MIXPATH: A UNIFIED APPROACH FOR ONE-SHOT NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Blending multiple convolutional kernels is proved advantageous in neural architecture design. However, current two-stage neural architecture search methods are mainly limited to single-path search spaces. How to efficiently search models of multi-path structures remains a difficult problem. In this paper, we are motivated to train a one-shot multi-path supernet to accurately evaluate the candidate architectures. Specifically, we discover that in the studied search spaces, feature vectors summed from multiple paths are nearly multiples of those from a single path. Such disparity perturbs the supernet training and its ranking ability. Therefore, we propose a novel mechanism called *Shadow Batch Normalization* (SBN) to regularize the disparate feature statistics. Extensive experiments prove that SBNs are capable of stabilizing the optimization and improving ranking performance. We call our unified multi-path one-shot approach as MixPath, which generates a series of models that achieve state-of-the-art results on ImageNet.

1 INTRODUCTION

Complete automation in neural network design is one of the most important research directions of automated machine learning (Tan et al., 2019; Liu et al., 2019). Among various mainstream paradigms, one-shot approaches (Guo et al., 2020; Chu et al., 2021; Li et al., 2019) make use of a weight-sharing mechanism that reduces a large amount of computational cost, but these approaches mainly focus on searching for single-path networks. Observing that a multi-path structure is beneficial for model performance as in Inception (Szegedy et al., 2015) and ResNeXT (Xie et al., 2017), it is necessary to incorporate multi-path structure into the search space. Noticeably, exploring multi-path search space is made possible in one-stage approaches like (Chu et al., 2020) and (Pham et al., 2018). However, it poses a challenge for two-stage methods to train a one-shot supernet that can accurately predict the performance of its multi-path submodels.

Although FairNAS (Chu et al., 2021) largely alleviates the ranking difficulty in the single-path case with a fairness strategy, it is inherently difficult to apply the same method in a multi-path scenario. It should be emphasized that the most critical point of the two-stage NAS is that the supernet can rank submodels as accurately as possible. However, we find that a vanilla training of a multi-path supernet (e.g. randomly pick a multi-path submodel to train at each step) is not stable to provide a confident ranking because of changing statistics during the sampling process. Therefore, we dive into its real causes and undertake a unified approach to tackle this issue. Our contributions can be summarized as follows.

- We propose a unified approach for multi-path (say at most m paths are allowed) one-shot NAS, as opposed to the current single-path methods. From this perspective, current one-shot methods (Guo et al., 2020; Chu et al., 2021) become one of our special cases when $m = 1$.
- We disclose why a vanilla multi-path training fails, and we propose a novel yet lightweight mechanism, called *shadow batch normalization* (SBN, see Fig. 1), to stabilize the supernet with a neglectable cost. By exploiting feature similarity from different paths, we further reduce the number of SBNs to be m, otherwise it would be exponential.
- We reveal that our multi-path supernet is trained with stability due to SBNs. It also comes with boosted ranking performance. Together with post BN calibration, it obtains a high Kendall tau (0.597) on a subset of NAS-Bench-101 (Ying et al., 2019).

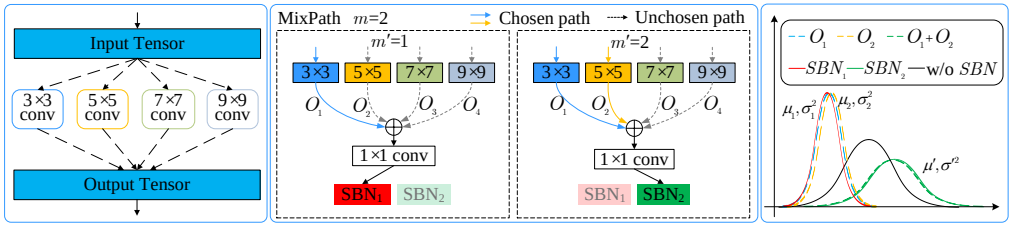


Figure 1: **Left**: Options in a demo block, where at most m paths can be chosen. **Middle**: An example of MixPath supernet training with Shadow Batch Normalizations (SBNs). Note 1x1 Conv is not a must. SBNs are standard BNs, except that SBNs are applied w.r.t the number of activated paths. E.g., SBN_1 is used whenever only $m' = 1$ path is activated, SBN_2 for $m' = 2$ paths. **Right**: SBNs catch each of the statistics in two modes (red for $m' = 1$, green for $m' = 2$), however, a single BN (black) can't capture both.

- We search proxylessly on ImageNet at a cost of **10 GPU days**. The searched models obtain state-of-the-art results on ImageNet, which are comparable with MixNet (Tan & Le., 2019) models searched with 200× more computing powers. Moreover, our model MixPath-B makes use of multi-branch feature aggregation and reaches higher accuracy than EfficientNet-B0 with fewer FLOPS and parameters.

2 Related Work

Conditional batch normalization Batch Normalization (BN) (Ioffe & Szegedy, 2015) has greatly facilitated the training of neural networks by normalizing layer inputs to have fixed means and variances. Slimmable neural networks(Yu et al., 2019) introduces a shared supernet that can run with switchable channels at four different scales (1×, 0.75×, 0.5×, 0.25×). Due to feature inconsistency, independent BNs are applied for each switch configuration to encode conditional statistics. However, it is impractical because it requires an increased number of BNs when it has arbitrary channel widths.

Model ranking correlation It should be emphasized that the ranking ability for the family of two-stage algorithms is of the uttermost importance (Bender et al., 2018), whose sole purpose is to evaluate networks. To quantitatively analyze their ranking ability, previous works like (Yu et al., 2020b; Chu et al., 2021; Li et al., 2020; Zheng et al., 2019) have applied a Kendall tau measure (Kendall, 1938). To this end, various proxies (Tan et al., 2019; Zoph & Le, 2017; Zoph et al., 2018), explicit or implicit performance predictors (Luo et al., 2018; Liu et al., 2018) are developed to avoid the intractable evaluation. Recent one-shot approaches (Bender et al., 2018; Guo et al., 2020; Chu et al., 2021) utilize a supernet where each submodel can be rapidly assessed with inherited weights.

3 MixPath: A Unified Approach

3.1 Motivation

There is a call for a multi-path one-shot approach. Informally, existing weight-sharing approaches (Liu et al., 2019; Chu et al., 2020; Guo et al., 2020; Chu et al., 2021) can be classified into four categories based on two dimensions: stage levels and multi-path support, as shown in Fig. 2. Specifically, DARTS (Liu et al., 2019) and Fair DARTS (Chu et al., 2020) both are one-stage methods while the latter allows multiple paths between any two nodes. One-shot methods (Guo et al., 2020; Chu et al., 2021) typically train a supernet in the first stage and evaluate submodels in the second stage for model selection. So far, the two-stage one-shot methods mainly consider a single-path search space. It is thus natural to devise their multi-path counterpart.

Moreover, it's reasonable to design such a search space regarding two factors. **First**, multi-path feature aggregation is proven to be useful, such as Inception series (Szegedy et al., 2015; Ioffe & Szegedy, 2015; Szegedy et al., 2016; 2017) and ResNeXt (Xie et al., 2017). **Second**, it has the potential to balance the trade-off between the performance and cost.

Without loss of generality, say an inverted bottleneck has an input feature of $C_{in} \times H \times W$, whose number of intermediate and output channels are C_{mid} and C_{out} (same as C_{in}) respectively. Its computational cost can be formulated as $c_{total} = 2HWC_{in}C_{mid} + k^2HWC_{mid} = 2HWC_{mid}(C_{in} + \frac{k^2}{2})$, where k is the kernel size of the depthwise convolution. Usually the value of k is set to 3 or 5. Since C_{in} typically dominates $\frac{k^2}{2}$, we can boost the representative power of depthwise transformation by mixing more kernels with neglectable increased cost. This design can be regarded as a straightforward combination of MixConv (Tan & Le., 2019) and ResNeXt (Xie et al., 2017).

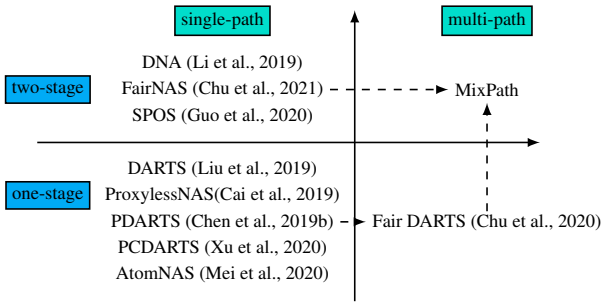


Figure 2: A brief taxonomy of weight-sharing NAS. MixPath fills the empty space of two-stage multiple-path methods.

Multi-path supernet is very hard to train. Vanilla training of the multi-path supernet suffers from severe training difficulties. We can simply train a multi-path supernet by randomly activating a multi-path model at a single step. Here we randomly activate or deactivate each operation. However, this training process is very unstable according to our pilot experiments conducted with the MixPath supernet on ImageNet (Deng et al., 2009), shown by the blue line in Fig. 3 (a).

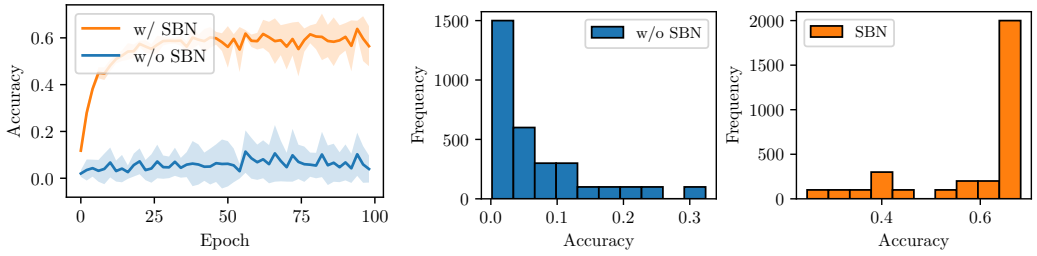
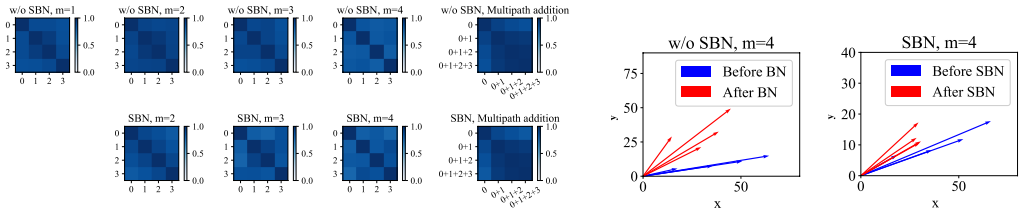


Figure 3: **(a)** Average one-shot validation accuracies (solid line) and their variances (shade) when training the MixPath supernet in S_2 (activating at most $m = 2$ paths in each MixPath block) on ImageNet. Twenty models are randomly sampled every two epochs. **(b)** Histogram of accuracies of randomly sampled 3k one-shot models. One-shot models generally come with good performance when SBNs are enabled during supernet training

What might have caused such a problem? It has been pointed out that high cosine similarity of features from different operations is important for training stability (Chu et al., 2021). To verify the issue, we draw the cosine similarity matrix of different features from our trained supernet in the top row of Fig. 4 (a). Surprisingly, not only are the features from single-path similar to each other, but also the multi-path features where they get added up. We discover that cosine similarity is limited because it measures the orientation only. The addition of high-dimensional feature vectors does not change their angles too much (as they are similar), but their magnitudes get scaled up, see blue arrows in Fig.4 (b). We can further make a bold postulation that *a simple superposition of multiple vectors renders very dynamic statistics, hence it causes training instability.* With this motivation in mind, we next scale the feature vectors down to the same magnitude to verify if we can make it stabilized.

3.2 Regularizing Multi-path Supernet

First of all, we can think of using a large amount of BNs to track the changing statistics. For a thorough regularization, we adopt multiple standard BNs to regularize feature statistics from different combinations of m paths. We call them *shadow batch normalizations* (SBNs) since they follow



(a) Similarity of multi-path feature vectors (b) Feature vectors projected to 2d space

Figure 4: **(a)** Features from various path combinations are quite alike (w/ and w/o SBNs). Shown here is the cosine similarity map of first-block feature vectors from the supernets (in S_1). Darker means higher similarity. **(b)** SBNs regularize the magnitudes of feature vectors (projected into 2-dimensional space here) while vanilla BN cannot. Blue arrows represent the feature vectors obtained before SBNs or a single BN, and red ones after them. Here SBNs transform the magnitudes (19.6, 37.2, 53.7, 68.7) to around 32. BN normalized magnitudes (32.6, 36.6, 50.6, 66.7) are still diverse.

the activated paths combinations as shadows. Strictly speaking, in order to regularize all feature combinations in a multi-path setting, the number of SBNs has to grow exponentially.[1]

3.2.1 Reduce the complexity from 2^m to m.

The above regularization is too costly. As (Chu et al., 2021) indicates for the single path case, the outputs of different operations at the same layer have higher similarities. It motivates us to reduce the number of shadow BN by diving into the underlying mechanisms further. Informally, if the features of every single path are similar (both magnitude and angles), we can derive that their combinations have similar statistics too, then the number of BNs can be reduced to m. Formally, this can be formulated as follows.

Definition 1. ***Zero-order condition****: Let two high-dimensional functions* $\mathbf{y} = \mathbf{f}(\mathbf{x})$ *and* $\mathbf{z} = \mathbf{g}(\mathbf{x})$ *represent two candidate operations, we say that* $\mathbf{y}$ *and* $\mathbf{z}$ *satisfy the zero-order condition if* $\mathbf{y} \approx \mathbf{z}$ *for any valid input* $\mathbf{x}$*.*

Lemma 3.1. *Let* $\mathbf{m}$ *be the maximum number of the activable paths, and each pair of operations satisfy the zero-order condition, there will be* $\mathbf{m}$ *kinds of expectation and variance for all combinations.*

The proof is provided in A.1 (appendix). It is pretty straightforward as the expectations and variances are simply scaled up according to the number of additions. For instance, $\mathbb{E}[\mathbf{y}+\mathbf{z}] \approx \mathbb{E}[\mathbf{u}+\mathbf{v}] \approx 2\mathbb{E}[\mathbf{y}]$. Lemma 3.1 assures us that we can take $SBN_i (i = 1, 2, \cdots, m)$ to track the combination that contains i paths. For the multi-path setting, we can observe similar results in Figure 4, which means the requirement of Definition 1 is also satisfied.

Statistical analysis of Shadow Batch Normalization. We look into the statistics from the supernet trained on CIFAR-10[2] in search space S_1 (elaborated later in Section C.2) (appendix). Without loss of generality, we set $m = 2$ and collect statistics of the four parameters of SBNs across all channels for the first choice block in Fig. 5. Note SBN_1 captures the statistics of one path, so does SBN_2 for two paths. Based on the theoretical analysis of Section 3.2, we should have $\mu_{bn_1} \approx 0.5\mu_{bn_2}$ and $\sigma^2_{bn_1} \approx 0.25\sigma^2_{bn_2}$. This is consistently observed in the first two columns of Fig. 5. It's also interesting to see that the other two learnable parameters β and γ are quite similar for SBN_1 and SBN_2. This is explainable as SBNs have transformed different statistics to a similar distribution, the parameters of β and γ become very close. Similar results are observed when $m = 3, 4$ and in different layers in Fig. 9 and Fig. 10 in C.3 (appendix).

3.3 MixPath Neural Architecture Search

Based on the above analysis, we train our multi-path supernet with *shadow batch normalizations*, which we call the MixPath supernet. Following One-Shot (Bender et al., 2018), the supernet is used

[1] $C^1_m + C^2_m + \cdots + C^m_m = 2^m - 1$

[2] It's cheaper to train the supernet on CIFAR-10, and similar observations are also reproducible on ImageNet.

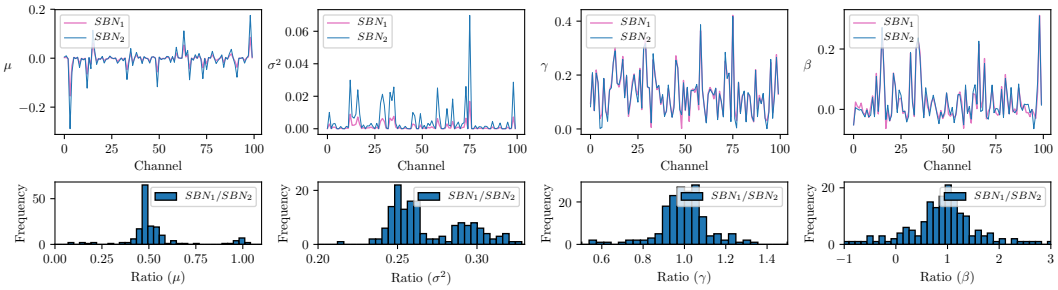


Figure 5: **Top**: Parameters $(\mu, \sigma^2, \gamma, \beta)$ of the first-layer SBNs in MixPath supernet trained on CIFAR-10 when at most $m = 2$ paths are activated. Specifically, SBN_1 and SBN_2 represent SBNs for a single path and two paths. **Bottom**: Histogram of ratios (SBN_1/SBN_2) for each BN parameter. They center around (0.5, 0.25, 1, 1). Note the mean of SBN_2 is roughly twice as that of SBN_1. It's interesting that we have previously forecasted this result.

to evaluate models in the predefined search space, which makes our approach a two-stage pipeline: 1) training supernet with SBNs and 2) searching for competitive models. Specifically, in the training stage, we apply random sampling so that multiple paths can be activated at each step, where they add up to a mixed output. We then attach a corresponding SBN to the current path combination. This process is detailed in Algorithm 1. Next, similar to (Chu et al., 2020), we progress with a well-known evolutionary algorithm NSGA-II (Deb et al., 2002) for searching. In particular, our objectives are to maximize classification accuracy while minimizing the FLOPS. This is detailed in Algorithm 2 (appendix). In addition, batch normalization calibration (Bender et al., 2018; Guo et al., 2020; Mei et al., 2020), as an orthogonal post-processing trick, will be verified effective to further improve the ranking ability.

Algorithm 1 : Stage 1 Supernet training with SBN.

Input: search space $S_{(n,L)}$ with n choice paths per layer and L layers in total, maximum optional paths m, supernet parameters $\Theta(n, L)$, training epochs N, training data loader D, loss function $Loss$.
Initialize every $\theta_{j,l}$ in $\Theta_{(m,L)}$, SBN set $\{SBN_1, SBN_2, \cdots, SBN_n\}$ in each layer.
for $i = 1$ **to** N **do**
  for $data, labels$ **in** D **do**
    for $l = 1$ **to** L **do**
      Randomly sample a number m' between 1 and m, then randomly sample m' paths without replacement, where each has output $\mathbf{o}_i^l$, and use $SBN_{m'}$ to act on the sum of outputs: $\mathbf{o}^l = SBN_{m'}\left(\sum_i \mathbf{o}_i^l\right)$
    end for
    Select $model$ from above sampled index.
    Clear gradients recorder for all parameters.
    Calculate gradients for $model$ based on $Loss$, $data$, $labels$, update $\theta_{(m,L)}$ by gradients.
  end for
end for

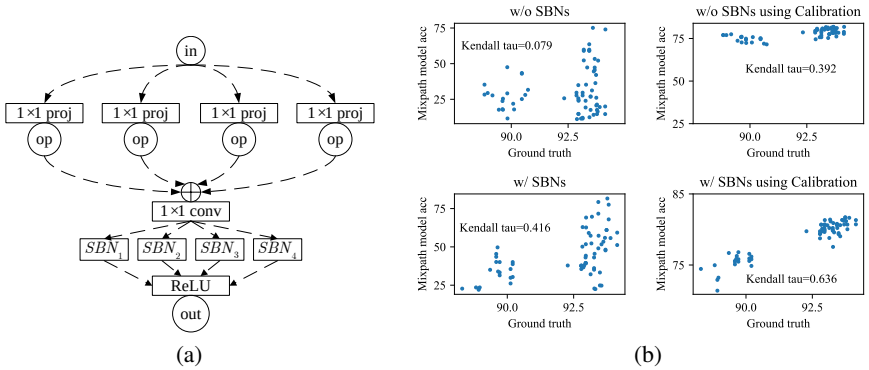
4 Experiments

4.1 Search spaces

We investigate four search spaces in this paper, from S_1 to S_4. For searching on CIFAR-10, we use S_1 that contains 12 inverted bottleneck blocks. For ImageNet, we search in S_2 and S_3 that are similar to MnasNet (Tan et al., 2019) and MixNet (Tan & Le., 2019). For Kendall Tau analysis, we devise a subset of a common benchmark NAS-Bench-101 (Yu et al., 2020b), denoted as S_4. The details of these search spaces are listed in Table 6 (appendix).

4.2 RANKING ABILITY ANALYSIS ON NAS-BENCH-101

To prove that SBNs can stabilize the supernet training and improve its ranking, we score our method on a subset of a common benchmark NAS-Bench-101 (Yu et al., 2020b). Each model is stacked by 9 cells, and every cell has at most 5 internal nodes. We make some adaptations to our method. The first 4 nodes are used to make candidate paths, each has 3 possible operations: 1×1 Conv, 3×3 Conv, and 3×3 Maxpool. The outputs of selected paths are summed up to give input to the fifth node (assigned as 1×1 Conv), after which the proposed SBNs are used. The designed cell space is shown in Fig. 6.



(a) (b)

Figure 6: **(a)** Cell space with SBNs in NAS-Bench-101 (Ying et al., 2019) **(b)** Ground truth accuracies vs. predictions by one-shot MixPath supernets ($m = 4$) trained with various strategies. We sample 70 models in NAS-Bench-101 (Ying et al., 2019) (sampling one time) to get the ground-truth. The supernet trained SBN with post BN calibration (top right) gives the best prediction. A vanilla BN alone (w/ SBNs) wrongly pushes models into a narrow range under 0.2.

Firstly, we conduct the baseline experiments on NAS-Bench-101 (Ying et al., 2019), training with and without SBNs. For the latter, the features from the multi-path are summed up and then divided by the number of activated paths, following a 1×1 Conv-BN-ReLU (i.e., a single BN).

To deeply analyze the model ranking capacity of our method and to understand the effects of SBNs, we conduct two pairs of experiments: (i) vanilla BN vs. SBN (ii) post-calibrated vanilla BN vs. post-calibrated SBN. For all experiments, we train the supernet for 100 epochs with a batch size of 96 and a learning rate of 0.025 on three different seeds. We randomly sample 70 models to find their ground-truth top-1 accuracy from NAS-Bench-101 to obtain Kendall tau (Kendall, 1938). We use $m = 4$ in this section.[3] The results are shown in Table 1 and Fig. 6 (b).

Strategy	w/o Calibration	w/ Calibration
w/o SBN	0.117±0.027	0.382±0.007
w/ SBN	0.393±0.037	**0.597±0.037**

Table 1: Comparison of Kendall taus between MixPath supernets ($m = 4$) trained with various strategies. For each control group, we sampled 70 models from NAS-Bench-101 (Ying et al., 2019) and repeated it 3 times on different seeds. †: after BN calibration

Notably, BN post-calibration can boost Kendall tau in each case, which confirms the validity of this post-processing trick (Bender et al., 2018; Guo et al., 2020; Mei et al., 2020). Compared with baseline experiments, the supernet trained with SBNs has a much stronger ranking capacity. Even without calibration, the proposed method still ranks architectures better than the calibrated vanilla BN (+0.011). The composition of two tricks can further boost the final score to 0.597. Next, we will search for models in the search space commonly used for CIFAR-10 (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009) dataset respectively.

[3]We place $m = 3$ case in ablation study.

We also search directly in this reduced space. To be comparable, this case is formulated as a single objective optimization problem: finding the best model with known ground truth (94.29%) in the space. As a strong baseline, we run DARTS (Liu et al., 2019) three times using different seeds. The result is shown in Table 2. Our method obtains 94.22% within 5 GPU hours, which outperforms DARTS with a large margin.

4.3 PROXYLESS SEARCH ON IMAGENET

We search proxylessly on ImageNet (Deng et al., 2009) in the search space of MnasNet (Tan et al., 2019), as this layer-wise search space with inverted bottleneck blocks (Sandler et al., 2018) is also very commonly used (Tan et al., 2019; Tan & Le, 2019; Wu et al., 2019; Chu et al., 2021; Tan & Le., 2019). We fix the expansion rate as in (Tan & Le., 2019) and search for the combinations of various depthwise convolutions. Specifically, we search from the kernel sizes (3, 5, 7, 9) and their combinations. We call this search space S_2. Moreover, we also construct a search space S_3 to incorporate architectures like MixNet (Tan & Le., 2019), where we evenly divide the channel dimension of the depthwise layer into 4 groups and search from the kernel sizes $(3, 5, 7, 9)$ for each group. We set $m = 2$ (in S_2) and $m = 1$ (in S_3). In each case, we utilize the same hyper-parameters: a batch size of 1024 and the SGD optimizer with 0.9 momentum and a weight decay of 10^{-5}. The initial learning rate is 0.1 with cosine decay and it decreases to 0 within 120 epochs. This takes about 150k times of backpropagation (10 GPU days on Tesla V100). Once the supernet is trained, we adopt NSGA-II (Deb et al., 2002) for model selection.

Table 2: Search results on the reduced NAS-Bench-101. The accuracy of known optimal is 94.29%.

Method	Top-1 Acc (%)	Search Cost (GPU Hours)
DARTS (Liu et al., 2019)	79.42±0.23	7
Ours	**94.22±0.06**	5

As per training stand-alone models, we use the same tricks as MixNet-M (Tan & Le., 2019), the results are listed in Table 3. To make fair comparisons, we disable distillation in NAS approaches because it can marginally boost the performance of found models. Our cost in Table 3 includes training supernet (10 GPU days) and NSGA-II (0.3 GPU day). MixPath-A, sampled from the Pareto front in S_3 (so is MixNet), uses 349M multiply-adds to obtain 76.9% accuracy on ImageNet. MixPath-B, in S_2, uses fewer FLOPS and parameters w.r.t EfficientNet-B0 to obtain 77.2%. Note SPOS, FBNet, ProxylessNAS are also in S_2 or similar. In the appendix we list the detailed structures of two models in Fig. 11 of C.5. We also transfer these models on COCO detection, see C.7.

Table 3: Comparison with state-of-the-art models on ImageNet. To make fair comparisons, we disable distillations for all methods. Search cost in GPU days. †: TPU days

Models	×+ (M)	#P (M)	Top-1 (%)	Top-5 (%)	Cost G · d
MobileNetV2 (Sandler et al., 2018)	300	3.4	72.0	91.0	-
ShuffleNetV2 (Ma et al., 2018)	299	3.5	72.6	-	-
GhostNet 1.3× (Han et al., 2020)	226	7.3	75.7	92.7	-
DARTS (Liu et al., 2019)	574	4.7	73.3	-	0.5
P-DARTS(Chen et al., 2019b)	557	4.9	75.6	92.6	0.3
PC-DARTS (Xu et al., 2020)	586	5.3	74.9	92.2	0.1
SGAS (Cri.2 best) (Li et al., 2020)	598	5.4	75.9	92.7	0.25
FairDARTS-C (Chu et al., 2020)	380	4.2	75.1	92.4	0.4
MnasNet-A2 (Tan et al., 2019)	340	4.8	75.6	92.7	≈4k
One-Shot Small (F=32) (Bender et al., 2018)	-	5.1	74.2	-	$4^\dagger$
FBNet-B (Wu et al., 2019)	295	4.5	74.1	-	9
MobileNetV3 (Howard et al., 2019)	219	5.4	75.2	92.2	≈3k
Proxyless-R (Cai et al., 2019)	320	4.0	74.6	92.2	8.3
FairNAS-A (Chu et al., 2021)	388	4.6	75.3	92.4	12
Single Path One Shot (Guo et al., 2020)	328	3.4	74.9	92.0	12
Single-Path NAS (Stamoulis et al., 2019)	365	4.3	75.0	92.2	1.25
OFA w/o PS (Cai et al., 2020)	235	4.4	72.4	-	50
OFA w/ distillation (Cai et al., 2020)	230	-	76.0	-	50
BigNAS-M w/o distillation (Yu et al., 2020a)	418	5.5	77.4	93.5	-
MixNet-M (Tan & Le., 2019)	360	5.0	77.0	93.3	≈3k
MixPath-A (ours)	349	5.0	76.9	93.4	10.3
MixPath-B (ours)	378	5.1	77.2	93.5	10.3

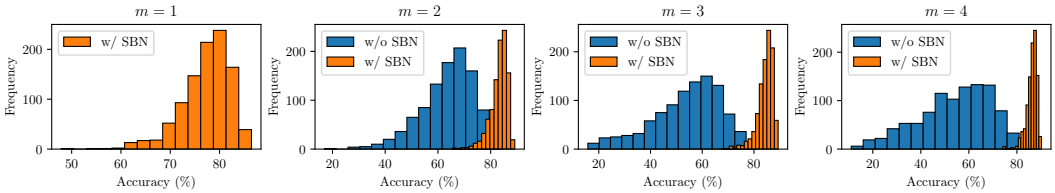


Figure 7: Histogram of one-shot model accuracies sampled from MixPath supernets trained on CIFAR-10 in S_1, activating at most $m = 1, 2, 3, 4$ paths respectively. In all cases, the supernet trained without SBNs severely underestimates a large proportion of candidate architectures. Note when $m = 1$ it falls back to a vanilla BN.

5 Ablation Study and Discussions

5.1 Supernet ranking performance with different strategies

To study the individual contribution of SBN and BN post-calibration, we profit from NAS-Bench-101 (Ying et al., 2019) for ranking calculation since it provides ground truths for all models. Moreover, it allows a fair comparison with other approaches in the same search space. Specifically, we investigate the case $m = 3$ and report the result of each control group using 3 different seeds. Kendall tau (Kendall, 1938) is calculated by randomly sampled 70 models.

Supernet with and without SBNs. Table 4 shows that the supernet trained without SBN is inadequate to evaluate sampled models ($\tau = 0.05$), which is much lower than the one with a linear number of SBNs ($\tau = 0.32$). We further quantify the effect of the additional post BN calibration, and both taus can be boosted. Particularly for SBN, it's improved to 0.59 ($+0.27$). Therefore, the combination of SBN and post BN calibration delivers the best ranking performance.

Table 4: Kendall taus' comparison between MixPath supernets ($m = 3$) trained with multiple strategies. (same settings as Table 1). ESBN: Exponential SBNs, LSBN: Linear SBNs

τ	w/o SBN	ESBN	LSBN
w/o Calibration	0.05±0.06	0.37±0.06	0.32±0.03
w/ Calibration	0.43±0.03	0.34±0.08	**0.59±0.02**

Linear number of SBNs vs. exponential. Through theoretical analysis, we conclude that a linear number of SBNs are enough to match all the combinations of activable paths, while we can still use its exponential counterpart where every single combination follows an SBN. Table 4 shows that linear SBN with BN calibration works best (with the highest τ), while Exp SBN (exponential) performs best if no calibration is applied. Why is it so? Linear SBNs make use of the zero-order condition which is an approximate relationship. Exp SBNs are more accurate since they catch statistics from each of the combinations. However, this makes a wider distribution of learnable parameters γ, β, which is difficult for BN calibration to fit them well. Therefore, we choose Linear SBNs with BN calibration instead. Regarding the Occam's Razor, the linear growing BN generalizes better than the exponential counterpart.

One-shot accuracy distribution of candidate models w.r.t. m

We run four experiments with $m = 1, 2, 3, 4$ to investigate the effect of SBNs in S_1, other settings are kept the same. In total, 1k models are randomly sampled to plot their test accuracy distributions on CIFAR-10 for each m in Fig. 7. When $m = 1$, MixPath falls back to single path. Whereas, SBN begins to demonstrate its power for $m > 1$, whose absence leads to a bad supernet with lower predicted accuracies and a much larger gap. This indicates the supernet without SBNs severely underestimates the performance of a large proportion of architectures.

Simple scaling vs. SBNs. Intuitively from the above motivation, simply scaling the magnitudes of all features to the same level in each block could also stabilize the supernet training. We hereby treat *simple scaling* as a baseline experiment. However, it is somewhat limited because it adjusts magnitudes only. We find that ranking ability after the calibration of the simple scaling case is higher than that of vanilla BNs and exponential SBNs, but it is lower than that of linear SBNs. This is consistent with the baseline experiment in Table 5. Our proposed method with linear SBNs is still the best among all the compared strategies. Note that the simple scaling approach can only roughly scale

the magnitudes of the output. In contrast, SBNs can adjust the magnitudes and angles together to better capture the changing distributions.

5.2 COMPARISON OF SEARCH STRATEGIES

We use the search space S_3 on ImageNet to compare the adopted NSGA-II search algorithm (Deb et al., 2002) with random search by charting the Pareto-front of models found by both methods in Fig. 13 (appendix). NSGA-II has a clear advantage that the Pareto-front models have higher accuracies and fewer multi-adds.

Table 5: Comparison of Kendall taus between MixPath supernets ($m = 4$) trained with various strategies. We sampled 70 models from NAS-Bench-101 (Ying et al., 2019) and repeated it 3 times on different seeds. †: after BN calibration

Strategy	Simple Scaling	w/ SBNs
w/o Calibration	0.293±0.016	0.393±0.017
Calibration	0.472±0.021	**0.597±0.037**

5.3 PIPELINE COMPONENT ANALYSIS

Apart from the ablation studies on NAS-Bench-101 (Ying et al., 2019), we make a component analysis of our pipeline in S_1. We focus on the supernet training with and without SBN by fixing the second-stage searching algorithm. Each pipeline is run three times on different random seeds. Our goal is to find the best models under 500M FLOPS. As a result, with SBN enabled, we obtain 97.44% top-1 accuracy, higher than the best 97.12% when SBN is disabled. This confirms that it mainly benefits from our novel training mechanism.

5.4 DISCUSSIONS

Why can SBNs stabilize supernet training and improve ranking ability? We have verified that high cosine similarity is not the only factor that guarantees the stable training of a multi-path supernet. It is more important to ensure that the feature distributions are consistent. Otherwise, the training is disrupted because a single BN is not enough to capture changing statistics from multiple paths. The proposed SBNs are able to track a variety of distributions, alleviating this instability. Empirically, better training of the supernet also gives more appropriate weights for each subnetwork, with which we can better estimate the ground-truth performance of the stand-alone models.

Why can SBNs work together with BN post-calibration? According to the analysis in Section 3.2, the parameters μ, σ of SBNs that follow a multi-path combination are multiples of those of a single-path, as shown in Fig. 5. But not all the parameters fit well into this relationship, post BN calibration can then readjust the mean and variance of the inputs to fit the learned parameters γ and β. To some extent, it can make up for the gap from the supernet to submodels. Noticeably, SBN is a type of regularization technique during the supernet training, while BN calibration is a post-processing trick. They can work in an orthogonal way and both contribute to the ranking.

Comparison with Switchable Batch Normalization Switchable BN (Yu et al., 2019) is only applied to learn the statistics of a limited number (K) of sub-architectures with $K = \{4, 8\}$ channel configurations. Our SBN is designed to match the changing feature statistics from a more flexible combination of different choice paths, while the number of channels is fixed. The former is used for network pruning while our SBN is to stabilize the training of the multi-path supernet.

6 CONCLUSION

In this paper, we propose a unified approach for two-stage one-shot neural architecture search, with a multi-path search space enabled. Existing single-path approaches can thus be regarded as a special case. The proposed method uses SBNs to catch the changing features from various branch combinations, which successfully solves two difficulties of vanilla multi-path methods: the unstable training of supernet and the unbearable weakness of model ranking. Moreover, by exploiting feature similarities from different path combinations, we reduced the number of required exponential SBNs to m, identical to the number of allowed paths. Extensive experiments on NAS-Bench-101 show that our method can boost the model ranking capacity of the one-shot supernet by clear margins. We achieved state-of-the-art architectures like MixPath-A (76.9%) and B (77.2%) on ImageNet.

REFERENCES

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and Simplifying One-Shot Architecture Search. In *ICML*, pp. 549–558, 2018.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*, 2019.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HylxE1HKwS.

Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019a.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive Differentiable Architecture Search: Bridging the Depth Gap between Search and Evaluation. In *ICCV*, 2019b.

Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: Eliminating Unfair Advantages in Differentiable Architecture Search. In *ECCV*, 2020.

Xiangxiang Chu, Bo Zhang, and Ruijun Xu. FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. In *ICCV*, 2021.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2): 182–197, 2002.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, pp. 248–255. IEEE, 2009.

Xuanyi Dong and Yi Yang. Searching for a Robust Neural Architecture in Four GPU Hours. In *CVPR*, pp. 1761–1770, 2019.

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. In *European Conference on Computer Vision*, pp. 544–560. Springer, 2020.

Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. GhostNet: More Features from Cheap Operations. In *CVPR*, 2020.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *ICCV*, 2019.

Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. *NeurIPS*, 2019.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, pp. 448–456, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

Maurice G Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938.

Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do Better Imagenet Models Transfer Better? In *CVPR*, pp. 2661–2671, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. Technical report, Citeseer, 2009.

Van Der Maaten Laurens and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(2605):2579–2605, 2008.

Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Blockwisely Supervised Neural Architecture Search with Knowledge Distillation. *arXiv preprint arXiv:1911.13053*, 2019.

Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Müller, Ali Thabet, and Bernard Ghanem. SGAS: Sequential Greedy Architecture Search. In *CVPR*, 2020.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *ECCV*, 2014.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *ICCV*, pp. 2980–2988, 2017.

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*, pp. 19–34, 2018.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *ICLR*, 2019.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural Architecture Optimization. In *NIPS*, pp. 7816–7827, 2018.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical Guidelines for Efficient CVV Architecture Design. In *ECCV*, pp. 116–131, 2018.

Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Yang Jianchao. AtomNAS: Fine-Grained End-to-End Neural Architecture Search. In *ICLR*, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pp. 8024–8035, 2019.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*, 2018.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, pp. 4510–4520, 2018.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours. In *ECML PKDD*, 2019.

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*, pp. 1–9, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, pp. 2818–2826, 2016.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *AAAI*, 2017.

Mingxing Tan and Quoc V Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*, 2019.

Mingxing Tan and Quoc V. Le. MixConv: Mixed Depthwise Convolutional Kernels. In *BMVC*, 2019.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*, 2019.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *CVPR*, 2019.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. In *CVPR*, pp. 1492–1500, 2017.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic Neural Architecture Search. In *ICLR*, 2019.

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. URL https://openreview.net/forum?id=BJlS634tPr.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*, pp. 7105–7114, 2019.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable Neural Networks. In *ICLR*, 2019. URL https://openreview.net/forum?id=H1gMCsAqY7.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pp. 702–717. Springer, 2020a.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating The Search Phase of Neural Architecture Search. In *ICLR*, 2020b. URL https://openreview.net/forum?id=H1loF2NFwr.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020. URL https://openreview.net/forum?id=H1gDNyrKDS.

Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial Distribution Learning for Effective Neural Architecture Search. In *ICCV*, pp. 1304–1313, 2019.

Barret Zoph and Quoc V Le. Neural Architecture Search with Reinforcement Learning. In *ICLR*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning Transferable Architectures for Scalable Image Recognition. In *CVPR*, volume 2, 2018.

A PROOFS

A.1 PROOF OF LEMMA 3.1

Proof. Let $\mathbf{y}_{p_{\mathbf{y}}(\mathbf{y})} = f(\mathbf{x})$, $\mathbf{z}_{p_{\mathbf{z}}(\mathbf{z})} = g(\mathbf{x})$, $\mathbf{x} \sim p_{\mathbf{x}}(\mathbf{x})$. For the case $m = 1$, the expectation of $\mathbf{y}$ and $\mathbf{z}$ can be written respectively as:

$$\begin{aligned} \mathbb{E}[\mathbf{y}] = \mathbb{E}[f(\mathbf{x})] = \int p_{\mathbf{x}}(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\,\mathbf{x} \\ \mathbb{E}[\mathbf{z}] = \mathbb{E}[g(\mathbf{x})] = \int p_{\mathbf{x}}(\mathbf{x}) g(\mathbf{x}) \mathrm{d}\,\mathbf{x} \end{aligned} \tag{1}$$

According to the zero-order condition, we have $f(\mathbf{x}) \approx g(\mathbf{x})$. And $p(\mathbf{x})$ is same for both $\mathbf{y}$ and $\mathbf{z}$, so $\mathbb{E}[\mathbf{y}] \approx \mathbb{E}[\mathbf{z}]$.

Now we prove $Var[\mathbf{y}] \approx Var[\mathbf{z}]$. Note that $Var[\mathbf{y}] = \mathbb{E}\left[\mathbf{y}^2\right] - (\mathbb{E}[\mathbf{y}])^2$ and $Var[\mathbf{z}] = \mathbb{E}\left[\mathbf{z}^2\right] - (\mathbb{E}[\mathbf{z}])^2$, thus we only need to prove $\mathbb{E}\left[\mathbf{y}^2\right] \approx \mathbb{E}\left[\mathbf{z}^2\right]$. It can be similarly proved as follows:

$$\begin{aligned} \mathbb{E}\left[\mathbf{y}^2\right] = \int p_{\mathbf{y}}(\mathbf{y}) \mathbf{y}^2 \mathrm{d}\mathbf{y} = \int p_{\mathbf{x}}(\mathbf{x}) f^2(\mathbf{x}) \mathrm{d}\mathbf{x} \\ \mathbb{E}\left[\mathbf{z}^2\right] = \int p_{\mathbf{z}}(\mathbf{z}) \mathbf{z}^2 \mathrm{d}\mathbf{z} = \int p_{\mathbf{x}}(\mathbf{x}) g^2(\mathbf{x}) \mathrm{d}\mathbf{x} \end{aligned} \tag{2}$$

According to the zero-order condition, we have $Var[\mathbf{y}] \approx Var[\mathbf{z}]$.

For the case of $m = 2$, when the two paths are both selected, the output becomes $\mathbf{y} + \mathbf{z}$, its expectation can be written as:

$$\mathbb{E}[\mathbf{y} + \mathbf{z}] = \mathbb{E}[\mathbf{y}] + \mathbb{E}[\mathbf{z}] \approx 2\mathbb{E}[\mathbf{y}] \tag{3}$$

And the variance of $\mathbf{y} + \mathbf{z}$ is,

$$Var[\mathbf{y} + \mathbf{z}] \approx Var[2\mathbf{y}] = 4Var[\mathbf{y}] \tag{4}$$

Therefore, there are two kinds of expectations and variances: $\mathbb{E}[\mathbf{y}]$ and $Var[\mathbf{y}]$ for $\{\mathbf{y}, \mathbf{z}\}$, and $2\mathbb{E}[\mathbf{y}]$ and $4Var[\mathbf{y}]$ for $\{\mathbf{y} + \mathbf{z}\}$. Similarly, in the case where $m \in [1, n]$, there will be m kinds of expectations and variances. □

B ALGORITHMS

Algorithm 2 : Stage 2-NSGA-II search strategy.

Input: Supernet S, the number of generations N, population size n, validation dataset D, constraints C, objective weights w.
Output: A set of K individuals on the Pareto front.
Uniformly generate the populations P_0 and Q_0 until each has n individuals satisfying C_{acc}, C_{FLOPs}.
for $i = 0$ **to** $N - 1$ **do**
  $R_i = P_i \cup Q_i$
  $F =$ non-dominated-sorting(R_i)
  Pick n individuals to form P_{i+1} by ranks and the crowding distance weighted by w.
  $Q_{i+1} = \emptyset$
  while $size(Q_{i+1}) < n$ **do**
    $M =$ tournament-selection(P_{i+1})
    $q_{i+1} =$ crossover(M)
    if $FLOPs(q_{i+1}) > FLOPs_{s_{max}}$ **then**
      continue
    end if
    Evaluate model q_{i+1} with S (BN calibration is recommended) on D
    if $acc(q_{i+1}) > acc_{min}$ **then**
      Add q_{i+1} to Q_{i+1}
    end if
  end while
end for
Select K equispaced models near Pareto-front from P_N

C EXPERIMENTS DETAILS

C.1 SEARCH SPACES

We show the list of used search spaces in Table 6.

Space	Dataset	m	Size	Details
S_1	CIFAR-10	1 2 3 4	8^{12} 36^{12} 92^{12} 162^{12}	There are 12 stacked inverted bottleneck blocks. Kernel sizes are in (3, 5, 7, 9), and expansion rates are in (3, 6).
S_2	ImageNet	2	10^{18}	There are 18 stacked inverted bottleneck blocks. Kernel sizes are in (3, 5, 7, 9). Expansion rate is fixed following MixNet.
S_3	ImageNet	1	256^{18}	There are 18 stacked mobile inverted bottlenecks. Depthwise layer channels are divided into 4 groups and there are 4 choice kernel sizes (3, 5, 7, 9) for each group. Expansion rate is also fixed as above.
S_4	CIFAR-10	4	255	There are 9 stacked cells, each with 5 internal nodes, where the first 4 nodes are candidate paths. Each node has 3 operation choices (1×1 Conv, 3×3 Conv, 3×3 Maxpool). See Fig. 6 (main text).

Table 6: Four search spaces used in this paper. m is the maximum number of allowed paths per layer

C.2 SEARCH ON CIFAR-10

We conduct our experiments on CIFAR-10 dataset (Krizhevsky et al., 2009) with a designed search space S_1 containing 12 inverted bottlenecks (Sandler et al., 2018), each of which has four choices of

kernel size (3, 5, 7, 9) for the depthwise layer and two choices (3, 6) for expansion rates. Hence, the size of search space S_1 ranges from 8^{12} (m=1) to 162^{12} (m=4). For each case, we directly train the supernet on CIFAR-10 for 200 epochs. The batch size is set to 96 and we use the SGD optimizer with a momentum of 0.9 and 3×10^{-4} weight decay. We also use a cosine scheduler for the learning rate strategy with an initial value of 0.025. It only takes about 6 GPU hours on a single Tesla V100 GPU in total. The comparison with recent state-of-the-art models is listed in Table 7 and 3. Our searched model MixPath-c obtains 97.4 % top-1 accuracy using 493M FLOPS on CIFAR-10, whose architecture is shown in Fig. 11 in C.5 (appendix).

Models	Params (M)	$\times+$ (M)	Top-1 (%)	Type	Cost G · d
NASNet-A (Zoph et al., 2018)	3.3	608	97.4	RL	1800
NASNet-A Large (Zoph et al., 2018)*	85.0	12030	98.0	TF	1800
ENAS (Pham et al., 2018)	4.6	626†	97.1	RL	0.5
DARTS (Liu et al., 2019)	3.3	528	97.1	GD	0.4
RDARTS (Zela et al., 2020)	-	-	97.1	GD	1.6
SNAS (Xie et al., 2019)	2.9	422	97.0	GD	1.5
GDAS (Dong & Yang, 2019)	3.4	519	97.1	GD	0.2
P-DARTS (Chen et al., 2019b)	3.4	532	97.5	GD	0.3
PC-DARTS (Xu et al., 2020)	3.6	558	97.4	GD	0.1
MixPath-c (ours)	5.4	493	97.4	OS	0.25
MixNet-M (Tan & Le., 2019)*	3.5	360	97.9	TF	≈3k
MixPath-a (ours)*	3.5	348	98.1	TF	10
MixPath-b (ours)*	3.6	377	98.2	TF	10

Table 7: Comparison of models on CIFAR-10. †: MultAdds computed from the provided code. *: transferred from ImageNet (whose search cost is on ImageNet). RL: Reinforcement learning, GD: Gradient-based, OS: one-shot, TF: Transferred.

C.3 More statistics analysis on SBNs

To further confirm our early postulation, we train MixPath supernet in the search space S_1 on CIFAR-10, allowing the number of activable paths $m = 3$ and $m = 4$. Other settings are kept the same as the case $m = 2$. The relationship of parameters in SBNs is shown in Fig. 9. As expected, SBNs capture feature statistics for different combinations of path activation. For instance, the mean of SBN_3 is three times that of SBN_1. The similar phenomenon can be observed in other layers as well, for instance, the statistics of the 6-th and 11-th layer are shown in Fig. 10.

SBNs have a strong impact on regularizing features from multiple paths This is more obvious when we draw a t-SNE visualization (Laurens & Hinton, 2008) of first-layer feature maps from our MixPath supernet ($m = 4$) trained on CIFAR-10 in Fig. 8. Before applying SBNs, features from different path combinations are quite distant from each other, while SBNs close up this gap and make them quite similar.

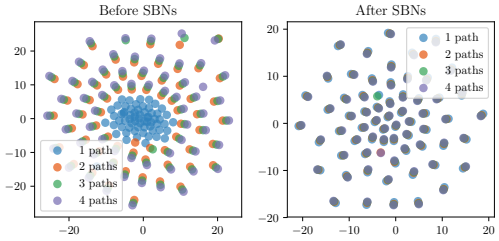
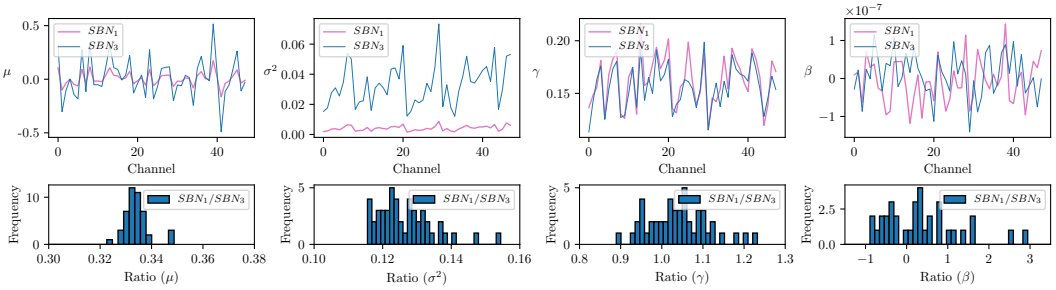


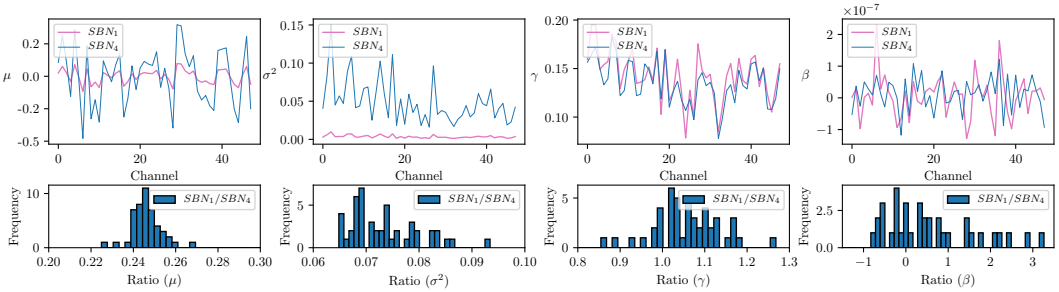
Figure 8: A t-SNE visualization of first-layer multi-path features before and after SBNs. We randomly sample 64 samples to get these features. Dots of the same color indicate the same multi-path combination. SBNs make distant features from multi-path combinations similar to each other (see closely overlapped dots on the right). Best viewed in color.

C.4 Cosine similarity and feature vectors on NAS-Bench-101

We also plot the cosine similarity of features from different operations along with their projected vectors before/after SBNs and vanilla BNs on

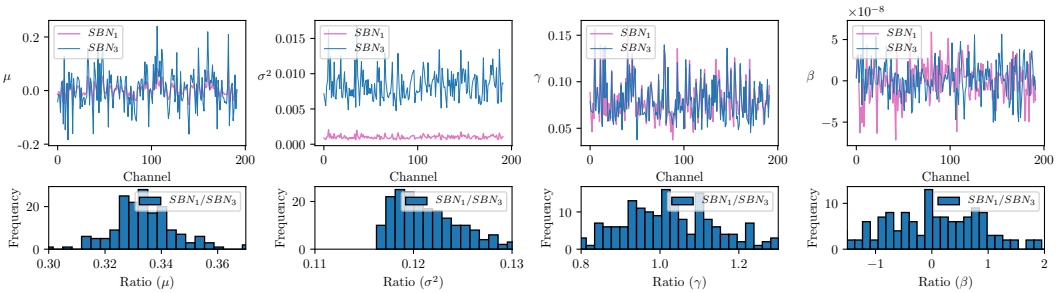


(a) $m = 3$

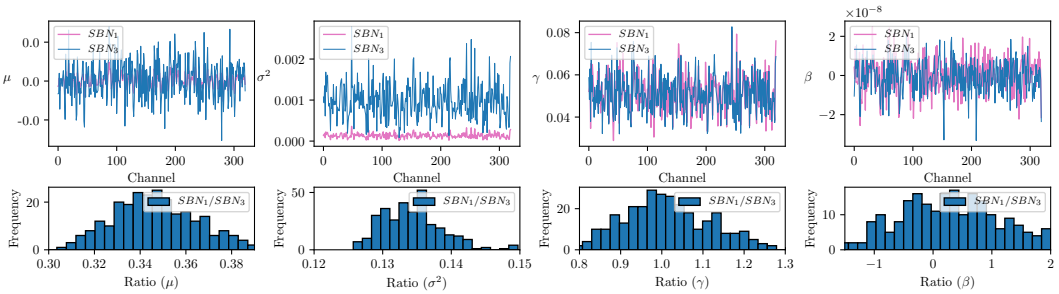


(b) $m = 4$

Figure 9: Parameters $(\mu, \sigma^2, \gamma, \beta)$ of the first-layer SBNs in MixPath supernet (in S_1) trained on CIFAR-10 when at most $m = 3, 4$ paths can be activated. SBN_n refers to the one follows n-path activations. The parameters of SBN_3 and SBN_4 are multiples of SBN_1 as expected.



(a) Layer 6



(b) Layer 11

Figure 10: Parameters $(\mu, \sigma^2, \gamma, \beta)$ of the SBNs of the 6th and 11th layer in MixPath supernet (in S_1) trained on CIFAR-10 when at most $m = 3$ paths can be activated. SBN_n refers to the one follows n-path activations. The parameters of SBN_3 are still multiples of SBN_1 as expected.

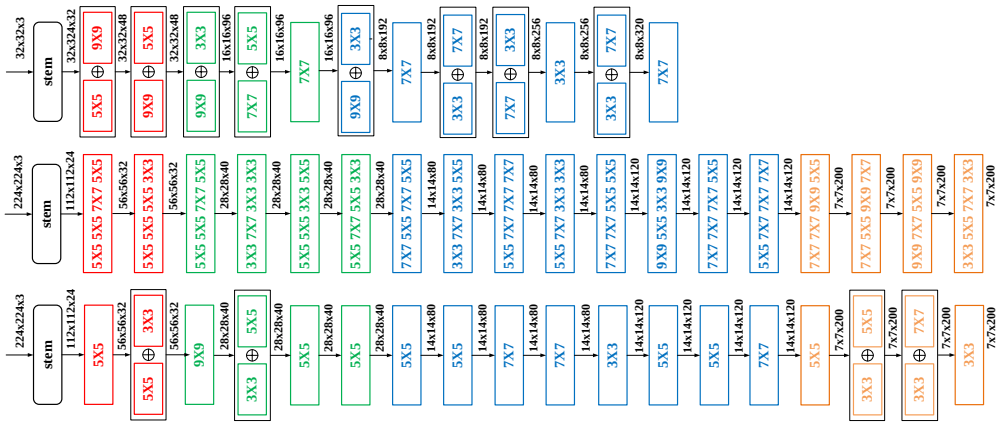


Figure 11: The architecture of MixPath-c (top), MixPath-A (middle) and MixPath-B (bottom). MixPath-B makes use of feature aggregation and outperforms EfficientNet-B0 with fewer FLOPS and parameters.

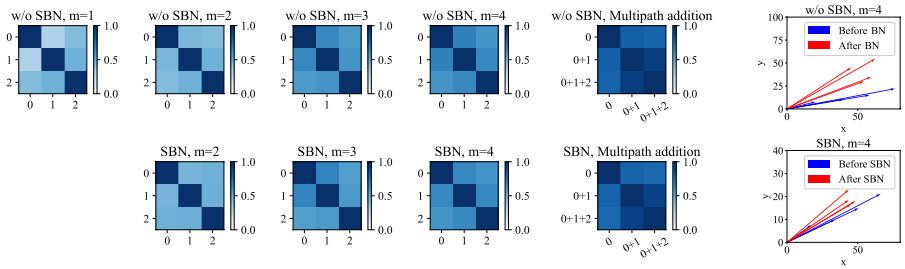


Figure 12: **(a)** Cosine similarity of first-block features from the supernets trained on NAS-Bench-101 with and without SBNs **(b)** Feature vectors projected into 2-dimensional space.

NAS-Bench-101 in Fig. 12. We can see that not only are the features from different operations similar, but so are the summations of features from multiple paths. At the same time, SBNs can transform the amplitudes of different vectors to the same level, while vanilla BNs can't. This is similar to the situation in the search space S_1 and matches with our theoretical analysis.

C.5 Searched architectures on CIFAR-10 and ImageNet

The architectures of MixPath-c, MixPath-A and MixPath-B are shown in Fig 11.

C.6 Transferring to CIFAR-10

We also evaluated the transferability of MixPath models on CIFAR-10 dataset, as shown in Table 3 (main text). The settings are the same as (Huang et al., 2019) and (Kornblith et al., 2019). Specifically, MixPath-b achieved 98.2% top-1 accuracy with only 377M FLOPS.

C.7 Transferring to object detection

We further verify the transferability of our models on object detection tasks and we only consider mobile settings. Particularly, we utilize the RetinaNet framework (Lin et al., 2017) and use our models as drop-in replacements for the backbone component. Feature Pyramid Network (FPN) is enabled for all experiments. The number of the FPN output channels is 256. The input features from the backbones to FPN are the output of the depth-wise layer of the last bottleneck block in four stages, which covers 2 to 5.

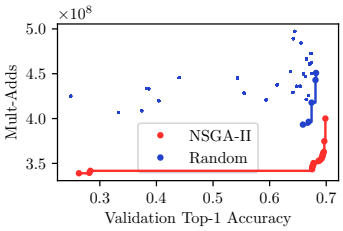


Figure 13: Pareto-front of models by NSGA-II vs. random search.

All the models are trained and evaluated on the MS COCO dataset (Lin et al., 2014) (train2017 and val2017 respectively) for 12 epochs with a batch size of 16. We use the SGD optimizer with 0.9 momentum and 0.0001 weight decay. The initial learning rate is 0.01 and multiplied by 0.1 at epochs 8 and 11. Moreover, we use the MMDetection toolbox (Chen et al., 2019a) based on PyTorch (Paszke et al., 2019). Table 8 shows that MixPath-A gives competitive results.

Backbones	×+ (M)	P (M)	Acc (%)	AP (%)	AP_{50} (%)	AP_{75} (%)	AP_S (%)	AP_M (%)	AP_L (%)
MobileNetV3	219	5.4	75.2	29.9	49.3	30.8	14.9	33.3	41.1
MnasNet-A2	340	4.8	75.6	30.5	50.2	32.0	16.6	34.1	41.1
SingPath NAS	365	4.3	75.0	30.7	49.8	32.2	15.4	33.9	41.6
MixNet-M	360	5.0	77.0	31.3	51.7	32.4	17.0	35.0	41.9
MixPath-A	349	5.0	76.9	31.5	51.3	33.2	17.4	35.3	41.8

Table 8: COCO Object detection with various drop-in backbones

C.8 Comparison of search strategies

We show the Pareto front of models searched by NSGA-II vs. Random in Fig. 13.

C.9 Applying SPOS on multi-path search space

We design a "branch composition" experiment $m = 4$ in NAS-Bench-101 with 3 candidate operations, leading to 34 possible blocks. We adopt the SPOS strategy [9] to train this supernet with the same hyper-parameters as ours, but we only reach a Kendall tau of **-0.225** while our method obtains **0.504** in terms of ranking the same sub-models (see Fig. 14). This result is not surprising. We blame the failure of the former approach for the use of an excessive number of blocks, which brings nearly **4**× more trainable parameters in the supernet (199.7M vs. ours 52.9M). In this case, it's impractical and infeasible for SPOS to work under such setting because it's hard to make candidate blocks fully optimized. In effect, thus-trained supernet is unable to ensure a reliable rank. For example, on average, one block receives an update every 34 iterations, resulting in poor ranking performance. Besides, this approach will encounter memory explosion at a growing m. Even if we can solve the ranking issue, it requires much more GPU memory to save the trainable parameters, which prevents proxyless search on large datasets or big models. In contrast, our method doesn't suffer from these issues.

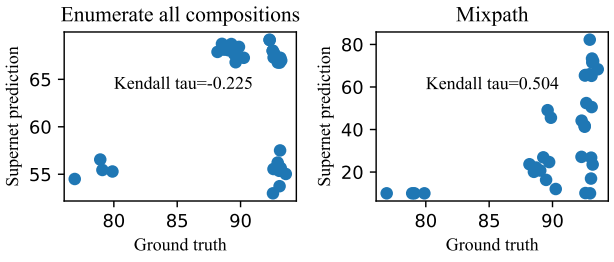


Figure 14: Rank performance comparison between enumerating all compositions and our method.