FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search

Xiangxiang Chu* Bo Zhang Ruijun Xu
{chuxiangxiang,zhangbo11,xuruijun}@xiaomi.com

Abstract

One of the most critical problems in weight-sharing neural architecture search is the evaluation of candidate models within a predefined search space. In practice, a one-shot supernet is trained to serve as an evaluator. A faithful ranking certainly leads to more accurate searching results. However, current methods are prone to making misjudgments. In this paper, we prove that their biased evaluation is due to inherent unfairness in the supernet training. In view of this, we propose two levels of constraints: expectation fairness and strict fairness. Particularly, strict fairness ensures equal optimization opportunities for all choice blocks throughout the training, which neither overestimates nor underestimates their capacity. We demonstrate that this is crucial for improving the confidence of models’ ranking. Incorporating the one-shot supernet trained under the proposed fairness constraints with a multi-objective evolutionary search algorithm, we obtain various state-of-the-art models, e.g., FairNAS-A attains 77.5% top-1 validation accuracy on ImageNet.

1. Introduction

The advent of neural architecture search (NAS) has brought deep learning into an era of automation [54]. Abundant efforts have been dedicated to searching within carefully designed search space [55, 37, 44, 31, 45]. Meanwhile, the evaluation of a network’s performance is an important building block for NAS. Conventional approaches evaluate an enormous amount of models based on resource-devouring training [55, 44]. Recent attention has been drawn to improve its efficiency via parameter sharing [2, 29, 35, 48].

Generally speaking, the weight-sharing approaches all involve training a supernet that incorporates many candidate subnetworks. They can be roughly classified into two categories: those who couple searching and training within one stage [35, 29, 4, 41, 48] and others who decouple them into two stages, where the trained supernet is treated as an evaluator for final searching [2, 1, 16, 32]. The supernet is a so-called one-shot model.

Despite being widely utilized due to searching efficiency, weight sharing approaches are roughly built on empirical experiments instead of solid theoretical ground. Several fundamental issues remain to be addressed. Namely, a) Why is there a large gap between the range of supernet predicted accuracies and that of “ground-truth” ones by stand-alone training from scratch [2, 1]? b) How to build a good evaluator that neither overestimates nor underestimates subnetworks? c) Why does the weight-sharing mechanism work, if under some conditions?

In this paper, we attempt to answer the above three questions for two-stage weight-sharing approaches. We present Fair Neural Architecture Search (FairNAS) in which we train the supernet under the proposed fairness constraints to improve evaluation confidence. Our analysis and experiments are conducted in a widely used search space as in [4, 48, 16, 41], as well as a cell-based search space from a common benchmark NAS-Bench-201 [14]. The contributions can be summarized as follows.

Firstly, we prove it is due to unfair bias that the supernet misjudges submodels’ performance, which is inevitable in current one-shot approaches [2, 1].

Secondly, we propose two levels of fairness constraints: Expectation Fairness (EF) and Strict Fairness (SF). They are enforced to alleviate supernet bias and to boost evaluation capability. Both outperform the existing unfair approaches while SF performs best with a ranking ($\tau$) of 0.7412 on NAS-Bench-201.

Thirdly, we unveil the root cause of the validity of single-path supernet training under our fairness perspective. That is, different choice blocks are interchangeable during the supernet training as they learn similar feature maps, according to their high cosine similarity measure, see Figure 2.

Last but not the least, our fair single-path sampling is memory-friendly, and its GPU costs can also be linearly amortized to the number of target models, see Fig. 1. We then incorporate our supernet with an EA-based multi-objective searching framework, from which we obtain three state-of-the-art networks within a single proxyless run at a cost of 12 GPU days on ImageNet.

*This work was done when all the authors were at Xiaomi AI Lab.
2. Fairness Taxonomy of Weight-sharing NAS

2.1. Review of Biased Supernets

On the one hand, supernet training and searching for good models are nested. In ENAS [35], the sampling policy $\pi(m, \Theta)$ of an LSTM controller [17] and a sampled subnetwork $m$ are alternatively trained. The final models are sampled again by the trained policy $\pi$, one who has the highest reward on a mini-batch of validation data is finally chosen. DARTS [29] combines the supernet training and searching within a bi-level optimization where each operation is associated with a coefficient denoting its importance. Both two methods treat all subnetworks unequally and introduce gradually increasing biases through optimization. Those who have better initial performance are more likely to be sampled or to maintain higher coefficients, resulting in a suboptimal or an even worse solution. For instance, architectures from DARTS usually contain an excessive number of skip connections [51, 26], which damage the outcome performance. Therefore, the prior-learning DARTS is biased as per skip connections, while a random approach doesn’t suffer [25]. DARTS overrated ‘bad’ models (jammed by skip connections), meantime many other good candidates are depreciated.

On the other hand, the rest one-shot methods consider the trained supernet as a confident proxy, which we also follow, to predict the real performance of all subnetworks [2, 1, 16]. We emphasize that a reliable proxy supernet should neither severely overestimate nor underestimate the ground-truth performance of any model. The next searching stage is decoupled from training and it can be implemented with random sampling, evolutionary algorithms, or reinforcement learning.

SMASH [2] invents a hyper network (referred to as HyperNet $H$) to generate the weights of a neural architecture by its binary encoding. This HyperNet resembles a typical supernet in that they can both produce weights for any architecture in the search space. At each step, a model is randomly sampled and trained based on the generated weights from $H$, and in turn, it updates the weights of $H$. For a set of randomly sampled models, a correlation between predicted validation errors and ground-truth exists, but it has a large discrepancy between the ranges, i.e., 40%-90% vs. 25%-30% on CIFAR-100 [24].

One-Shot [1] involves a dynamic dropout rate for the supernet, each time only a subset is optimized. Apart from its training difficulty, there is also an evident performance gap of submodels with inherited weights compared with their ground-truth, i.e., 30%-90% vs. 92%-94.5% on CIFAR-10 [24].
2.2. Evaluating Supernets by Ranking Ability

Regardless of how supernets are trained, what matters most is how well they predict the performance of candidate models. To this end, a recent work [40] evaluates weight-sharing supernets with Kendall Tau (τ) metric [22]. It measures the relation between one-shot models and stand-alone trained ones. The range of τ is from -1 to 1, meaning the rankings are totally reversed or completely preserved, whereas 0 indicates no correlation at all. Surprisingly, most weight-sharing supernets with Kendall Tau (τ) metric [22].

In order to reduce the above mentioned bias in Section 2.1, we first consider the case \(P(m, n, l)\). Selecting a block from layer \(l\) is subject to a categorical distribution. In this case, each basic event occurs with an equal probability \(p(X = l_i) = \frac{1}{m}\). For \(n\) steps, the expectation and variance of \(Y_i\), can be written as,

\[
E(Y_i) = n \cdot p_i = n/m \\
\text{Var}(Y_i) = n \cdot p_i (1 - p_i) = \frac{n(m-1)}{m^2}
\]

That’s to say, all choices share the same expectation and variance.

2.3. Formal Formulation of Fairness

What kind of fairness can we think of? Will fairness help to improve supernet performance and ranking ability? First of all, to remove the training difference between a supernet and its submodels, we scheme an equality principle on training modality.

**Definition 2.1. Equality Principle.** Training a supernet satisfies the equality principle if and only if it is in the same way how a submodel is trained.

Only those who train a single path model at each step meet this principle by its definition. On the contrary, other methods like DARTS [29] train the supernet with all paths altogether, One-Shot [1] dynamically drops out some paths, and ProxylessNAS [4] uses two paths, directly violating the principle.

Formally, we discuss fairness in a common supernet that consists of \(L\) layers, each with several choice blocks. Without loss of generality, we suppose each layer has an equal number of choices, say \(m\). A model is generated by sampling a block layer by layer. The weights are updated for \(n\) times in total. Therefore, we can describe the training process as \(P(m, n, L)\).

2.3.1 First Attempt: Expectation Fairness

In order to reduce the above mentioned bias in Section 2.1, a natural way is to guarantee all choices blocks have equal expectations after \(n\) steps. We define this basic requirement as expectation fairness in Definition 2.2.

**Definition 2.2. Expectation Fairness.** On the basis of Definition 2.1, let \(\Omega\) be the sampling space containing \(m\) basic events \(\{l_1, l_2, ..., l_m\}\), which are generated by selecting a block from layer \(l\) with \(m\) choice blocks. Let \(Y_i\) be the number of times that the outcome \(l_i\) is observed (updated) over \(n\) trials. Then the expectation fairness is that for \(P(m, n, L)\), \(E(Y_{i1}) = E(Y_{i2}) = ... = E(Y_{in})\) holds, \(\forall l \in L\).

2.3.2 An EF Example: Uniform Sampling

Let us check a single-path routine [16] which uses uniform sampling. As sampling on any layer \(l\) is independent of others, we first consider the case \(P(m, n, l)\). Selecting a block from layer \(l\) subject to a categorical distribution. In this case, each basic event occurs with an equal probability \(p(X = l_i) = \frac{1}{m}\). For \(n\) steps, the expectation and variance of \(Y_i\), can be written as,

\[
E(Y_i) = n \cdot p_i = n/m \\
\text{Var}(Y_i) = n \cdot p_i (1 - p_i) = \frac{n(m-1)}{m^2}
\]

That’s to say, all choices share the same expectation and variance. Consequently, uniform sampling meets Expectation Fairness by Definition 2.2 and it seems superficially fair for various choices. However, Expectation Fairness is not enough. For example, we can randomly sample each model and keep it training for \(k\) times, then switch to another. This procedure also meets Definition 2.2, but it’s very unstable to train.

Even in SPOS [16] with uniform sampling, there is a latent ordering issue. For a sequence of choices \((M_1, M_2, M_3)\), it implies an inherent training order \(M_1 \rightarrow M_2 \rightarrow M_3\). Since each model is usually trained by back-propagation, the trained weights of \(M_1\) are immediately updated to the supernet and those of \(M_2\) are renewed next while carrying the effect of the former update, so for \(M_3\). A simple permutation of \((M_1, M_2, M_3)\) does comply with Expectation Fairness but yields different results. Besides, if the learning rate \(lr\) is changed within the sequence, the situation thus becomes even more complicated.

Generally, for \(P(m, n, L)\) where \(m, n, L\) are positive integers, assume the sampling times \(n\) can be divided by \(m(\geq 2)\). If we adopt uniform sampling, as \(n\) goes infinite, it is impossible for \(m\) choices to be sampled for an exactly equal number of times. This is formally stated as below.

**Lemma 2.1.** Regarding \(P(m, n, L)\), \(\forall n \in \{x: x\%m = 0, x \in N_+\}\), \(\lim_{n \to +\infty} p(Y_{i1} = Y_{i2} = ... = Y_{in}) = 0\).

**Proof.** Let \(f(m, n) = p(Y_{i1} = Y_{i2} = ... = Y_{in})\).

\[
f(m, n) = \binom{n}{m} \binom{n}{m-1} ... \binom{n}{m-n+1} \frac{1}{m^n} = \frac{n!}{(n/m)!} \frac{1}{m^n}
\]

(2)

Firstly, we prove the existence of limitation, \(f(n)\) strictly decreases monotonically with \(n\) and \(f(n) \geq 0\), therefore, its limitation exists.
Secondly, we calculate its limitation using equivalent infinity replacement based on Stirling’s approximation about factorial [47].

\[
\lim_{n \to +\infty} f(m, n) = \lim_{n \to +\infty} \frac{n!}{(\frac{n}{m})^m \times m^n} = \lim_{n \to +\infty} \frac{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n}{\sqrt{2\pi m} \left(\frac{m}{e}\right)^m} = \lim_{n \to +\infty} \frac{\sqrt{m}}{2\pi n} = 0
\]

Q.E.D.

Lemma 2.1 is somewhat counter-intuitive and thereby neglected in previous works. To throw light on this phenomenon, we plot this probability curve in Figure 3. We see that \( f(2, n) \) decreases below 0.2 when \( n \geq 20 \). In most cases, \( n \geq 10^6 \), which suffers severely from this issue.

![Figure 3. The function curve of \( f(m, n) \) in Lemma 2.1. When sampling uniformly from \( m \) blocks for \( n \) trials, the probability of having an equal sampling number for each block quickly reaches zero.](image)

2.3.3 A Meticulous Overhaul: Strict Fairness

Our insights come from the above overlooked phenomenon. We propose a more rigorous requirement that ensures the parameter of every choice block be updated the same amount of times at any stage, which is called strict fairness and formally as Definition 2.3.

**Definition 2.3. Strict Fairness.** Regarding \( P(m, n, L) \), \( \forall n \in \{ x : x \% m = 0, x \in N_+ \} \), \( Y_{i_1} = Y_{i_2} = \ldots = Y_{i_m} \) holds.

Definition 2.3 imposes a constraint more demanding than Definition 2.2. That is, \( p(Y_{i_1} = Y_{i_2} = \ldots = Y_{i_m}) = 1 \) holds at any time. It seems subtle but it will be later proved to be crucial. Nevertheless, we have to be aware that it is not ultimate fairness since different models have their own optimal initialization strategy and hyperparameters, which we single them out for simplicity.

3. Fair Neural Architecture Search

Our NAS pipeline is divided into two stages: training the supernet and searching for competitive models.

### 3.1. Stage 1: Train Supernet with Strict Fairness

We first propose a fair sampling and training algorithm to strictly abide by Definition 2.3. We use uniform sampling without replacement and sample \( m \) models at step \( l \) so that each choice block must be activated and updated only once. This is detailed in Algorithm 1 and depicted by Figure 4.

**Algorithm 1**: Stage 1 - Fair Supernet Training.

**Input**: training steps \( n \), search space \( S_{(m,L)} \), \( m \times L \) supernet parameters \( \Theta(m, L) \), search layer depth \( L \), choice blocks \( m \) per layer, training epochs \( N \), training data loader \( D \), loss function \( Loss \).

1. Initialize every \( \theta_{j,l} \) in \( \Theta(m,L) \).
2. For \( i = 1 \) to \( N \) do
   a. For \( data, labels \in D \) do
      i. For \( l = 1 \) to \( L \) do
         1. \( c_l = \) an uniform index permutation for the choices of layer \( l \)
         2. Clear gradients recorder for all parameters.
         3. Calculate gradients for \( \Theta_{c_l,1} \), \( \nabla \theta_{c_{l,2}}, \ldots, \nabla \theta_{c_{l,L}} \).
         4. Accumulate gradients for activated parameters, \( \nabla \theta_{c_{l,1}}, \nabla \theta_{c_{l,2}}, \ldots, \nabla \theta_{c_{l,L}} \).
      end for
   end for
   a. Update \( \theta_{(m,L)} \) by accumulated gradients.
3. End for

To reduce the bias from different training orders, we don’t perform back-propagation and update parameters immediately for each model as in the previous works [1, 16]. Instead, we define one supernet step as several back-propagation operations (BPs) accompanied by a single parameter update. In particular, given a mini-batch of training data, each of \( m \) single-path models is trained with back-propagation. Gradients are then accumulated across the selected \( m \) models but supernet’s parameters get updated only when all \( m \) BPs are done. This approach also doesn’t suffer from the ordering issue as each choice block is updated regardless of external learning rate strategies.

**Strict Fairness Analysis.** We now check whether our proposed Algorithm 1 satisfies Strict Fairness. By its design, each choice block is activated only once during a parameter update step. Thus \( Y_{i_1}' = Y_{i_2}' = \ldots = Y_{i_m}' \) holds. In particular, \( Y_{i_1}' = Y_{i_2}' = \ldots = Y_{i_m}' = n/m \) holds. Here, we write its

---

3 We use \( n \) to denote the total number of BPs operations to match Eq 1.
expectation and variance as follows:

\[ E(Y'_t) = n/m, \quad \text{Var}(Y'_t) = 0 \]  

(4)

Compared with Equation 1, the obvious difference lies in the variance. For the single-path approach with uniform sampling [16], the variance spreads along with \( n \), which gradually increases the bias. However, our approach calibrates this inclination and assures fairness at every step.

### 3.2. Stage 2: Searching with Supernet

For searching, we can either choose random search, vanilla evolutionary algorithms, or reinforcement learning. In practice, there are many requirements and objectives to achieve, e.g., inference time, multiply-adds, and memory costs, etc. This leads us to adopt a multi-objective solution. Besides, as the search space is too vast to enumerate all models, we need an efficient approach to balance the exploration and exploitation trade-off instead of a random sampling strategy. Here we adopt a searching algorithm from NSGA-II [10] with a small variation by using Proximal Policy Optimization [39] as the default reinforcing algorithm. The whole process is given in Algorithm 2. Benefitting from the fast evaluation of the weight-sharing supernet, we can achieve tremendous speed-up in terms of GPU days by two orders of magnitudes.

### 4. Experiments

#### 4.1. Setup

**Search Space.** We use several search spaces in this paper. The first one is that of NAS-Bench-201 [14], which is a new benchmark for NAS methods. Besides, We adopt two extra search spaces to compare with other NAS methods on ImageNet. (a) A search space for fairness analysis, which is based on MobileNetV2’s inverted bottleneck blocks as done in [4]. In particular, we retain the same amount of layers

\[ \text{with standard MobileNetV2 [38]. Convolution kernels are with the size in (3, 5, 7) and expansion rates are of (3, 6). We keep the number of filters unchanged. Besides, the squeeze-and-excitation block [20] is excluded. In total, it has a size of } 6^{19}. (b) A search space of 19 layers as ProxylessNAS [4], whose size spreads to } 6^{19}. This is to be on par with various state-of-the-art methods.

**Training Hyperparameters.** For NAS-Bench-201, we train the supernet for 50 epochs using a batch size of 128. The initial learning rate is 0.025 and decayed to zero by the cosine schedule.

For search space (a), we train the supernet for 150 epochs using a batch size of 256 and adopt a stochastic gradient descent optimizer with a momentum of 0.9 [42] based on standard data augmentation as [38]. A cosine learning rate decay strategy [30] is applied with an initial learning rate of 0.045. Moreover, We regularize the training with L2 weight

![Figure 4. Our strict fairness sampling and training strategy for supernet. A supernet training step t consists of training m models, each on one batch of data. The supernet gets its weights updated after accumulating gradients from each single-path model. All operations are thus ensured to be equally sampled and trained within every step t. There are \((6!)^8\) choices per step in our experiments.](image)

---

\[ \text{Algorithm 2 : Stage 2 - Search Strategy.} \]

**Input:** Supernet \( SN \), the number of generations \( G \), validation dataset \( D \)

**Output:** A set of \( K \) individuals on the Pareto front.

Train supernet \( SN \) with Algorithm 1.

Uniform initialization for the populations \( P_1 \) and \( Q_1 \).

for \( i = 1 \) to \( G \) do

\( R_i = P_i \cup Q_i \)

end for

\( F = \text{non-dominated-sorting}(R_i) \)

Pick \( N \) individuals to form \( P_{i+1} \) by ranks and the crowding distance.

\( M = \text{tournament-selection}(P_{i+1}) \)

\( Q_{i+1} = \text{crossover}(M) \cup \text{hierarchical-mutation}(M) \)

end for

Select \( K \) evenly-spaced models from \( P_{G+1} \) to train
decay ($4 \times 10^{-5}$). Our supernet is thus trained to fullness in 10 GPU days.

For search space (b), we follow the same strategy as above for training the supernet, but we adopt vanilla data processing as well as training tricks in [44] for stand-alone models. Regarding the stand-alone training of sampled models, we use similar training tricks. To be consistent with the previous works, we don’t employ tricks like cutout [11] or mixup [52], although they can further improve the scores on the test set.

4.2. Search Result

4.2.1 Search on ImageNet.

Figure 2 (supplementary) exhibits the resulting FairNAS-A, B and C models, which are sampled from our Pareto front to meet different hardware constraints. The quantitative result is shown in Table 1.

Notably, FairNAS-A obtains a highly competitive result 75.3% top-1 accuracy for ImageNet classification, which surpasses MnasNet-92 (+0.5%) and Single-Path-NAS (+0.3%). FairNAS-B matches Proxyless-GPU with much fewer parameters and multiply-adds. Besides, it surpasses Proxyless-R Mobile (+0.5%) with a comparable amount of multiply-adds.

Our models also reach a new state of the art when equipped with combined tricks such as Squeeze-and-Excitation [20], Swish activations [36] and AutoAugment [9]. Namely, FairNAS-A obtains 77.5% top-1 accuracy by using similar FLOPS as EfficientNet-B0. Even without mixed kernels [46], FairNAS-B (77.2%) outperforms MixNet-M (77.0%) with 11M fewer FLOPS. The most light-weight model FairNAS-C (76.7%) also outperforms EfficientNet-B0 with about 20% fewer FLOPS.

4.2.2 Search on NAS-Bench-201.

To be comparable with existing methods, we formulate our problem as a single objective one: finding the best model. Specially, we use a standard evolutionary algorithm in the second stage after the supernet is fairly trained. The result is shown in Table 2. Our method outperforms the other baselines in most datasets with the lowest search cost.

4.3. Transferred Results on CIFAR

To validate transferability of FairNAS models, we adapt the pre-trained models on ImageNet to CIFAR-10 and CIFAR-100 following the configuration of GPipe [21] and [23]. Table 3 shows that FairNAS models outperform the rest transferred models with higher top-1 accuracy.

4.4. Transferability on Object Detection

For object detection, we treat FairNAS models as drop-in replacements for RetinaNet’s backbone [27]. We follow the same setting as [27] and exploit MMDetection toolbox [5] for training. All the models are trained and evaluated on MS COCO dataset (train2017 and val2017 respectively) [28] for 12 epochs with a batch size of 16. The initial learning rate is 0.01 and decayed by 0.1 × at epochs 8 and 11.

The input features from these backbones to the FPN module are from the last depthwise layers of stage 2 to 5. The number of output channels of FPN is kept 256 as [27]. We also use $\alpha = 0.25$ and $\gamma = 2.0$ for the focal loss. Given longer training epochs and other tricks, the detection performance can be improved further. However, it’s sufficient to compare the transferability of various methods. The results are given in Table 4, we have the best transferability.

4.5. Transferability on Semantic Segmentation

We further evaluate FairNAS models as a feature extractor with DeepLabv3+[6] on the mobile semantic segmentation task, which confirms FairNAS backbones are competitive. All models are first pre-trained on COCO dataset [28], then coarsely trained on VOC2012 [15] extra annotated images.

---

Table 1. Comparison of mobile models on ImageNet. M: Memory cost at all sampled sub-models (SM), a single path or two paths (SP/TP), and a whole supernet (SN). *: from code, †: w/ SE and Swish, P: Number of parameters, ‡: reportedly 3× EfficientNet Training.
Table 2. Comparison of NAS-Bench-201 [14]. Averaged on three runs of searching.

Table 3. Comparison of state-of-the-art methods on CIFAR. \( ^{†} \): w/ SE and Swish. \( ^{⋆} \) based on our reimplementation.

Table 4. Object detection on COCO with various drop-in backbones. \( ^{†} \): w/ SE and Swish

Table 5. Semantic Segmentation on VOC 2012. OS: Output Stride.

5. Ablation Study

5.1. Model Ranking Capacity

As stated, the most important role of the supernet in the two-stage methods is to score models’ relative performance, and fine-tuned on VOC2012 fine annotated images. The results are given in Table 5 where the Atrous Spatial Pyramid Pooling (ASPP) module and multi-scale contextual information are not used. We also don’t flip inputs left or right during test.

Table 5. Semantic Segmentation on VOC 2012. OS: Output Stride

i.e. model ranking.

For supernet training, we set up three control groups that meet Expectation Fairness as our baselines. a) EF \( lr \), uniformly sampling one path and train \( k \) times, followed by parameter update. b) EF \( 1/6lr \): same as the first one except that the learning rate is scaled by \( 1/6 \). In practice, we set \( k = 6 \) to make it comparable to FairNAS. c) SPOS: an implementation of Single-Path One-Shot [16]. Other hyperparameters are kept the same. Note a), c) and FairNAS all use the same learning rate \( lr \).

We run the search pipeline for 200 epochs with a population size of 64, sampling 12,800 models in total. It takes only 2 GPU days due to accelerated evaluation. Due to high training cost, we sampled 13 models at approximately equal distances on the Pareto front and trained them from scratch to get the ranking, which is shown to the right of Figure 1. We observe that the FairNAS supernet gives a highly relevant ranking while Single-Path One-Shot [16] doesn’t. The training process of sampled models is plotted in Figure 3 (see supplementary).

We further adopt Kendall Tau [22] for the ranking analysis following a recent work [40] that evaluates NAS approaches. A method based on incomplete training reaches an average \( \tau \) of 0.474 [53]. Instead, we hit a new high record of the Kendall rank correlation coefficient \( \tau = 0.9487 \). We show our ranking comparison with baseline groups in Table 6. In general, methods with EF have a better ranking than those without EF, while SF is the best of all, which discloses the relevance of fairness to ranking in one-shot approaches.
5.2. Comparisons of Searching Algorithms

For the second-stage, we adopt multi-objective optimization where three objectives are considered: accuracies, multiply-adds, and the number of parameters. Specifically, we apply MoreMNAS with a minor modification in which PPO [39] is utilized instead of REINFORCE [43].

We construct several comparison groups that cover the main searching algorithms: a) **EA**: NSGA-II with reinforced mutation, b) **random search**, c) **Multi-objective RL**: MnasNet which uses PPO with a mixed multi-objective reward [44]. The results are shown in Figure 5, control groups generally align within our Pareto front and are constricted within a narrow range, affirming an excellent advantage in the MoreMNAS variant.

![Figure 5. Pareto front (last generation elists $P_{G+1}$) of the MoreMNAS variant (adopted) compared with Left: NSGA2 (EA-like baseline) with RL mutator and random search (random baseline), Right: MnasNet (RL baseline). Each samples 1,088 models.](image)

5.3. Component Contribution Analysis

Being a two-stage method, which stage contributes more to the final performance of the architecture? The experiment in NAS-Bench-201 has answered this question. To be complete, we further compare various supernet training strategies while fixing the second stage using the ImageNet dataset. Considering it’s not affordable to train the entire models from Pareto front, we impose an explicit constraint of maximum 400M FLOPS. The best models from One-Shot [2] (no EF) and SPOS (EF) [7] reach 74.0% and 74.6% top-1 accuracies, indicating that our result (75.3%) benefits mainly from the ranking capacity of the supernet in the first stage.

6. Discussions

6.1. Why Does Single-Path Training Work?

Our supernet generates a relatively small range of one-shot accuracies, from which we postulate that choice blocks be quite alike in terms of capacity. In fact, given an input of a chickadee image, the choice blocks of the first layer yield similar feature maps on the same channel, as shown in Figure 2. But how much do they resemble each other? We involve the **cosine similarity** [34] to measure the distance among various feature vectors. It ranges from -1 (opposite) to 1 (identical), where 0 indicates no correlation. In Figure 6, each $6 \times 6$ symmetric matrix shows the cross-block distances per channel, they are very similar (above 0.9).

![Figure 6. Cross-block channel-wise cosine similarity matrix on feature maps of 6 choice blocks in Layer 1. We observe that each choice block learns very similar features on the same channel](image)

In summary, the channel-wise feature maps generated by our supernet come with high similarities. We conclude that this important characteristic significantly stabilizes the whole training process. For layer $l+1$, its input is randomly from choice blocks in the previous layer $l$. As different choices have highly similar channel-aligned features, the random sampling constructs a mechanism mimicking feature augmentation, which boosts the supernet training.

6.2. Fairness Closes Supernet Accuracy gap

As discussed in Section 2.1, previous one-shot methods [2, 1] have a large accuracy gap between the one-shot and stand-alone models. We define it as supernet accuracy gap, $\lambda = |\delta_{\text{oneshot}} - \delta_{\text{standalone}}|$, where $\delta_{\text{oneshot}}$ is the accuracy range of one-shot models, and $\delta_{\text{standalone}}$ for stand-alone models. Ideally, $\delta_{\text{oneshot}}$ can be obtained by evaluating all paths from the supernet but not affordable since the search space is enormous. Instead, we can approximate $\delta_{\text{oneshot}}$ by covering a wide range of models. We randomly sample 1,000 models from our supernet, then we evaluate these models directly on the ImageNet validation set. Their top-1 accuracies (see Figure 1) range from 0.666 to 0.696, which leads to $\delta_{\text{oneshot}} = 0.03$, hence it reduces $\lambda$ as well.

7. Conclusion

In this work, we scrutinize the weight-sharing neural architecture search with a fairness perspective. Observing that unfairness inevitably incurs a severely biased evaluation of one-shot model performance, we propose two degrees of fairness enhancement, where Strict Fairness (SF) works best. Our supernet trained under SF then acts as a performance evaluator. In principle, the fair supernet can be incorporated in any search pipeline that requires an evaluator. To demonstrate its effectiveness, we adopt a multi-objective evolutionary backend. After searching proxylessly on ImageNet for 12 GPU days, we harvest three state-of-the-art models of different magnitudes nearby Pareto Optimality. Future works remain as to study fairness under heterogenous search spaces and to improve the evaluation performance of the supernet.
References


[41] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyanta, Jie Liu, and Diana Marculescu. Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2019. 1, 6, 7


