TF-NAS: Rethinking Three Search Freedoms of Latency-Constrained Differentiable Neural Architecture Search

Yibo Hu[1,2], Xiang Wu[1], and Ran He[1⋆]

[1] CRIPAC & NLPR, CASIA, Beijing, China
[2] JD AI Research, Beijing, China
{huyibo871079699,alfredxiangwu}@gmail.com, rhe@nlpr.ia.ac.cn

Abstract. With the flourish of differentiable neural architecture search (NAS), automatically searching latency-constrained architectures gives a new perspective to reduce human labor and expertise. However, the searched architectures are usually suboptimal in accuracy and may have large jitters around the target latency. In this paper, we rethink three freedoms of differentiable NAS, i.e. operation-level, depth-level and width-level, and propose a novel method, named Three-Freedom NAS (TF-NAS), to achieve both good classification accuracy and precise latency constraint. For the operation-level, we present a **bi-sampling** search algorithm to moderate the operation collapse. For the depth-level, we introduce a **sink-connecting** search space to ensure the mutual exclusion between skip and other candidate operations, as well as eliminate the architecture redundancy. For the width-level, we propose an **elasticity-scaling** strategy that achieves precise latency constraint in a progressively fine-grained manner. Experiments on ImageNet demonstrate the effectiveness of TF-NAS. Particularly, our searched TF-NAS-A obtains 76.9% top-1 accuracy, achieving state-of-the-art results with less latency. Code is available at https://github.com/AberHu/TF-NAS.

Keywords: Differentiable NAS, Latency-constrained, Three Freedoms

1 Introduction

With the rapid developments of deep learning, ConvNets have been the *de facto* method for various computer vision tasks. It takes a long time and substantial effort to devise many useful models [14,18,19,23,28,29], boosting significant improvements in accuracy. However, instead of accuracy improvement, designing efficient ConvNets with specific resource constraints (e.g. FLOPs, latency, energy) is more important in practice. Manual design requires a huge number of exploratory experiments, which is time-consuming and labor intensive. Recently, Neural Architecture Search (NAS) has attracted lots of attentions [21,22,26,33,39]. It learns to automatically discover resource-constrained architectures, which can achieve better performance than hand-craft architectures.

⋆ corresponding author

Most NAS methods are based on reinforcement learning (RL) [30,39,40] or evolutionary algorithms (EA) [6,8,26], leading to expensive or even unaffordable computing resources. Differentiable NAS [4,22,33] couples architecture sampling and training into a supernet to reduce huge resource overhead. This supernet supports the whole search space with three freedoms, including the operation-level, the depth-level and the width-level freedoms. However, due to the various combinations of search freedoms and the coarse-grained discreteness of search space, differentiable NAS often makes the searched architectures suboptimal with specific resource constraints. For example, setting the GPU latency constraint to 15ms and carefully tuning the trade-off parameters, we search for architectures based on the latency objective from ProxylessNAS [4]. The searched architecture has 15.76ms GPU latency, exceeding the target by a large margin. More analyses are presented in Sec. 4.5.

To address the above issue, in this paper, we first rethink the operation-level, the depth-level and the width-level search freedoms, tracing back to the source of search instability. For the operation-level, we observe operation collapse phenomenon, where the search procedure falls into some fixed operations. To alleviate such collapse, we propose a bi-sampling search algorithm. For the depth-level, we analyze the special role of skip operation and explain the mutual exclusion between skip and other operations. Furthermore, we also illustrate architecture redundancy by a simple case study in Fig. 3. To address these phenomena, we design a sink-connecting search space for NAS. For the width-level, we explore that due to the coarse-grained discreteness of search space, it is hard to search target architectures with precise resource constraints (e.g. latency). Accordingly, we present an elasticity-scaling strategy that progressively refines the coarse-grained search space by shrinking and expanding the model width, to precisely ensure the latency constraint. Combining the above components, we propose Three-Freedom Neural Architecture Search (TF-NAS) to search accurate latency-constrained architectures. To summarize, our main contributions lie in four-folds:

- Motivated by rethinking the operation-level, the depth-level and the width-level search freedoms, a novel TF-NAS is proposed to search accurate architectures with latency constraint.
- We introduce a simple bi-sampling search algorithm to moderate operation collapse phenomenon. Besides, the mutual exclusion between skip and other candidate operations, as well as the architecture redundancy, are first considered to design a new sink-connecting search space. Both of them ensure the search flexibility and stability.
- By investigating the coarse-grained discreteness of search space, we propose an elasticity-scaling strategy that progressively shrinks and expands the model width to ensure the latency constraint in a fine-grained manner.
- Our TF-NAS can search architectures with precise latency on target devices, achieving state-of-the-art performance on ImageNet classification task. Particularly, our searched TF-NAS-A achieves 76.9% top-1 accuracy with only 1.8 GPU days of search time.

2 Related Work

Micro Search focuses on finding robust cells [25,26,27,34,40] and stacking many copies of them to design the network architecture. AmoebaNet [26] and NASNet [40], which are based on Evolutionary Algorithm (EA) and Reinforcement Learning (RL) respectively, are the pioneers of micro search algorithms. However, these approaches take an expensive computational overhead, i.e. over 2,000 GPU days, for searching. DARTS [22] achieves a remarkable efficiency improvement (about 1 GPU day) by formulating the neural architecture search tasks in a differentiable manner. Following gradient based optimization in DARTS, GDAS [11] is proposed to sample one sub-graph from the whole directed acyclic graph (DAG) in one iteration, accelerating the search procedure. Xu et al. [36] randomly sample a proportion of channels for operation search in cells, leading to both faster search speed and higher training stability. P-DARTS [5] allows the depth of architecture to grow progressively in the search procedure, to alleviate memory/computational overheads and weak search instability. Comparing with accuracy, it is obvious that micro search algorithms are unfriendly to constrain the number of parameters, FLOPs and latency for neural architecture search.

Macro Search aims to search the entire neural architecture [4,6,30,31,33,39], which is more flexible to obtain efficient networks. Baker et al. [1] introduce MetaQNN to sequentially choose CNN layers using Q-learning with an ϵ-greedy exploration strategy. MNASNet [30] and FBNet [33] are proposed to search efficient architectures with higher accuracy but lower latency. One-shot architecture search [2] designs a good search space and incorporates path drop when training the over-parameterized network. Since it suffers from the large memory usage to train an over-parameterized network, Cai et al. [4] propose ProxylessNAS to provide a new path-level pruning perspective for NAS. Different from the previous neural architecture search, EfficientNet [31] proposes three model scaling factors including width, depth and resolution for network designment. Benefiting from compounding scales, they achieve state-of-the-art performance on various computer vision tasks. Inspired by EfficientNet [31], in order to search for flexible architectures, we rethink three search freedoms, including **operation-level**, **depth-level** and **width-level**, for latency-constrained differentiable neural architecture search.

3 Our Method

3.1 Review of Differentiable NAS

In this paper, we focus on differentiable neural architecture search to search accurate macro architectures constrained by various inference latencies. Similar with [11,22,33], the search problem is formulated as a bi-level optimization:

$$\min_{\alpha \in A} L_{\text{val}}\left(\omega^*, \alpha\right) + \lambda C\left(LAT(\alpha)\right) \tag{1}$$

$$\text{s.t.} \quad \omega^* = \arg\min_{\omega} L_{\text{train}}\left(\omega, \alpha\right) \tag{2}$$

where ω and α are the supernet weights and the architecture distribution parameters, respectively. Given a supernet A, we aim to search a subnet $\alpha^* \in A$ that minimizes the validation loss $L_{val}(\omega^*, \alpha)$ and the latency constraint $C(LAT(\alpha))$, where the weights ω^* of supernet are obtained by minimizing the training loss $L_{train}(\omega, \alpha)$ and λ is a trade-off hyperparameter.

Different from RL-based [30,39,40] or EA-based [6,8,26] NAS, where the outer objective Eq. (1) is treated as reward or fitness, differentiable NAS optimizes Eq. (1) by gradient descent. Sampling a subnet from supernet A is a non-differentiable process w.r.t. the architecture distribution parameters α. Therefore, a continuous relaxation is needed to allow back-propagation. Assuming there are N operations to be searched in each layer, we define op_i^l and α_i^l as the i-th operation in layer l and its architecture distribution parameter, respectively. Let x^l present the input feature map of layer l. A commonly used continuous relaxation is based on Gumbel Softmax trick [11,33]:

$$x^{l+1} = \sum_i u_i^l \cdot \text{op}_i^l\left(x^l\right), u_i^l = \frac{\exp\left(\left(\alpha_i^l + g_i^l\right)/\tau\right)}{\sum_j \exp\left(\left(\alpha_j^l + g_j^l\right)/\tau\right)} \tag{3}$$

$$LAT(\alpha) = \sum_l LAT\left(\alpha^l\right) = \sum_l \sum_i u_i^l \cdot LAT\left(\text{op}_i^l\right) \tag{4}$$

where τ is the temperature parameter, g_i^l is a random variable i.i.d sampled from $Gumbel(0, 1)$, $LAT\left(\alpha^l\right)$ is the latency of layer l and $LAT\left(\text{op}_i^l\right)$ is indexed from a pre-built latency lookup table. The superiority of Gumbel Softmax relaxation is to save GPU memory by approximate N times and to reduce search time. That is because only one operation with max u_i^l is chosen during forward pass. And the gradients of all the α_i^l can be back-propagated through Eq. (3).

3.2 The Search Space

In this paper, we focus on latency-constrained macro search. Inspired by EfficientNet [31], we build a layer-wise search space, which is depicted in Fig. 1 and Tab. 1. The input shapes and the channel numbers are the same as EfficientNet-B0 [31]. Different from EfficientNet-B0, we use ReLU in the first three stages. The reason is that the large resolutions of the early inputs mainly dominate the inference latency, leading to worse optimization during architecture searching.

Layers from stage 3 to stage 8 are searchable, and each layer can choose an operation to form the operation-level search space. The basic units of the candidate operations are MBInvRes (the basic block in MobileNetV2 [28]) with or without Squeeze-and-Excitation (SE) module, which are illustrated in Supp. 1. In our experiments, there are 8 candidate operations to be searched in each searchable layer. The detailed configurations are listed in Tab. 1. Each candidate operation has a kernel size $k = 3$ or $k = 5$ for the depthwise convolution, and a continuous expansion ratio $e \in [2, 4]$ or $e \in [4, 8]$, which constitutes to the width-level search space. Considering the operations with SE module, the SE expansion ratio is $e_{se} = 1$ or $e_{se} = 2$. In Tab. 1, the ratio of e_{se} to e for all

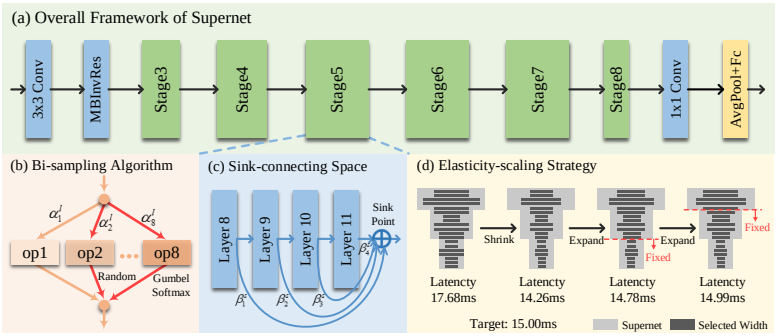

Fig. 1. The search space of TF-NAS. It contains (b) operation-level, (c) depth-level and (d) width-level search freedoms.

Stage	Input	Operation	C_{out}	Act	L
1	$224^2 \times 3$	3×3 Conv	32	ReLU	1
2	$112^3 \times 32$	MBInvRes	16	ReLU	1
3	$112^2 \times 16$	OPS	24	ReLU	$[1, 2]$
4	$56^2 \times 24$	OPS	40	Swish	$[1, 3]$
5	$28^2 \times 40$	OPS	80	Swish	$[1, 4]$
6	$14^2 \times 80$	OPS	112	Swish	$[1, 4]$
7	$14^2 \times 112$	OPS	192	Swish	$[1, 4]$
8	$7^2 \times 192$	OPS	320	Swish	1
9	$7^2 \times 320$	1×1 Conv	1280	Swish	1
10	$7^2 \times 1280$	AvgPool	1280	-	1
11	1280	Fc	1000	-	1

OPS	Kernel	Expansion	SE Expansion
$k3\_e3$	3	$[2, 4]$	-
$k3\_e3\_e_{se}1$	3	$[2, 4]$	1
$k5\_e3$	5	$[2, 4]$	-
$k5\_e3\_e_{se}1$	5	$[2, 4]$	1
$k3\_e6$	3	$[4, 8]$	-
$k3\_e6\_e_{se}2$	3	$[4, 8]$	2
$k5\_e6$	5	$[4, 8]$	-
$k5\_e6\_e_{se}2$	5	$[4, 8]$	2

Table 1. Left: Macro architecture of the supernet. "OPS" denotes the operations to be searched. "MBInvRes" is the basic block in [28]. "C_{out}" means the output channels. "Act" denotes the activation function used in a stage. "L" is the number of layers in a stage, where $[a, b]$ is a discrete interval. If necessary, the down-sampling occurs at the first operation of a stage. **Right**: Candidate operations to be searched. "Expansion" defines the width of an operation and $[a, b]$ is a continuous interval. "SE Expansion" determines the width of the SE module.

the candidate operations lies in $[0.25, 0.5]$. $e3$ or $e6$ in the first column of Tab. 1 defines the expansion ratio is 3 or 6 at the beginning of searching, and e can vary in $[2, 4]$ or $[4, 8]$ during searching. Following the same naming schema, MBInvRes at stage 2 has a fixed configuration of $k3\_e1\_e_{se}0.25$. Besides, we also construct a depth-level search space based on a new sink-connecting schema. As shown in Fig. 1(c), during searching, the outputs of all the layers in a stage are connected to a sink point, which is the input to the next stage. After searching, only one connection, i.e. depth, is chosen in each stage.

3.3 Three-Freedom NAS

In this section, we investigate the operation-level, depth-level and width-level search freedoms, respectively, and accordingly make considerable improvements of the search flexibility and stability. Finally, our Three-Freedom NAS is summarized at the end of section.

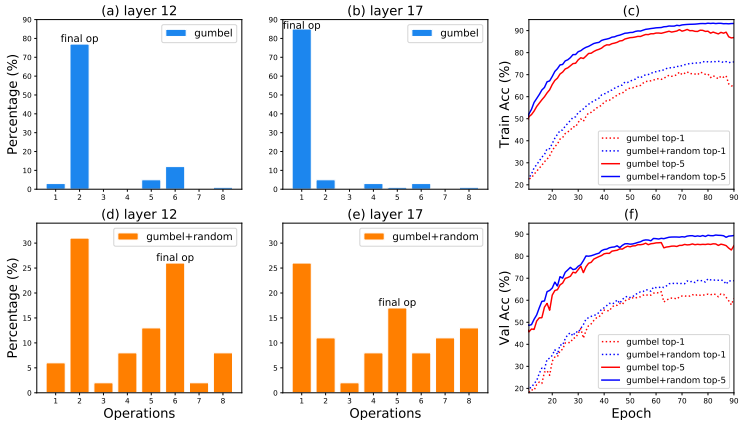

Fig. 2. (a)-(b): The counting percentages of the derived operations during searching by Gumbel Softmax relaxation. (d)-(e): The counting percentages of the derived operations during searching by bi-sampling algorithm. (c): Training accuracy of the supernet. (f): Validating accuracy of the supernet. Zoom in for better view.

Rethinking Operation-level Freedom. As demonstrated in Sec. 3.1, NAS based on Gumbel Softmax relaxation samples one operation per layer during forward pass. It means when optimizing the inner objective Eq. (2), only one path is chosen and updated by gradient descent. However, due to the alternative update between ω and α in the bi-level optimization, one path sampling strategy may focus on some specific operations and update their parameters more frequently than others. Then the architecture distribution parameters of these operations will get better when optimizing Eq. (1). Accordingly, the same operation is more likely to be selected in the next sampling. This phenomenon may cause the search procedure to fall into the specific operations at some layers, leading to suboptimal architectures. We call it operation collapse. Although there is a temperature parameter τ to control the sampling, we find that the operation collapse still occurs in practice. We conduct an experiment based on our search space with the Gumbel Softmax relaxation, where τ linearly decreases from 5.0 to 0.2. The results are shown in Fig. 2(a)-(b), where we count the derived operations for layer 12 and 17 during searching (after each search epoch). It can be observed that almost 80% architecture derivations fall into specific operations in both layer 12 and 17, illustrating the occurrence of operation collapse.

To remedy the operation collapse, a straightforward method is early stopping [20,35]. However, it may lead to suboptimal architectures due to incomplete supernet training and operation exploration (Supp. 5). In this paper, we propose a simple bi-sampling search algorithm, where two independent paths are sampled for each time. In this way, when optimizing Eq. (2), two different paths are chosen and updated in a mini-batch. We implement it by conducting two times forward but one time backward. The second path is used to enhance the competitiveness of other operations against the one operation sampling in

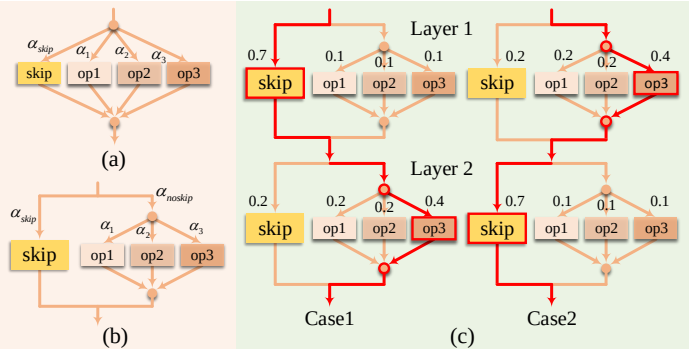

Fig. 3. (a)-(b): The mutual exclusion between skip and other operations. (c): A case study for architecture redundancy.

Gumbel Softmax. In Sec. 4.3, we conduct several experiments to explore various sampling strategies for the second path and find random sampling is the best one. Similarly, we also conduct an experiment based on our bi-sampling search algorithm and present the results in Fig. 2(d)-(e). Compared with Gumbel Softmax based sampling, our bi-sampling strategy is able to explore more operations during searching. Furthermore, as shown in Fig. 2(c) and Fig. 2(f), our bi-sampling strategy is superior to the Gumbel Softmax based sampling in both the supernet accuracy on the training and the validating set.

Rethinking Depth-level Freedom. In order to search for flexible architectures, an important component of differentiable NAS is depth-level search. Previous works [6,33] usually add a skip operation in the candidates and search them together (Fig. 3(a)). In this case, skip has equal importance to other operations and the probability of $op2$ is $P(\alpha_2)$. However, it makes the search unstable, where the derived architecture is relatively shallow and the depth has a large jitter, especially in the early search phase, as shown in orange line in Fig. 4(a). We argue that it is because the skip has higher priority to rule out other operations during searching, since it has no parameter. Therefore, the skip operation should be independent of other candidates, as depicted in Fig. 3(b). We call it as the mutual exclusion between skip and other candidate operations. In this case, skip competes with all the other operations and the probability of $op2$ is $P(\alpha_2, \alpha_{noskip})$. However, directly applying such a scheme will lead to architecture redundancy. Assuming there are two searchable layers in Fig. 3(c). Case 1: we choose skip in layer 1 and $op3$ in layer 2. Case 2: we choose $op3$ in layer 1 and skip in layer 2. Both cases have the same derived architectures $op3$ but quite different architecture distributions. As the number of searchable layers increases, such architecture redundancy will be more serious.

To address the above issue, we introduce a new sink-connecting search space to ensure the mutual exclusion between skip and other candidate operations, as well as eliminate the architecture redundancy. The basic framework is illustrated in Fig. 1(c), where the outputs of all the layers in a stage are connected to a

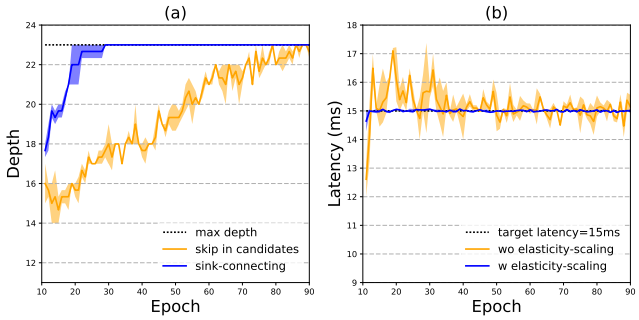

Fig. 4. (a): The searched depth for different depth-level search spaces. (b): The searched latency w/wo elasticity-scaling. All the search procedures are repeated 5 times, and we plot the mean, the maximum and the minimum. Zoom in for better view.

sink point. During searching, the weighted sum of the output feature maps is calculated at the sink point, which is the input to the next stage. When deriving architectures, only one connection, i.e. depth, is chosen in each stage. Obviously, our sink-connecting search space makes the skip operation independent of the other candidates and has no architecture redundancy, because if a layer is skipped, then all the following layers in the same stage are also skipped. Let β_l^s be the architecture distribution parameter of l-th connection in stage s. We employ a Softmax function as the continuous relaxation:

$$x^{s+1} = \sum_{l \in s} v_l^s \cdot x^l, \; v_l^s = \frac{\exp\left(\beta_l^s\right)}{\sum_k \exp\left(\beta_k^s\right)} \tag{5}$$

$$Lat(\alpha, \beta) = \sum_s \sum_{l \in s} v_l^s \cdot Lat(\alpha^l) = \sum_s \sum_{l \in s} \sum_i v_l^s \cdot u_i^l \cdot Lat(op_i^l) \tag{6}$$

Blue line in Fig. 4(a) shows the search on sink-connecting search space. It is obvious that the search procedure is stable and the derived depth converges quickly. We do not sample for depth-level search, because if bi-sampling for β, we must independently sample 2 paths of depth and operation, respectively, leading to 4 times forward, which notably increases GPU memory and search time.

Rethinking Width-level Freedom. Due to the coarse-grained discreteness of search space, current NAS methods cannot satisfy the precise latency constraints. Each searchable layer has a fixed number of channels for the candidate operations, which means each layer has a fixed number of latency options. Furthermore, in each stage, all the layers excluding the first one have the same input and output shapes, so the latency options of these layers are all the same. Although the search space of NAS is huge (e.g. it is 10^{21} for FBNet [33]), the statuses of architectures with different latencies are finite and discrete. Due to the coarse-grained search space for latency, some target latency cannot be precisely satisfied, leading to instability during architecture searching. For example, setting the target latency to be 15ms, we search two architectures: one is 14.32ms

and the other is 15.76ms. Both of them have around 0.7 ms gaps for the target latency. More analyses are presented in Sec. 4.5.

In order to refine the coarse-grained search space for latency, previous works [12,33] introduce a global scaling factor or add additional candidate operations for width-level search. However, these methods are not flexible. Inspired by MorphNet [13], we propose an elasticity-scaling approach that adaptively shrinks and expands the model width to precisely satisfy the latency constraint in a progressively fine-grained manner. Our approach does not increase additional GPU memory and is insensitive to hyperparameter settings.

Given a supernet, we derive a discrete seed network (sn) based on the current architecture distribution parameters, where the strongest operation in each layer and the strongest depth in each stage are chosen. We can multiply sn by a scaling factor γ to control the width. Let $\gamma \cdot sn_{i:j}$ be a network whose layer width from stage i to stage j is multiplied by γ. Our elasticity-scaling strategy is presented in Algorithm 1, including a global scaling ($i = 3$) and a series of progressively fine-grained scaling ($i = 4 \dots 8$). Note that the searchable stages are from stage 3 to stage 8 in our search space. More implementation details can be found in Supp. 3. In Fig. 4 (b), we observe that our elasticity-scaling strategy is effective in stabilizing the architecture search with the precise latency constraint.

Algorithm 1 Elasticity-scaling Strategy

1: Derive a seed network sn from the supernet A.
2: **for** $i = 3, \dots, 8$ **do**
3: Find the largest γ such that $LAT(\gamma \cdot sn_{i:8}) \leq lat_{\text{target}}$.
4: Set $sn = \gamma \cdot sn_{i:8}$.
5: **end for**
6: Put sn back to the supernet A.
7: **return** A;

Overall Algorithm. Our Three-Freedom NAS (TF-NAS) contains all above components: the bi-sampling search algorithm, the sink-connecting search space and the elasticity-scaling strategy. It finds latency-constrained architectures from the supernet (Tab. 1) by solving the following bi-level problem:

$$\min_{\alpha,\beta} L_{\text{val}}\left(\omega^*, \alpha, \beta\right) + \lambda C\left(LAT(\alpha, \beta)\right) \tag{7}$$

$$\text{s.t. } \omega^* = \arg\min_{\omega} L_{\text{t\_g}}\left(\omega, \alpha, \beta\right) + L_{\text{t\_r}}\left(\omega, \alpha, \beta\right) \tag{8}$$

where $L_{\text{t\_g}}$ and $L_{\text{t\_r}}$ denote the training losses for Gumbel Softmax based sampling and random sampling, respectively. The latency-constrained objectives in [33,4] do not employ the target latency, leading to imprecise latency compared with the target one. Therefore, we introduce a new objective that explicitly contains the target latency lat_{target}:

$$C\left(LAT(\alpha, \beta)\right) = \max\left(\frac{LAT(\alpha, \beta)}{lat_{\text{target}}} - 1, 0\right) \tag{9}$$

The continuous relaxations of α and β are based on Eq. (3)-(4) and Eq. (5)-(6), respectively. We employ elasticity-scaling after each searching epoch, making it barely increase the search time. After searching, the best architecture is derived from the supernet based on α and β, where the strongest operation in each layer and the strongest depth in each stage are chosen.

4 Experiments

4.1 Dataset and Settings

All the experiments are conducted on ImageNet [9] under the mobile setting. Similar with [3], the latency is measured with a batch size of 32 on a Titan RTX GPU. We set the number of threads for OpenMP to 1 and use Pytorch1.1+cuDNN7.6.0 to measure the latency. Before searching, we pre-build a latency look up table as described in [3,33]. To reduce the search time, we choose 100 classes from the original 1000 classes to train our supernet. The supernet is trained for 90 epochs, where the first 10 epochs do not update the architecture distribution parameters. This procedure takes about 1.8 days on 1 Titan RTX GPU. After searching, the derived architecture is trained from scratch on the whole ImageNet training set. For fair comparison, we train it for 250 epochs with standard data augmentation [4], in which no auto-augmentation or mixup is used. More experimental details are provided in Supp. 2.

4.2 Comparisons with Current SOTA

We compare TF-NAS with various manually designed and automatically searched architectures. According to the latency, we divide them into four groups. For each group, we set a target latency and search an architecture. Totally, there are four latency settings, including 18ms, 15ms, 12ms and 10ms, and the final architectures are named as TF-NAS-A, TF-NAS-B, TF-NAS-C and TF-NAS-D, respectively. The comparisons are presented in Tab. 2. There is a slight latency error for each model. As shown in [4], the error mainly comes from the slight difference between the pre-built lookup table and the actual inference latency.

As shown in Tab. 2, our TF-NAS-A achieves 76.9% top-1 accuracy, which is better than NASNet-A [40] (+2.9%), PC-DARTS [36] (+1.1%), MixNet-S [32] (+1.1%) and EfficientNet-B0 [31] (+0.6%). For the GPU latency, TF-NAS-A is 6.2ms, 2.15ms, 1.83ms and 1.23ms better than NASNet-A, MdeNAS, PC-DARTS, MixNet-S and EfficientNet-B0, respectively. In the second group, our TF-NAS-B obtains 76.3% top-1 accuracy with 15.06ms. It exceeds the micro search methods (DARTS [22], DGAS [11], SETN [10], CARS-I [37]) by an average of 2.1%, and the macro search methods (SCARLET-C [6], DenseNAS-Large [12]) by an average of 0.5%. For the 12ms latency group, our TF-NAS-C is superior to ShuffleNetV1 2.0x [38], AtomNAS-A [24], FBNet-C [33] and ProxylessNAS (GPU) [4] both in accuracy and latency. Besides, it is comparable with MobileNetV3 [16] and MnasNet-A1 [30]. Note that MnasNet-A1 is trained for

more epochs than our TF-NAS-C (350 vs 250). Obviously, training longer makes an architecture generalize better [15]. In the last group, our TF-NAS-D achieve 74.2% top-1 accuracy, outperforming MobileNetV1 [17] (+3.6%), ShuffleNetV1 1.5x [38] (+2.6%) and FPNASNet [7] (+0.9%) by large margins.

Further to investigate the impact of the SE module, we remove SE from our candidate operations and search new architectures based on the four latency settings. The result architectures are marked as TF-NAS-A-wose, TF-NAS-B-wose, TF-NAS-C-wose and TF-NAS-D-wose. As shown in Tab. 2, they obtain 76.5%, 76.0%, 75.0% and 74.0% top-1 accuracy, respectively, which are competitive with or even superior to the previous state-of-the-arts. Due to the page limitation, more results are presented in our supplementary materials.

Architecture	Top-1 Acc(%)	GPU Latency	FLOPs (M)	Training Epochs	Search Time (GPU days)	Venue
NASNet-A [40]	74.0	24.23ms	564	-	2,000	CVPR'18
PC-DARTS [36]	75.8	20.18ms	597	250	3.8	ICLR'20
MixNet-S [32]	75.8	19.86ms	256	-	-	BMVC'19
EfficientNet-B0 [31]	76.3	19.26ms	390	350	-	ICML'19
TF-NAS-A-wose (Ours)	76.5	18.07ms	504	250	1.8	-
TF-NAS-A (Ours)	**76.9**	**18.03ms**	457	250	1.8	-
DARTS [22]	73.3	17.53ms	574	250	4	ICLR'19
DGAS [11]	74.0	17.23ms	581	250	0.21	CVPR'19
SETN [10]	74.3	17.42ms	600	250	1.8	ICCV'19
MobileNetV2 1.4x [28]	74.7	16.18ms	585	-	-	CVPR'18
CARS-I [37]	75.2	17.80ms	591	250	0.4	CVPR'20
SCARLET-C [6]	75.6	15.09ms	280	-	12	ArXiv'19
DenseNAS-Large [12]	76.1	15.71ms	479	240	2.67	CVPR'20
TF-NAS-B-wose (Ours)	76.0	15.09ms	433	250	1.8	-
TF-NAS-B (Ours)	**76.3**	**15.06ms**	361	250	1.8	-
ShuffleNetV1 2.0x [38]	74.1	14.82ms	524	240	-	CVPR'18
AtomNAS-A [24]	74.6	12.21ms	258	350	-	ICLR'20
FBNet-C [33]	74.9	12.86ms	375	360	9	CVPR'19
ProxylessNAS (GPU) [4]	75.1	12.02ms	465	300	8.3	ICLR'18
MobileNetV3 [16]	75.2	12.36ms	219	-	-	ICCV'19
MnasNet-A1 [30]	75.2	11.98ms	312	350	288	CVPR'18
TF-NAS-C-wose (Ours)	75.0	12.06ms	315	250	1.8	-
TF-NAS-C (Ours)	**75.2**	**11.95ms**	284	250	1.8	-
MobileNetV1 [17]	70.6	**9.73ms**	569	-	-	ArXiv'17
ShuffleNetV1 1.5x [38]	71.6	10.84ms	292	240	-	CVPR'18
FPNASNet [7]	73.3	11.60ms	300	-	0.83	ICCV'19
TF-NAS-D-wose (Ours)	74.0	10.10ms	286	250	1.8	-
TF-NAS-D (Ours)	**74.2**	10.08ms	219	250	1.8	-

Table 2. Comparisons with state-of-the-art architectures on the ImageNet classification task. For the competitors, we directly cite the FLOPs, the training epochs, the search time and the top-1 accuracy from their original papers or official codes. For the GPU latency, we measure it with a batch size of 32 on a Titan RTX GPU.

4.3 Analyses of Bi-sampling Search Algorithm

As described in Sec. 3.3, our bi-sampling algorithm samples two paths in the forward pass. One path is based on Gumbel Softmax trick and the other is selected from the remaining paths. In this subsection, we set the target latency to 15ms and employ four types of sampling methods for the second path, including the Gumbel Softmax (Gumbel), the minimum architecture distribution parameter ($\min \alpha^l$), the maximum architecture distribution parameter ($\max \alpha^l$) and the random sampling (Random). As shown in Tab. 3, compared with other methods, random sampling achieves the best top-1 accuracy. As a consequence, we employ random sampling in our bi-sampling search algorithm. Another interesting observation is that *Gumbel+Gumbel* and *Gumbel+*$\max \alpha^l$ are inferior to one path *Gumbel* sampling strategy. This is due to the fact that both *Gumbel+Gumbel* and *Gumbel+*$\max \alpha^l$ will exacerbate the operation collapse phenomenon, leading to inferior architectures. Compared with one path *Gumbel* sampling, our bi-sampling algorithm increases the search time by 0.3 GPU day, but makes a significant improvement in top-1 accuracy (76.3% vs 75.8%).

Sampling	Top-1 Acc(%)	GPU Latency	FLOPs(M)	Search Time
Gumbel	75.8	15.05ms	374	1.5 days
Gumbel+Gumbel	75.7	15.04ms	371	1.8 days
Gumbel+$\min \alpha^l$	76.0	15.11ms	368	1.8 days
Gumbel+$\max \alpha^l$	75.5	14.92ms	354	1.8 days
Gumbel+Random	76.3	15.06ms	361	1.8 days

Table 3. Comparisons with different sampling methods for the second path in bi-sampling search algorithm.

4.4 Analyses of Sink-connecting Search Space

As mentioned in Sec. 3.3, skip operation has a special role in depth-level search. Ensuring the mutual exclusion between skip and other candidate operations, as well as eliminating the architecture redundancy are important stability factors for the architecture search procedure. In this subsection, we set the target latency to 15ms and compare our sink-connecting search space with the other two depth-level search spaces. The results are presented in Tab. 4, where "skip in candidates" means adding the skip operation in the candidates (Fig. 3(a)), and "skip out candidates" denotes putting the skip operation independent of the candidates (Fig. 3(b)). Obviously, our "sink-connecting" achieves the best top-1 accuracy, demonstrating its effectiveness in finding accurate architectures during searching. The "skip out candidates" beats the "skip in candidates" by about 0.5% top-1 accuracy, and the "sink-connecting" is 0.2% higher than the "skip out candidates". The former achieves more improvement than the later, indicating that the mutual exclusion between skip and other operations is more important than the architecture redundancy.

Method	Mutual Exclusion	Architecture Redundancy	Top-1 Acc(%)	GPU Latency	FLOPs(M)
skip in candidates	×	×	75.6	15.10ms	384
skip out candidates	√	×	76.1	15.07ms	376
sink-connecting	√	√	76.3	15.06ms	361

Table 4. Comparisons with different depth-level search spaces.

4.5 Analyses of Elasticity-scaling Strategy

The key to search latency-constrained architectures is the differentiable latency objective $C\left(LAT(\alpha, \beta)\right)$ in Eq. (7). Previous methods [33,4] employ diverse latency objectives with one or two hyperparameters. We list them in Tab. 5 and name them as C1 and C2, respectively. By tuning the hyperparameters, both C1 and C2 can be trade-off between the accuracy and the latency. We set the target latency to 15ms and directly employ C1 and C2 (without elasticity-scaling strategy) to search architectures. We try our best to fine-tune the hyperparameters in C1 and C2, so that the searched architectures conform to the latency constraint as much as possible. The search procedure is repeated 5 times for each latency objective, and we plot the average latencies of the derived architectures during searching (orange lines in Fig. 5(a)-(b)). It is obvious that both C1 and C2 cannot reach the target latency before the first 50 epochs. After that, the architecture searched by C1 fluctuates down and up around the target latency, but the architecture searched by C2 always exceeds the target latency. We also plot the results of our proposed latency objective Eq. (9) (orange line in Fig. 4(b)) and find it is more precise than C1 and C2 after the first 30 epochs. The reason is that the target latency term is explicitly employed in our latency objective.

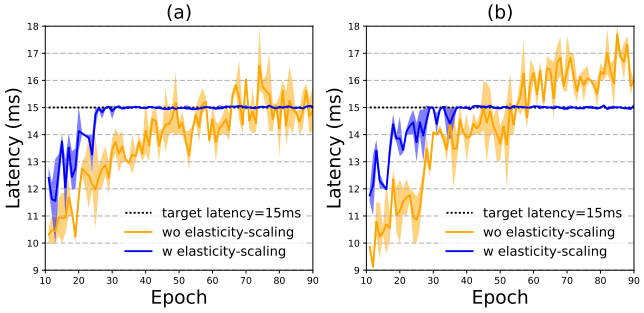

Fig. 5. (a): The searched latency by C1 w/wo elasticity-scaling. (b): The searched latency by C2 w/wo elasticity-scaling. All the search procedures are repeated 5 times, and we plot the mean, the maximum and the minimum. Zoom in for better view.

The proposed elasticity-scaling strategy is the vital component in our TF-NAS to ensure the searched architectures precisely satisfy the target latency.

By employing it, all the objectives are able to quickly search latency-satisfied architectures (blue lines in Fig. 4(b), Fig. 5(a) and Fig. 5(b)), demonstrating the effectiveness and the versatility of our elasticity-scaling strategy. Furthermore, we also evaluate the searched architectures based on C1, C2 and our proposed objective with and without elasticity-scaling. As shown in Tab. 5, our method achieves the best top-1 accuracy at 15ms latency constraint, which is slightly superior to C2 and beats C1 by a large margin no matter with or without elasticity-scaling. Therefore, explicitly introducing the target latency into the latency-constrained objective not only stabilizes large latency changes but also facilitates more accurate architecture discovery. Another observation is that under the similar backbone, the searched architectures with less/greater latencies than the target usually obtain lower/higher top-1 accuracies, especially when the latency gap is large. For example, C1 with elasticity-scaling achieves 75.9% top-1/15.05ms, which beats its counterpart without elasticity-scaling (75.6% top-1/14.32ms) by 0.3% top-1 accuracy and the latency gap is approximate 0.7ms.

Name	Formulation	Elasticity-scaling	Top-1 Acc(%)	GPU Latency
C1 [33]	$\lambda_1 \log\left[(LAT(\alpha,\beta))\right]^{\lambda_2}$	×	75.6	14.32ms
		√	75.9	15.05ms
C2 [4]	$\lambda_1 \left(LAT(\alpha,\beta)\right)$	×	76.2	15.76ms
		√	76.1	15.08ms
Ours	$\lambda_1 \max\left(\frac{LAT(\alpha,\beta)}{lat_{\text{target}}} - 1, 0\right)$	×	76.3	15.28ms
		√	76.3	15.06ms

Table 5. Comparisons with different latency objectives w/wo elasticity-scaling.

5 Conclusion

In this paper, we have proposed Three-Freedom NAS (TF-NAS) to seek an architecture with good accuracy as well as precise latency on the target devices. For operation-level, the proposed bi-sample search algorithm moderates the operation collapse in Gumbel Softmax relaxation. For depth-level, a novel sink-connecting search space is defined to address the mutual exclusion between skip operation and other candidate operations, as well as architecture redundancy. For width-level, an elasticity-scaling strategy progressively shrinks or expands the width of operations, contributing to precise latency constraint in a fine-grained manner. Benefiting from investigating the three freedoms of differentiable NAS, our TF-NAS achieves state-of-the-art performance on ImageNet classification task. Particularly, the searched TF-NAS-A achieves 76.9% top-1 accuracy with less latency and training epochs.

Acknowledgement This work is partially funded by Beijing Natural Science Foundation (Grant No. JQ18017) and Youth Innovation Promotion Association CAS (Grant No. Y201929).

References

1. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. In: ICLR (2017)
2. Bender, G., Kindermans, P., Zoph, B., Vasudevan, V., Le, Q.V.: Understanding and simplifying one-shot architecture search. In: ICML (2018)
3. Cai, H., Gan, C., Han, S.: Once for all: Train one network and specialize it for efficient deployment. In: NeurIPS (2019)
4. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. In: ICLR (2019)
5. Chen, X., Xie, L., Wu, J., Tian, Q.: Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In: ICCV (2019)
6. Chu, X., Zhang, B., Li, J., Li, Q., Xu, R.: Scarletnas: Bridging the gap between scalability and fairness in neural architecture search. arXiv (2019)
7. Cui, J., Chen, P., Li, R., Liu, S., Shen, X., Jia, J.: Fast and practical neural architecture search. In: ICCV (2019)
8. Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., Vajda, P., Uyttendaele, M., Jha, N.K.: Chamnet: Towards efficient network design through platform-aware model adaptation. In: CVPR (2019)
9. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Li, F.: Imagenet: A large-scale hierarchical image database. In: CVPR (2009)
10. Dong, X., Yang, Y.: One-shot neural architecture search via self-evaluated template network. In: ICCV (2019)
11. Dong, X., Yang, Y.: Searching for a robust neural architecture in four GPU hours. In: CVPR (2019)
12. Fang, J., Sun, Y., Zhang, Q., Li, Y., Liu, W., Wang, X.: Densely connected search space for more flexible neural architecture search. In: CVPR (2020)
13. Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T., Choi, E.: Morphnet: Fast & simple resource-constrained structure learning of deep networks. In: CVPR (2018)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
15. Hoffer, E., Hubara, I., Soudry, D.: Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In: NeurIPS (2017)
16. Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H.: Searching for mobilenetv3. In: ICCV (2019)
17. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv (2017)
18. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: CVPR (2018)
19. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: CVPR (2017)
20. Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., Li, Z.: DARTS+: improved differentiable architecture search with early stopping. arXiv (2019)
21. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A.L., Huang, J., Murphy, K.: Progressive neural architecture search. In: ECCV (2018)
22. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: ICLR (2019)

23. Ma, N., Zhang, X., Zheng, H., Sun, J.: Shufflenet V2: practical guidelines for efficient CNN architecture design. In: ECCV (2018)
24. Mei, J., Li, Y., Lian, X., Jin, X., Yang, L., Yuille, A.L., Yang, J.: Atomnas: Fine-grained end-to-end neural architecture search. In: ICLR (2020)
25. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: ICML (2018)
26. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: AAAI (2019)
27. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: ICML (2017)
28. Sandler, M., Howard, A.G., Zhu, M., Zhmoginov, A., Chen, L.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: CVPR (2018)
29. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: CVPR (2016)
30. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: CVPR (2019)
31. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: ICML (2019)
32. Tan, M., Le, Q.V.: Mixconv: Mixed depthwise convolutional kernels. In: BMVC (2019)
33. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: CVPR (2019)
34. Xie, S., Zheng, H., Liu, C., Lin, L.: SNAS: stochastic neural architecture search. In: ICLR (2019)
35. Xiong, Y., Mehta, R., Singh, V.: Resource constrained neural network architecture search: Will a submodularity assumption help? In: ICCV (2019)
36. Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G., Tian, Q., Xiong, H.: PC-DARTS: partial channel connections for memory-efficient differentiable architecture search. In: ICLR (2020)
37. Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., Xu, C.: Cars: Continuous evolution for efficient neural architecture search. In: CVPR (2020)
38. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: CVPR (2018)
39. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: ICLR (2017)
40. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: CVPR (2018)