters of architecture distribution and neural operations respectively. This is exactly the *task-specific end-to-end NAS* objective.

To optimize parameters $\boldsymbol{\theta}$ and architecture $\boldsymbol{\alpha}$ simultaneously with Eq. 2, (termed as *single-level optimization* in [14]), SNAS relaxes the discrete one-hot random variable $\boldsymbol{Z}$ to a continuous random variable $\tilde{\boldsymbol{Z}}$ with the gumbel-softmax trick. However, the continuous relaxation requires to store the whole parent network in GPU, preventing it from directly applying to large-scale networks. In Xie et al. [27], SNAS is still a two-stage method.

If the temperature in SNAS's gumble-softmax trick can be directly pushed to zero, SNAS can be extended to large-scale networks trivially. However, it is not the case. Take a look at the search gradient given in Xie et al. [27]:

$$\frac{\partial \mathcal{L}}{\partial \alpha_{i,j}^k} = \frac{\partial \mathcal{L}}{\partial x_j} \boldsymbol{O}_{i,j}^T(x_i)(\boldsymbol{\delta}(k'-k) - \tilde{\boldsymbol{Z}}_{i,j}) Z_{i,j}^k \frac{1}{\lambda \alpha_{i,j}^k}, \tag{3}$$

ones can see that the temperature λ is not valid to be zero for the search gradient. Xie et al. [27] only gradually annealed it to be close to zero. In this work, we seek for an alternative way to differentiate Eq. 2, combining the efficiency of discrete sampling and the robustness of continuous differentiation. And we start from SNAS's credit assignment.

3.2. Discrete SNAS (DSNAS)

In original SNAS [27], to prove its efficiency over ENAS, a policy gradient equivalent of the search gradient is provided

$$\begin{aligned} &\mathbb{E}_{\tilde{\boldsymbol{Z}}\sim p(\tilde{\boldsymbol{Z}})}[\frac{\partial \mathcal{L}}{\partial \alpha_{i,j}^k}] \\ &= \mathbb{E}_{\tilde{\boldsymbol{Z}}\sim p(\tilde{\boldsymbol{Z}})}[\nabla_{\alpha_{i,j}^k} \log p(\tilde{\boldsymbol{Z}})[\frac{\partial \mathcal{L}}{\partial x_j} \boldsymbol{O}_{i,j}^T(x_i) \tilde{\boldsymbol{Z}}_{i,j}]_c], \end{aligned} \tag{4}$$

where $\tilde{\boldsymbol{Z}}_{i,j}$ is the gumbel-softmax random variable, $[\cdot]_c$ denotes that $\cdot$ is a *cost* independent from $\boldsymbol{\alpha}$ for gradient calculation. In other words, Eq. 4 and Eq. 3 both optimize the *task-specific end-to-end NAS* objective *i.e.* Eq. 2.

In order to get rid of SNAS's continuous relaxation, we push the λ in the PG equivalent (4) to the limit 0, with the insight that only reparameterization trick needs continuous relaxation but policy gradient doesn't. The expected *search gradient* for architecture parameters at each edge becomes:

$$\begin{aligned} &\lim_{\lambda\to 0} \mathbb{E}_{\tilde{\boldsymbol{Z}}\sim p(\tilde{\boldsymbol{Z}})}[\frac{\partial \mathcal{L}}{\partial \alpha_{i,j}^k}] \\ &= \lim_{\lambda\to 0} \mathbb{E}_{\tilde{\boldsymbol{Z}}\sim p(\tilde{\boldsymbol{Z}})}[\nabla_{\alpha_{i,j}^k} \log p(\tilde{\boldsymbol{Z}}_{i,j})[\frac{\partial \mathcal{L}}{\partial x_j} \tilde{\boldsymbol{O}}_{i,j}(x_i)]_c] \\ &= \mathbb{E}_{\boldsymbol{Z}\sim p(\boldsymbol{Z})}[\nabla_{\alpha_{i,j}^k} \log p(\boldsymbol{Z}_{i,j})[\frac{\partial L}{\partial x_j} \boldsymbol{O}_{i,j}^T(x_i) \boldsymbol{Z}_{i,j}]_c] \\ &= \mathbb{E}_{\boldsymbol{Z}\sim p(\boldsymbol{Z})}[\nabla_{\alpha_{i,j}^k} \log p(\boldsymbol{Z}_{i,j})[\frac{\partial L}{\partial x_j} \sum_k O_{i,j}^k(x_i) Z_{i,j}^k]_c], \end{aligned} \tag{5}$$

where $\boldsymbol{Z}_{i,j}$ is a strictly one-hot random variable, $Z_{i,j}^k$ is the kth element in it, $[\cdot]_c$ denotes that $\cdot$ is a *cost* independent from $\boldsymbol{\alpha}$ for gradient calculation. Line 3 is derived from line 2 since $p(\boldsymbol{Z}_{i,j}) = \lim_{\lambda\to 0} p(\tilde{\boldsymbol{Z}}_{i,j})$ [16], $L = \lim_{\lambda\to 0} \mathcal{L}$.

Exploiting the one-hot nature of $\boldsymbol{Z}_{i,j}$, *i.e.* only $Z_{i,j}^s$ on edge (i, j) is 1, others *i.e.* $Z_{i,j}^{\not> s}$ are 0, the *cost* function can be further reduced to

$$\begin{aligned} C(\boldsymbol{Z}_{i,j}) &= \sum_k \frac{\partial L}{\partial x_j} O_{i,j}^k(x_i) Z_{i,j}^k \\ &= \frac{\partial L}{\partial x_j^i} O_{i,j}^s(x_i) Z_{i,j}^s = \frac{\partial L}{\partial x_j^i} x_j^i \\ &= \frac{\partial L}{\partial x_j^i} \frac{\partial x_j^i}{\partial Z_{i,j}^s} = \frac{\partial L}{\partial Z_{i,j}^s}, \end{aligned} \tag{6}$$

as long as $|\frac{\partial \mathcal{L}}{\partial x_j} O_{i,j}^{\not> s}(x_i)| \neq \infty$. Here $x_j^i = O_{i,j}^s(x_i) Z_{i,j}^s$ is the output of the operation $O_{i,j}^s$ chosen at edge (i, j). The equality in line 3 is due to $Z_{i,j}^s = 1$.

3.3. Implementation

The algorithmic fruit of the mathematical manipulation in Eq. 6 is a parallel-friendly implementation of Discrete SNAS, as illustrated in Fig. 2. In SNAS, the network construction process is a pass of forward of stochastic computational graph. The whole network has to be instantiated with the *batch* dimension. In DSNAS we offer an alternative implementation. Note that $C(\boldsymbol{Z}_{i,j}) = \frac{\partial L}{\partial Z_{i,j}^s}$ only needs to be calculated for the sampled subnetworks. And apparently it is also the case for $\frac{\partial L}{\partial \boldsymbol{\theta}}$. That is to say, the back-propagation of DSNAS only involves the sampled network, instead of the whole parent network. Thus we only instantiate the

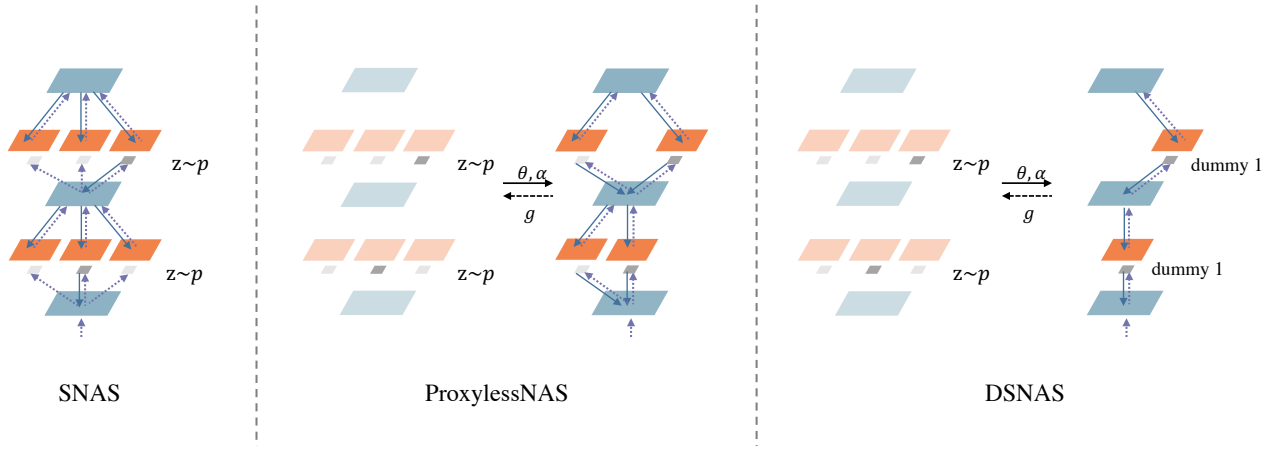

Figure 2. Forward and backward on SNAS, ProxylessNAS and DSNAS. Blue lumps stand for feature maps, orange ones for operation candidates. Blue arrow lines indicate forward data flows, purple dashed lines indicate backward ones. Semi-transparent lumps stand for parent networks that are not instantiated with *batch* dimension, a technique to reduce computation in ProxylessNAS and DSNAS. $dummy$ 1 highlights the smart implementation introduced in Sec. 3.3.

Algorithm 1 Discrete SNAS

Require: parent network, operation parameters $\boldsymbol{\theta}$ and categorical arch distribution $p_{\boldsymbol{\alpha}}(\boldsymbol{Z})$
Initialize $\boldsymbol{\theta}$, $\boldsymbol{\alpha}$
while not converged **do**
Sample one-hot random variables $\boldsymbol{Z}$ from $p_{\boldsymbol{\alpha}}(\boldsymbol{Z})$
Construct child network with $\boldsymbol{\theta}$ according to $\boldsymbol{Z}$, multiply a 1^{dummy} after each feature map X
Get a batch from data and forward to get L
Backward L to both $\boldsymbol{\theta}$ and 1^{dummy}, backward $\log p_{\boldsymbol{\alpha}}(\boldsymbol{Z})$ to $\boldsymbol{\alpha}$
Update $\boldsymbol{\theta}$ with $\frac{\partial L}{\partial \boldsymbol{\theta}}$, update $\boldsymbol{\alpha}$ with $\frac{\partial \log p_{\boldsymbol{\alpha}}(\boldsymbol{Z})}{\partial \boldsymbol{\alpha}} \frac{\partial L}{\partial 1^{dummy}}$
end while

subnetwork with the *batch* dimension for forward and backward. However, the subnetwork derived in this way does not necessarily contain $\boldsymbol{Z}_{i,j}$. If it was not with Line 3 of Eq. 6, we would have to calculate $C(\boldsymbol{Z}_{i,j})$ with $\frac{\partial L}{\partial x_j^i} x_j^i$. Then the policy gradient loss would explicitly depend on the intermediate result $x_j^i = O_{i,j}^s(x_i)$, which may need an extra round of forward if it is not stored by the automated differentiation infrastructure. With a smart mathematical manipulation in Eq. 6, ones can simply multiply a $dummy$ 1 to the output of each selected operation, and calculate $C(\boldsymbol{Z}_{i,j})$ with $\frac{\partial L}{\partial 1_{i,j}^{dummy}}$. The whole algorithm is shown in Alg. 1

3.4. Complexity analysis

In this subsection, we provide a complexity analysis of DSNAS, SNAS, and ProxylessNAS. Without loss of generality, we define a parent network with l layers and each layer has n candidate choice blocks. Let the forward time on a sampled subnetwork be P, its backward time be Q, and the memory requirement for this round be M.

As the original SNAS instantiates the whole graph with *batch* dimension, it needs n times GPU memory and n times calculation comparing to a subnetwork. It is the same case in DARTS.

Method	Forward time	Backward time	Memory
Subnetwork	$O(P)$	$O(Q)$	$O(M)$
SNAS	$O(nP)$	$O(nQ)$	$O(nM)$
ProxylessNAS*	$O(nP)$	$O(nQ)$	$O(nM)$
ProxylessNAS	$O(2P)$	$O(2Q)$	$O(2M)$
DSNAS	$O(P)$	$O(Q)$	$O(M)$

Table 1. Computation complexity comparison between SNAS, ProxylessNAS and DSNAS. ProxylessNAS* indicates its theoretical complexity.

This memory consumption problem of differentiable NAS was first raised by [3]. And they proposed an approximation to DARTS's optimization objective, with the BinaryConnect [6] technique:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}_{i,j}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{Z}}_{i,j}} \frac{\partial \hat{\boldsymbol{Z}}_{i,j}}{\partial \boldsymbol{\alpha}_{i,j}} \approx \sum_k \frac{\partial \mathcal{L}}{\partial Z_{i,j}^k} \frac{\partial \hat{Z}_{i,j}^k}{\partial \boldsymbol{\alpha}_{i,j}}, \tag{7}$$

where $\hat{\boldsymbol{Z}}_{i,j}$ denotes the attention-based estimator as in DARTS [14], distinct from the discrete random variable $\boldsymbol{Z}_{i,j}$, highlighting how the approximation is being done. But this approximation does not directly save the memory and computation. Different from Eq. 5 and Eq. 6, theoretically, the calculation of Eq. 7 still involves the whole network, as indicated by the summation $\sum$. To reduce memory consumption, they further empirically proposed a path sampling heuristic to decrease the number of paths from n to 2. Table 1 shows the comparison.

3.5. Progressive early stop

One potential problem in sample-based differentiable NAS is that empirically, the entropy of architecture distribution does not converge to zero, even though comparing to attention-based NAS [14] they are reported [27] to converge with smaller entropy. The non-zero entropy keeps the sampling going on until the end, regardless of the fact that sampling at that uncertainty level does not bring significant gains. To the opposite, it may even hinder the learning on other edges.

To avoid this side-effect of architecture sampling, DSNAS applies a progressive early stop strategy. Sampling and optimization stop at layers/edges in a progressive manner. Specifically, a threshold h is set for the stopping condition:

$$\min\{\alpha_{i,j}^{k} - \alpha_{i,j}^{m}, \forall m \neq k || \alpha_{i,j}^{k} = \max\{\boldsymbol{\alpha}_{i,j}\}\} \geq h. \quad (8)$$

Once this condition is satisfied on any edge/layer, we directly select the operation choice with the highest probability there, stop its sampling and architecture parameters update in the following training.

3.6. Comparison with one-shot NAS

Different from all differentiable NAS methods, one-shot NAS only do architecture optimization in one-shot, before which they obtain a rough estimation of the graph through either pretraining [1, 7] or an auxiliary hypernetwork [2]. All of them are two-stage methods. The advantage of DSNAS is that it optimizes architecture alongside with parameters, which is expected to save some resources in the pretraining stage. Intuitively, DSNAS rules out non-promising architectures in an adaptive manner by directly optimizing the objective in an end-to-end manner. Although one-shot methods can also have an end-to-end realization, by investing more resources in pretraining, it may take them more epochs to achieve comparable performance as DSNAS. They can also do finetuning, but still parameters of the optimal networks are updated less frequently than DSNAS. Ones can expect better performance from DSNAS given equivalent training epochs.

4. Experimental Results

In this section, we first demonstrate why the proposed *task-specific end-to-end* is an open problem for NAS, by investigating the performance correlation between *searching* stage and *evaluation* stage of the two-stage NAS. We then validate the effectiveness and efficiency of DSNAS under the proposed *task-specific end-to-end* metric on the same search space as SPOS [7]. We further provide a breakup of time consumption to illustrate the computational efficiency of DSNAS.

4.1. Accuracy correlation of two-stage NAS

Since the validity of the *searching* in two-stage NAS relies on a high correlation in the performance of *searching* stage and *evaluation* stage, we check this assumption with a ranking correlation measure, Kendall Tau metric τ [10].

$$\tau = \frac{2(N_{concordant} - N_{disconcordant})}{N(N-1)}, \quad (9)$$

where N is the total number of pairs (x_i, y_i) from the *searching* stage and *evaluation* stage consisting of $N_{concordant}$ concordant ranking pairs ($x_1 > x_2, y_1 > y_2$ or $x_1 < x_2, y_1 < y_2$) and $N_{disconcordant}$ disconcordant ranking pairs ($x_1 > x_2, y_1 < y_2$ or $x_1 < x_2, y_1 > y_2$). Kendall Tau metric ranges from -1 to 1, which means the ranking order changes from reversed to identical. τ being close to 0 indicates the absence of correlation.

We measure the ranking correlation by calculating Kendall Tau metric from two perspectives: (1) The τ_{inter} is calculated based on the top-k model performance of the *searching* and *evaluation* stage in one single searching process; (2) The Kendal Tau metric τ_{intra} is calculated by running the two-stage NAS methods several times with different random seeds using the top-1 model performance in each searching process. As shown in Table 2, the performance correlation between the *searching* stage and *evaluation* stage in both SPOS and ProxylessNAS is fairly low. This indicates the necessity of *task-specific end-to-end* NAS problem formulation. Fairly low correlation may also imply reproducibility problems.

Model	τ_{inter}	τ_{intra}
Single Path One-Shot[7]	0.33	-0.07
ProxylessNas [3]	-	-0.33

Table 2. Kendall Tau metric τ calculated with the top-k model performance in the searching and evaluation stage. τ_{inter} measures the correlation of top-k model performance of the *searching* and *evaluation* stage in one single searching process while τ_{intra} measures the correlation of top-1 model performance from different searching processes.

4.2. Single-path architecture search

Motivation To compare the efficiency and accuracy of derived networks from DSNAS versus existing two-stage methods, we conduct experiment in single-path setting. Results are compared in the *task-specific end-to-end* metrics.

Dataset All our experiments are conducted in a mobile setting on the ImageNet Classification task [18] with a resource constraint $FLOPS \leq 330M$. This dataset consists of around 1.28×10^6 training images and 5×10^4 validation images. Data transformation is achieved by the standard pre-processing techniques described in the supplementary material.

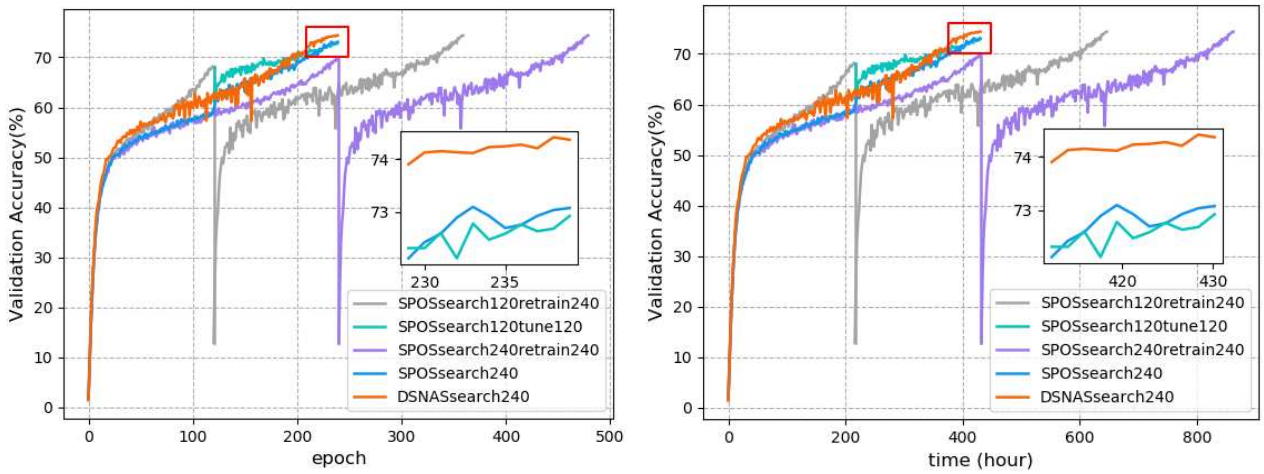

Figure 3. Searching process of two-stage SPOS and single-stage SPOS/DSNAS. `SPOSsearch120retrain240` and `SPOSsearch240retrain240` search for 120/240 epochs then retrain the derived architecture for 240 epochs. Instead of retraining, `SPOSsearch120tune120` finetunes the result for 120 epochs. `DSNASsearch240` and `SPOSsearch240` utilize one-stage training for 240 epochs. `DSNASsearch240` applies progressive early stop, `SPOSsearch240` applies one-shot EA at 120th epoch.

Search Space The basic building block design is inspired by ShuffleNet v2 [15]. There are 4 candidates for each choice block in the parent network, i.e., choice_3, choice_5, choice_7, and choice_x. These candidates differ in the kernel size and the number of depthwise convolutions, spanning a search space with 4^{20} single path models. The overall architecture of the parent network and building blocks are shown in the supplementary material.

Training Settings We follow the same setting as SPOS [7] except that we do not have an *evaluation stage* in our searching process. We adopt a SGD optimizer with a momentum of 0.9 [22] to update the parent network weight parameters. A cosine learning rate scheduler with an initial learning rate of 0.5 is applied. Moreover, an L2 weight decay ($4 \times 10e^{-5}$) is used to regularize the training process. The architecture parameters are updated using the Adam optimizer with an initial learning rate of 0.001. All our experiments are done on 8 NVIDIA TITAN X GPUs.

Searching Process To demonstrate the efficiency of DSNAS, we compare the whole process needed to accomplish *task-specific end-to-end* NAS in ImageNet classification with two-stage NAS methods. Among all existing two-stage methods, SPOS [7] is the one with state-of-the-art accuracy and efficiency. SPOS can be regarded as a weight-sharing version of random search, where the search is conducted with one-shot evolution algorithm after training the uniformly sampled parent network.

Figure 3 shows DSNAS's advantage over several different configurations of SPOS. We purposefully present curves in terms of both epoch number and time to illustrate that even though DSNAS updates architecture in an iteration-basis, almost no extra computation time is introduced. Among the four configurations of SPOS, `SPOSsearch120retrain240` is the original one as in Guo et al. [7], using the two-stage paradigm. Obviously, DSNAS achieves comparable accuracy in an end-to-end manner, with roughly 34% less computational resources. As `SPOSsearch120retrain240` updates block parameters for only 120 epochs[2], we run the `SPOSsearch120tune120` and `SPOSsearch240retrain240` configurations for fair comparison. At the end of the 240th epoch, the accuracy of SPOS models is around 1.4% and 4% lower than DSNAS's respectively.

In addition, for the ablation study of DSNAS's progressive early stop strategy, we call the EA algorithm of SPOS at the 120th epoch in the one-stage `DSNASsearch240` configuration. Continuing the parameter training, the selected models experience a leap in accuracy and converge with accuracy 1.3% lower than DSNAS's. However, seeking this EA point is fairly *ad hoc* and prone to random noise.

Searching Results The experimental results are reported in Table 3. Comparing with all existing two-stage NAS methods, DSNAS shows comparable performance using at least 1/3 less computational resources. More importantly, the standard deviation in DSNAS's accuracy is lower than those from both *searching* and *evaluation* stage from EA-based SPOS (0.22 vs 0.38/0.36). This exhibits as a differentiable NAS framework, DSNAS is a more robust method in the *task-specific end-to-end* metric.

4.3. Time consumption breakup

In last subsection, we show DSNAS can achieve comparable performance under the *task-specific end-to-end* met-

[2]Same learning rate scheduler is used in DSNAS and SPOS.

Model	FLOPS	Search		Retrain		No proxy	Time (GPU hour)	
		Top-1 acc(%)	Top-5 acc(%)	Top-1 acc(%)	Top-5 acc(%)		Search	Retrain
MobileNet V1 (0.75x)[8]	325M	Manual		68.4	-		Manual	
MobileNet V2 (1.0x)[19]	300M	Manual		72.0	91.00		Manual	
ShuffleNet V2 (1.5x)[15]	299M	Manual		72.6	90.60		Manual	
NASNET-A(4@1056)[29]	564M	-	-	74.0	91.60	False	48000	-
PNASNET[13]	588M	-	-	74.2	91.90	False	5400	-
MnasNet-A1[23]	312M	-	-	75.2	92.50	False	$40000^{\S}$	-
DARTS[14]	574M	-	-	73.3	91.30	False	24	288
SNAS[27]	522M	-	-	72.7	90.80	False	36	288
Proxyless-R (mobile)[3]	320M	62.6*	84.7*	74.6	92.20	True	$300^{\ddagger}$	$\geq$384
Single Path One-Shot[7]	319M	$68.7^{\dagger}$	-	74.3	-	True	250	384
Single Path One-Shot*	323M	68.2±0.38	88.28	74.3±0.36	91.79	True	250	384
Random Search	$\leq$330M	$\leq$68.2	$\leq$88.31	$\leq$73.9	$\leq$91.8	True	250	384
DSNAS	324M	**74.4±0.22**	91.54	74.3±0.27	91.90	True	420	

Table 3. Results of choice block search. The time is measured based on NVIDIA TITAN X and accuracy is calculated on the validation set. * is our implementation with the original paper setting. $40000^{\S}$ is the GPU hour converted from 6912 TPUv2 hours with a ratio of roughly 1:6. $300^{\ddagger}$ is the GPU hour converted from V100 GPU with a ratio of 1:1.5. $68.7^{\dagger}$ is the accuracy on the search set.

ric with much less computation than one-shot NAS methods. In this subsection, we further break up the time consumption of DSNAS into several specific parts, i.e. forward, backward, optimization and test[3], and conduct a controlled comparison with other differentiable NAS methods. We also hope such a detailed breakup can help readers gain insight into further optimizing our implementation.

We first compare the computation time of SNAS and DSNAS on CIFAR-10 dataset. The average time of each splited part[4] is shown in Table 4. Under the same setting, our DSANS is almost **five** folds faster than SNAS and consumes only $\mathbf{1/n}$ of GPU memory as SNAS (n is the total number of candidate operations in each edge).

Method	Train			Test
	Forward	Backward	Opt	
SNAS	0.26s	0.46s	0.14s	0.18s
DSNAS	0.05s	0.07s	0.13s	0.04s

Table 4. Computation time of SNAS and DSNAS

We further compare the average time of each splited part between DSNAS and ProxylessNAS in a mobile setting on the ImageNet Classification task. As shown in Table 5, the average time [5] is calculated on the same search space of ProxylessNAS [3] with a total batch size of 512. With a fair comparison, DSNAS is roughly **two** folds faster than ProxylessNAS.

Method	Train			Test
	Forward	Backward	Opt	
ProxylessNAS	3.3s	2.3s	3.6s	1.2s
DSNAS	1.9s	1.3s	2.6s	0.9s

Table 5. Computation time of ProxylessNAS and DSNAS

5. Summary and future work

In this work, we first define a *task-specific end-to-end NAS* problem, under the evaluation metrics of which we scrutinize the efficiency of two-stage NAS methods. We then propose an efficient differentiable NAS framework, DSNAS, which optimizes architecture and parameters in the same round of back-propagation. Subnetworks derived from DSNAS are *ready-to-deploy*. One competitive counterpart would be EfficientNet[24], which tries to bridge two stages with extra grid search on network scale after NAS. However, its total cost is larger than DSNAS. More accuracy gain can be achieved in DSNAS if scales are searched similarly. As a framework, DSNAS is orthogonal to the random wiring solution, which focuses on graph topology search [26]. We look forward to their combination for a joint search of topology, operations, and parameters.

Acknowledgement

This work is mainly done at SenseTime Research Hong Kong. SH and XL are also partially supported by by Hong Kong Research Grants Council General Research Fund No. 14200218 and Shun Hing Institute of Advanced Engineering Project No. MMT-p1-19.

[3] To clarify, we also do evaluation on testing set, retraining parameters is what we do not do.

[4] The average time of each splited part in one batch is calculated on one NVIDIA TITAN X GPU with the same setting (batchsize 64) as in [27].

[5] As shown in Table 1 that Proxyless NAS takes 2 times GPU memory as DSNAS, we use 8 TITAN X GPUs for ProxylessNAS and 4 for DSNAS to calculate the time.

References

[1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 549–558, 2018. 6

[2] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017. 6

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 1, 2, 3, 5, 6, 8

[4] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. Detnas: Neural architecture search on object detection. *arXiv preprint arXiv:1903.10979*, 2019. 1

[5] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019. 2

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015. 5

[7] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019. 1, 2, 3, 6, 7, 8

[8] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 8

[9] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2016–2025, 2018. 1, 3

[10] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. 6

[11] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019. 2, 3

[12] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Autodeeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019. 1

[13] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 8

[14] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1, 2, 4, 5, 6, 8

[15] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018. 7, 8

[16] 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. 4

[17] 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. 2

[18] 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. 6

[19] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 8

[20] Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019. 2, 3

[21] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002. 1

[22] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013. 7

[23] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 8

[24] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 8

[25] Martin Wistuba. Finding competitive network architectures within a day using uct. *arXiv preprint arXiv:1712.07420*, 2017. 1

[26] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*, 2019. 8

[27] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018. 1, 2, 4, 6, 8

[28] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1, 3

[29] 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*, pages 8697–8710, 2018. 8