FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search

Xiangxiang Chu Bo Zhang Ruijun Xu
JixiangLi
Xiaomi AI Lab
{chuxiangxiang, zhangbo11, xuruijun, lijixiang }@xiaomi.com

Abstract

One of the most critical problems in neural architecture search is evaluation, which ranks various candidate models. Recently, there is a rapidly increasing interest in weight-sharing approaches. Although being very efficient with orders of magnitude faster than traditional methods, they are prone to misjudgments of candidate architectures.

In this paper, we first prove in current one-shot weight-sharing approaches, biased evaluation is inevitable due to inherent unfairness. To rectify it, we propose two levels of fairness constraints: ***expectation fairness*** *and* ***strict fairness****. Among several comparison groups, strict fairness works best both theoretically and empirically. Incorporating our supernet trained under such a constraint with a multi-objective evolutionary search algorithm, we obtain three state-of-the-art models on ImageNet. Especially, FairNAS-A attains* ***75.34%*** *top-1 accuracy. Finally, we give an in-depth analysis of the proposed method.*

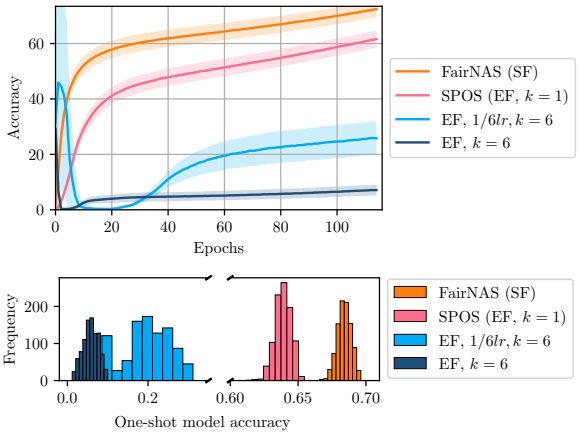

Figure 1. Supernet training on ImageNet. **Top:** The average top-1 accuracies of one-shot models, with variances shown in shades. **Bottom:** Histogram of validation accuracies from a stratified sample (960 each) of one-shot models. SF: Strict Fairness (proposed). EF: Expectation Fairness (baseline, sampling one path and perform k iterations each step), SPOS [7] (a case of EF where $k=1$). All methods use the same lr except for EF $1/6lr$ (light blue).

1. Introduction

The advent of neural architecture search (NAS) has brought deep learning into an era of automation [37]. Abundant efforts have been dedicated to searching within carefully designed search space [38, 21, 30, 17, 31]. 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 [38, 30]. Recent attention has been drawn to improve its efficiency via parameter sharing [3, 15, 20, 33].

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 [20, 15, 4, 27, 33] and others who decouple them into two stages, where the trained supernet is treated as an evaluator for final searching [3, 2, 7, 18].

Despite being widely utilized due to searching efficiency, weight sharing approaches are roughly built on empirical experiments instead of solid theoretical ground. There are several fundamental issues that 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 [3, 2]? 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) to fairly train the supernet (see Figure 1) as an evaluator, which in turn narrows the accuracy gap. Our analysis and experiments are conducted in a widely used search space [4, 33, 7, 27].

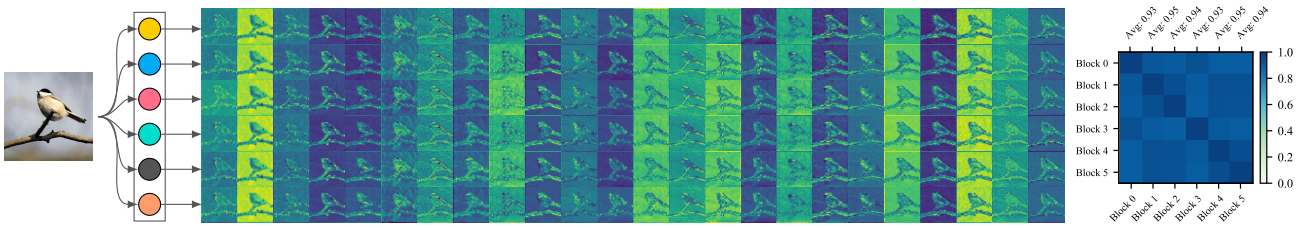

Figure 2. **Left:** Feature maps activated from 6 blocks of the first layer in our supernet trained with *strict fairness*. **Right:** Cross-block cosine similarity averaged on each channel. Each block learns very similar feature maps to others (all above 0.9).

The contributions can be summarized as follows.

Firstly, we prove it is due to *unfair bias* that the supernet misjudges submodels' performance. It is inevitable in current one-shot approaches [3, 2, 7].

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 capacity. Both outperform the existing unfair approaches while SF delivers a state-of-the-art ranking (τ) of 0.9487.

Thirdly, we unveil the root cause of the validity of single-path supernet training under our fairness perspective. That is, different choice blocks of the same layer learn similar feature maps on the corresponding channel, according to their high *cosine similarity* measure, see Figure 2.

Last but not the least, we incorporate the memory-friendly fair supernet with an EA-based multi-objective searching framework. We obtain three state-of-the-art networks within a single run at the cost of 12 GPU days, proxylessly on ImageNet. Among them, FairNAS-A achieves **75.34%** top-1 validation accuracy.

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 [20], the sampling policy $\pi(m, \theta)$ of an LSTM controller [8] and a sampled subnetwork m are alternatively trained. The final models are sampled again by the trained policy π, one who has the highest reward on a mini-batch of validation data is finally chosen. DARTS [15] combines the supernet training and searching within a bi-level optimization where all operations are 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 even worse solution. For instance, architectures from DARTS usually contain an excessive number of skip connections [34, 14], which damage the outcome performance. Therefore, the prior-learning DARTS is biased as per skip connections, while a random approach doesn't suffer [13]. 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 [3, 2, 7]. 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, and reinforcement learning.

SMASH [3] 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 (40%-90% vs. 25%-30% on CIFAR-100 [12]).

One-Shot [2] 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 (30%-90% vs. 92%-94.5% on CIFAR-10 [12]). SPOS [7] uniformly samples a single-path from the supernet during the training so that all architectures can be optimized simultaneously. However, we find it offers limited fairness and its supernet performance is somewhat restricted, see Figure 1.

Given the above biased supernets and the obvious range disparity, we are motivated to revisit one-shot approach under a novel fairness perspective.

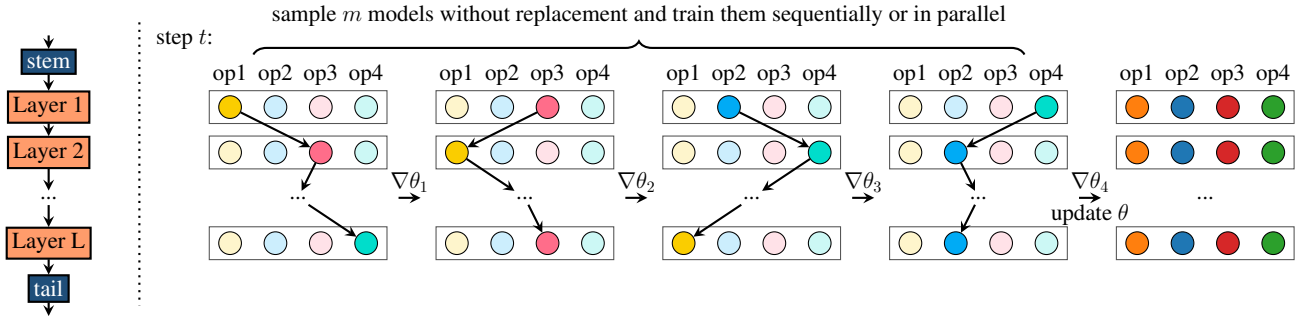

Figure 3. Our *strict fairness sampling and training* strategy for supernet. A supernet training step t consists of training m models, each on one batch of data. The supernet gets its weight updated after accumulating gradients from each model. All operations are thus ensured to be equally sampled and trained within every step t. There are $(6!)^{18}$ choices per step in our experiments[2].

2.2. 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 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 [15] train the supernet with all paths altogether, One-Shot [2] 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 $P(m, n, L)$.

2.2.1 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.

Definition 2. ***Expectation Fairness.*** *On the basis of Definition 1, let Ω 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_{l_i} be the number of times that the outcome l_i is observed (updated) over n trials.*

Then the expectation fairness is that for $P(m, n, L)$, $E(Y_{l_1}) = E(Y_{l_2}) = ... = E(Y_{l_m})$ holds, $\forall l \in L$.

2.2.2 Is Uniform Sampling Fair Enough?

Let us check a single-path routine [7] 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 is subject to the 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_{l_i} can be written as,

$$\begin{aligned} E(Y_{l_i}) &= n * p_{l_i} = n/m \\ Var(Y_{l_i}) &= n * p_{l_i}(1 - p_{l_i}) = \frac{n(m-1)}{m^2} \end{aligned} \tag{1}$$

That's to say, all choices share the same expectation and variance. Consequently, **uniform sampling meets Expectation Fairness** by Definition 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, but it's very unstable to train, shown as EF ($k = 6$) in Figure 1.

Even in [7] with uniform sampling where $k = 1$, 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 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

[2]It can be calculated by $(6^{19} * 5^{19} * 4^{19} * 3^{19} * 2^{19} * 1)/6! = (6!)^{18}$.

$m (m \geq 2)$. 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 stated formally as Lemma 1.

Lemma 1. *Regarding* $P(m, n, L)$, $\forall\ n \in \{x : x\%m = 0, x \in N_+\}$, $\lim_{n \to +\infty} p(Y_{l1} = Y_{l2} = ... = Y_{lm}) = 0$.

Proof. Let $f(m, n) = p(Y_{l1} = Y_{l2} = ... = Y_{lm})$.

$$f(m,n) = C_n^{\frac{n}{m}} C_{\frac{n(m-1)}{m}}^{\frac{n}{m}} ... C_{\frac{n}{m}}^{\frac{n}{m}} \frac{1}{m^n} = \frac{n!}{(\frac{n}{m}!)^m} \frac{1}{m^n} \quad \text{(a)}$$

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 [32].

$$\begin{aligned} \lim_{n \to +\infty} f(m,n) &= \lim_{n \to +\infty} \frac{n!}{(\frac{n}{m}!)^m \times m^n} \\ &= \lim_{n \to +\infty} \frac{\sqrt{2\pi n}(\frac{n}{e})^n}{\sqrt{2\pi \frac{n}{m}}^m (\frac{n}{e})^n} \\ &= \lim_{n \to +\infty} \frac{\sqrt{m}}{\frac{2\pi n}{m}^{\frac{m-1}{2}}} \\ &= 0 \end{aligned} \quad \text{(b)}$$

Q.E.D. □

Lemma 1 is somewhat counter-intuitive and thereby neglected in previous works. To throw light on this phenomenon, we plot this probability curve in Figure 4. 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.

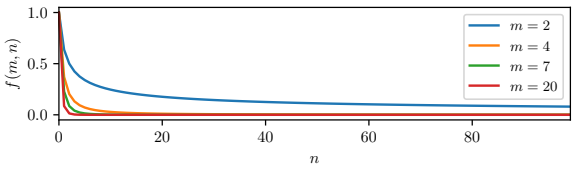

Figure 4. The function curve of $f(m, n)$ in Lemma 1. When sampling uniformly from m blocks for n trials, the probability of having an equal sampling number for each block quickly reaches zero.

2.2.3 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 3.

NAS Methods	M	C_t	C_s	EF	SF
SMASH [3]	SN	-	-	✗	✗
One-Shot [2]	SN	$16^{\ddagger}$	3.3	✗	✗
DARTS [15]	SN	$4^{\dagger}$	0	✗	✗
FBNet [33]	SP	20	0	✗	✗
ProxylessNAS [4]	TP	15	0	✗	✗
Single Path One-Shot [7]	SP	12	<1	✓	✗
Single-Path NAS [27]	SP	$1.25^{\ddagger}$	0	✓	✗
FairNAS (Ours)	SP	10	2	✓	✓

Table 1. Comparison of state-of-the-art weight-sharing NAS methods as per cost and fairness basis. M: Memory cost at a single path (SP), two paths (TP), and a whole supernet (SN). C_t, C_s: train and search cost measured in GPU days. EF: Expectation Fairness, SF: Strict Fairness. †: searched on CIFAR-10, ‡: TPU.

Definition 3. ***Strict Fairness.*** *Regarding P(m, n, L),* $\forall\ n \in \{x : x\%m = 0, x \in N_+\}$, $Y_{l_1} = Y_{l_2} = ... = Y_{l_m}$ *holds.*

Definition 3 imposes a constraint more demanding than Definition 2. That is, $p(Y_{l_1} = Y_{l_2} = ... = Y_{l_m}) = 1$ holds at any time. It seems subtle but it will be later proved to be crucial. Nevertheless, we have to be aware this is not ultimate fairness since different models have their own optimal initialization process and hyperparameters, which we single them out for simplicity.

2.2.4 A Fairness Taxonomy

As a summary, we compare current weight-sharing NAS methods based on the previous discussion in Table 1. Only single-path methods [27, 7] satisfy Expectation Fairness, while the rest doesn't. Next, we build our method FairNAS to meet both Expectation Fairness and Strict Fairness.

3. Fair Neural Architecture Search

Following One-Shot [2] and Single-Path One-Shot [7], we use the supernet to evaluate the performance of a multitude of models in our search space. Hence we can divide our fair neural architecture search into two stages: training the supernet and searching for competitive models.

3.1. Stage One: Supernet with Strict Fairness

We first propose a fair sampling and training algorithm to strictly abide by Defintion 3, see Algorithm 1. We use *uniform sampling without replacement* and sample m models at step t so that each choice block must be activated and updated only once, depicted by Figure 3.

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 [2, 7]. Instead, we define one *supernet step* as several back-propagation operations (BP) accompanied by a single parameter update. In particular, given a mini-batch of training data, each of m one-shot model is trained with back-

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 $Loss$
initialize every $\theta_{j,l}$ in $\Theta_{(m,L)}$.
for $i = 1$ **to** N **do**
  for $data, labels$ **in** D **do**
    for $l = 1$ **to** L **do**
      c_l = an uniform index permutation for the choices of layer l
    end for
    Clear gradients recorder for all parameters
    $\nabla\theta_{j,l} = 0, j = 1, 2, ..., m, l = 1, 2, ..., L$
    for $k = 1$ **to** m **do**
      Build $model_k = (c_{1_k}, c_{2_k}, .., c_{L_k})$ from sampled index
      Calculate gradients for $model_k$ based on $Loss$, data, labels.
      Accumulate gradients for activated parameters, $\nabla\theta_{c_{1_k},1}, \nabla\theta_{c_{2_k},2}, ..., \nabla\theta_{c_{L_k},L}$
    end for
    update $\theta_{(m,L)}$ by accumulated gradients.
  end for
end for

propagation. Gradients are then accumulated across the selected m models but supernet's parameters get updated only when all m BPs are done. Algorithm 1 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 $Y'_{l_1} = Y'_{l_2} = ... = Y'_{l_m}$ holds. In particular, $Y'_{l_1} = Y'_{l_2} = ... = Y'_{l_m} = n/m$ holds[3]. Here, we write its expectation and variance as follows:

$$\begin{aligned} E(Y'_{l_i}) &= n/m \\ Var(Y'_{l_i}) &= 0 \end{aligned} \tag{2}$$

Compared with Equation 1, the obvious difference lies in the variance. For the single-path approach with uniform sampling [7], the variance spread along with n, which gradually increases the bias. However, our approach calibrates this inclination and assures fairness at every step.

3.2. Stage Two: Supernet as an Evaluator

In stage two of searching, we utilize so-trained supernet to accurately evaluate each submodel's performance. It is important to notice that we differ from DARTS [15] as it learns priors towards a promising candidate, while we suppress the prior for fair evaluation. As there are many other requirements and objectives to achieve in real applications, e.g., inference time, multiply-adds, and memory costs, etc., we naturally adopt a multi-objective solution.

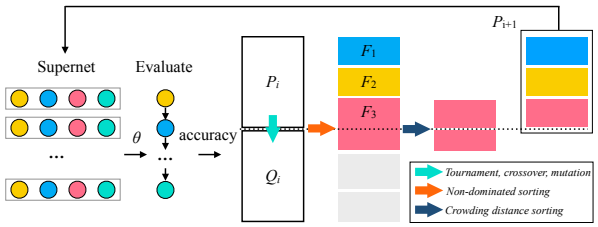

Figure 5. Evolutionary searching with the supernet trained with *strict fairness*. In each generation, candidate models in the current population inherit weights from the supernet for evaluation. Their estimated accuracies are fed into the searching pipeline as one of the objectives. The evolution loops till Pareto optimality.

Besides, 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 integrate MoreMNAS [5] by replacing its incomplete-training evaluator with our fairly trained supernet. By doing so, we achieve tremendous speed-up in terms of GPU days by two orders of magnitudes. Moreover, we use Proximal Policy Optimization as the default reinforcing algorithm [24]. The searching pipeline is detailed in Algorithm 3 (supplementary) and shown in Figure 5. Since our new approach is based on fair sampling and training of the supernet, we name the whole framework Fair Neural Architecture Search (FairNAS).

4. Experiments

4.1. Setups

Search Space. We adopt two types of search spaces, one for ranking analysis and the other for comparison with other NAS methods. **(a)** The search space for ranking analysis is designed based on MobileNetV2's inverted bottleneck blocks as done in [4]. In particular, we retain the same amount of layers with standard MobileNetV2 [23]. 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 [9] is excluded. In total, it has a size of 6^{16}. **(b)** To be on par with various state-of-the-art methods, we use the same search space of 19 layers as ProxylessNAS [4], whose size spreads to 6^{19}.

Dataset. We perform all experiments on ImageNet [22] and randomly select 50,000 images from the training set as our validation set (50 samples from each class). The remaining training set is used as our training set, while the

[3]Here we use n to represent the total number of BP operations to match Equation 1.

original validation set is taken as the test set to measure the final performance of each model.

Training Hyperparameters. 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 [28] based on standard data augmentation as [23]. A cosine learning rate decay strategy [16] is applied with an initial learning rate of 0.045. Moreover, We regularize the training with L2 weight decay (4×10^{-5}). To be consistent with the previous works, we don't employ any other tricks like dropout [26], cutout [6] or mixup [35], although they can further improve the scores on the test set. Our supernet is thus trained to fullness in 10 GPU days. Regarding the stand-alone training of sampled models, we use the same hyperparameters of the supernet.

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 [30] for stand-alone models.

4.2. Comparisons with State-of-the-art Methods

Memory consumption and fairness classification of various neural architecture search methods are reported in Table 1. It is worth to note we don't require extra tactics for stabilizing supernet or to prevent it from over-regularization, as apposed to [2].

Our candidate models FairNAS-A,B,C are sampled from our Pareto front (Section 3.2) with equal distance over multiply-adds to meet different requirements. The result is shown in Table 2. Although the latency is not considered as one of our objectives, our models still outperform or parallel with other methods in this regard.

We draw FairNAS models in Figure 7. FairNAS-A hits a new state-of-the-art result **75.34%** top-1 accuracy for ImageNet classification, which surpasses MnasNet-92 (+0.55%) and Single-Path-NAS (+0.38%). 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.

Three models seem to agree with high expansion rates and large kernels at the tail end, which enables full use of high-level features. FairNAS-A tends to choose a small expansion rate operator at the first two stages to cut down the computational cost, but it continues with a large expansion rate in the following stages when the feature resolution has been reduced. Unlike ProxylessNAS mobile, which prefers to append a large kernel and expansion rate after a downsampling operation, it's interesting to see that our FairNAS-B instead appreciates a larger kernel. FairNAS-C adopts lots of blocks with a small kernel 3×3, an expansion rate of 3 to keep as lightweight as possible, and it selects large kernels and expansion rates only at the tail to work with high-level features.

Methods	$\times+$ (M)	P (M)	Lat (ms)	μ_1 (%)	μ_5 (%)
MobileNetV2 [23]	300	3.4	78	72.0	91.0
NASNet-A [38]	564	5.3	183	74.0	91.6
MnasNet [30]	317	4.2	76	74.0	91.8
MnasNet-92 [30]	388	3.9	92	74.79	92.1
DARTS [15]	574	4.7	-	73.3	91.3
FBNet-B [33]	295	4.5	-	74.1	-
Proxyless-R [4]	320†	4.0	78	74.6	92.2
Proxyless GPU [4]	465†	7.1	124	75.1	-
Single Path One-Shot [7]	323	3.5	-	74.4	91.0
Single-Path NAS [27]	365	4.3	79	74.96	92.2
FairNAS-A (Ours)	388	4.6	104	**75.34**	**92.4**
FairNAS-B (Ours)	345	4.5	90	75.10	92.3
FairNAS-C (Ours)	321	4.4	83	74.69	92.1

Table 2. Comparison of mobile models on ImageNet. P: Params, μ_1, μ_5: Top-1 and Top-5 accuracies, †: Based on its published code. Mobile latencies (Lat) are measured on a Google Pixel 1 using a single large core of CPU.

4.3. Ablation Study

4.3.1 Expectation Fairness vs. Strict Fairness

For supernet training, we set up three control groups that meet *Expectation Fairness* as our baselines. a) **EF** $k = 6$, uniformly sampling one path and train k times, followed by parameter update. b) **EF** $k = 6$, $1/6lr$: same as the first one except that the learning rate is scaled by $\frac{1}{k}$. In practice, we set $k = 6$ to make it comparable to FairNAS. c) **EF** $k = 1$: an reimplementation of Single-Path One-Shot [7]. Other hyperparameters are kept the same. Note a), c) and FairNAS all use the same lr.

We draw average training accuracies of one-shot models over 120 epochs for these strategies compared with FairNAS, shown at the top of Figure 1. Our supernet that adheres to **Strict Fairness** steadily boosts the validation accuracy, which reaches up to 60% after 40 epochs. On the contrary, the first baseline **EF** has trouble stabilizing the training process. We suspect that the repeated k updates along a single activated path could cause overfitting on a mini batch of data. It pushes weights too far away so that oscillation is incurred on the next activated path. To validate this hypothesis, we inspect the second baseline, **EF** $1/6lr$, in which the learning rate is calibrated by scaling. It eliminates the oscillation but it demonstrates a quite slow learning speed. As for **EF** $k = 1$, We notice that average accuracies have been largely improved but they are still lower than FairNAS.

4.3.2 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 [24] is utilized instead of REINFORCE [29].

We construct several comparison groups that cover main searching algorithms: a) **EA**: NSGA-II with reinforced mutation, b) **random** search, c) **RL**: MnasNet which uses PPO with a mixed multi-objective reward [30]. The results are shown in Figure 6, control groups generally align within our Pareto front and are constricted within a narrow range, affirming an excellent advantage in the MoreMNAS variant.

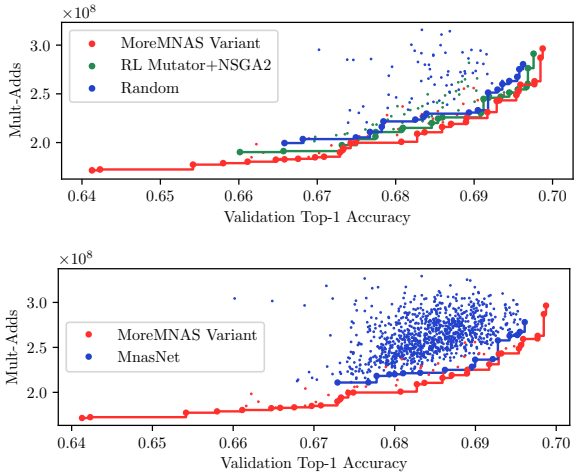

Figure 6. Pareto front of the MoreMNAS variant (adopted) compared with **Top:** NSGA2 (EA-like baseline) with RL mutator and random search (random baseline), **Bottom:** MnasNet (RL baseline). Each samples 1,088 models. The adopted MoreMNAS variant generally outperforms other mainstream searching methods. Last generation elitists P_{G+1} are lined.

4.4. Kendall Rank Analysis

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 in Figure 8. We observe that FairNAS supernet gives a highly relevant ranking while Single-Path One-Shot [7] doesn't. The training process of sampled models is plotted in Figure 10 (see supplementary).

We also adopt Kendall Tau [11] for the ranking analysis following a recent work [25] that evaluates NAS approaches. Kendall Tau (τ) measures the ranking relation between one-shot models and fully trained ones. The range of τ is from -1 to 1, meaning the rankings are totally reversed or completely preserved, whereas 0 means there is no correlation at all. Surprisingly, most weight-sharing approaches behave incredibly poorly on this metric [18, 20, 25]. A method based on incomplete training reaches an average τ of 0.474 [36]. 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 3. 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.

Methods	Fairness	τ
One-Shot [2]†	None	0
Uniform ($k = 6$, baseline)	EF	0.4871
Uniform ($k = 1, 1/6lr$)	EF	0.4871
SPOS [7] ($k = 1$)†	EF	0.6153
FairNAS‡	SF	**0.9487**

Table 3. Ranking ability of methods satisfying Expected Strictness vs. Strict Strictness. In total, 13 evenly-spaced models are fully trained on ImageNet to obtain their ground-truth ranking order. †: Reimplemented. ‡: With or without recalculating batch normalization, τ holds the same. For EF methods, k iterations are performed at each training step.

Is recalibration for batch normalization a must? Notably, the statistics for Batch Normalization [10] must be recalibrated in other approaches [2, 7] to boost their model ranking performances. We compare the ranking of FairNAS with and without this extra process. As Table 3 shows, both cases have the same $\tau = 0.9487$, which we attribute to the proposed fairness mechanism. Thus, we can evaluate the sampled architectures on the fly without the time-consuming recalibration.

4.5. Discussion about the Supernet Accuracy gap

As discussed in Section 2.1, previous one-shot methods [3, 2] have a large accuracy gap between the one-shot and stand-alone models. We call this difference as *supernet accuracy gap*, $\lambda = |\delta_{oneshot} - \delta_{standalone}|$, where $\delta_{oneshot}$ is the accuracy range of one-shot models, and $\delta_{standalone}$ for stand-alone models. Ideally, $\delta_{oneshot}$ can be obtained by evaluating all paths from the supernet. However, it's non-trivial to calculate because the search space is enormous. Instead, we can approximate $\delta_{oneshot}$ by covering a wide range of models.

In practice, we randomly sample 1,000 models from our supernet, then we evaluate these models directly on the ImageNet validation set. The histogram of their top-1 accuracies are reported at the bottom of Figure 1, ranging from 0.666 to 0.696. This leads to our $\delta_{oneshot} = 0.03$. By contrast, the accuracies of sampled models from the One-Shot [2] supernet trained on the CIFAR-10 dataset [12] have a range of $[0.3, 0.9]$, while stand-alone equivalents are within $[0.92, 0.945]$. Thus, we have its $\delta_{oneshot} = 0.6$ and $\lambda = 0.575$. It was hypothesized that this abnormal accuracy distribution is a result of learning only useful operations, the removal of which causes a large drop in accuracy [2]. According to our experiments and analysis, we blame the unfair training process for this gap. They may suffer from a *rich-get-richer* phenomenon, as noted in [1].

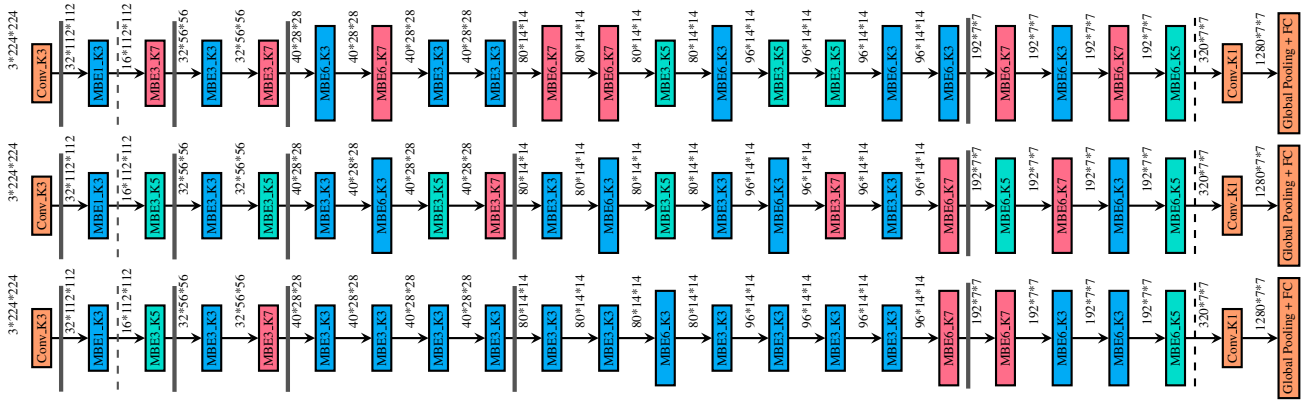

Figure 7. Architectures of FairNAS-A,B,C (from top to bottom). Ex_Ky means an expansion rate of x and a kernel size of y for its depthwise convolution. Grey thick lines refer to downsampling points. Dashed lines separate the stem and end layers from the backbone.

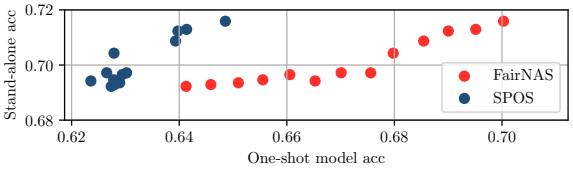

Figure 8. Top-1 accuracies of 13 fully trained stand-alone models (ground-truth) vs. one-shot models. Exact same models are used for both methods. FairNAS (SF) one-shot accuracies exhibit higher relation to their ground-truth, than SPOS [7] (EF).

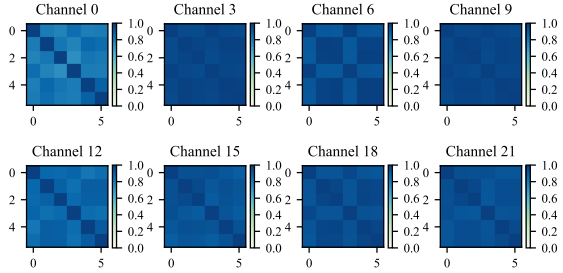

Figure 9. Cross-block channel-wise cosine similarity matrix on feature maps of 6 choice blocks in Layer 1. We observe that each choice block learns very similar features on the same channel.

5. Why Does Single Path Training Work?

First of all, our supernet generates a relatively small range of one-shot accuracies, from which we postulate that choice blocks be quite alike in 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 hereby involve the *cosine similarity* [19] to measure the distance among various feature vectors. It ranges from -1 (opposite) to 1 (identical), where 0 indicates no correlation. In Figure 9, each 6×6 symmetric matrix shows the distances on the same channel from block to block, they are mainly above 0.9 (very similar). Besides, we calculate the cross channel similarities and put the results in Figure 14 and 12 (both in the supplementary), which indicate that cross channels are quite irrelevant.

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 are randomly from choice blocks in previous layer l. As different choices have highly similar channel-aligned features, the random sampling constructs a mechanism mimicking **feature augmentation**, which boosts the stable training of the supernet. Moreover, this augmentation is however weak and thus doesn't alter the feature distribution too much, therefore, recalibrating batch normalization is no longer required for evaluation.

6. Conclusion

In this work, we scrutinize neural architecture search with a fairness perspective, especially for weight-sharing approaches. We prove that unfairness inevitably incurs a severely biased evaluation of one-shot model performance. We propose several degrees of fairness enhancement to alleviate such neglected unfairness, among which *Strict Fairness* (SF) works best. Our supernet trained under SF acts as a performance evaluator. Besides, satisfying SF is also memory friendly, there is only a single path trained at each step. To the best of our knowledge, we are the first to give a theoretical analysis of why single-path training is more beneficial.

In principle, the fair supernet can be incorporated in any NAS pipeline requiring an evaluator. Hence, a multi-

objective evolutionary searching backend is adopted to demonstrate its effectiveness. After searching proxylessly on ImageNet at cost of 12 GPU days, we harvest three state-of-the-art models of different magnitudes nearby *Pareto Optimality*. Future works remain as to study fairness in more complex search spaces, e.g., full of dense connections, and to design a fair approach to learn a robust prior for those without an explicit predictor like DARTS.

References

[1] George Adam and Jonathan Lorraine. Understanding Neural Architecture Search Techniques. *arXiv preprint. arXiv:1904.00438*, 2019. 7

[2] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and Simplifying One-Shot Architecture Search. In *International Conference on Machine Learning*, pages 549–558, 2018. 1, 2, 3, 4, 6, 7

[3] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. SMASH: One-Shot Model Architecture Search through HyperNetworks. In *International Conference on Learning Representations*, 2018. 1, 2, 4, 7

[4] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *International Conference on Learning Representations*, 2019. 1, 3, 4, 5, 6

[5] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Hailong Ma. Multi-Objective Reinforced Evolution in Mobile Neural Architecture Search. *arXiv preprint. arXiv:1901.01074*, 2019. 5, 11

[6] Terrance DeVries and Graham W Taylor. Improved Regularization of Convolutional Neural Networks with Cutout. *arXiv preprint. arXiv:1708.04552*, 2017. 6

[7] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single Path One-Shot Neural Architecture Search with Uniform Sampling. *arXiv preprint. arXiv:1904.00420*, 2019. 1, 2, 3, 4, 5, 6, 7, 8, 11

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[9] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-Excitation Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018. 5

[10] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 7

[11] Maurice G Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938. 7

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. Technical report, Citeseer, 2009. 2, 7

[13] Liam Li and Ameet Talwalkar. Random Search and Reproducibility for Neural Architecture Search. *Conference on Uncertainty in Artificial Intelligence*, 2019. 2

[14] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved Differentiable Architecture Search with Early Stopping. *arXiv preprint arXiv:1909.06035*, 2019. 2

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*, 2019. 1, 2, 3, 4, 5, 6

[16] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations*, 2017. 6

[17] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. NSGA-NET: A Multi-Objective Genetic Algorithm for Neural Architecture Search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 419–427, 2019. 1

[18] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural Architecture Optimization. In *Advances in Neural Information Processing Systems*, pages 7816–7827, 2018. 1, 7

[19] Hieu V Nguyen and Li Bai. Cosine Similarity Metric Learning for Face Verification. In *Asian Conference on Computer Vision*, pages 709–720. Springer, 2010. 8

[20] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. In *International Conference on Machine Learning*, 2018. 1, 2, 7

[21] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized Evolution for Image Classifier Architecture Search. *International Conference on Machine Learning, AutoML Workshop*, 2018. 1

[22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 5

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 5, 6

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint. arXiv:1707.06347*, 2017. 5, 6, 11

[25] Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the Search Phase of Neural Architecture Search. *arXiv preprint. arXiv:1902.08142*, 2019. 7

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 6

[27] 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. *arXiv preprint. arXiv:1904.02877*, 2019. 1, 4, 6

[28] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the Importance of Initialization and Momentum

in Deep Learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013. 6

[29] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018. 6

[30] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 1, 6, 7

[31] Mingxing Tan and Quoc V Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning*, 2019. 1

[32] Ian Tweddle. *James Stirlings Methodus Differentialis: An Annotated Translation of Stirlings Text*. Springer Science & Business Media, 2012. 4

[33] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 1, 4, 6

[34] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and Robustifying Differentiable Architecture Search. *arXiv preprint arXiv:1909.09656*, 2019. 2

[35] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond Empirical Risk Minimization. In *International Conference on Learning Representations*, 2018. 6

[36] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial Distribution Learning for Effective Neural Architecture Search. In *International Conference on Computer Vision*, 2019. 7

[37] Barret Zoph and Quoc V Le. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations*, 2017. 1

[38] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning Transferable Architectures for Scalable Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 2018. 1, 6

Algorithm 2 Speed up in parallel under some conditions.

Input:
initialize every $\theta_{j,l}$ in $\Theta_{(m,L)}$.
for $i = 1$ **to** N **do**
  for $data, labels$ **in** D **do**
    for $l = 1$ **to** L **do**
      c_l = an uniform index permutation for the choices of layer l
    end for
    Clear gradients recorder for all parameters
    $\nabla\theta_{j,l} = 0, j = 1, 2, ..., m, l = 1, 2, ..., L$
    for $k = 1$ **to** m **(in parallel) do**
      Build $model_k = (c_{1_k}, c_{2_k}, .., c_{L_k})$ from sampled index
      Calculate gradients for $model_k$ based on $Loss$, data, labels.
      update $\theta_{(m,L)}$.
    end for
  end for
end for

A. Algorithms

A.1. Paralleled Training of Supernet

Algorithm 1 (in main text) can reach at least the same training efficiency as uniform sampling approach [7]. But it usually demonstrates faster training speed in practice because the sampled mini-batch data are reused to perform BP operations for m times, thus alleviating the data generation overhead for the underlying data loader and fully utilizing the power of GPU or TPU machines.

In fact, it can be further accelerated under some conditions described by Algorithm 2 (supplementary), ideally linear to the number of paralleled workers. When the whole supernet needs to be explored like Figure 3 (in main text) and each choice block within a layer has its own parameters, training m models at each step can be absolutely decoupled into m tasks so that both back-propagation and parameter update can be run in parallel, i.e., synchronized update for parameters is no longer needed. Most of the deep learning frameworks can support paralleled training of such type.

A.2. Evolutionary Searching Pipeline

With our supernet fairly trained as a model evaluator, we adopt an evolutionary-based algorithm for searching, detailed in Algorithm 3. Generally, it is built on the ground of MoreMNAS [5] by replacing its incomplete-training evaluator with our fairly trained supernet. FairNAS supernet exhibits tremendous speed-up in terms of GPU days by two orders of magnitudes. We also use Proximal Policy Optimization as the default reinforcing algorithm [24].

Algorithm 3 : 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**
  $R_i = P_i \cup Q_i$
  for all $p \in R_i$ **do**
    Evaluate model p with inherited weights from SN on D
  end for
  $F =$ non-dominated-sorting(R_i)
  Pick N individuals to form P_{i+1} by ranks and the crowding distance.
  $M =$ tournament-selection(P_{i+1})
  $Q_{i+1} =$ crossover$(M) \cup$ hierarchical-mutation(M)
end for
Select K evenly-spaced models from P_{G+1} to train

B. Experiment Details

B.1. Hyperparameters for MoreMNAS variant

We list the hyperparameters for the adopted MoreMNAS [5] variant in Table 4. It has a population N of 64 models. It also takes a hierarchical mutation strategy. Respectively, p_{rm}, p_{re}, p_{pr} indicate probabilities for random mutation, reinforce mutation and prior regulator, where p_{re} again is divided into p_{K-M} for *roulette wheel selection*, and p_M for reinforced controller.

Table 4. Hyperparameters for the whole pipeline.

ITEM	VALUE	ITEM	VALUE
POPULATION N	64	MUTATION RATIO	0.8
p_{rm}	0.2	p_{re}	0.65
p_{pr}	0.15	p_M	0.7
p_{K-M}	0.3		

B.2. Training of stand-alone models

We picked 13 models for full train whose one-shot accuracies are approximately evenly spaced, ranges in $[0.641, 0.7]$. They are trained with the exactly same hyperparameters as the supernet. Their corresponding stand-alone accuracies are within $[0.692, 0.715]$. Figure 10 plots the training process, from which we observe the ranking of one-shot models are generally maintained. The model-metas of these 13 models are listed in Table 5. Besides, the mapping from basic element of model-meta to a search operation is given in Table 6.

B.3. Evolution Process

FairNAS evolution process based on MoreMNAS variant [5] is shown in Figure 11. At each generation, 64 models are

Table 5. Model-metas of stand-alone models.

IDX	MODEL META
0	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
2	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0]
3	[0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0]
4	[0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0]
5	[0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0]
6	[0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 2, 0]
7	[0, 1, 0, 1, 1, 4, 1, 1, 1, 1, 1, 0, 0, 1, 2, 0]
8	[0, 1, 4, 1, 0, 3, 1, 1, 1, 1, 1, 1, 0, 2, 0, 0]
9	[0, 1, 0, 0, 1, 5, 1, 1, 0, 5, 1, 1, 0, 1, 2, 3]
10	[3, 1, 4, 1, 3, 4, 1, 4, 1, 3, 1, 1, 3, 1, 2, 0]
11	[0, 1, 4, 3, 1, 3, 1, 1, 1, 3, 4, 1, 3, 1, 2, 3]
12	[1, 5, 3, 2, 1, 4, 3, 4, 1, 5, 1, 1, 3, 5, 5, 3]

Table 6. Mapping between model-meta index and operations.

MODEL META IDX	KERNEL	EXPANSION RATE
0	3	3
1	5	3
2	7	3
3	3	6
4	5	6
5	7	6

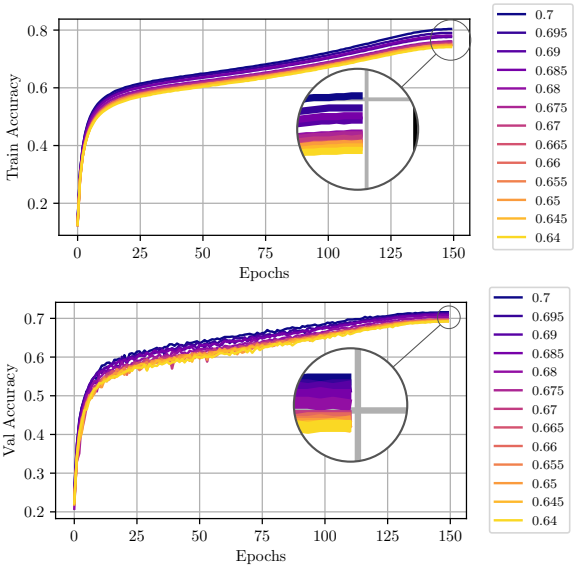

Figure 10. Train and validation accuracies (ground truth) of all 13 stand-alone models when being fully trained with the same hyper-parameters. Lines are labelled with corresponding one-shot accuracies (predicted) sorted in descending order (as reflected by color gradient).

evaluated by our fair supernet, after 200 generations, the evolution converges, the Pareto-front is shown in bright yellow,

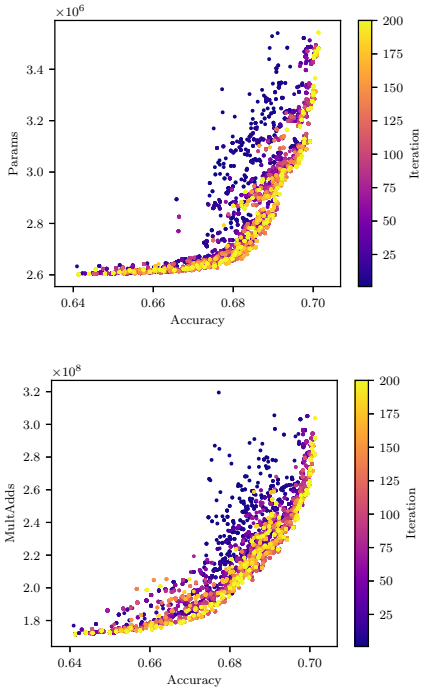

Figure 11. FairNAS evolution process of 200 generations, with 64 models sampled in each generation. Number of parameters, multiply-adds are charted with top-1 accuracies on the ImageNet validation set.

each dot represents a candidate network.

C. Single-Path Training Analysis

We have shown that cross-block features in the same channel are quite alike. Here we give all channel results in Figure 13. How about cross-channel features? As a result, cross-channel feature maps of the same block are nearly irrelevant, see each 24×24 matrix in Figure 12. The same result holds also for cross-channel features of different blocks, see Figure 14.

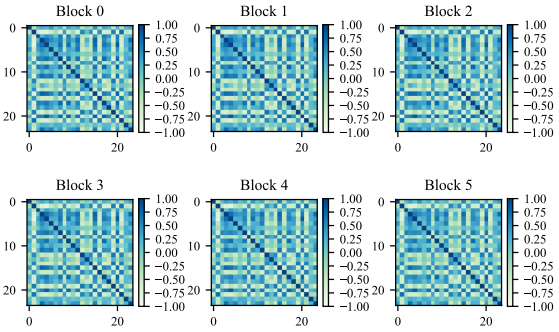

Figure 12. Cross-channel cosine similarity matrix on feature maps of each choice block in Layer 1. Different channels for the same block learn very diverse features.

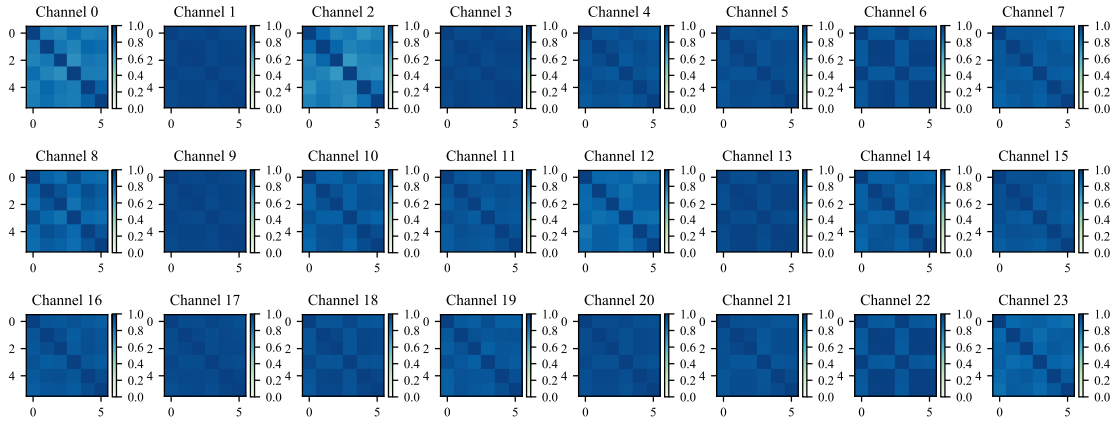

Figure 13. Channel-wise cosine distance on feature maps of 6 choice blocks in Layer 1. We observe that each choice block learns very similar features.

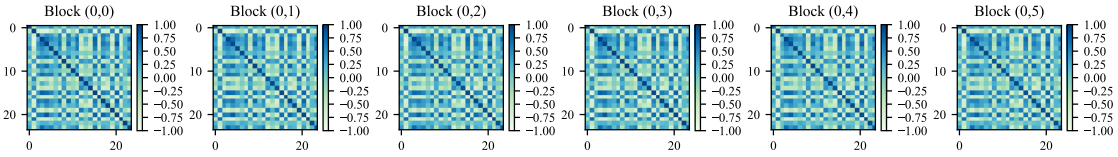

Figure 14. Cross-channel cosine similarity matrix on feature maps from choice block 0 to other blocks in Layer 1. Different channels of various blocks learn very diverse features.