FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search

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

Abstract

The ability to rank models by its real strength is the key to Neural Architecture Search. Traditional approaches adopt an incomplete training for such purpose which is still very costly. One-shot methods are thus devised to cut the expense by reusing the same set of weights. However, it is uncertain whether shared weights are truly effective. It is also unclear if a picked model is better because of its vigorous representational power or simply because it is overtrained.

In order to remove the suspicion, we propose a novel idea called Fair Neural Architecture Search (FairNAS), in which a strict fairness constraint is enforced for fair inheritance and training. In this way, our supernet exhibits nice convergence and very high training accuracy. The performance of any sampled model loaded with shared weights from the supernet strongly correlates with that of stand-alone counterpart when trained fully. This result dramatically improves the searching efficiency, with a multi-objective reinforced evolutionary search backend, our pipeline generated a new set of state-of-the-art architectures on ImageNet: FairNAS-A attains 75.34% top-1 validation accuracy on ImageNet, FairNAS-B 75.10%, FairNAS-C 74.69%, even with lower multi-adds and/or fewer number of parameters compared with others. The models and their evaluation code are made publicly available online[1].

1. Introduction

The advent of neural architecture search (NAS) has brought deep learning into an era of automation [29]. Abundant efforts have been dedicated to various methods that guide the search within carefully designed search space [30, 16, 24, 13, 25]. As the conventional NAS approaches evaluate an enormous amount of models based on resource-devouring training, recent attention is drawn to improve the estimation efficiency via parameter sharing [3, 11, 15, 27]. For instance in differential architecture search [11], a super network is built with categorically parameterized operations, whose output is a mixture of all operations. The goal is to reduce the validation loss through joint optimization of operations and weights. The target architecture is then induced from the set of operations based on their mixing probabilities. In doing so, all possible models are optimized with the same set of weights, remarkably cutting down the training cost.

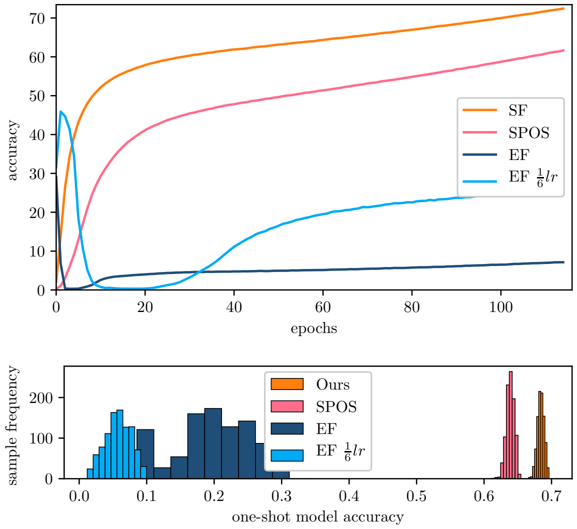


Figure 1. Training process of supernets. **Top:** The average top-1 accuracy on ImageNet training set of one-shot models under different fairness constraints. **Bottom:** Histogram of accuracies on ImageNet validation set from a stratified sample (960 each) of one-shot models. Note the statistics of batch normalization in each model is not recalculated. SPOS: Single-Path One-Shot [7], EF: Expectation Fairness, SF: Strict Fairness (Ours).

The doubt, however, arises about whether such shared weights are still effective across a wide range of architectures. One-shot methods like [3, 2] try to ensure that models with such inherited weights can best predict the true accuracy. Moreover, in view of huge memory consumption of a super network, current one-shot methods train only one

[1] http://github.com/fairnas/FairNAS

model at each optimization step [4, 22, 7].

We continue diving into the one-shot approach by discussing its fairness problem, which is rarely mentioned in previous works. In this paper, we present Fair Neural Architecture Search, which clears up two fundamental unanswered problems,

- Is it really fair enough to tell the difference of submodels with the training of a one-shot supernet and the sampling techniques presented in previous methods?
- How can we quickly rank models by its performance with a strong belief?

Our contributions can be summarized in several aspects. Firstly, observing that more frequently trained models certainly have advantages to perform better, our framework strengthens the one-shot method by complying with what we call *strict fairness* for both sampling and training. We found that fair training of supernet is so critical that no operations are incredibly inferior to others. They behaved poorly in previous works like [15, 2] just because they are unfairly trained, hence it was due to a *rich-get-richer* phenomenon, as noted in [1].

Secondly, as shown in Figure 1, our experiments attest that under *strict fairness* constraint, the average accuracies enjoy a steady improvement rather than oscillation. The range of accuracies from a stratified sample of one-shot models has been substantially narrowed compared with [2]. This is significant progress since now we can evaluate a model rapidly, at the same time, accurately.

Thirdly, although the one-shot approach tremendously speeds up the estimation, we are confronted with multiple real-world constraints and vast search space. We choose a multi-objective NAS method [5] to serve our need. To prove its efficiency, we conducted several ablation studies showing that it is exceptional compared with random, pure RL and EA methods.

Lastly, with our pipeline, a new set of state-of-the-art architectures is produced for the classification task on ImageNet: FairNAS-A achieves 75.34% top-1 validation accuracy, FairNAS-B 75.10%, FairNAS-C 74.69%. All three models have comparable sizes with previous works.

2. Related Works

The most time-consuming part of common neural architecture search is that in order to rank thousands of sampled architectures, they have to be trained from scratch for certain epochs. Reasoning on whether it is feasible to train a single set of weights for many heterogeneous architectures has led to a new paradigm called *one-shot* model architecture search. The key to these approaches lies in that the performance of candidate models can be highly predictable.

SMASH [3] devised a stand-alone hypernetwork that generates weights for all possible architectures in the search space. It is trained to reduce the loss of each sampled model on some mini-batch of dataset. A single train of the hypernetwork is good once and for all, any candidate network can then be evaluated directly. However, the design of a hypernetwork requires delicate expertise to obtain a strong correlation between the true performance of a sampled model and that with generated weights.

One-Shot [2] made a step further by constructing a one-shot model that covers all operations in search space. At the evaluation stage, a child network is emulated by zeroing out other incoming connections. Noticing the different impact that various operations pose on the prediction capability, the model automatically focuses on important ones. Unfortunately, it relies on hyperparameters like drop-out rate and the number of operation choices for each block, which renders this method less robust while specific care has to be taken to stabilize the training and to prevent over-regularization. Moreover, such a one-shot model suffers a memory explosion problem as it subsumes all architectures, it simply becomes too big to train when the search space grows. Many one-shot variants appear and the designs and training strategies of super-net can be roughly classified into three branches: training the whole super-net based on drop-connect tricks [2], training the choice weights and network parameters jointly (alternately) [4, 11, 27], training in single-path way [7, 22].

Our work is most closely related to single path one-shot [7], which can be considered as an extreme case of one-shot [2], where all paths are dropped but one. It stresses the principle that all architectures have to be optimized simultaneously. At each step of optimization, only a single model has its weights updated. It also adopts an empirical random sampling on a fixed distribution of architectures, which decouples the previous joint optimization of both weight and architecture.

3. Training Fairness of One-Shot Methods

3.1. The Bias of One-Shot Proxies

In a way, all one-shot methods are different proxies for performance predictor of any single path model within a defined search space. A good proxy should neither severely overestimate nor underestimate the score of models. To our knowledge, this topic hasn't been deeply studied and most previous works simply concentrate on searching several models with good scores. In order to reduce the prior bias from the process of supernet training, a basic and direct requirement can be defined as,

Definition 1. *Training a supernet is in the same way how a submodel is trained. Every submodel has an equal opportunity to be sampled and trained within each iteration.*

It's trivial to see that only single path one-shot approaches meet the above definition.

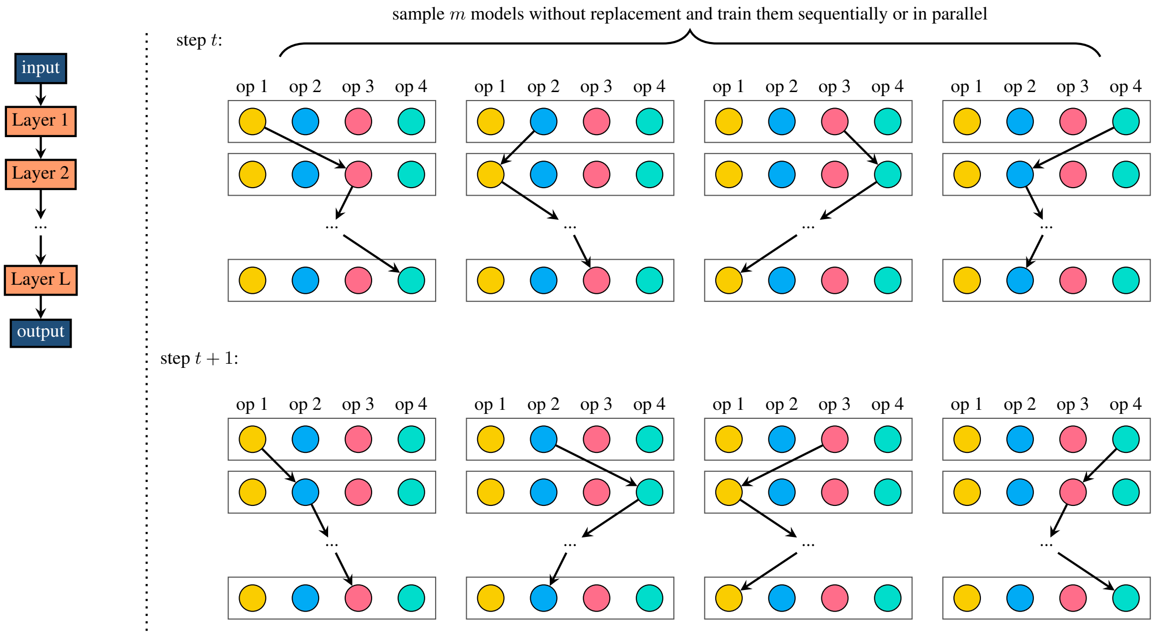

sample m models without replacement and train them sequentially or in parallel
step t:
step $t+1$:
input
Layer 1
Layer 2
...
Layer L
output
op 1
op 2
op 3
op 4

Figure 2. Our one-shot architecture and sampling strategy. All operations are equally trained within one certain step.

Another interesting fact is that a model getting more frequently trained is more likely to perform better at the stage of model evaluation. The key point of one-shot approaches lies in the training process of the supernet and all subsequent steps depend on its quality. In fact, training fairness for all choice blocks is one of the most critical guarantees for such quality.

3.2. Problem Definition

For a common search space with L layers, each layer contains several choices. Without loss of generality, we suppose that the number of choices within each layer is M. One model can be generated by sampling a block from each layer sequentially, which can be represented by a tuple $(l_1, l_2, ..., l_L)$. Besides, the parameters of the supernet are updated N times in total. Therefore, we can define this problem using a tuple $P(M, N, L)$.

It's easy to see that at each step of supernet training, only the parameters of correspondingly activated choice blocks are updated. Loosely speaking, as this update aims to decrease the loss on a mini-batch of data, it creates a bias while it helps the activated choice block to score higher than those non-activated ones. Therefore, a straight and basic requirement we call **Expectation Fairness** to reduce this bias can be defined below:

Definition 2. *On the basis of Definition 1, let Ω be the sampling space containing m basic events $\{x_{l1}, x_{l2}, ..., x_{lm}\}$, which are generated by selecting a block from layer l_l with m choice blocks. Let Y_{li} be the number of times that the outcome x_{li} is observed (updated) over n trials.*

Then the expectation fairness is that for $P(m, n, L)$, $E(Y_{l1}) = E(Y_{l2}) = ... = E(Y_{lm})$ holds.

3.3. Training Fairness and Bias

Let us check the single-path routine over this definition. Selecting a block from layer l_l with m choice blocks is subject to the categorical distribution. We can simplify the expectation calculation because sampling on any layer l_i is independent of operations on other layers. In case of the uniform sampling, each basic event occurs with equal probability $p(X = x_{li}) = \frac{1}{m}$. The expectation and variance can be written as,

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

That's to say, different variables share the same expectation and variance. Consequently, uniform sample meets Definition 2 and seems superficially fair to various choices.

However, **Expectation Fairness** alone is not enough. For example, we can randomly sample a model and keep it training for K times, then switch to another. This procedure also meets Definition 2, Obviously, it's unstable to sample and train in this way.

Even with uniform single path approach where $K = 1$, there is still a new issue about the sampling order. As an example, for a sequence of choices (M_1, M_2, M_3), there is an inherent training order $M_1 \rightarrow M_2 \rightarrow M_3$. Since each model is usually trained by the back-propagation algorithm, the parameters related to M_1 in the supernet are immediately updated and the parameters of M_2 are renewed next while carrying the effect of the former update and so for M_3. A permutation of (M_1, M_2, M_3) does comply with **Expectation Fairness** but yields different results. Besides, if the learning rate is changed within the sequence, the situation becomes even more complicated. Clearly, we need a constraint of more strength to address the bias problem more thoroughly.

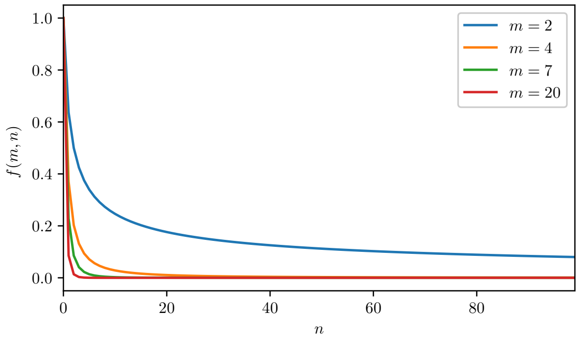


Figure 3. Plot of $f(m, n)$.

Let us simplify it with the case $P(2, n, 1)$[2], where $L = 1$ and $m = 2$, and the total sampling number n is even. We thus derive Lemma 1.

Lemma 1. *Regarding* $P(2, n, 1)$, $\lim_{n\to+\infty} p(Y_{11} = Y_{12}) = 0$.

Proof. Let $f(m, n) = p(Y_{l1} = Y_{l2} = ... = Y_{lm})$ for the general case and we solve $f(2, n)$ here. $Y_{11} + Y_{12} = n$. Thus, $Y_{11} = Y_{12}$ is equivalent to $Y_{11} = \frac{n}{2}$. We only prove the case $\lim_{n\to+\infty} p(Y_{11} = \frac{n}{2}) = 0$. We have,

$$p(Y_{11} = \frac{n}{2}) = C_n^{\frac{n}{2}} \frac{1}{2^n} = \frac{n!}{(\frac{n}{2}!)^2 \times 2^n} \tag{a}$$

Firstly, we prove the existence of limitation, $f(n)$ strictly decreases monotonically with n and $p(Y_{11} = \frac{n}{2}) \geq 0$, therefore, its limitation exists.

Secondly, we calculate its limitation using equivalent infinity replacement based on Stirling's approximation about factorial [26].

$$\begin{aligned}\lim_{n\to+\infty} p(Y_{11} = \frac{n}{2}) &= \lim_{n\to+\infty} \frac{n!}{(\frac{n}{2}!)^2 \times 2^n} \\ &= \lim_{n\to+\infty} \frac{\sqrt{2\pi n}(\frac{n}{e})^n}{2\pi n(\frac{n}{e})^n} \\ &= \lim_{n\to+\infty} \frac{1}{\sqrt{2\pi n}} \\ &= 0\end{aligned} \tag{b}$$

□

The conclusion from Lemma 1 is a bit counter-intuitive because it claims that it's impossible to distribute the same training opportunity for two choice blocks within a layer in practice. To throw light on this phenomenon, we plot the function curve in Figure 3. $f(2, n)$ decreases below 0.2 when $n \geq 20$ and $f(4, n)$ decreases much faster, which is below 0.1 when $n \geq 4$. Considering that a typical number of parameter updates for a supernet is more than 10^6 times, most one-shot researches with $4 \leq m \leq 10$ violate the training fairness severely. Similarly, we can draw a more general conclusion as Lemma 2, with its proof listed in Section B.

Lemma 2. *Regarding* $P(m, n, L)$, $\forall\ n \in \{x : x\%m = 0, x \in N_+\}$, $\lim_{n\to+\infty} p(Y_{l1} = Y_{l2} = ... = Y_{lm}) = 0$.

Our insights come from this neglected unfairness. Here, we propose a more rigorous requirement called **Strict Fairness** for both sampling and training fairness, which can be defined in the following definition.

Definition 3. *Regarding P(m, n, L),* $\forall\ n \in \{x : x\%m = 0, x \in N_+\}$, $Y_{l1} = Y_{l2} = ... = Y_{lm}$ *holds.*

Definition 3 imposes a stricter constraint than Definition 2. It ensures the parameter of every choice block be updated the same amount of times at any stage, i.e., $p(Y_{l1} = Y_{l2} = ... = Y_{lm}) = 1$ holds at any time. The next section introduces our method to meet this definition which reduces the bias accordingly. Strictly speaking, it's almost impossible to create absolute fairness because different models have their own optimal initialization process and hyper-parameters, which are not well investigated by now.

4. Fair Neural Architecture Search

4.1. Fair Sampling and Training of the Supernet

We propose a fair sampling and training algorithm (Algorithm 1) to strictly abide by Defintion 3. We use uniform sampling without replacement and sample m models at one step so that each choice block must be activated once and once only per update, as depicted in Figure 2.

[2] L doesn't affect our analysis owing to the independence of choice over different layers.

Algorithm 1 Fair Sampling and 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
```

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 redefine one step as several back-propagation operations (BP) along with parameter update only once. To be exact, given a mini-batch of training data, a total of m back-propagation operations are triggered and each one-shot model has its own BP. Gradients are then accumulated across the selected m models but parameters are updated only once after the accumulation has ended. A byproduct benefit of Algorithm 1 is that each choice block is updated regardless of external learning rate strategies.

4.2. Fairness Analysis

Let Y'_{lk} have the same meaning as before based on our fair algorithm. Since Algorithm 1 is specially designed to make sure each choice block is activated once and once only during a parameter update step. Thus $Y'_{l1} = Y'_{l2} = ... = Y'_{lm}$ holds, which meets the requirement of **Strict Fairness**. In particular, $Y'_{l1} = Y'_{l2} = ... = Y'_{lm} = n/m$ holds[3]. Here, we calculate its expectation and variance as follows:

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

Compared with Equation 1, the obvious difference lies in the variance. For the uniform single path one-shot approach, the variance expands along with n, which worsens fairness violation and increases the bias. However, our approach calibrates this inclination and assures fairness at every step.

4.3. Efficiency Analysis

Algorithm 1 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, Algorithm 1 can be further accelerated under some conditions described by Algorithm 2 in Appendix A (ideally linear to the number of paralleled workers). When the whole supernet needs to be explored like Figure 2 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.

4.4. Supernet from the Respective of Model Score Predictor

The above sections address how to reduce the bias introduced in the training process of the supernet. So far, the supernet alone cannot deliver favorable models directly. In fact, there are many other requirements and objectives to accomplish in real applications, such as inference time, multiply-adds, and memory cost, etc.

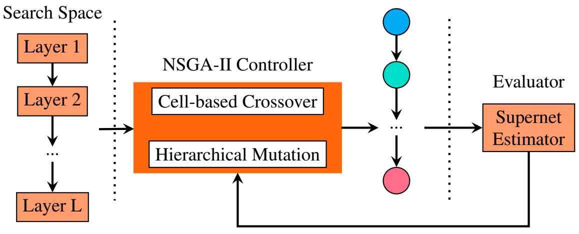


Figure 4. The Structure of Fair Neural Architecture Search

In general, 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. Moreover, we need a powerful algorithm to deal with a multi-objective problem (MOP). Here we integrate MoreMNAS [5] to solve these problems by replacing

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

its incomplete-train evaluator with our fairly trained supernet. In this way, we can achieve speed-up by two orders of magnitudes than MoreMNAS in terms of GPU days. Unlike [5], we use Proximal Policy Optimization as the default reinforcing algorithm [19]. Since our new approach is based on fair sampling and training of the supernet, we call the whole framework Fair Neural Architecture Search (FairNAS), shown in Figure 4.

5. Accuracy Gap between One-Shot and Stand-alone Models

The most exciting advantage of the one-shot approach is that the supernet can be used to differentiate the quality of various models. However, such a critical topic hasn't been deeply investigated. A recent work [20] evaluates the search phase based on Kendall Tau [10], which measures the ranking relation between models during the search phase and fully trained ones. It can be simply calculated by counting the number of concordant and discordant rankings.

$$\tau = \frac{2(N_{concordant} - N_{discordant})}{n(n-1)} \tag{3}$$

Kendall Tau ranges from -1 to 1, meaning the rankings are totally reversed or completely preserved, whereas 0 means there is no correlation at all. While it seems ideal, most of the studies on neural architecture search behave incredibly poorly with this metric [14, 15], with the best reported τ be 0.282 [20].

For a group of sampled models from the supernet, in our case on the classification task, we also notice a gap of accuracy range between the search phase and full-train stage, i.e., the accuracies of sampled one-shot models usually spread much wider than those of stand-alone ones which are completely trained. We define this gap as follows,

$$\begin{aligned} M_{gap} &= |R_{search} - R_{train}| \\ R_{search} &= \max(P_{search}) - \min(P_{search}) \\ R_{train} &= \max(P_{train}) - \min(P_{train}) \end{aligned} \tag{4}$$

where R_{search} is the difference of top-1 accuracies P_{search} between the best performing model and the worst one at the search stage. For the one-shot approach, it's can be obtained by enumerating all paths from the supernet to render models and subtracting its maximum with minimum. However, it's nontrivial to calculate the exact value of M_{gap} because the space is way too expansive to reckon all models[4]. Instead, we approximate M_{gap} by sampling a large set of models, see Figure 1.

Let's discuss Equation 4 in further detail. First, it is not necessary to pursue $M_{gap} = 0$, instead we only expect the ranking to be instructive to discriminate models. Second, R_{search} can be neither too big nor too small, as it either severely underestimates some promising models or fails to make a distinction. Both two cases introduce big prediction biases which mislead the ranking. In fact, some one-shot methods [3, 2] encounter the former problem and a single path approach [7] suffers the latter.

The above-defined values are firmly correlated with search space and most of the previous works concentrate only on the top ranking models and report few results about the lower ranking models at the full-train stage. Therefore, the question remains to thoroughly analyze their relationships in these methods.

A special case is when $M_{gap} = 0$, i.e., $R_{search} = R_{train}$. Ideally, no extra training processing is required and we can sample well-trained models and apply them directly. From this viewpoint, we try to minimize M_{gap}. As mentioned above, it's nontrivial to address it. The difficulty comes from two aspects: huge search space to explore, and not all-fitting training hyperparameters. For the former, we can estimate it by uniformly sampling some models. For the latter, we have no better solution but to use a fixed set of hyperparameters.

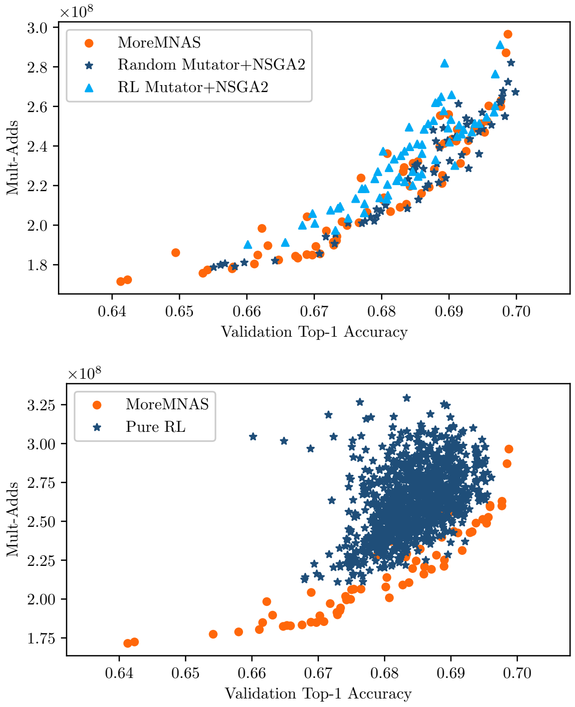


Figure 5. **Top:** Pareto front of MoreMNAS (equipped with a hierarchical mutator) compared with that of a random mutator and an RL mutator. **Bottom:** Pareto-front of MoreMNAS encompasses the models found by a pure RL NAS method with a mixed multi-objective reward. All models are tested on the ImageNet validation set. Each method has sampled 1,088 models.

[4] Typically, the capacity of the search space is larger than 6^{13}.

6. Experiments

6.1. Setups

Search Space. Our search space is designed based on MobileNetV2's inverted bottleneck block as done in [4]. In particular, we retain the same amount of layers with standard MobileNetV2 [18]. We search among convolution kernels (3, 5, 7) and expansion rates (3, 6), and we keep the number of filters unchanged. Besides, squeeze and excitation block is not included here regarding fairness [8]. Thus, the total search space contains 6^{16} child models, which is too large to enumerate them all.

Dataset. We perform all experiments on ImageNet [17] 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 original validation set is taken as the test set to measure the final performance of each model.

Training Hyperparameters. We train the supernet for 150 epochs using a batch size of 256. We adopt a stochastic gradient descent optimizer with a momentum of 0.9 [23] based on standard data augmentation as [18]. A cosine learning rate decay strategy [12] is applied with an initial learning rate of 0.045. Moreover, We regularize the training with L2 weight decay (4×10^{-5}). In order to be consistent with the previous works, we don't employ any other tricks like dropout [21], cutout [6] or mixup [28], although they can further improve the scores on the test set.

Regarding the stand-alone training of sampled models, we use the same hyperparameters as the supernet.

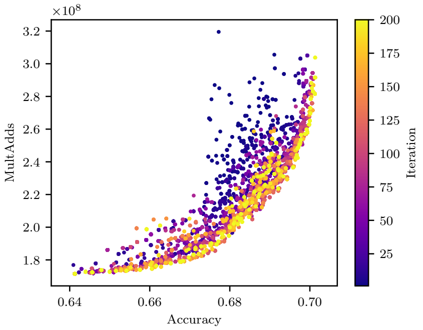


Figure 6. FairNAS evolution process of 200 generations, with 64 models sampled in each generation. Number of parameters are charted with Top-1 accuracies on the ImageNet validation set.

6.2. Comparisons with Various Model Selection Strategies

Our supernet is thus trained to fullness for 10 GPU days. Treating the supernet as a model estimator, we proceed with the traditional NAS. In practice, successful utilization of deep learning models requires dealing with several objectives that usually conflict with each other. Therefore, here we only examine NAS methods tailored for multi-objective optimization. We apply MoreMNAS [5] and construct several comparison groups: NSGA-II with reinforced mutation, NSGA-II with random mutation, PPO with a mixed multi-objective reward as defined in [24] [5]. The results are shown in Figure 5, affirming an outstanding advantage with the proposed method for that the control group all align within our Pareto front. In this paper, we consider three objectives: accuracy, multiply-adds, and parameters. We don't use latency here because we restrain ourselves from searching fast and accurate models that only suit for specific devices.

We run the pipeline for 200 epochs with a population size of 64, sampling 12,800 models in total. The searching takes 2 GPU days due to estimation speed-up. Unlike most of previous works [30, 16, 24], we sampled 13 models at approximately equal distances on the Pareto front and trained them fully to get the ranking, which is shown in Figure 7. According to Equation 3, we hit a new high record of Kendall rank correlation coefficient $\tau = 0.9487$.

6.3. Comparisons with various State-of-the-art Methods

We use the same search space as [4] to make comparisons be on par with various state-of-the-art methods. We search among convolution kernels (3, 5, 7) and expansion rates (3, 6) for 19 layers, therefore it contains 6^{19} submodels in total. Besides, we use the same hyper-parameters to train the supernet as Section 6.1 and the training setting and vanilla data processing tricks are the same as [24] to train the stand-alone models, which are sampled from our Pareto front with equal distance over multiply-adds to meet different computing requirements. The result is shown in Table 2. Although the latency is not considered as one of the objectives and it's a little unfair for FairNAS, we still report it here for the integrity of our work. FairNAS-A hits a new state-of-the-art result **75.34%** (top-1 accuracy for Imagenet 1k classification) under the same search space settings, which surpasses MnasNet-92 (+0.55%), Single-Path-NAS (+0.38%) with a comparable amount of multiply-adds. FairNAS-B matches Proxyless-GPU with much fewer parameters and multiply-adds. Besides, it surpasses Proxyless-R Mobile (+0.5%), One-Shot Small (+0.9%) with comparable amount of multiply-adds.

Three models seem to agree with high expansion rates and large kernels at the tail end, which enables a 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

[5] The latency is replaced with multiply-adds, and T = 300M.

after a downsampling operation, it's interesting to see that our FairNAS-B instead appreciates a larger kernel as in Figure 8. FairNAS-C apparently 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.

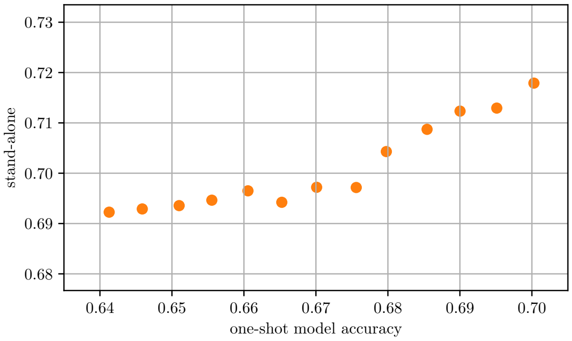


Figure 7. Top-1 validation accuracies on ImageNet of stand-alone models vs. one-shot models.

6.4. Expectation Fairness vs. Strict Fairness

We set up two other experiment groups that meet **expectation fairness** to form up our baselines. The training strategy for the first baseline is to update the activated path for extra $k-1$ times which follows a uniform model sampling and parameter update. For the second baseline, it is the same as the first one except that the learning rate is scaled to $\frac{1}{k}$ as that of FairNAS. In practice, we set $k=6$ to make it comparable to FairNAS. FairNAS is trained based on our proposed algorithms. Other hyperparameter settings are kept exactly the same.

We plot the average training accuracy over 120 epochs for various strategies, shown at the top of Figure 1. In particular, we calculate the Top-1 training accuracy by averaging each model's score on the corresponding mini-batch of data. The strategy that adheres to the **strict fairness** requirements boosts the accuracy of supernet accuracy steadily and rapidly, which reaches up to 60% after 40 epochs.

As for the first baseline, it has trouble to stabilize the training process. We suspect that the repeated k updates along a single activated path could cause the corresponding parameters overfitting for the sampled mini-batch of data while other paths perform rather poorly. While the next sampled path is getting activated, it pushes those weights too far away from the last update, thus forming the oscillation. Another factor about the oscillation is that the repeated k updates might have overshot.

To validate the above hypothesis, we inspect the second baseline in which the learning rate is calibrated and scaled. It eliminates the oscillation but it demonstrates a quite slow learning speed.

We further compare various neural architecture search methods based on our fairness definition in Table 1.

6.5. Analysis of Supernet Accuracy Gap

We randomly sample 1,000 models from the supernet trained with our fair strategy, then we evaluate these models directly on the ImageNet validation set. Unlike [7], no special measures are taken to recalculate the statistics for batch normalization [9]. Instead, we evaluate the sampled architectures on the fly.

The top-1 accuracies on the ImageNet validation set are reported at the bottom of Figure 1, ranging from 0.666 to 0.696. This leads to $R_{search} = 0.03$. By contrast, the accuracies of sampled models from the One-Shot supernet trained on the CIFAR-10 dataset [2] extend from 30% to 90%, while those of stand-alone models are bounded between 92.0% and 94.5%, thus $R_{search} = 0.6$ and $M_{gap} = 0.6 - 0.025 = 0.575$ by Equation 4. Bender *et al.* explained this abnormal accuracy distribution by hypothesizing the one-shot models learns useful operations, the removal of which causes a large drop in accuracy. Be that as it may, according to our analysis and experiments, we blame the unfair training process for this gap.

7. Conclusion

In the paper, we have thoroughly investigated the previously undiscussed fairness problem in weight-sharing neural architecture search approaches. We have discovered that biased weight-sharing methods like [2, 4, 7] either underestimate or overestimate the performance of the chosen model. For this reason, we have enforced a strict fairness constraint that helps to equitably train each possible operations. Any model thereby constructed can manifest its real strength thus good models can stand out during the search process. The one-shot model accuracy is demonstrated to be highly related with that of its corresponding stand-alone model. With a stable ranking at hand, we exploited the power of a multi-objective reinforced evolutionary method and innovatively utilized the fairly trained supernet as its fast model evaluator (altogether we call FairNAS). Our approach has proved to be effective in that it generated a set of new state-of-the-art architectures on the ImageNet dataset, with FairNAS-A achieving 75.34% top-1 validation accuracy at a size comparable to other NAS generated models.

References

[1] George Adam and Jonathan Lorraine. Understanding neural architecture search techniques. *arXiv preprint arXiv:1904.00438*, 2019. 2

[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, 5, 6, 8, 9

NAS Methods	Type	Memory Consumption (Supernet)	Supernet Train Cost (GPU days)	Search Cost (GPU days)	EF	SF
ENAS [15]	RL + Fine Tune	Single path	n/a	n/a	✗	✗
DARTS [11]	Gradient-based	A whole supernet	4 †	0	✗	✗
One-Shot [2]	Supernet	A whole supernet	16 ‡	3.3	✗	✗
FBNet [27]	Gradient-based	A whole supernet	20*	0	✗	✗
ProxylessNAS [4]	Gradient-based/RL	Two paths	15*	0	✓	✗
Single Path One-Shot [7]	Supernet+EA	Single path	12	<1	✓	✗
Single-Path NAS [22]	Supernet	Single path with super kernels	1.25‡	0	✓	✗
FairNAS (ours)	Fair Supernet+EA+RL	Single path	10	2	✓	✓

Table 1. Comparison of state-of-the-art NAS methods as per fairness basis. EF: Expectation Fairness, SF: Strict Fairness, †: searched on CIFAR-10, ‡: TPU, *: reported by [7].

Methods	Mult-Adds (M)	Params (M)	Top-1 Accuracy (%)	Top-5 Accuracy (%)
MobileNetV2 1.0 [18]	300	3.4	72.0	91.00
NASNet-A (4@1056) [30]	564	5.3	74.0	91.60
MnasNet [24]	317	4.2	74.0	91.78
MnasNet-92 [24]	388	3.9	74.79	92.05
DARTS [11]	574	4.7	73.3	91.30
One-Shot Small (F=32) [2]	-	5.1	74.2	-
FBNet-B [27]	295	4.5	74.1	
Proxyless-R Mobile [4]	320†	4.0	74.6	92.2
Proxyless GPU [4]	465†	7.1	75.1	-
Single-Path NAS [22]	365	4.3	74.96	92.21
FairNAS-A (ours)	388	4.6	**75.34**	**92.38**
FairNAS-B (ours)	345	4.5	75.10	92.30
FairNAS-C (ours)	321	4.4	74.69	92.12

Table 2. Comparison of mobile models on ImageNet. The input size is set to 224×224. †: Based on our calculation.

[3] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. In *6th International Conference on Learning Representations*, 2018. 1, 2, 6

[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. 2, 7, 8, 9

[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. 2, 5, 6, 7

[6] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 7

[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, 5, 6, 8, 9

[8] 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. 7

[9] 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. 8

[10] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. 6

[11] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *International Conference on Learning Representations*, 2019. 1, 2, 9

[12] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *International Conference on Learning Representations*, 2017. 7

[13] 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. *arXiv preprint arXiv:1810.03522*, 2018. 1

[14] 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. 6

[15] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter

sharing. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. 1, 2, 6, 9

[16] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *ICML AutoML Workshop*, 2018. 1, 7

[17] 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. 7

[18] 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. 7, 9

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 6

[20] 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. 6

[21] 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. 7

[22] 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. 2, 9

[23] 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. 7

[24] 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, 7, 9

[25] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36-th International Conference on Machine Learning*, 2019. 1

[26] Ian Tweddle. *James Stirlings methodus differentialis: an annotated translation of Stirlings text*. Springer Science & Business Media, 2012. 4, 11

[27] 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 differentiableneural architecture search. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 2, 9

[28] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. 7

[29] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017. 1

[30] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2(6), 2018. 1, 7, 9

Algorithm 2 Speed up in parallel under some conditions.

```
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 $m$ (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

We give the parallel version of Algorithm 1 in Algorithm 2.

B. Proof

Here is the proof for Lemma 2.

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

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

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 [26].

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

□

C. Experiment Details

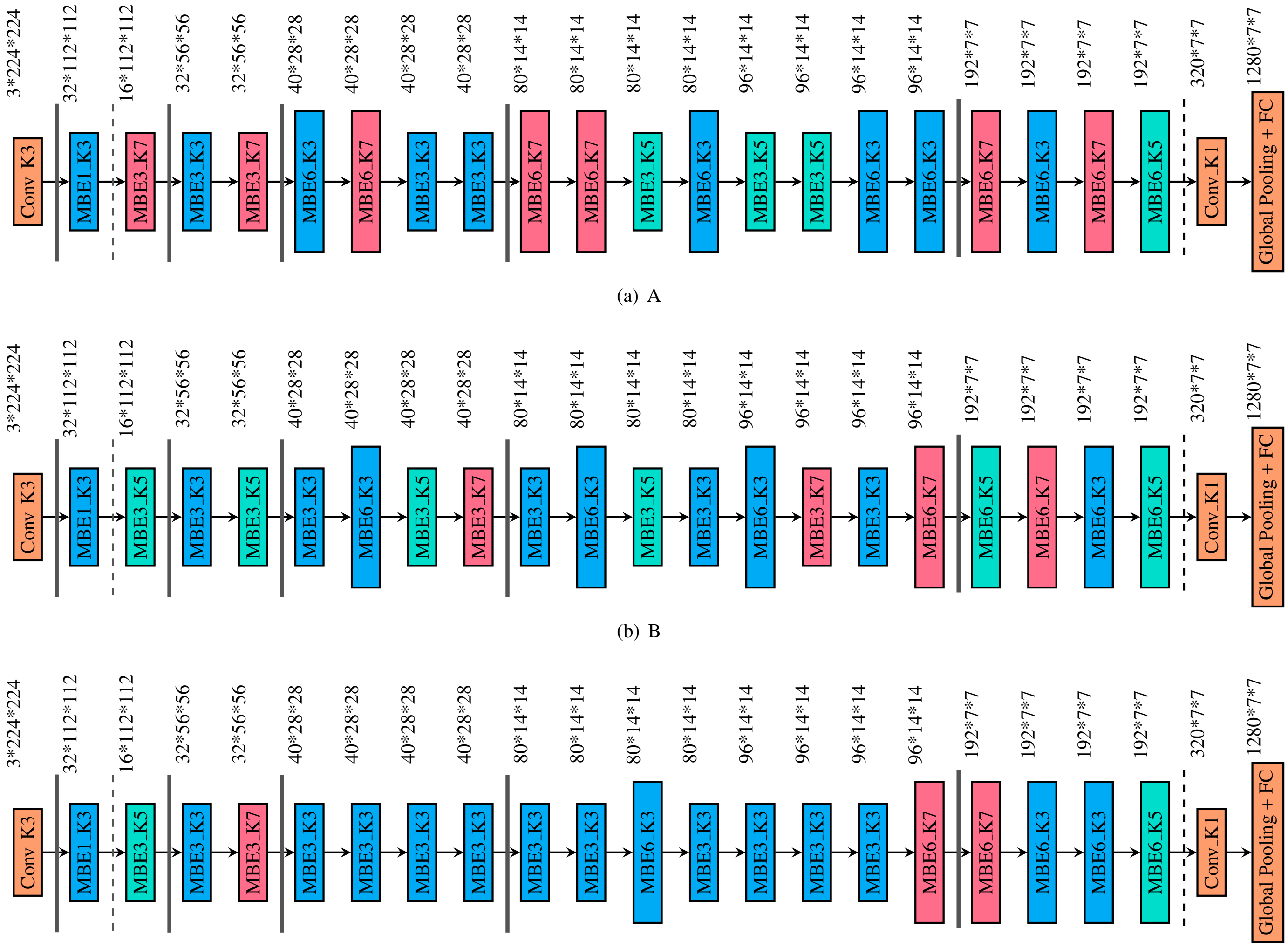
C.1. Hyperparameters for MoreMNAS

We list them in Table 3.

Table 3. 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		

C.2. FairNAS Models



(a) A

(b) B

(c) C

Figure 8. The Architectures of FairNAS-A, B, C. Note Ex_Ky means an expansion rate of x for its expansion layer and a kernel size of y for its depthwise convolution layer. Grey thick lines refer to downsampling points. Dashed lines separate the stem and end layers from the backbone.