One-Shot Neural Architecture Search: Maximising Diversity to Overcome Catastrophic Forgetting

Miao Zhang, Huiqi Li, *Senior Member, IEEE,* Shirui Pan, Xiaojun Chang, Chuan Zhou, Zongyuan Ge, and Steven Su, *Senior Member, IEEE*

Abstract—One-shot neural architecture search (NAS) has recently become mainstream in the NAS community because it significantly improves computational efficiency through weight sharing. However, the supernet training paradigm in one-shot NAS introduces catastrophic forgetting, where each step of the training can deteriorate the performance of other architectures that contain partially-shared weights with current architecture. To overcome this problem of catastrophic forgetting, we formulate supernet training for one-shot NAS as a constrained continual learning optimization problem such that learning the current architecture does not degrade the validation accuracy of previous architectures. The key to solving this constrained optimization problem is a novelty search based architecture selection (**NSAS**) loss function that regularizes the supernet training by using a greedy novelty search method to find the most representative subset. We applied the NSAS loss function to two one-shot NAS baselines and extensively tested them on both a common search space and a NAS benchmark dataset. We further derive three variants based on the NSAS loss function, the NSAS with depth constrain (**NSAS-C**) to improve the transferability, and **NSAS-G** and **NSAS-LG** to handle the situation with a limited number of constraints. The experiments on the common NAS search space demonstrate that NSAS and it variants improve the predictive ability of supernet training in one-shot NAS with remarkable and efficient performance on the CIFAR-10, CIFAR-100, and ImageNet datasets. The results with the NAS benchmark dataset also confirm the significant improvements these one-shot NAS baselines can make.

Index Terms—AutoML, neural architecture search, continual Learning, catastrophic forgetting, novelty search.

1 Introduction

Neural architecture search (NAS) has recently attracted massive interest from the deep learning community because experts do not have inordinate amounts of time and labor designing neural networks [13], [18], [29], [35], [42], [48], [51], [72]. Early NAS methods were based on a nested approach that trained numerous separate architectures from scratch and then used reinforcement learning (RL) or an evolutionary algorithm (EA) to find the most promising architectures, based on validation accuracy [19], [47], [73]. However, these methods are so computationally-expensive as to be impractical for most machine learning practitioners. For example, it would take more than 1800 GPU days through RL to find promising architectures for the problem outlined in [73], and Real et al. [47] spent 7 days with 450 GPUs searching for promising architectures with an EA. Recent studies have shown that NAS can significantly improve computational efficiency [3], [6], [64]. Weight sharing, in particular, also called one-shot NAS [4], [46], [67], has attracted enormous attention for automating neural architecture design. This is because it not only finds state-of-the-art architectures but also significantly reduces the search hours needed. One-shot NAS encodes the search space as a supernet, where all possible architectures directly inherit weights from the supernet for evaluation without needing to be trained from scratch. Since one-shot NAS only trains the supernet for architecture searches, this learning paradigm might reduce the time a search takes from many days down to several hours.

Pioneer studies on one-shot NAS follow two sequential steps [4], [14], [30], [46]. They first adopt an architecture sampling controller to sample architectures for training the supernet. Then, a heuristic search method finds promising architectures over a discrete search space based on the trained supernet [20], [30], [46], [59]. Later studies [7], [16], [38], [39], [56], [60], [65] have further employed continuous relaxation to differentiate between architectures so that the gradient descent can be used to optimize the architecture with respect to validation accuracy. The architecture parameters and supernet weights are alternatively optimized through a bilevel optimization method, and the most promising architecture is obtained once the supernet is trained.

Since one-shot NAS evaluates candidate architectures based on the validation accuracy of the weights it inherits from the supernet as opposed to training them from scratch, the success of one-shot NAS relies on a critical assumption that the validation accuracy should approximate the test accuracy after training from scratch or be highly predictive. The authors of the first study on one-shot NAS [4] observed a strong positive correlation between the validation accuracy and the test accuracy when the supernet was trained through random path dropout. Subsequent studies

- *M. Zhang and H. Li are with the School of Information and Electronics, Beijing Institute of Technology, Beijing, China, 100081. E-mail: huiqili@bit.edu.cn*
- *S. Pan and M. Zhang are with the Faculty of Information Technology, Monash University, Australia. E-mail: shirui.pan@monash.edu*
- *X. Chang is with the Faculty of Information Technology, Monash University Australia, and a Distinguished Adjunct Professor with the Faculty of Computing and Information Technology, King Abdulaziz University.*
- *C. Zhou is with Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China.*
- *Z. Ge is with Monash e-Research Centre, Monash University, Australia.*
- *S. Su and M. Zhang are with the Faculty of Engineering and Information Technology, University of Technology Sydney, NSW, Australia.*
- *Corresponding authors: Huiqi Li and Shirui Pan.*

Manuscript received March 15, 2020; revised July 22, 2020; accepted October 26, 2020

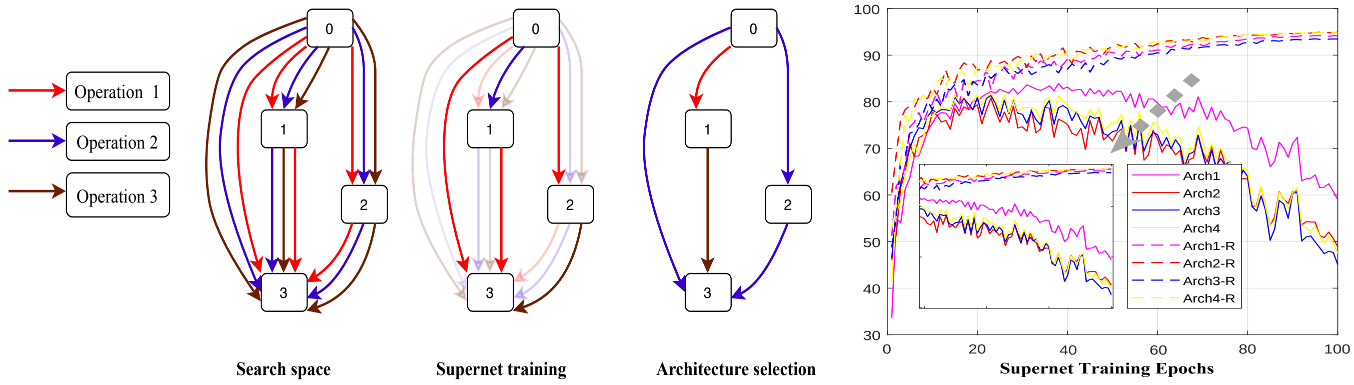


Fig. 1: **Left**: The general process of one-shot NAS. First, the search space is defined as a supernet containing all candidate architectures. Then a single path of the supernet (an architecture) is trained in each step of the supernet training process. Promising architectures are selected based on the validation accuracy of weights inherited from the trained supernet without the need for training from scratch. **Right**: The validation accuracy for four different architectures during the supernet training. The solid lines ("Arch") are the accuracies returned using weights inherited from the supernet; the dashed lines ("Arch-R") are the accuracies after retraining.

all rightly considered this assumption to be true for all one-shot NAS methods. However, several recent studies have revealed that this assumption may not hold in most popular one-shot NAS approaches. For instance, Sciuto et al. [62] show that there is no observable correlation between the validation and test accuracy of the weight-sharing paradigm with ENAS [46], and Adam et al. [1] show that the RNN controller in ENAS does not depend on past sampled architectures, which means its performance is the same as a random search. Similarly, Singh et al. [52] find that there is no visible progress in terms of the retrained performance for found architectures based on supernet during the architecture search phase, implying the supernet training is useless for improving the predictive ability of one-shot NAS. Further, Yang et al. [58] conducted extensive experiments that demonstrated that the current one-shot NAS techniques struggle to outperform naive baselines. Rather, the success of one-shot NAS is mostly due to the design of the search space.

Most one-shot NAS approaches [7], [14], [20], [30] adopt a single-path training method for their supernet training, where only a single path (one architecture) in the supernet is trained in each step. This is the scenario we consider. However, Benyahia et al. [5] observed that when training multiple models (architectures) with partially-shared weights for a single task, the training of each model may lower the performance of other models. Benyahia et al. [5] defined this phenomenon as multi-model forgetting, also known as catastrophic forgetting. They also observed this catastrophic forgetting in one-shot NAS. For example, consider a large supernet containing multiple models with shared weights across them. Sequentially training each model on a single task could mean that the accuracy of each model tends to drop when training another model containing partially-shared weights [5], [62]. This multi-model forgetting in one-shot NAS is illustrated in Fig.1 in terms of the validation accuracy of four different architectures during supernet training. What is clear from the figure is that inheriting weights makes performance deteriorate even further during supernet training.

So, although weight sharing can greatly reduce computation hours, it can also introduce catastrophic forgetting into the supernet training, which results in unreliable architecture rankings. Addressing multi-model forgetting during supernet training is an urgent issue if we are to better leverage one-shot NAS and improve the predictive ability of supernets. Hence, we have formulated supernet training as a constrained optimization problem for continual learning to avoid degrading the performance of previous architectures when training a new one.

That said, it is intractable to consider all previously visited architectures. Therefore, only the most representative subset of previous architectures is used to regularize learning of the current architecture. We have devised an efficient greedy novelty search method based on maximizing diversity to select the constraints. We have also implemented the approach in two one-shot baselines. The experimental results demonstrate that our strategy is able to relieve multi-model forgetting in one-shot NAS methods. A summary of our main contributions follows.

- We first formulate supernet training with one-shot NAS as a constrained optimization problem of continual learning, where learning the current architecture should not degrade the performance of previous architectures with partially-shared weights.
- We have also designed an efficient greedy novelty search method based on maximizing diversity to select a subset of the constraints that best approximate the feasible region formed by all previous architectures.
- With these two techniques, we then incorporate this **NSAS** loss function (novelty search-based architecture selection) into the RandomNAS [30] and GDAS [16] one-shot NAS baselines to form RandomNAS-NSAS and GDAS-NSAS, with the goal of reducing the level of multi-model forgetting during supernet training. Our best-found models from the common search space [38] returned a competitive test error of 2.59% on CIFAR-10 and only took 0.7 GPU days of search time.
- To improve transferability, we further devised a variant of NSAS, called **NSAS-C**, which searches for "deeper" architectures in the convolutional cell search. Experiments on the common search space demonstrate increased transferability of the found models, and competitive test errors of 16.69% on CIFAR-100 and 25.5% on ImageNet.
- A series of experiments conducted with the NAS benchmark dataset NAS-Bench-201 [17] also verify that **NSAS** significantly reduce forgetting and improve the perfor-

mance of one-shot NAS baselines.

This paper outlines some significant extensions to our recent conference paper [66]. These include: **(1)** improvements to the transferability of the found models through a variant of novel search strategy called **NSAS-C**. **NSAS-C** has a depth constraint for convolutional architecture searches. Experiments with ImageNet demonstrate its efficacy; **(2)** a new NAS search space, NSA-Bench-201, along with comparative experiments against relevant baselines; **(3)** two new variants of NSAS, **NSAS-G** and **NSAS-LG**, to handle situations with a limited number of constraints. Comparison experiments with the NAS-Bench-201 dataset are provided to showcase these extensions; and **(4)** an impact analysis of the hyperparameters settings and constraint selection strategy in Section 5.2.2 and Section 5.2.3.

2 Background

2.1 Neural Architecture Search

The goal of NAS is to automatically design deep neural networks without human intervention. In general, the architecture of a deep neural network α is usually represented as a directed acyclic graph (DAG), which is also a subgraph of the whole search space $\alpha \in \mathcal{A}$. A deep neural network could be defined as $\mathcal{U}(\alpha, w_\alpha)$, where w_α are the weights associated with architecture α. With NAS, one tries to find the architecture with the best validation performance according to:

$$\alpha^* = \underset{\alpha \in \mathcal{A}}{\texttt{argmin}}\, \mathcal{L}_{\texttt{val}}(\mathcal{U}(\alpha, w_\alpha^*)), \tag{1}$$

where w_α^* is derived by training architecture α on the training set while minimizing the training loss function $\mathcal{L}_{train}$:

$$w_\alpha^* = \underset{w}{\texttt{argmin}}\, \mathcal{L}_{\texttt{train}}(\mathcal{U}(\alpha, w_\alpha)). \tag{2}$$

Early studies on NAS usually used a nested approach to finding promising architectures by training numerous architectures from scratch and leveraging EA [47] or RL [72] to reveal the promising ones. However, from a practical standpoint, it is computationally inefficient and often unaffordable to evaluate numerous architectures in this way. Therefore, more recently, researchers have shifted their attention to reducing computation costs with strategies such as performance prediction [3], [55], weights generation [6], [64], weight sharing [38], [46], and so on [73].

2.2 One-Shot Neural Architecture Search

One-shot NAS encodes a search space $\mathcal{A}$ as a supernet $\mathcal{W}_A$ that consumes all possible candidates. Only the supernet is trained, while all candidate architectures α directly inherit weights from the supernet without needing to be trained from scratch. Search times are therefore greatly reduced because only one neural network needs to be trained during the architecture search phase. The most promising architecture α^* is based on validation performance with weights inherited from the supernet:

$$\begin{aligned} \underset{\alpha \in \mathcal{A}}{\texttt{min}} \quad & \mathcal{L}_{\texttt{val}}(\mathcal{W}_{\mathcal{A}}^*(\alpha)) \\ \texttt{s.t.} \quad & \mathcal{W}_{\mathcal{A}}^*(\alpha) = \texttt{argmin}\, \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}(\alpha)). \end{aligned} \tag{3}$$

Eq. (3) is more than a challenging bilevel optimization problem: the discrete characteristic of the architecture space makes it impossible to use a gradient-based method to solve the formula directly. For this reason, ENAS [46] uses an LSTM controller to sample the architectures. Whereas, [20] and [30] train the supernet based on a uniform sampling strategy and the best-performing architecture from the trained supernet is found through a random search or evolutionary method.

Several state-of-the-art one-shot methods use continuous relaxation to transform discrete architectures into a continuous space $\mathcal{A}_\theta$ with a **softmax** function to further improve efficiency [16], [38], [44], [56]. The supernet weights and architecture parameters can be jointly optimized through:

$$(\alpha_\theta^*, \mathcal{W}_{\mathcal{A}_\theta}(\alpha_\theta^*)) = \underset{\alpha_\theta, \mathcal{W}}{\texttt{argmin}}\, \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}_\theta}(\alpha_\theta^*)), \tag{4}$$

making it possible to continually optimize the architecture search. The best architecture α^* is determined through argmax based on the continuous architecture representation α_θ^*.

Since Eq.(4) is supposed to train the entire supernet in each step, it has a much higher memory requirement than ENAS. Hence, ProxylessNAS [7] transforms the real-valued architecture parameters into binary representations through binary gates, and only a single path is activated during the supernet training. In this way, the memory requirement for ProxylessNAS is the same as training a single architecture. GDAS [16] introduces a gradient-based sampler to sample the single path for each training step. Additionally, the distribution of architectures and the supernet weights can be jointly optimized, which means the memory requirement also equates to only training a single architecture. Yao et al. [60] developed a constrained optimization method to force each step of the architecture optimization process in the continuous space to arrive at a binary result, thus reducing the memory requirement of supernet training. Unlike continuous relaxation, NAO [39] uses an LSTM-based autoencoder to transform discrete neural architectures into continuous representations. A differentiable method is then used to search for architectures in the continuous space.

2.3 Multi-model Forgetting in One-Shot NAS

Catastrophic Forgetting is a common problem in artificial general intelligence and multi-task learning. It describes the phenomenon of where a model forgets what it has learned about a previous task(s) after being trained on a new task [9], [22], [26], [28], [34], [45]. Formally, a model with optimal parameters θ_A^* for dataset $\mathcal{D}_A$ will perform substantially less well on $\mathcal{D}_A$ after it has trained on another dataset $\mathcal{D}_B$. Methods to resolve such issues are defined as *continual learning*. Some examples include learning without forgetting (LwF) [33], which adds a response from the old task as a regularization term to prevent catastrophic forgetting, and elastic weight consolidation (EWC) [26], which maximizes the likelihood of a conditional probability $p(\theta \mid \mathcal{D})$, where $\mathcal{D}$ containing two independent data sets $\mathcal{D}_A$ and $\mathcal{D}_B$, and $\mathcal{D}_A$ is not available when trained on $\mathcal{D}_B$.

Multi-model Forgetting occurs when training multiple models with a single dataset. Unlike training a model on several tasks sequentially, one-shot NAS applies different models to a single dataset $\mathcal{D}$, e.g., $\theta_a = (\theta_a^p, \theta^s)$ and $\theta_b = (\theta_b^p, \theta^s)$, to a single dataset $\mathcal{D}$, where θ^s is the shared weight and θ_a^p and θ_b^p are private weights. Several recent studies [31], [54], [62] have shown that the interactions between networks can degrade the performance of a whole network, and that catastrophic forgetting with one-shot NAS can lower the performance of previous architectures after training a new architecture in the supernet. To alleviate this problem, Benyahia et al. [5] proposed a weight plasticity loss (WPL), which maximizes the posterior probability $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathcal{D})$ as:

$$\begin{aligned} p(\theta \mid \mathcal{D}) &= \frac{p(\theta_a^p, \theta_b^p, \theta^s, \mathcal{D})}{p(\mathcal{D})} = \frac{p(\theta_a^p \mid \theta_b^p, \theta^s, \mathcal{D}) p(\theta_b^p, \theta^s, \mathcal{D})}{p(\mathcal{D})} \\ &= \frac{p(\theta_a^p, \theta^s \mid \mathcal{D}) p(\mathcal{D} \mid \theta_b^p, \theta^s) p(\theta_b^p, \theta^s)}{p(\theta^s, \mathcal{D})} \\ &= \frac{p(\theta_a \mid \mathcal{D}) p(\mathcal{D} \mid \theta_b) p(\theta_b)}{p(\theta^s, \mathcal{D})}. \end{aligned} \tag{5}$$

However, it is intractable to calculate $(\theta^s, \mathcal{D})$ in Eq.(5), so Benyahia et al. [5] made several presuppositions to make the calculation feasible: a) that θ_a^p and θ_s are independent, and b) that the weights θ_a for previous model are in optimal points. This way, $p(\theta^s, \mathcal{D})$ can be estimated by the distance of θ^s to the optimal θ_s^* with the diagonal of the Fisher information defining the importance of each parameter. In WPL, the loss function to maximize the likelihood of $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathcal{D})$ is calculated as:

$$\mathcal{L}_{WPL}(\theta_b) = \mathcal{L}_c(\theta_b) + \frac{\eta}{2}(\|\theta_b^p\|^2 + \|\theta^s\|^2) + \sum_{\theta_{s_i} \in \theta_s} \frac{\varepsilon}{2} F_{\theta_{s_i}} (\theta_{s_i} - \theta_{s_i}^*), \tag{6}$$

where $\mathcal{L}_c$ is the cross-entropy loss function, and $F_{\theta_{s_i}}$ is the diagonal element of the Fisher information matrix corresponding to parameter θ_{s_i}. $F_{\theta_{s_i}}$ is estimated by presupposing parameters (θ_a^p, θ_b^p) are independent, and that θ_s^* are the shared parameters θ^s after the previous model has been trained, which are assumed to be in the optimal points. A detailed derivation of Eq.(6) can be found in [5].

Limitations weight plasticity loss (WPL) only considers one previous architecture in each step of supernet training. This method is also based on the assumption that the shared weights are optimal. However, these two assumptions are hard to hold when training a supernet in a one-shot NAS scheme given numerous architectures shared weights with the current architecture. Plus, the shared weights are usually far away from the optimal points. To address these concerns, we formulated supernet training with one-shot NAS as a constrained optimization problem, where learning the current architecture does not degrade the performance of previously-visited architectures. We consider a subset of previous architectures as constraints to regularize the learning of the current architecture. We also demonstrate that the loss function of the posterior probability $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathcal{D})$ can be calculated without assuming that the shared weights are optimal when maximizing the diversity of the selected architectures.

3 METHODOLOGY

3.1 Problem Formulation

Unlike jointly optimizing the posterior probability under the assumption that θ_a is near-optimal as per WPL [5] or keeping the shared weights fixed as per Learn to Grow [32], we formulate supernet training as a constrained optimization problem. Specifically, we enforce the architectures with inherited weights in the current step so as to perform better than the last step. Without loss of generality, we consider a typical scenario where only one architecture in the supernet is trained in each step, and the constrained optimization problem is defined as:

$$\begin{aligned} &\mathcal{W}_{\mathcal{A}}^t = \underset{\theta \in \mathcal{W}_{\mathcal{A}}(\alpha^t)}{\texttt{argmin}} \quad \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}(\alpha^t)), \\ &\texttt{s.t.} \quad \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}^t(\alpha^i)) \le \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}^{t-1}(\alpha^i));\ \forall i \in \{0 ... t-1\}. \end{aligned} \tag{7}$$

Algorithm 1 Greedy Novelty Search

Input: constraints archive $\mathcal{M}$, recent architectures archive $\mathcal{C}$, selected architecture α^m, n.

```
1: $N(\alpha^m, \mathcal{M}) \leftarrow$ calculate the novelty score of $\alpha^m$ in $\mathcal{M}$ based on Eq.(9);
2: for $i = 1, 2, ..., n$ do
3:     randomly sample an architecture $\alpha^r$ from $\mathcal{C}$;
4:     if $N(\alpha^r, \mathcal{M}) > N(\alpha^m, \mathcal{M})$ then
5:         replace $\alpha^m$ with $\alpha^r$;
6:     end if
7: end for
```

Here, $\mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}(\alpha)) = \mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha)) + \lambda \mathcal{R}(\mathcal{W}_{\mathcal{A}}(\alpha))$, and $\mathcal{W}_{\mathcal{A}}$ represents the total of all weights in the supernet. α^t is the current architecture in step t, and $\mathcal{W}_{\mathcal{A}}(\alpha^t)$ is the weights of α^t inherited from the supernet, and only $\mathcal{W}_{\mathcal{A}}(\alpha^t)$ is optimized in each step t.

3.2 Constraints Selection based on Novelty Search

The constraints in Eq.(7) prevent the learning the current architecture from degrading the performance of previous architectures as a strategy to overcome multi-model forgetting in one-shot NAS. However, the number of constraints in Eq.(7) increases linearly with the step, which makes it intractable to consider all constraints in the optimization. Therefore, it is more practical to try and select a subset of M constraints from the previous architectures that forms as close a feasible region to the original feasible region as possible. Intuitively, maximizing the diversity of the subset is an efficient way to find the most representative samples from the previous architectures. Based on this observation and motivated by [2], we propose a surrogate for constraint selection:

$$\begin{aligned} &\texttt{maximize}_{\mathcal{M}} \sum_{\alpha^i, \alpha^j \in \mathcal{M}} dis(\alpha^i, \alpha^j), \\ &\texttt{s.t.} \quad \mathcal{M} \subset \{\alpha^1 ... \alpha^{t-1}\}; |\mathcal{M}| = M, \end{aligned} \tag{8}$$

where $dis(\alpha^i, \alpha^j)$ is a function to calculate the distance between architectures. Further, to solve this equation, we proposed a greedy novelty search method to maximize the diversity of the subset. Before the archive is full, all the new coming architectures are added into the subset. Once full, the most similar one to the current architecture is chosen to replace the one that maximizes the novelty score of the archive. Algorithm 1 sets out a simple implementation of our greedy novelty search method. A simple and standard novelty measurement, defined as $N(\alpha, \mathcal{M})$, measures the mean distance of its k-nearest neighbors in $\mathcal{M}$:

$$\begin{aligned} N(\alpha, \mathcal{M}) &= \frac{1}{|S|} \sum_{\alpha^j \in S} dis(\alpha, \alpha^j) \\ S &= kNN(\alpha, \mathcal{M}) = \{\alpha^1, \alpha^2, ..., \alpha^k\}. \end{aligned} \tag{9}$$

In this paper, we measure the difference of the input edges for each node in an architecture. The input edge of the same node for two architectures is considered to be the same only when the two edges have the same input node and the same operations. M constraint architectures are then selected from $|\mathcal{C}|$ recent architectures rather than all previous architectures.

3.3 The NSAS Loss Function

After finding the M most representative architectures $\{\alpha_1, ..., \alpha_M\}$ by maximizing diversity, we need to forcibly optimize the learning of the current architecture in the feasible region

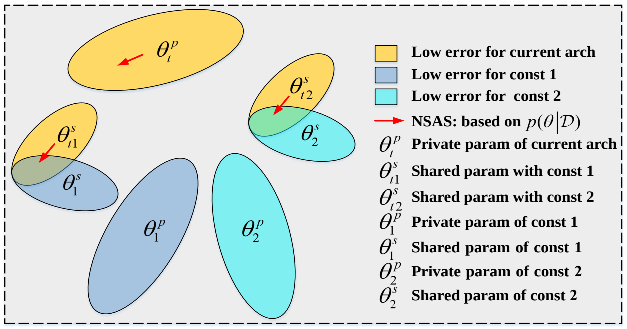

θ_t^p
θ_{t1}^s
θ_1^s
θ_1^p
θ_2^p
θ_{t2}^s
θ_2^s
Low error for current arch
Low error for const 1
Low error for const 2
NSAS: based on $p(\theta|\mathcal{D})$
θ_t^p Private param of current arch
θ_{t1}^s Shared param with const 1
θ_{t2}^s Shared param with const 2
θ_1^p Private param of const 1
θ_1^s Shared param of const 1
θ_2^p Private param of const 2
θ_2^s Shared param of const 2

Fig. 2: **NSAS** loss function ensures that the learning of current architecture will not deteriorate the performance of previous architectures in the constraint subset.

formed by these constraints. A common approach is to convert the constraints into a soft regularization loss or apply a replay buffer [2]. The weights of these architectures in the subset are described as $\{\theta_1, ..., \theta_M\}$. When the selected constraints are converted to a soft regularization loss, the loss function for the constrained optimization problem in Eq. (7) could be described as Eq.(10):

$$\begin{aligned}\mathcal{L}_N(\mathcal{W}_\mathcal{A}(\alpha^t)) &= (1-\beta)(\mathcal{L}_c(\mathcal{W}_\mathcal{A}(\alpha^t)) + \lambda\mathcal{R}(\mathcal{W}_\mathcal{A}(\alpha^t))) \\ &+ \frac{\beta}{M}\sum_{i=1:M}(\mathcal{L}_c(\mathcal{W}_\mathcal{A}(\alpha^i)) + \lambda\mathcal{R}(\mathcal{W}_\mathcal{A}(\alpha^i))),\end{aligned} \tag{10}$$

where $\mathcal{L}_c$ is the cross-entropy loss function, $\mathcal{R}$ is the ℓ_2 regularization term, and β are the trade-offs. The term $\mathcal{L}_N(\mathcal{W}_\mathcal{A}(\alpha^t))$ is the **NSAS** loss function. While learning the current architecture α_t, **NSAS** protects the performance of those constraint architectures by ensuring the shared parameters stay in a region of low error for these constraints, as shown schematically in Fig. 2.

3.4 From Weight Plasticity Loss (WPL) to NSAS

WPL [5] regularizes the learning of current architecture by maximizing the posterior probability $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathcal{D})$, where $\theta_a = \{\theta_a^p, \theta^s\}$ is the weights of the last architecture, $\theta_b = \{\theta_b^p, \theta^s\}$ is the weights of the current architecture, and θ^s is their shared weights. Unlike WPL, which only considers one previous architecture, we consider a subset of previously visited architectures - $\theta_a = \{\theta_1, ..., \theta_M\} = \{(\theta_1^p, \theta_1^s), ..., (\theta_M^p, \theta_M^s)\}$ - where θ_i^p is the private weights, and θ_i^s is the weights shared with the current architecture. When selected constraints make the following two assumptions hold true, then **Lemma** 1 describes the relationship between WPL and NSAS.

Assumption 1: The architectures in the constraint subset cover all weights of the current architecture α_t that $\theta_b^{(e)} \subseteq \{\theta_1^{(e)} \cup ... \cup \theta_M^{(e)}\}$ for every edge e in α_t, where $\theta_m^{(e)}$ is the weight of the operations assigned to each edge of α_m.

Assumption 2: There are no shared weights among these constraint architectures, i.e, $\theta_1 \cap \theta_2 = ... = \theta_{M-1} \cap \theta_M = \emptyset$.

Lemma 1. When the selected constraint architectures satisfy the above two assumptions, the **NSAS** loss function with the WPL can be formulated as:

$$\mathcal{L}_N(\mathcal{W}_\mathcal{A}(\alpha^t)) = \mathcal{L}_c(\mathcal{W}_\mathcal{A}(\alpha^t)) + \gamma\mathcal{L}_{WPL}(\mathcal{W}_\mathcal{A}(\alpha^t)). \tag{11}$$

Proof Since the weights of current architecture θ_b are shared by the constraints described in **Assumption 1**, $\theta_b^{(e)} \subseteq \{\theta_1^{(e)} \cup ... \cup \theta_M^{(e)}\}$, and for every edge e in α_t, we have $\theta_b^p = \emptyset$. Further, θ_i and θ_j $(i, j = 1...M)$ are independent as the architectures are trained separately without shared weights, as described in **Assumption 2**. Now the posterior probability in the WPL can be written as:

$$\begin{aligned}p(\theta \mid \mathcal{D}) = p(\theta_1...\theta_M, \theta_b \mid \mathcal{D}) &= \frac{p(\theta_1^p...\theta_M^p, \theta_1^s...\theta_M^s, \mathcal{D})}{p(\mathcal{D})} \\ &= \frac{p(\theta_1...\theta_M, \mathcal{D})}{p(\mathcal{D})} = \frac{p(\theta_1 \mid \theta_2...\theta_M, \mathcal{D})p(\theta_2...\theta_M, \mathcal{D})}{p(\mathcal{D})} \\ &= \prod_{i=1:M} p(\theta_i \mid \mathcal{D}) \propto \prod_{i=1:M} p(\mathcal{D} \mid \theta_i)p(\theta_i) \\ &= p(\theta)\prod_{i=1:M} p(\mathcal{D} \mid \theta_i) = p(\theta^t)\prod_{i=1:M} p(\mathcal{D} \mid \theta_i),\end{aligned} \tag{12}$$

where θ_i is the weights of architecture α^i in the constraint subset. As only architecture α^t is trained, $p(\theta) = p(\theta^t)$, where θ^t is the weights of the current architecture α^t, and θ is all the considered weights. Eq.(12) derives the posterior probability without the assumption that θ^s in the previous step is optimal. Hence, now, the WPL to optimize the posterior probability $p(\theta \mid \mathcal{D})$ can be expressed as:

$$\mathcal{L}_{WPL}(\mathcal{W}_\mathcal{A}(\alpha^t)) = \epsilon\mathcal{R}(\mathcal{W}_\mathcal{A}(\alpha^t)) + \sum_{i=1:M}\mathcal{L}_c(\mathcal{W}_\mathcal{A}(\alpha^i)), \tag{13}$$

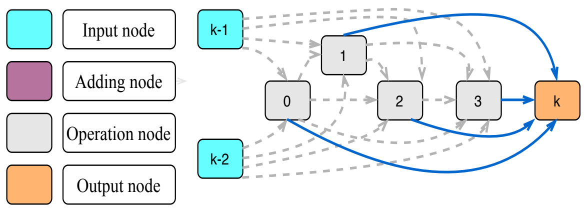
where ϵ is the trade-off factor. And the proposed **NSAS** loss function with the WPL can be expressed as:

$$\begin{aligned}\mathcal{L}_N(\mathcal{W}_\mathcal{A}(\alpha^t)) &= (1-\beta)(\mathcal{L}_c(\mathcal{W}_\mathcal{A}(\alpha^t)) + \lambda\mathcal{R}(\mathcal{W}_\mathcal{A}(\alpha^t))) \\ &+ \frac{\beta}{M}\sum_{i=1:M}(\mathcal{L}_c(\mathcal{W}_\mathcal{A}(\alpha^i)) + \lambda\mathcal{R}(\mathcal{W}_\mathcal{A}(\alpha^i))) \\ &= \mathcal{L}_c(\mathcal{W}_\mathcal{A}(\alpha^t)) + \gamma\mathcal{L}_{WPL}(\mathcal{W}_\mathcal{A}(\alpha^t)).\end{aligned} \tag{14}$$

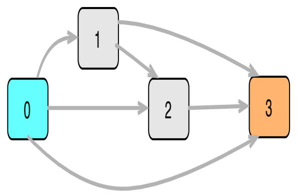
Lemma 1 demonstrates that the NSAS loss function not only contains the WPL but also optimizes learning of the current architecture when the appropriate constraints have been selected. Additionally, when a specific number of constraints for a densely-connected search space are set, the strategy of selecting constraints based on maximizing diversity has the potential to see the two assumptions hold true. Take the search space of NAS-Bench-201 [17] (as defined in Section 4.2) as an example. When the number of candidate operations in each edge $M = 5$ and the diversity of the constraint subset is maximized, those five constraint architectures should cover all possible operations for all edges (**Assumption 1**), and each edge in each constraint architecture should contain different operations (**Assumption 2**).

3.5 One-Shot NAS with Novelty Search based Architecture Selection

We implemented our loss function into two popular one-shot NAS: RandomNAS [30] and GDAS [16]. Like the most weight sharing NAS methods, only a single path is trained in each step of the architecture search phase. Therefore, incorporating NSAS into RandomNAS is relatively easy. However, most gradient-based NAS methods, like DARTS [38], train the whole supernet in each step of the supernet training, which would violate both the assumptions set out above. For this reason, we chose GDAS [16] as the gradient method, which uses the Gumbel-Max trick [24], [41], [56] to relax the discrete architecture distribution so as to be continuous and differentiable. The **argmax** function reparameterizes the architecture distribution and samples only one architecture in each supernet training step during the forward pass. The **softmax** function is applied during the backward pass for architecture learning. Algorithm 2 outlines the one-shot NAS with the **NSAS** loss function, called one-shot NAS-NSAS.



(a) Common Search Space for CNN



(b) Search Space in NAS-Bench-201

Fig. 3: The search spaces for the two different datasets. (a) The cell structure of a CNN with a common search space takes the two previous cells' output as the cell inputs. There are four operation nodes in each cell, and each operation node selects two outputs from the former nodes associated with those operations as inputs. The output of this cell is the sum of the outputs of all operation nodes. (b) The cell in NAS-Bench-201 is a densely-connected structure, where the operation nodes and output nodes select all former nodes7 with the applied operations as their inputs.

Algorithm 2 One-Shot NAS-NSAS

```
Input: $\mathcal{D}_{train}$, $\mathcal{D}_{val}$, $\mathcal{W}$, constraints archive $\mathcal{M} = \emptyset$, $M$, neural architecture search iteration $T$, batch size $b$
    for $t = 1, 2, ..., (T * \texttt{size}(\mathcal{D}_{train})/b)$ do
2:      if $\texttt{size}(\mathcal{M}) < M$ then
            sample $\alpha^t$ based on gradient search or random search, and update the weights $\mathcal{W}_A(\alpha)$ by normal loss function, and add architecture $\alpha$ into $\mathcal{M}$;
4:      else
            sample $\alpha^t$ based on gradient search or random search, select the architecture $\alpha^m$ that is most similar to $\alpha^t$ from $\mathcal{M}$, and replace $\alpha^m$ with $\alpha^t$ to maximize the diversity of $\mathcal{M}$ based on Algorithm 1. Update the weights $\mathcal{W}_A(\alpha)$ by our proposed loss function in Eq.(10) or a replay buffer;
6:      end if
    end for
8:  Obtain $\alpha^*$ based on Eq.(3) (RandomNAS-NASA) or Eq.(4) (GDAS-NASA).
```

4 Experimental settings

To evaluate the performance of **NSAS** loss function, we compared baseline versions of RandomNAS [30] and GDAS [16] with our **NSAS** implementations denoted as **RandomNAS-NSAS** and **GDAS-NSAS**. We considered two different search spaces: a common search space adopted by most state-of-the-art one-shot NAS methods [30], [38], and a second NAS-Bench-201 space [17]. The NAS-Bench-201 dataset was specifically designed for one-shot NAS research, so it comes with a guarantee of fair comparison between one-shot NAS methods. The NAS-Bench-201 search space is much smaller than the common search space, and, accordingly, the best test performances for all candidate architectures on all datasets were reported with this search space, relieving the computational concern in the further analysis of one-shot NAS approaches. We first apply **RandomNAS-NSAS** and **GDAS-NSAS** to search for promising neural architectures in the common search space and compared the results with the two baselines and many other current one-shot NAS algorithms. We then further analyzed **NSAS** loss function with the NAS-Bench-201 benchmark dataset. Fig. 3 illustrates the differences between the two different search spaces. In the next two subsections, we describe the experimental settings for each of these search spaces.

4.1 One-Shot NAS Common Search Space

The design on the search space is important with NAS, and a common search space [38] is typically regarded as best for a fair comparison. The cell structure in this space contains eight different types of operations: 3×3 max pooling and average pooling operation, 3×3 and 5×5 separable convolution operation, 3×3 and 5×5 dilated separable convolutions operation, identity, and $zero$. There are seven nodes in each cell: two input nodes, four operation nodes, and one output node. The inputs to a cell are the outputs of two of its former cells, and the output of the cell is the sum of the outputs of all operation nodes. In a CNN structure, there are two types of cells with the same search space: a normal cell α_{normal} and a reduction cell α_{reduce}. The reduction cells are located in the 1/3 and 2/3 depths of the network as residual blocks. The cell structure is repeatedly stacked to form the final CNN structure.

The number of cells for the CNN supernet training was set to only 8. We used the momentum SGD optimizer for supernet to learn the weights, and an Adam optimizer to optimize the architecture parameters. The initial learning rate for SGD was set to 0.025 with a cosine schedule to annealed down to 0. The momentum of SGD and the weight decay were set to 0.9 and 3×10^{-4}, respectively. The initial learning rate for Adam was set to 3×10^{-4}, and the momentum and weight decay were set to (0.5,0.999) and 10^{-3}, respectively. The dropout probability was 0.4 with 16 initial channels. The CIFAR-10 dataset was used as the training set, divided into two halves at the architecture search stage. One half was used for the supernet weight learning, and the other for the architecture parameter optimization. Training took place with a batch size of 64 to derive the most promising cell structure. After supernet training and selecting the promising cells, we stacked 20 cells for full training with a batch size of 96.

The best-found cell structure with CIFAR-10 was then transferred to CIFAR-100 with the same hyperparameter settings as with the CIFAR-10 experiments. We also transferred the the best-found cell structure to ImageNet following the mobile settings in [30], [38], [56], and restricted the number of FLOPs to less than 600M. The weight decay is 3×10^{-5}, and the initial SGD learning rate is 0.1 with a decayed factor of 0.97. The network is stacked by 14 cells with batch size 128 with 250 epochs training.

As described in the previous Section 3.2 and outlined in Algorithm 1, one-shot NAS-NSAS selects M architectures from $|\mathcal{C}|$ previously visited architectures based on Algorithm 1. We set

TABLE 1: Results with the existing NAS approaches on CIFAR-10 and CIFAR-100.

Method	Test Error (%) CIFAR-10	Test Error (%) CIFAR-100	Param. (M)	FLOPs (M)	Search Cost (GPU Days)	Memory Consumption	Search Method
NASNet-A [72]	2.65	17.81	3.3	-	1800	Single path	RL
AmoebaNet-A [47]	3.34±0.06	-	3.2	-	3150	Single path	EA
Hierarchical Evo [37]	3.75±0.12	-	15.7	-	300	Single path	EA
PNAS [36]	3.41±0.09	17.63	3.2	-	225	P Single path	SMBO
IRLAS [19]	2.60	-	3.91	-	-	Single path	RL
IRLAS-differential [19]	2.71	-	3.43	-	-	Single path	RL
NAO [39]	3.18	-	10.6	-	1000	Single path	Gradient
NAO-WS [39]	3.53	-	2.5	-	-	Single path	Gradient
SETN (T=1K) [15]	2.69	17.25	4.6	606	1.8	Single path	Gradient
ENAS [46]	2.89	18.91	4.6	-	0.5	Single path	RL
SNAS [56]	2.85±0.02	20.09	2.8	422	1.5	Whole Supernet	Gradient
PARSEC [8]	2.86±0.06	-	3.6	485	0.6	Single path	Gradient
BayesNAS [71]	2.81±0.04	-	3.4	-	0.2	Whole Supernet	Gradient
RENAS [12]	2.88±0.02	-	3.5	-	6	-	RL&EA
MdeNAS [70]	2.55	17.61	3.8	599	0.16	Single path	MDL
DSO-NAS [68]	2.84±0.07	-	3.0	-	1	Whole Superne	Gradient
WPL [5]	3.81	-	-	-	-	Single path	RL
XNAS [44]	**2.57±0.09***	**16.34**	3.7	596	0.3	-	Gradient
PDARTS [11]	**2.50**	**16.63**	3.4	532	0.3	-	Gradient
PC-DARTS [57]	**2.57±0.07**	17.11	3.6	557	0.3	-	Gradient
Random baseline [38]	3.29±0.15	-	3.2	-	4	-	Random
DARTS (1st) [38]	2.94	-	2.9	501	1.5	Whole Supernet	Gradient
DARTS (2nd) [38]	2.76±0.09	17.54	3.4	528	4	Whole Supernet	Gradient
GDAS [16]	2.93	18.38	3.4	519	0.21	Single path	Gradient
GDAS-NSAS	2.75±0.08	18.02±0.05	3.5	528	0.4	Single path	Gradient
GDAS-NSAS-C	2.70±0.07	**16.70±0.08**	3.3	520	0.4	Single path	Gradient
RandomNAS [30]	2.85±0.08	17.63	4.3	612	2.7	Single path	Random
RandomNAS-NSAS	**2.59±0.06**	17.56±0.05	3.1	489	0.7	Single path	Random
RandomNAS-NSAS-C	**2.65±0.05**	**16.69±0.06**	3.5	552	0.7	Single path	Random

The first block contains the NAS methods without weight sharing. The approaches in the second block are the one-shot NAS methods. "*" indicates the results were reproduced with the best-reported cell structures in the original paper but with the same experimental settings as all the other comparators. Methods with "-" in the CIFAR-100 experiment were not reproduced because either they had different search spaces or did not report their best structures. All models were trained for 600 epochs, and we trained our best-searched architecture with 3 different random seeds to get the statistical results. "P Single path" means that the search space progressively increases during the architecture search, while only a single path is trained at each step of supernet training.

$M = 8$ and $|\mathcal{C}| = 50$ for the common CNN search space. The trade-off in Eq.(10) is set as $\beta = 0.5$.

4.2 NAS-Bench-201

The NAS-Bench-201 search space is similar to the recent cell-based NAS methods [30], [38], which repeatedly stack computational cells to form the final structure. The architectural skeleton of this search space contains three stages connected by a basic residual block [21]] with a stride of 2 between them. In each stage, the cell structure was stacked $N = 5$ times. In this search space, the cell structure is represented as a **densely** connected directed acyclic graph (DAG) with four nodes. There are six different edges between these nodes, and each edge is associated with five candidate operations, resulting in $5^6 = 15625$ candidate cell structures. The candidate operations included: a 1×1 convolution, 3×3 convolution, 3×3 average pooling, skip connection, and $zero$. The $zero$ helps to drop the associated edge. The initial channel c for the supernet was set to 16 and trained with an SDG optimizer. The learning rate was decayed from 0.025 to 0.001 with a cosine schedule, and the weight decay and the momentum were set to 0.0005 and 0.9, respectively. We followed the experimental setup in [17], and trained the supernet with a batch size of 64. After the architecture search phase and the most promising architectures in hand, we directly indexed the test performance of each architecture based on NAS-Bench-201 dataset [17] without training from scratch.

As described in Sec. 3.2, another important hyperparameter in our proposed method is the number of constraints in Eq.10. Since the search space of NAS-Bench-201 is very small, we default set $M = 2$ in this search space, and the trade-off for the constraints regularization term as $\beta = 0.2$.

5 Experiments and Results

Our first set of experiments was to conduct a neural architecture search on the common search space and compare with all methods, including our implementations, the baselines and a range of current and state-of-the-arts methods. The in-depth experiments with **NSAS** loss function and baselines on the NAS-Bench-201 dataset [17] followed.

5.1 Experimental Results on Common Search Space

5.1.1 Architecture Search on CIFAR-10

With this experiment, we searched for micro-cell structures in the search space and formed the final structure by stacking the cells in series. To compare the performance of one-shot NAS-NSAS with state-of-the-art NAS methods, we follow DARTS's experimental setting in [38]. We conducted the architecture search several times

TABLE 2: Results with existing manual-designed architectures and NAS approaches on the ImageNet dataset.

Method	Test Error (%) ImageNet	Param. (M)	FLOPs (M)
Inception-v1 [53]	30.2	6.6	1448
MobileNet [23]	29.4	4.2	569
MobileNet V2 [10], [49]	25.3	6.9	585
ShuffleNet 2× (V1) [69]	26.4	5	524
ShuffleNet 2× (V2) [40]	25.1	5	591
NASNet-A [73]	26.0	5.3	564
AmoebaNet-A [47]	25.5	5.1	555
PNAS [36]	25.8	5.1	588
SNAS [56]	27.3	4.3	522
SETN [15]	25.7	5.4	599
PARSEC [8]	26.3	5.5	-
BayesNAS [71]	26.5	3.9	-
MdeNAS [70]	26.8	6.1	595
DSO-NAS [68]	26.2	4.7	571
PDARTS [11]	25.9*	4.9	557
XNAS [44]	**25.3***(**24.7**†)	5.3	590
PC-DARTS [57]	25.7* (25.1†)	5.3	586
DARTS (2nd) [38]	26.7	4.7	574
GDAS [16]	27.5	4.4	497
GDAS-NSAS	26.7	5.1	564
GDAS-NSAS-C	25.9	5.2	565
RandomNAS [30]	27.1	5.4	595
RandomNAS-NSAS	26.1	5.2	581
RandomNAS-NSAS-C	**25.5** (**24.65**†)	5.4	593

The first block contains manually-designed architectures and the NAS methods without weight sharing. The second block contains one-shot NAS methods. We trained RandomNAS-NSAS with 52 initial channels C and RandomNAS-NSAS-C with $C = 50$. GDAS-NSAS and GDAS-NSAS-C were set to $C = 50$, and the FLOPs were restricted to less than 600M. * indicates that the architecture evaluation is reproduced following the common DARTS [38] setting, same as remaining methods. † indicates that the architecture evaluation settings are following PC-DARTS [57], with a warm-up linear learning rate scheduler.

with different random seeds to obtain the architectures, and then retrained them to pick the best architectures based on retraining validation performance. The comparison results are provided in Table 1 and can be summarized as follows:

- Compared to RandomNAS and GDAS, RandomNAS-NSAS and GDAS-NSAS greatly improve the search results. The **NSAS** loss function decreased the test errors from 2.85% for RandomNAS to 2.59%, from 2.93% for GDAS to 2.75%, demonstrating the effectiveness of NSAS at improving the predictive ability of the supernet.
- RandomNAS-NSAS' results were competitive compared to the other NAS methods, with a 2.59% test error and only 489M FLOPs. This is an inspiring result to validate our strategy for overcoming multi-model forgetting.
- Our **NSAS** evaluate more architectures during supernet training, so the search cost is slightly higher than the baselines. However, it still efficient in the sense that the supernet training in RandomNAS-NSAS only took 0.7 GPU days for and only 0.4 GPU days for GDAS-NSAS.

5.1.2 *Convolutional Cell Search with Depth Constraint to Improve Transferability*

In the next experiments, we transferred the best-found architectures from CIFAR-10 to CIFAR-100 and ImageNet datasets to evaluat their transferability. The results on CIFAR-100 are reported in Table 1 and ImageNet are reported in Table 2.

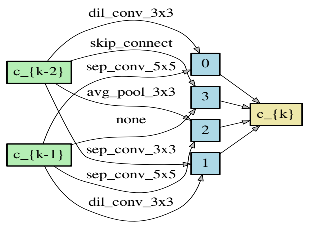
From Table 1 and Table 2, we can see that the **NSAS** loss function improves the performance of RandomNAS and GDAS with both datasets. However, although the **NSAS** methods yielded remarkable performance with CIFAR-10, the performance was not as impressive with CIFAR-100 and ImageNet. For example, **NSAS** decrease RandomNAS' test error from 2.85% to 2.59% on CIFAR-10, but only from 17.63% to 17.56% with CIFAR-100. Similarly, with ImageNet, the improvement was only 27.1% to 26.1%. More importantly though, the architectures returned by RandomNAS-NSAS on CIFAR-10 were competitive, while XNAS [44] was the superior method with CIFAR100 and ImageNet, thus demonstrating better transferability. XNAS [44] suggests that the architectures with "deeper" cell structures should provide superior performance with the ImageNet dataset. The authors also observe that most NAS methods usually return shallower cells with a larger width after searching CIFAR-10, noting that a visualization of the CNN models found from all one-shot baselines is provided in Appendix 2. For example, the architectures found by PC-DARTS and **NSAS** on CIFAR-10 are extremely shallow, which gave excellent results with CIFAR-10 but poor results with ImageNet. Conversely, the architectures found by XNAS, PDARTS, and PC-DARTS on ImageNet were much deeper and the results were impressive. A recent study on neural network optimization [50] gives a hint as to why most NAS methods prefer wider networks. The authors observe that width is a key factor affecting the convergence speed of neural networks, and therefore wider networks are easier to train. Based on this observation, the wider (shallower) architecture in the NAS search space reduces the loss with limited supernet training epochs, and has a higher probability of being chosen.

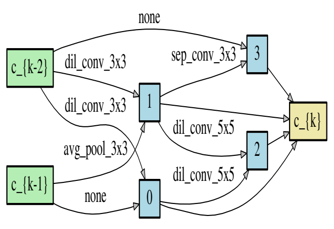
These findings suggest that encouraging NAS methods to search for "deeper" architectures could improve transferability; hence, our variant of **NSAS-C**, NSAS with depth constraint . The depth constraint in NSAS-C force NAS to search for "deeper" architectures. Simply put, the structure of the architectures are "fixed" to a depth so that the inputs of each node are the outputs of its previous node and the output of the previous cell. Figure 4 (c) and (d) show an example. This way, we only need to determine the operation of each edge in an architecture. All remaining experimental settings stay the same. Table 1 and 2 report the transferability of the models found by **NSAS-C** with CIFAR-100 and ImageNet. Compared to NSAS, the results are excellent. The best found cells by NSAS-C are shown in Fig. 4 (c) and (d), with competitive performance of 2.69%, 16.58%, and 25.5% test errors on CIFAR-10, CIFAR-100 and ImageNet, respectively. The codes and trained models are available online [1]. From Table 1, we can see that restricting the architecture depth is a very effective way of improving the transferability of NAS methods, e.g., 16.75% for GDAS-NSAS compared to 18.02% for GDAS with CIFAR-100, and RandomNAS-NSAS from 17.56% to 16.69%. Similarly, as shown in the ImageNet results in Table 2, NSAS-C again improves transferability, with improving GDAS-NSAS from 26.7% to 25.9%, and RandomNAS-NSAS from 26.2% to 25.5%.

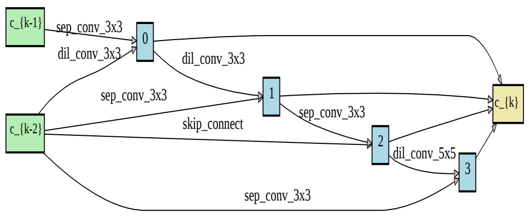
5.1.3 *Supernet Predictive Ability Comparison*

Multi-model Forgetting in One-Shot NAS To demonstrate catastrophic forgetting in a neural architecture search, we conducted

1. https://github.com/MiaoZhang0525/NSAS_FOR_CVPR.


(a) Normal cell with NSAS


(b) Reduction cell with NSAS


(c) Normal cell with NSAS-C

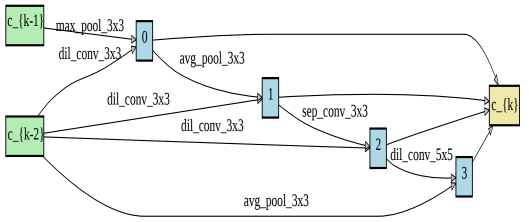

(d) Reduction cell with NSAS-C

Fig. 4: The best found cells with NSAS and NSAS-C on CIFAR-10.

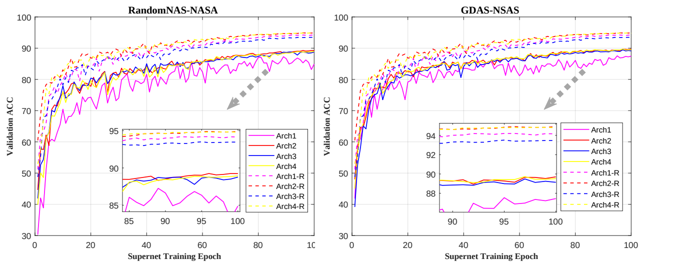


Fig. 5: The validation accuracy during supernet training for four different architectures with RandomNAS-NSAS and GDAS-NSAS. The solid lines ("Arch") indicate the validation accuracy with weights inherited from the supernet, and the dashed lines ("Arch-R") represent the validation accuracy after retraining.

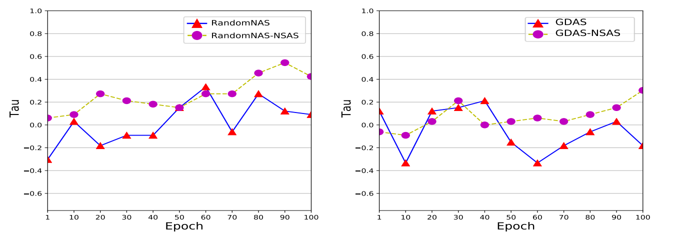
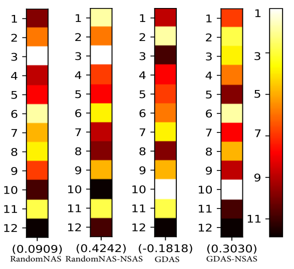


Fig. 6: The Kendall Tau metric (τ) of architecture ranking based on weight sharing and retraining.

experiments with a convolutional cell search task. The results show the differences between weight sharing and a retraining-based architecture ranking strategy. We tracked the validation accuracy of inheriting weights for several fixed sampled architectures with GDAS and also plotted the validation accuracy over 100 epochs when retraining these separate architectures from scratch in Fig. 1. From the results, we find that the validation accuracy of the architectures that directly inherit weights from the supernet fluctuate tremendously, making it hard to verify the quality of the architecture. What is worse is that the architecture ranking results completely violate the primary hypothesis of weight sharing NAS, i.e., that architectures with higher validation performance based on weight sharing should yield better retraining performance. It is worth noting that the performance of the architectures that inherited weights gets even worse during the supernet training, as shown in Fig. 1.

We also tracked the validation accuracy of weight sharing and retraining during the supernet training with RandomNAS-NSAS and GDAS-NSAS. The results are given in Fig. 5. We find that the NSAS loss function substantially alleviates multi-model forgetting with one-shot NAS. The plots of the validation accuracy with the inherited weight methods are much smoother, especially for architectures 2, 3, and 4. Moreover, performance does not decrease during supernet training. This is clearly a more reliable method.

Supernet Predictive Ability Comparison RandomNAS-NSAS and GDAS-NSAS should also alleviate ranking errors. The experiments we conducted to verify the architecture ranking predictions are shown in Figures 6 and 7. For these experiments, we sampled four of the best architectures over four rounds with RandomNAS and RandomNAS-NSAS, and four randomly sampled from the previous experiment. Then we individually trained these 12 architectures from scratch and calculated the correlation between the architecture ranking and the validation accuracy for each of the weight sharing and retraining approaches. Fig. 6 presents the Kendall Tau (τ) metric [25], [70] of the architecture rankings based on weight sharing and retraining. The results show the difference in rankings between the normal cross-entropy loss function and the NSAS loss function. Fig. 7 (a) gives the final Kendall Tau (τ) metric values for RandomNAS and GDAS with different loss functions after supernet training. Here, the normal loss function has poor supernet predictive ability, with only $\tau = 0.0909$


(a) Architectures ranking difference after supernet training

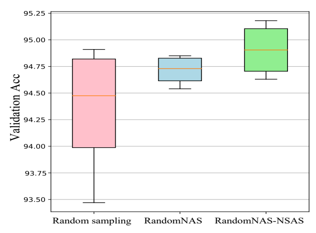

(b) Retraining validation accuracy

Fig. 7: (a) The architecture ranking differences between retraining and inheriting weights from a trained supernet with RandomNAS, RandomNAS-NSAS, GDAS, and GDAS-NSAS (from left to right, respectively). (b) The mean retraining validation accuracy for the architectures found through different methods.

TABLE 3: Results of one-shot NAS baselines on NAS-Bench-201.

Method	CIFAR-10		CIFAR-100		ImageNet-16-120	
	Valid(%)	Test(%)	Valid(%)	Test(%)	Valid(%)	Test(%)
ENAS [46]	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
DARTS (1st) [38]	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS (2nd) [38]	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
SETN [15]	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
RandomNAS [30]	80.42±3.58	84.07±3.61	52.12±5.55	52.31±5.77	27.22±3.24	26.28±3.09
GDAS [16]	90.00±0.21	93.51±0.13	71.14±0.27	70.61±0.26	41.70±1.26	41.84±0.09
RandomNAS* [30]	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
RandomNAS-NSAS	**89.20±0.31**	**92.61±0.10**	**68.62±1.94**	**68.47±1.73**	**41.17±2.16**	**41.68±1.91**
GDAS* [16]	89.88±0.33	93.40±0.49	70.95±0.78	70.33±0.87	41.28±0.46	41.47±0.21
GDAS-NSAS	**89.99±0.29**	**93.55±0.16**	**71.17±0.44**	**70.69±0.33**	**41.85±1.71**	**42.14±1.40**

"*" indicates that we reproduce the results with same random seeds as our approaches. All results in the first block are from [17]. The hyperparameters M and β were set to 5 and 0.5 for RandomNAS-NSA, 2 and 0.2 for GDAS-NSAS. We run each scenario for 4 independent times with random seed { *0, 1, 100, 101* } following the experimental settings in [17].

TABLE 4: Analysis of one-shot NAS with various settings for β and M on the NAS-Bench-201 dataset.

Method	β	CIFAR-10		CIFAR-100		ImageNet-16-120	
		Valid Acc(%)	Test Acc(%)	Valid Acc(%)	Test Acc(%)	Valid Acc(%)	Test Acc(%)
RandomNAS-NSAS ($M = 2$)	0	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
	0.2	86.38±4.35	89.58±2.82	63.64±6.36	64.72±4.47	34.68±7.80	34.19±5.72
	0.5	85.13±1.43	88.09±1.23	58.73±6.82	60.77±3.85	31.67±3.58	30.35±4.18
	0.8	86.82±2.44	90.14±2.35	64.41±4.34	64.27±3.26	35.06±4.81	34.82±6.06
RandomNAS-NSAS ($M = 3$)	0	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
	0.2	88.38±4.35	90.74±1.31	63.64±6.36	64.83±4.05	36.53±6.50	36.08±4.68
	0.5	87.93±1.63	91.23±1.03	66.03±3.46	66.17±4.12	38.14±2.76	38.72±3.11
	0.8	85.58±2.59	88.78±2.23	64.12±4.55	65.06±3.35	34.80±4.24	34.37±5.57
RandomNAS-NSAS ($M = 4$)	0	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
	0.2	86.73±1.30	90.64±0.99	62.38±4.57	66.42±1.47	36.68±3.80	37.59±3.72
	0.5	87.13±1.43	91.04±0.43	64.43±4.82	64.77±3.61	36.86±3.70	36.35±4.15
	0.8	88.52±0.74	92.04±0.50	67.40±2.22	67.62±1.94	39.91±4.50	40.61±3.51
RandomNAS-NSAS ($M = 5$)	0	85.30±0.59	88.14±0.21	62.60±3.56	63.40±4.52	33.60±4.36	33.83±3.17
	0.2	88.45±0.47	91.36±0.74	65.79±0.59	65.58±0.42	37.69±1.23	37.31±2.42
	0.5	**89.20±0.31**	**92.61±0.10**	**68.62±1.94**	**68.47±1.73**	**41.17±2.16**	**41.68±1.91**
	0.8	88.42±0.30	91.56±0.36	66.77±4.83	66.58±4.99	39.53±4.85	38.64±5.13

and $\tau = -0.1818$ for RandomNAS and GDAS, respectively. Although the supernet trained with the NSAS loss function was not able to provide identical architecture rankings, the positive correlations matched the Kendall Tau metrics ($\tau = 0.4242$ and $\tau = 0.3030$ for RandomNAS-NSAS and GDAS-NSAS, respectively). From this we surmise that a supernet with better predictive ability tends to provide architectures with better retraining performance. Fig. 7 (b) plots the mean retraining validation accuracy of the sampled architectures with various methods. We found that RandomNAS-NSAS achieved better results than RandomNAS, further verifying its effectiveness.

5.2 Experimental Results on NAS-Bench-201

Evaluating architectures in one-shot NAS is much more computationally intensive than an architecture search, so most state-of-the-art one-shot NAS methods only report the results of their best-found architectures. Comprehensive statistical analyses of the results are usually also overlooked due to computational limitations. Several concurrent studies [17], [27], [61], [63] have tried to address this problem by building benchmark datasets for NAS. With these datasets, researchers can analyze their one-shot NAS methods without evaluating numerous architectures. To analyze our approach in this way, we chose NASBench-201 [17] as a benchmark evaluation set. NAS-Bench-201 is easy to use and can be directly applied to most one-shot NAS algorithms. It also reports the performance of all candidate architectures on CIFAR-10, CIFAR-100, and ImageNet, making it sufficient to evaluate one-shot NAS algorithms. We did not restrict the width of architectures in the NAS-Bench-201 search space because the architectures are densely connected and have the same depth, making that constraint somewhat moot. To verify and further analyze the effectiveness of the NSAS loss function, we conducted three sets of experiments with this search space: 1) a comparison of the baselines; 2) a study of the hyperparameter settings; and 3) a study of the constraint selection strategies.

5.2.1 Empirical Comparison with Baselines

The results of the comparison study are presented in Table 3, and all experimental settings follow [17]. The statistical results were calculated from independent searches with four different *random seeds*. We found the NSAS loss function significantly improved the performance of the two baselines. RandomNAS-NSAS, in particular, achieved a test accuracy of 92.61%±0.10 on CIFAR-

TABLE 5: Analysis of the one-shot NAS methods with various constraint selection strategies on CIFAR-10.

METHOD	β	NSAS Test Acc(%)	NSAS-G Test Acc (%)	NSAS-LG Test Acc(%)	RG Test Acc(%)	LoW Test Acc(%)	LoW-R Test Acc (%)
RandomNAS	0.2	89.58±2.82	**91.74±1.16**	91.11±2.16	89.24±2.24	90.13±4.25	89.72±3.64
Test Acc(%)	0.5	88.09±1.23	90.37±2.14	**91.19±0.79**	77.30±19.76	89.86±2.91	86.85±10.29
(88.14±0.21)	0.8	90.14±2.35	**92.52±0.48**	91.18±1.73	89.53±0.24	89.39±3.88	85.54±7.18
GDAS	0.2	**93.55±0.16**	93.37±0.27	93.52±0.30	93.29±0.19	93.51±0.14	93.40±0.26
Test Acc(%)	0.5	93.49±0.24	**93.51±0.14**	93.46±0.27	93.31±0.30	93.47±0.29	93.36±0.21
(93.40±0.49)	0.8	93.32±0.18	93.29±0.29	**93.55±0.20**	93.40±0.28	**93.55±0.20**	**93.55±0.16**

10 compared to RandomNAS at only 88.14%±0.21. Similarly, GDAS-NSAS yielded a test accuracy of 93.55%±0.16 on CIFAR-10 compared to the 93.40%±0.49 of GDAS. Furthermore, the architectures searched by RandomNAS-NSAS and GDAS-NSAS also performed better when transferred to the larger CIFAR-100 and ImageNet datasets.

5.2.2 Hyperparameter Study

As described in Eq.(10), the trade-off β and the number of constraints M are important hyperparameters for the NSAS loss function $\mathcal{L}_N$. we studied the impact of β concurrent with M.

First, we considered to fix the number of constraints, and investigated the impact of four different settings of β on multi-model forgetting with one-shot NAS. In this experiment, we took the RandomNAS-NSAS with $M = 5$ as example. The results, shown in the last four rows of Table 4, indicate that using constraints to regularize the supernet training can greatly improve test performance and, additionally, that RandomNAS-NSAS is somewhat sensitive to β. More important, with different number of constraints ($M = 2, 3, 4$), the results all demonstrated our regularization method can enhance the performance, where our RandomNAS-NSAS with different M and β all outperformed the baseline.

We then fixed $\beta = 0.2$ and varied M, also to analyzed the impact on forgetting. Given there are only five candidate operations in the NAS-Bench-201 search space, there are no shared weights among constraints only when $M \leq 5$. The four settings for M in this experiment were 2, 3, 4, and 5. From the results, we found that, again, RandomNAS-NSAS seemed sensitive to the number of constraints, and the larger $M = 5$ gave much better results than the other scenarios for RandomNAS-NSAS with CIFAR10, CIFAR-100, and ImageNet. More interestingly, there was a large performance gain between $M = 4$ and $M = 5$ with RandomNAS-NSAS. One underlying reason may be that $M = 5$ has the potential to ensure the two assumptions hold true, as discussed in Section 3.3. Similarly, the tendency also exists in the remain 2 different β, that increasing the number of constrained architectures can enhance the performance. More details on these results can be found in Table 4.

Overall, Table 4 considered three settings for β and four settings for M, and presented the results of RadnomNAS-NSAS on CIFAR-10, CIFAR-100, and ImageNet-16-120. In general, a larger β and a larger M provided the better results. In the next subsection, we discuss the benefits of holding to the two assumptions with NSAS, with respect to the constrained architecture selection strategy.

5.2.3 Analysis of Constraints Selection

Although we demonstrate the theoretical benefits of the NSAS loss function in relieving catastrophic forgetting Section 3.3, and the experiments in Sections 5.2.1 and 5.2.2 support these theories, at least for one-shot NAS, is it still open to debate as to whether these improvements are due to the constraint selection strategy or simply because of the regularization. In this section, we further conduct an ablation study to investigate the impact of different architecture selection strategies.

Directly maximizing the diversity of the constraint subset, as per Section 3.4, easily holds **Assumption 2**, but it does not guarantee that **Assumption 1** will hold. Therefore, we devised two variants of the NSAS loss function, both of which strictly observe the assumptions when selecting constraints. These are **NSAS-G** and **NSAS-LG**. The difference between the two concerns treatment of the last architecture. More specifically, with **NSAS-G**, the constraints are generated randomly, maximizing diversity, but the last constraint θ_M is generated by complementing the operations contained in the current architecture α^t that have not been covered in the previous constraints. This means all selected architectures $\{\theta_1, ..., \theta_M\}$ covering all parameters of α_t such that $\theta_t \subseteq \{\theta_1 \cup ... \cup \theta_M\}$. With **NSAS-LG**, however, the last architecture α^{t-1} is first added into the subset, and remaining constraints are generated as following **NSAS-G**. This is to test the common thinking on catastrophic forgetting that the last architecture deteriorates performance the most.

We evaluated all three loss functions - **NSAS**, **NSAS-G**, and **NSAS-LG** - along with three naive architecture selection methods added to the RandomNAS and GDAS baseline to regularize the supernet training. Thus, the six loss functions were:

- **NSAS** - which selects constraints through maximizing diversity.
- **NSAS-G** - a variant of **NSAS** described above.
- **NSAS-LG** - a variant of **NSAS** described above.
- **RG** - randomly generates architectures to form the constraint subspace.
- **LoW** - only adds the last architecture α^{t-1} to the constraint subset.
- **LoW-R** - adds the last architecture α^{t-1} to the constraint subset plus randomly generated constraints.

Table 5, 6, and 7 show the test accuracies for the CIFAR-10, CIFAR-100, and ImageNet-16-120 datasets. We set the number of constraints $M = 2$ in this experiment, to more precisely investigate the effect of architecture selection and quantitatively analyze the constraint selection strategies. The results for one-shot NAS without relieving forgetting are shown in the first column of Table 5, 6, and 7. All NSAS methods improved performance but, interestingly, some of the naive constraint selection methods did

TABLE 6: Analysis of the one-shot NAS methods with various constraint selection strategies on CIFAR-100.

METHOD	β	NSAS Test Acc(%)	NSAS-G Test Acc (%)	NSAS-LG Test Acc(%)	RG Test Acc(%)	LoW Test Acc(%)	LoW-R Test Acc (%)
RANDOMNAS	0.2	64.72±4.47	**67.22±2.20**	66.58±3.43	63.66±3.97	64.06±7.89	60.68±9.04
Test Acc(%)	0.5	60.77±3.85	65.30±3.49	**66.74±2.66**	47.28±27.17	64.27±6.43	59.47±13.13
(63.40±4.52)	0.8	64.27±3.26	**67.83±1.66**	66.01±2.94	64.13±0.56	61.37±8.95	58.32±8.82
GDAS	0.2	70.69±0.33	70.49±0.61	**70.86±0.90**	70.43±0.56	70.53±0.34	70.40±0.51
Test Acc(%)	0.5	70.53±0.30	70.57±0.23	**70.69±0.67**	70.21±0.44	70.10±0.70	70.25±0.38
(70.33±0.87)	0.8	70.33±0.41	70.28±0.58	**70.80±0.55**	70.40±0.60	70.78±0.19	70.38±0.45

TABLE 7: Analysis of the one-shot NAS methods with various constraint selection strategies on ImageNet-16-120.

METHOD	β	NSAS Test Acc(%)	NSAS-G Test Acc (%)	NSAS-LG Test Acc(%)	RG Test Acc(%)	LoW Test Acc(%)	LoW-R Test Acc (%)
RANDOMNAS	0.2	34.19±5.72	**39.58±2.60**	37.79±5.11	34.38±5.02	36.32±8.07	30.64±13.19
Test Acc(%)	0.5	30.35±4.18	35.14±3.33	**39.81±2.81**	31.96±26.53	36.81±5.47	29.11±17.77
(33.83±3.17)	0.8	34.82±6.06	**40.14±2.60**	38.34±4.65	34.94±2.29	33.75±7.91	28.38±9.84
GDAS	0.2	42.14±1.40	**42.26±0.20**	41.71±0.57	41.35±0.13	42.16±1.30	41.68±1.18
Test Acc(%)	0.5	42.20±1.31	42.16±1.30	**42.29±1.00**	42.45±1.07	41.32±1.56	42.20±1.25
(41.47±0.21)	0.8	41.78±0.89	**42.21±0.16**	41.64±1.01	41.68±0.96	41.84±1.01	41.64±1.21

as well, which indicates that overcoming catastrophic forgetting is a promising research direction for one-shot NAS. For example, **LoW** improved performance simply by including the last visited architecture in the regularization. However, these results also show the importance of constraint selection strategy, as randomly selecting constraints can reduce performance. We observed that **RG** was the worst method in all cases, with a reduction in test accuracy from 88.14±0.21 to 77.30% ±19.76 for RandomNAS, and from 93.40±0.49 to 93.29±0.19 for GDAS in CIFAR-10. Moreover, the **LoW-R** strategy of adding more randomly generated constraints into the replay buffer yielded even worse results than **LoW** in most cases. These results suggest that randomly selecting constraints does not alleviate multi-model forgetting with one-shot NAS.

As for the **NSAS** and its variants, **NSAS-G** and **NSAS-LG**, all improved performance significantly. It is interesting that **NSAS** and **NSAS-G** achieved similar results with GDAS. This indicates that Assumption 1, which requires the constraints to cover all parameters of α_t, may not be so important for relieving catastrophic forgetting with gradient-based one-shot NAS while holding to this assumption with RandomNAS did help. Overall, **NSAS-G** achieved much better results than **NSAS** and, in most cases, **NSAS-G** and **NSAS-LG** produced the best results. Thus, simultaneously considering the last visited architecture and maximizing the diversity of constraints combined are the two key factors that need to be addressed to relieve catastrophic forgetting with both random sampling-based and gradient-based one-shot NAS.

5.3 Discussion

We can draw several conclusions from this series of experiments.

- RandomNAS tends to achieve better performance than GDAS with a common search space, whereas GDAS outperforms RandomNAS with the NASBench-201 space no matter the loss function. This may be because gradient-based methods typically arrive at the local optimal solution once the supernet is trained. RandomNAS, however, must perform a subsequent model selection process using either a random search or an EA to find a global optimal solution from the trained supernet. Since common search spaces are much more complicated than the one in NAS-Bench-201, a global optimization method will usually outperform a gradient method, while gradient-based NAS is more efficient and effective with simple search spaces.
- It is clear that the NSAS loss function can increase the predictive ability of the supernet, which, in turn, greatly improves the performance of the architectures found by RandomNAS and GDAS. However, supernet training in one-shot NAS is still a problem with much room for further advancements. Devising a more appropriate loss function than the status quo appears to be a promising direction for improving the performance of one-shot NAS methods.
- Lastly, the ablation study indicates that adding recently visited architectures into the constraint subset and maximizing its diversity are two efficient ways to mitigate catastrophic forgetting with one-shot NAS.

6 Conclusion and Future Works

In this paper, we formulated supernet training as a constrained optimization problem to reduce some of the negative impacts of catastrophic forgetting with one-shot NAS, and multi-model forgetting in particular. Our strategy is to select a representative subset of constraints with a greedy novelty search method. Then the supernet training is regularized in a feasible region with a new novelty search-based architecture selection loss function, i.e., **NSAS** to overcome multi-model forgetting.

We implemented **NSAS** into two one-shot NAS baselines - RandomNAS and GDAS - and compared the quality of the architecture selections with and without the new loss function. The results of experiments on the common search space of a neural architecture show **NSAS** and two of its variants improve the predictive ability of the supernet with both convolutional and recurrent cell search. Experiments with the NAS-Bench-201 dataset also suggest that **NSAS** can substantially offset performance degradation due to forgetting with one-shot NAS. In future research, we plan to focus on searching on a latent space by transforming discrete architectures into continuous representations. Further, we will look to leveraging expert knowledge with DNN searches to design architectures with greater transferability.

Acknowledgment

This study was supported in part by the NSFC under Grant No. 61702415 and No. 61972315, the Australian Research Council (ARC) under a Discovery Early Career Researcher Award (DECRA) No. DE190100626, the Air Force Research Laboratory, and DARPA under Agreement No. FA8750- 19-2-0501.

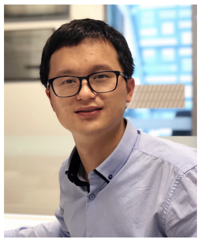
References

[1] George Adam and Jonathan Lorraine. Understanding neural architecture search techniques. *arXiv preprint arXiv:1904.00438*, 2019.
[2] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *NeurIPS*, 2019.
[3] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. 2018.
[4] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018.
[5] Yassine Benyahia, Kaicheng Yu, Kamil Bennani Smires, Martin Jaggi, Anthony C Davison, Mathieu Salzmann, and Claudiu Musat. Overcoming multi-model forgetting. In *ICML*, 2019.
[6] Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas J Weston. Smash: One-shot model architecture search through hypernetworks. In *ICLR*, 2018.
[7] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.
[8] Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116*, 2019.
[9] Xiaojun Chang, Yaoliang Yu, Yi Yang, and Eric P. Xing. Semantic pooling for complex event analysis in untrimmed videos. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(8):1617–1632, 2017.
[10] Sheng Chen, Yang Liu, Xiang Gao, and Zhen Han. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. In *CCBR*, 2018.
[11] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019.
[12] Yukang Chen, Gaofeng Meng, Qian Zhang, Shiming Xiang, Chang Huang, Lisen Mu, and Xinggang Wang. Renas: Reinforced evolutionary neural architecture search. In *CVPR*, 2019.
[13] Xuelian Cheng, Yiran Zhong, Mehrtash Harandi, Yuchao Dai, Xiaojun Chang, Hongdong Li, Tom Drummond, and Zongyuan Ge. Hierarchical neural architecture search for deep stereo matching. In *NeurIPS*, 2020.
[14] 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.
[15] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *ICCV*, 2019.
[16] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *CVPR*, 2019.
[17] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020.
[18] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.
[19] Minghao Guo, Zhao Zhong, Wei Wu, Dahua Lin, and Junjie Yan. Irlas: Inverse reinforcement learning for architecture search. In *CVPR*, 2019.
[20] 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:1906.07590*, 2019.
[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.
[22] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.
[23] 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.
[24] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.
[25] Maurice G Kendall. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945.
[26] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.
[27] Aaron Klein and Frank Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.
[28] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *NeurIPS*, 2017.
[29] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Blockwisely supervised neural architecture search with knowledge distillation. In *CVPR*, 2020.
[30] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *UAI*, 2019.
[31] Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot nas by suppressing the posterior fading. *arXiv preprint arXiv:1910.02543*, 2019.
[32] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *ICML*, 2019.
[33] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.
[34] Zhihui Li, Lina Yao, Xiaojun Chang, Kun Zhan, Jiande Sun, and Huaxiang Zhang. Zero-shot event detection via event-adaptive concept relevance mining. *Pattern Recognit.*, 88:595–603, 2019.
[35] Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta learning. In *ICLR*, 2020.
[36] 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 *ECCV*, 2018.
[37] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018.
[38] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019.
[39] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018.
[40] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, pages 116–131, 2018.
[41] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.
[42] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. In *ICLR*, 2020.
[43] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. In *ICLR*, 2018.
[44] Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik-Manor. Xnas: Neural architecture search with expert advice. In *NeurIPS*, 2019.
[45] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.
[46] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.
[47] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.
[48] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv:2006.02903*, 2020.
[49] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.
[50] Karthik A. Sankararaman, Soham De, Zheng Xu, W. Ronny Huang, and Tom Goldstein. Analyzing the effect of neural network architecture on training performance. In *ICML*, 2020.
[51] Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. In *ICLR*, 2020.
[52] Prabhant Singh, Tobias Jacobs, Sebastien Nicolas, and Mischa Schmidt. A study of the learning progress in neural architecture search techniques. *arXiv preprint arXiv:1906.07590*, 2019.

[53] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015.
[54] Weiyao Wang, Du Tran, and Matt Feiszli. What makes training multi-modal networks hard? *arXiv preprint arXiv:1905.12681*, 2019.
[55] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *arXiv preprint arXiv:1910.11858*, 2019.
[56] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In *ICLR*, 2019.
[57] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *ICLR*, 2020.
[58] Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. In *ICLR*, 2020.
[59] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *CVPR*, 2020.
[60] Quanming Yao, Ju Xu, Wei-Wei Tu, and Zhanxing Zhu. Efficient neural architecture search via proximal iterations. In *AAAI*, 2020.
[61] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*, 2019.
[62] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *ICLR*, 2020.
[63] Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *arXiv preprint arXiv:2001.10422*, 2020.
[64] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *ICLR*, 2019.
[65] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Zongyuan Ge, and Steven Su. Differentiable neural architecture search in equivalent space with exploration enhancement. In *NeurIPS*, 2020.
[66] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, and Steven Su. Overcoming multi-model forgetting in one-shot nas with diversity maximization. In *CVPR*, 2020.
[67] Miao Zhang, Huiqi Li, Shirui Pan, Taoping Liu, and Steven Su. One-shot neural architecture search via novelty driven sampling. In *IJCAI*, 2020.
[68] Xinbang Zhang, Zehao Huang, and Naiyan Wang. You only search once: Single shot neural architecture search via direct sparse optimization. *arXiv preprint arXiv:1811.01567*, 2019.
[69] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, pages 6848–6856, 2018.
[70] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial distribution learning for effective neural architecture search. In *ICCV*, 2019.
[71] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, 2019.
[72] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.
[73] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.

Miao Zhang received a Ph.D. from Beijing Institute of Technology (BIT), China. He is also pursuing a dual-degree of Ph.D. in the University of Technology Sydney (UTS). His major research interests are AutoML, neural architecture search, Bayesian optimization, continual learning, and deep learning.

Huiqi Li received Ph.D. degree from Nanyang Technological University, Singapore in 2003. She is currently a professor at Beijing Institute of Technology. Her research interests are image processing and computer-aided diagnosis.



Shirui Pan received a Ph.D. in computer science from the University of Technology Sydney (UTS), Ultimo, NSW, Australia. He is currently a lecturer with the Faculty of Information Technology, Monash University, Australia. Prior to this, he was a Lecturer with the School of Software, University of Technology Sydney. His research interests include data mining and machine learning. To date, Dr Pan has published over 60 research papers in top-tier journals and conferences.

Xiaojun Chang is currently a Faculty Member with the Faculty of Information Technology, Monash University, Clayton, VIC, Australia. He is also a Distinguished Adjunct Professor with the Faculty of Computing and Information Technology, King Abdulaziz University. He is an ARC Discovery Early Career Researcher Award (DECRA) Fellow from 2019 to 2021. He has achieved top performance in various international competitions, such as TRECVID MED, TRECVID SIN, and TRECVID AVS.

Chuan Zhou obtained Ph.D. degree from Chinese Academy of Sciences in 2013. He won the outstanding doctoral dissertation of Chinese Academy of Sciences in 2014, the best paper award of ICCS-14, and the best student paper award of IJCNN-17. Currently, he is an Associate Professor at the Academy of Mathematics and Systems Science, Chinese Academy of Sciences. His research interests include socail network analysis and graph mining. To date, he has published more than 70 papers, including IEEE TKDE, ICDM, AAAI, CIKM, IJCAI and WWW.

Zongyuan Ge is a full-time Senior Research Fellow employed by Monash University Vice Chancellor and Provost Office with specific interest and expertise in Medical AI development. He has a strong background in statistical analysis, machine learning and computer vision research. So far, he has published more than 40 peer-reviewed publications and patents, which are first/senior author. Zongyuan was selected as one of the 200 Most Qualified Young Researchers in Computer and Mathematics by the Scientific Committee of the Heidelberg Laureate Forum Foundation in 2017 and received Monash Exceptional Research Award 2019.

Steven Su received Ph.D. degree in Control Engineering from RSISE the Australian National University (ANU). He is currently an Associate Professor in University of Technology Sydney (UTS). His research interest are system modeling and control, machine learning, wearable health monitoring, and rehabilitation engineering.

TABLE 8: Comparison results with the existing NAS methods on PTB.

Method	Perplexity(PTB) Valid	Perplexity(PTB) Test	Parameters (M)	Search Cost	Memory Consumption	Search Method
LSTM [72]	60.7	58.8	24	-	-	Manual Design
LSTM+SC [43]	60.9	58.3	24	-	-	Manual Design
NAS baseline [72]	-	64	25	$1e^4$	Single path	RL
Random baseline [38]	61.8	59.4	23	2	Single path	Random
ENAS [30]	60.8	58.6	24	0.5	Single path	RL
WPL [5]	-	61.9	-	-	Single path	RL
DARTS (1st) [38]	60.2	57.6	23	0.5	Whole Supernet	Gradient
DARTS (2nd) [38]	58.8	56.6	23	1	Whole Supernet	Gradient
RandomNAS* [30]	59.7*	57.16*	23	0.25	Single path	Random
RandomNAS-NSAS	**59.22**	**56.84**	23	0.62	Single path	Random
GDAS [16]	59.8	57.5	23	0.4	Single path	Gradient
GDAS-NSAS	59.74	57.24	23	0.50	Single path	Gradien

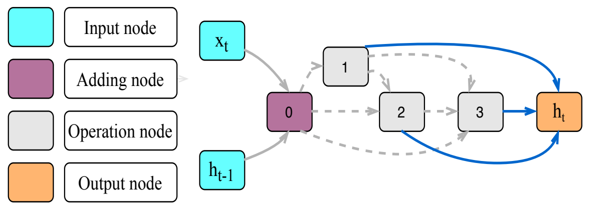
"*" indicates the results were reproduced with the best-reported cell structures in the original paper but with the same experimental settings as all the other competitors. The first block includes manually-designed neural architectures, and the second block includes the NAS methods without weight sharing and the one-shot NAS methods.

APPENDIX1: ARCHITECTURE SEARCH FOR RECURRENT CELLS

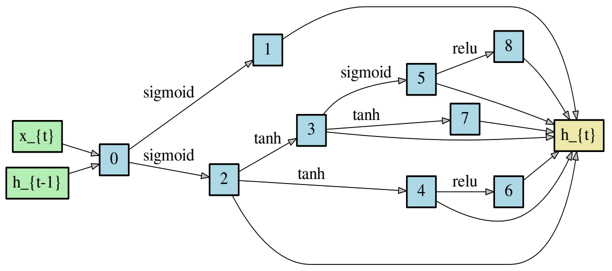
Beyond the convolutional cell structure search, we also conducted experiments on RNN cell structure search with the PTB dataset. For a fair comparison, the search space and hyperparameters were set following [30], [38]. The RNN's search space only contains four different types of operations: $identity$, $relu$, $tanh$, $sigmoid$. The recurrent cell contains 12 nodes: 2 input nodes, 1 adding nodes, 8 operation nodes, and 1 output node. The adding node adds the two inputs and applies the $tanh$ activation function. The input of each node is the output of one of its previous nodes, and the output of the cell is the summation of outputs of all operation nodes. Unlike the CNN structure, our RNN architecture only contains a single cell. Fig.8 (a) shows the common search spaces of the RNN cell structure.

We used an SGD optimizer for the recurrent architecture search, without momentum for weight learning. The learning rate was set to 20.0, and the weight decay was set to 5×10^{-7}. Like the CNN search, we also used an Adam optimizer for architecture optimization. The initial learning rate was set to 3×10^{-3} , the weight decay was 10^{-3}, with a momentum of (0.9,0.999). Variational dropout was used with probabilities of 0.2 for the word embeddings, 0.75 for the cell input, and 0.25 for all the hidden nodes. Dropout was also applied to the output layer with a probability of 0.75. We trained the supernet for 300 epochs and a batch size of 256 to arrive at the most promising RNN cell structure. Then, to train the best-found recurrent cell, we changed the embedding and the hidden sizes to 850 with a batch size of 64. The other hyperparameters had the same settings as before. All setups of the search space for CNN and RNN can also be found in [30], [38].

A comparison of the results across all the baselines is presented in Table 8. The best model discovered by RandomNAS-NSAS had a validation and test perplexity of 59.22 and 56.84, respectively, which are on par with the state-of-the-art approaches. Further, the two baselines with the NSAS loss function showed an improved supernet predictive ability, as evidenced by RandomNAS-NSA' inferior test perplexity with the normal loss function of 56.84 compared to 57.16 with NSAS and GDAS with 57.24 compared to 57.69 with NSAS. Fig. 8 (b) visualizes the best-found cells with the RandomNAS-NSAS approach for the RNN models.



(a) Common Search Space for RNN



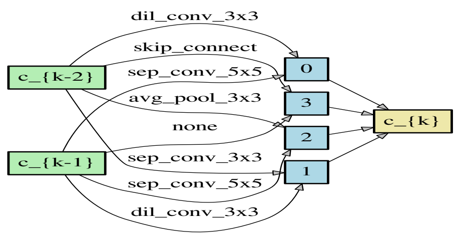
(b) Reccurent cell learned on PTB

Fig. 8: The search space and the best discovered RNN cell.

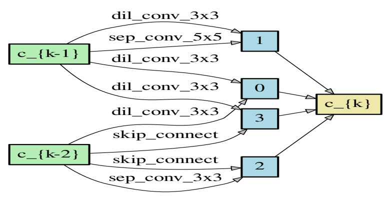
APPENDIX2: VISUALIZATION FOR THE BEST SEARCHED CNN ARCHITECTURES AND DEPTH

As discussed in Section 5.1.2, most existing NAS approaches are expected to find shallow CNN structures when searching the CIFAR-10 dataset, but deliver poor performance when transferred to ImageNet dataset. Several recent studies, including XNAS [44], PDARTS [11], and PC-DARTS [57], suggest that encouraging a deeper search is an effective way to improve the transferability of the found models. In this section, we first visualize the best found normal cell structures through several existing one-shot NAS methods in Fig. 9. From the results, we find the first five cells are much shallower than the rest of the structures.

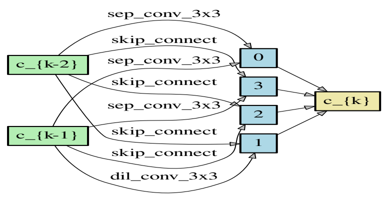
We also followed the metric in Section 10 of XNAS to measure the cell "depth". This metric calculates the average of the input



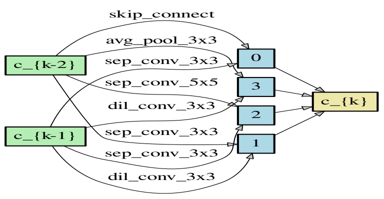
(a) Normal cell with our NSAS



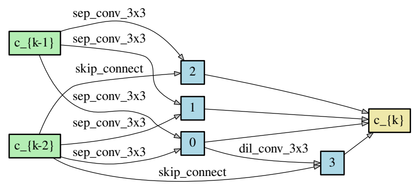
(b) Normal cell with PARSEC [8]



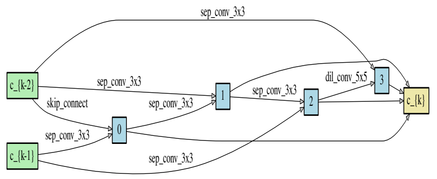
(c) Normal cell with SNAS [56]



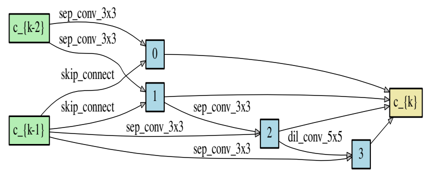
(d) Normal cell with PC-DARTS-C [44]



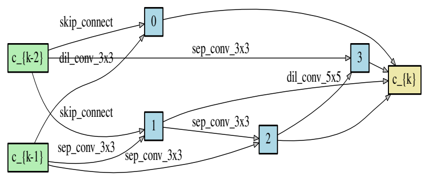
(e) Normal cell with DARTS (2nd) [38]



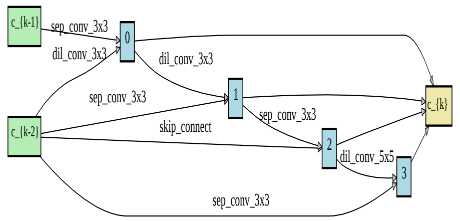
(f) Normal cell with XNAS [44]



(g) Normal cell with PC-DARTS-I [44]



(h) Normal cell with PDARTS [11]



(i) Normal cell with our NSAS-C

Fig. 9: Several example normal cell structures searched one-shot NAS methods. PC-DARTS-I is the same as ours that the model searched on CIFAR-10 and transferred to ImageNet, while PC-DARTS-C is the model directly searched on ImageNet.

TABLE 9: The depth of normal cells with the NAS approaches on CIFAR-10 and their performance on ImageNet.

Method	Test Error (%)		Depth
	CIFAR-10	ImageNet	
SNAS [56]	2.85±0.02	27.3	0.5
DARTS (2nd) [38]	2.76±0.09	26.7	0.625
PARSEC [8]	2.86±0.06	26.3	0.5
Our NSAS	2.59±0.05	26.2	0.5
PDARTS [11]	2.50	25.9*	1.25
PC-DARTS-C [57]	2.57±0.07	25.7*	0.5
SETN [15]	2.69	25.7	1.375
Our NSAS-C	2.69±0.05	**25.5**	1.25
XNAS [44]	2.57±0.09*	25.3*	1.375

The one-shot NAS methods are ranked in descending order of performance on ImageNet. * indicates that the architecture evaluation on ImageNet follows the common DARTS [38] setting.

index of all the nodes in a cell. Since there are only two reduction cells compared to 12 normal cells in the final CNN structure with the ImageNet dataset, the depth of a normal cell is the main indicator for measuring the depth of the final CNN structure.

To more intuitively reveal the depth and the transferability of the found models, we also present the depth of normal cells searched on CIFAR-10 and ImageNet using the one-shot NAS methods in Table 9. Similar to XNAS, we observed that, in general, those architectures with a larger depth performed better and had higher transferability with ImageNet.

Another interesting finding from Table 9 is that our reproduced results for PDARTS [11], PC-DARTS [57], and XNAS [44] are not as good as the results reported in the original studies. PC-DART should have resulted in a 25.1% test error with ImageNet, but we found a test error of 25.7%. Similarly, PCDART obtained a 25.9% test error with ImageNet rather than 24.4%. Likewise, XNAS's test error should have been 24.0% with ImageNet but, in attempting to reproduce that result, we arrived at 25.3%. With CIFAR-10, we found a test error of 2.52% compared to its reported 1.81% in XNAS. With PDARTS [11] and PC-DARTS [57], the authors changed the architecture evaluation settings and added some tricks to improve performance with the ImageNet dataset. However, although these tricks may have significantly improved performance with some architectures, validation accuracy with others was reduced and even deteriorated in some cases. Similarly, XNAS [57] improved performance with its selected model with CIFAR-10 by increasing the number of initial filters and training epochs, augmenting the data, and other tricks. Although they achieved state-of-the-art performance through these changes, we could not reproduce these results in a scenario with settings common to all baselines. A standard framework for fairly evaluating architectures is still an open problem in the neural architecture search community.