GreedyNAS: Towards Fast One-Shot NAS with Greedy Supernet

Shan You[1,2]*, Tao Huang[1,3]*, Mingmin Yang[1]*, Fei Wang[1], Chen Qian[1], Changshui Zhang[2]
[1]SenseTime [2]Department of Automation, Tsinghua University
[3]Dian Group, School of CST, Huazhong University of Science and Technology
{youshan,huangtao,yangmingmin,wangfei,qianchen}@sensetime.com zcs@mail.tsinghua.edu.cn

Abstract

*Training a supernet matters for one-shot neural architecture search (NAS) methods since it serves as a basic performance estimator for different architectures (paths). Current methods mainly hold the assumption that a supernet should give a reasonable ranking over all paths. They thus treat all paths equally, and spare much effort to train paths. However, it is harsh for a single supernet to evaluate accurately on such a huge-scale search space (*e.g.*, 7^{21}). In this paper, instead of covering all paths, we ease the burden of supernet by encouraging it to focus more on evaluation of those potentially-good ones, which are identified using a surrogate portion of validation data. Concretely, during training, we propose a multi-path sampling strategy with rejection, and greedily filter the weak paths. The training efficiency is thus boosted since the training space has been greedily shrunk from all paths to those potentially-good ones. Moreover, we further adopt an exploration and exploitation policy by introducing an empirical candidate path pool. Our proposed method GreedyNAS is easy-to-follow, and experimental results on ImageNet dataset indicate that it can achieve better Top-1 accuracy under same search space and FLOPs or latency level, but with only ~60% of supernet training cost. By searching on a larger space, our GreedyNAS can also obtain new state-of-the-art architectures.*

1. Introduction

By dint of automatic feature engineering, deep neural networks (DNNs) have achieved remarkable success in various computer vision tasks, such as image classification [37, 36, 41, 33, 32, 15, 39], visual generation [34, 35], image retrieval [40, 7, 8, 12] and semantic comprehension [18, 17]. In contrast, neural architecture search (NAS) aims at automatically learning the network architecture to further boost the performance for target tasks [10, 20, 43, 2, 19]. Nevertheless, previous NAS methods in general suffer from huge computation budget, such as 2000 GPU days of reinforcement learning [43] and 3150 GPU days of evolution [26] with hundreds of GPUs.

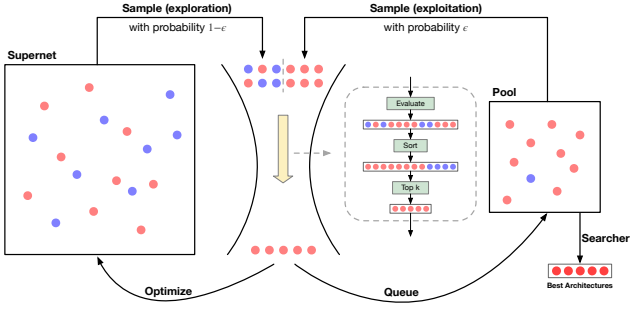


Figure 1: Diagram of supernet training for our proposed GreedyNAS. The supernet greedily shrinks its training space from all paths (red and blue dots) into potentially-good paths (red dots), and further into candidate pool.

Current One-shot NAS methods boost the search efficiency by modeling NAS as a one-shot training process of an over-parameterized supernet. As a result, various architectures can be derived from the supernet, and share the same weights. For example, DARTS [21] and its variants [38, 1] parameterize the supernet with an additional categorical distribution for indicating what operations we want to keep. In contrast, recent single path methods adopt a non-parametric architecture modeling, and split the searching into two consecutive stages, *i.e.*, supernet training and architecture sampling. For training supernet, only a single path consisting of a single operation choice is activated and gets optimized by regular gradient-based optimizers. After the supernet is trained well, it is regarded as a performance estimator for all architectures (*i.e.*, paths). Then the optimal architecture can be searched using a hold-out validation dataset via random search [16] or (reinforced) evolutionary [11, 4] algorithms under specified hardware constraint (*e.g.*, FLOPs and latency). As only one path is activated for training, the memory cost coheres with that of traditional network training, and scales well on large-scale datasets (*e.g.*, ImageNet [27]).

*Equal contributions.

Supernet matters for it serves as a fundamental performance estimator of different architectures (paths). Current methods [16, 11, 4, 3] hold the assumption that the supernet should estimate the (relative) performance accurately for *all* paths, and thus all paths are treated equally and trained simultaneously. However, the paths contained in the supernet are of fairly huge scale (*e.g.*, 7^{21}). Hence it can be harsh for a single supernet to evaluate and give reasonable ranking on such a quantity of paths at the same time. In fact, the ultimate aim of supernet is only to identify a bunch of optimal paths. But the huge search space implies significant variance and variety of paths; there exist many architectures of inferior quality in terms of accuracy performance.[1] Since the weights of all paths are highly shared, if a weak path is sampled and gets trained, it would disturb the weights of those potentially-good paths. This disturbance will undermine their eventual performance estimation and affect the searched optimal architecture accordingly. The supernet is thus not supposed to care much on these weak paths and get updated for them. Besides, training on those weak paths actually involves unnecessary update of weights, and slows down the training efficiency more or less.

In this paper, we ease the training burden by encouraging a greedy supernet. A greedy supernet is capable of shifting its focus on performance estimation of those potentially-good paths instead of all paths. Concretely, during the supernet training, we propose a multi-path sampling strategy with rejection to filter the weak paths, so the supernet will greedily train those potentially-good paths. This path filtering can be efficiently implemented via evaluation using a surrogate portion of validation dataset, without harming the computation cost too much. Moreover, we also adopt an exploration and exploitation policy [14, 24] by introducing a candidate pool, which dynamically tracks those potentially-good paths discovered during training. In this way, the supernet improves its training efficiency by switching its training space from all paths into those potentially-good ones, and further into candidate pool by sampling from it, as shown in Figure 1.

We implement our proposed method GreedyNAS on the large-scale benchmark ImageNet dataset [27], and extensive experimental results indicate our superiority in terms of accuracy performance and supernet training efficiency. For example, with the same search space, our method can achieve higher Top-1 accuracy than that of other comparison methods under the same FLOPs or latency level, but reduces approximate 40% of supernet training cost. By searching on a larger space, we can also obtain new state-of-the-art architectures.

[1]For example, in a same supernet, MobileNetV2 [28] can achieve 72.0% Top-1 accuracy on ImageNet dataset while an extreme case of almost all identity operations only has 24.1% [3].

2. Related Work

One-shot NAS methods mainly aim to train an over-parameterized network (a.k.a supernet) that comprises all architectures (paths), which share the same weights mutually. Then the optimal architecture can be derived or searched from the supernet. There are mainly two categories of one-shot NAS methods [9], which differ in how the architectures are modeled and elaborated as follows.

Parameterized architectures. To use the gradient-based optimizers for direct searching, an real-valued categorical distribution (architecture parameter) is usually introduced in the supernet, and can be thus jointly learned with the supernet weights, such as DARTS [21], FBNet [38] and MdeNAS [42]. When the supernet training is finished, the optimal architecture can be induced by sampling from the categorical distribution. However, it may suffer from the huge GPU memory consumption. ProxylessNAS [1] alleviates this issue by factorizing the searching into multiple binary selection tasks while *Single-Path*-NAS [29] uses superkernels to encode all operation choices. Basically, they are difficult to integrate a hard hardware constraint (*e.g.*, FLOPs and latency) during search but resort to relaxed regularization terms [38, 1].

Sampled single-path architectures. By directly searching the discrete search space, the supernet is trained by sampling and optimizing a single path. The sampling can be uniform sampling [11, 16] or multi-path sampling with fairness [4]. After the supernet is trained, it is supposed to act as a performance estimator for different paths. And the optimal path can be searched by various searchers, such as random search and evolutionary algorithms [6]. For example, ScarletNAS [3] employs a multi-objective searcher [22] to consider classification error, FLOPs and model size for better paths. Different to the previous parameterized methods, the hard hardware constraint can be easily integrated in the searchers. Our proposed method GreedyNAS is cast into this category.

3. Rethinking path training of supernet

In Single-path One-shot NAS, we utilize an over-parameterized supernet $\mathcal{N}$ with parameter Ω to substantialize a search space, which is formulated as a directed acyclic graph (DAG). In the DAG, feature maps act as the nodes, while the operations (or transformations) between feature maps are regarded as edges for connecting sequential nodes. Assume the supernet $\mathcal{N}$ has L layers, and each layer $\mathcal{N}^l$ is allocated with O operation choices $\mathcal{O} = \{o_i\}$, which can be basic convolution, pooling, identity or different types of building blocks, such as MobileNetV2 block [28] and ShuffleNetV2 block [23]. Then each architecture (*i.e.*, path) denoted as $\boldsymbol{a}$ can be represented by a tuple of size L, *i.e.*, $\boldsymbol{a} = (o^1, o^2, ..., o^L)$ where $o^j \in \mathcal{O}$, $\forall j = 1, 2, ..., L$. As

a result, the search space $\mathcal{A}$ is discrete, and there will be O^L (*e.g.*, 7^{21}) architectures in total, namely, $|\mathcal{A}| = O^L$.

Training supernet matters since it is expected to serve as a fundamental performance estimator. Due to the consideration of memory consumption, single-path NAS methods implement training by sampling a single path $\boldsymbol{a}$ from $\mathcal{A}$, then the sampled paths are all optimized on the training data $\mathcal{D}_{tr}$. It can be formulated as minimizing an expected loss over the space $\mathcal{A}$, *i.e.*,

$$\Omega^* = \arg\min_{\Omega} \mathop{\mathbb{E}}_{\boldsymbol{a}\sim p(\mathcal{A})} [\![\mathcal{L}(\omega_{\boldsymbol{a}}; \mathcal{D}_{tr})]\!], \tag{1}$$

where $\omega_{\boldsymbol{a}}$ refers to the parameter of path $\boldsymbol{a}$, and $p(\mathcal{A})$ is a discrete sampling distribution over $\mathcal{A}$.

After the supernet $\mathcal{N}(\Omega^*)$ is trained well, we can evaluate the quality of each path by calculating its (Top-1) accuracy (ACC) on the validation dataset $\mathcal{D}_{val}$, and the optimal path $\boldsymbol{a}^*$ corresponds to the maximum ACC, *i.e.*,

$$\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}\in\mathcal{A}} \ \text{ACC}(\omega^*_{\boldsymbol{a}}, \mathcal{D}_{val}), \tag{2}$$

where $\omega^*_{\boldsymbol{a}} \subset \Omega^*$ w.r.t. path $\boldsymbol{a}$ in the trained supernet $\mathcal{N}(\Omega^*)$.

3.1. Reshaping sampling distribution $p(\mathcal{A})$

Current methods assume that the supernet should provide a reasonable ranking over all architectures in $\mathcal{A}$. Thus *all* paths $\boldsymbol{a}$ are treated equally, and optimized simultaneously [16, 11, 4, 3]. Then the sampling distribution $p(\mathcal{A})$ amounts to a uniform distribution $p(\mathcal{A}) = U(\mathcal{A})$ over $\mathcal{A}$, *i.e.*,

$$p(\boldsymbol{a}) = \frac{1}{|\mathcal{A}|}\mathbb{I}(\boldsymbol{a} \in \mathcal{A}), \tag{3}$$

where $\mathbb{I}(\cdot)$ is an indicator function. However, as previously discussed, it is a demanding requirement for the supernet to rank accurately for all paths at the same time. In the huge search space $\mathcal{A}$, there might be some paths of inferior quality. Since the weights are highly shared in the same supernet, training on these weak paths does have negative influence on the evaluation of those potentially-good paths. To alleviate this disturbance, an intuitive idea is to block the training of these weak paths.

For simplifying the analysis, we assume the search space $\mathcal{A}$ can be partitioned into two subsets $\mathcal{A}_{good}$ and $\mathcal{A}_{weak}$ by an Oracle good but unknown supernet $\mathcal{N}_o$, where

$$\mathcal{A} = \mathcal{A}_{good} \bigcup \mathcal{A}_{weak}, \ \mathcal{A}_{good} \bigcap \mathcal{A}_{weak} = \emptyset, \tag{4}$$

and $\mathcal{A}_{good}$ indicates the potentially-good paths while $\mathcal{A}_{weak}$ is for weak paths, *i.e.*,

$$\text{ACC}(\boldsymbol{a}, \mathcal{N}_o, \mathcal{D}_{val}) \geq \text{ACC}(\boldsymbol{b}, \mathcal{N}_o, \mathcal{D}_{val}) \tag{5}$$

holds for all $\boldsymbol{a} \in \mathcal{A}_{good}, \boldsymbol{b} \in \mathcal{A}_{weak}$ on validation dataset $\mathcal{D}_{val}$. Then to screen the weak paths and ease the burden of the supernet training, we can just sample from the potentially-good paths $\mathcal{A}_{good}$ instead of all paths $\mathcal{A}$.

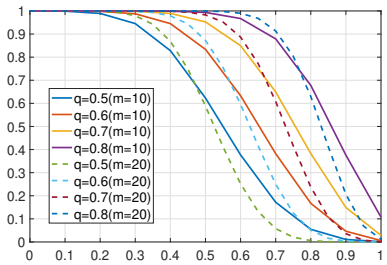


Figure 2: Probability of sampling at least k potentially-good paths out of m paths. X-axis: $r = k/m$. $q = |\mathcal{A}_{good}|/|\mathcal{A}|$.

The sampling distribution $p(\mathcal{A})$ is equivalently reshaped by truncation on $\mathcal{A}_{good}$, *i.e.*, $\boldsymbol{p}(\mathcal{A}) = U(\mathcal{A}_{good}; \mathcal{N}_o, \mathcal{D}_{val})$ and

$$p(\boldsymbol{a}; \mathcal{N}_o, \mathcal{D}_{val}) = \frac{1}{|\mathcal{A}_{good}|}\mathbb{I}(\boldsymbol{a} \in \mathcal{A}_{good}). \tag{6}$$

In this way, the supernet is expected to thoroughly get trained on the potentially-good paths and thus give decent performance ranking. Besides, since the valid search space has been shrunken from $\mathcal{A}$ into $\mathcal{A}_{good}$, the training efficiency of supernet is improved accordingly.

3.2. Greedy path filtering

Nevertheless, in the supernet training the Oracle supernet $\mathcal{N}_o$ is unknown, thus we can not sample paths according to Eq.(6) since it relies on $\mathcal{N}_o$. In this paper, we propose to use greedy strategy and during training, current supernet $\mathcal{N}_\dagger$ is progressively regarded as a proxy of the Oracle $\mathcal{N}_o$. Thus during the supernet training, we greedily sample paths according to the reshaped sampling distribution given by current $\mathcal{N}_\dagger$, namely, $\boldsymbol{p}(\mathcal{A}) = U(\mathcal{A}_{good}; \mathcal{N}_\dagger, \mathcal{D}_{val})$. The sampled paths will get optimized, then the supernet is get updated and evolves to a decent performance estimator over $\mathcal{A}_{good}$.

However, a natural question arises: even given a supernet $\mathcal{N}_\dagger$, how can we sample from the shaped distribution $\boldsymbol{p}(\mathcal{A}) = U(\mathcal{A}_{good}; \mathcal{N}_\dagger, \mathcal{D}_{val})$? In other words, how can we accurately identify whether a path is from $\mathcal{A}_{good}$ or $\mathcal{A}_{weak}$? Note that the partition of $\mathcal{A}$ is determined by traversing all paths in $\mathcal{A}$ as Eq.(5), which is not affordable in computation. Since we can not accurately know whether a single path is good or weak, to solve this issue, we propose a multi-path sampling strategy with rejection.

Suppose we uniformly sample a path from $\mathcal{A}$, then it amounts to be sampled from $\mathcal{A}_{good}$ with probability $q = |\mathcal{A}_{good}|/|\mathcal{A}|$, and sampled from $\mathcal{A}_{weak}$ with probability $1 - q$. In this way, if we sample multiple paths independently at a time, we have the following results based on binomial distribution.

Theorem 1. *If m paths are sampled uniformly i.i.d. from $\mathcal{A}$, and $\mathcal{A}_{good}$ and $\mathcal{A}_{weak}$ are defined as Eq.*(4) *and Eq.*(5) *based on supernet $\mathcal{N}_\dagger$, then it holds that at least k ($k \leq m$)*

Algorithm 1 Greedy path filtering w.t/w.o candaidate pool.

Input: supernet $\mathcal{N}$ with parameter Ω,validation data $\mathcal{D}_{val}$, number of sampled multiple paths m, number of kept paths k, candidate pool $\mathcal{P}$ with sampling probability ϵ.

if without candidate pool $\mathcal{P}$ **then**
  sample m paths $\{\boldsymbol{a}_i\}_{i=1}^m$ i.i.d. w.r.t. $\boldsymbol{a}_i \sim U(\mathcal{A})$
else
  sample m paths $\{\boldsymbol{a}_i\}_{i=1}^m$ i.i.d. w.r.t. $\boldsymbol{a}_i \sim (1-\epsilon)\cdot U(\mathcal{A}) + \epsilon \cdot U(\mathcal{P})$
end if
randomly sample a batch $\hat{\mathcal{D}}_{val}$ in $\mathcal{D}_{val}$
evaluate the loss ℓ_i of each path $\boldsymbol{a}_i$ on $\hat{\mathcal{D}}_{val}$
rank the paths by ℓ_i, and get Top-k indexes $\{t_i\}_{i=1}^k$
return k paths $\{\boldsymbol{a}_{t_i}\}_{i=1}^k$ and filter the rest

paths are from $\mathcal{A}_{good}$ with probability

$$\sum_{j=k}^{m} \mathbb{C}_m^j q^j (1-q)^{m-j}, \tag{7}$$

where $q = |\mathcal{A}_{good}|/|\mathcal{A}|$.

From Theorem 1, we can see by sampling m paths, the probability that at least k paths are from $\mathcal{A}_{good}$ is very high when the proportion of potentially-good paths q is medially large or k is medially small (see Figure 2). For example, if we conservatively assume 60% paths have the potential to be good (*i.e.*, $q = 0.6$), we will have 83.38% confidence to say at least 5 out of 10 paths are sampled from $\mathcal{A}_{good}$. In this way, based on the definition of Eq.(4) and Eq.(5), we just rank the sampled m paths using validation data $\mathcal{D}_{val}$, keep the Top-k paths and reject the remaining paths.

However, ranking m paths involves calculation of ACC over all validation dataset $\mathcal{D}_{val}$ as Eq.(5), which is also computationally intensive during the supernet training.[2] In fact, in our multi-path sampling, what we care about is the obtained ranking; we empirically find that it suffices to rank based on the loss ℓ (*e.g.*, cross entropy loss for classification) over a surrogate subset of $\mathcal{D}_{val}$ (*e.g.*, 1k images on ImageNet dataset), denoted as $\hat{\mathcal{D}}_{val}$. The consistency between this rank and that given by ACC on all $\mathcal{D}_{val}$ is fairly significant. More details and analysis refer to the ablation studies in Section 5.3.1. Then the sampling works as Algorithm 1.

As a result, path filtering can be efficiently implemented for it can run in a simple feed-forward mode (*e.g.*, `eval()` mode in Pytorch) on a small portion of validation data. In this sense, we block the weak paths greedily during the supernet training. And the validation data $\hat{\mathcal{D}}_{val}$ acts as a rough filter to prevent the training of those low-quality or even harmful paths, so that the supernet can get sufficient training on those potentially-good ones.

[2]For example, the size of $\mathcal{D}_{val}$ on ImageNet dataset is 50k.

Algorithm 2 Greedy training of supernet.

Input: supernet $\mathcal{N}$ with parameter Ω, training data $\mathcal{D}_{tr}$, validation data $\mathcal{D}_{val}$, number of sampled multiple paths m, number of kept paths k, max iteration T, training data loader D

initialize candidate pool $\mathcal{P} = \emptyset$,
set a Scheduler of pool sampling probability ϵ
for $\tau = 1, .., T/k$ **do**
  get the pool sampling probability ϵ by Scheduler
  sample k paths $\{\boldsymbol{a}_{t_i}\}_{i=1}^k$ out of m paths using Algorithm 1 with pool sampling probability ϵ
  update candidate pool $\mathcal{P}$ using $\{\boldsymbol{a}_{t_i}\}_{i=1}^k$
  for $i = 1, .., k$ **do**
    get a training batch from D
    update the weights $\omega_{\boldsymbol{a}_{t_i}}$ of path $\boldsymbol{a}_{t_i}$ using gradient-based optimizer
  end for
end for

4. Proposed Approach: GreedyNAS

In this section, we formally illustrated our proposed NAS method (a.k.a. GreedyNAS) based on a greedy supernet. Our GreedyNAS is composed with three procedures, *i.e.*, supernet training, searching paths and retraining the searched optimal path. The last retraining corresponds to conventional training a given network. We mainly elaborate the first two as follows.

4.1. Greedy training of supernet

As previously discussed, we propose to maintain a greedy supernet during its training. By doing this, we gradually approximate the sampling $\boldsymbol{p}(\mathcal{A}) = U(\mathcal{A}_{good}; \mathcal{N}_\dagger, \mathcal{D}_{val})$ by keeping the Top-k paths and filtering the bottom $m-k$ paths by evaluating using $\hat{\mathcal{D}}_{val}$. Then those weak paths are prevented from getting trained, which allows the supernet to focus more on those potentially-good paths and switch its training space from $\mathcal{A}$ into $\mathcal{A}_{good}$.

4.1.1 Training with exploration and exploitation

After the greedy path filtering, we have actually identified some potentially-good paths, which amount to some empirically-good ones given by current supernet. Then to further improve the training efficiency, inspired by the Monte Carlo tree search [14] and deep Q-learning (DQN) [24], we propose to train the supernet with an exploration and exploitation (E-E) strategy by reusing these paths.

Concretely, we introduce a *candidate pool* $\mathcal{P}$ to store the potentially-good paths discovered during training. Each path $\boldsymbol{a}$ is represented as a tuple of operation choices. Besides, each $\boldsymbol{a}_i$ is also allocated with an evaluation loss ℓ_i.

The candidate pool is thus formulated as a fixed-size ordered queue with priority ℓ. With more potentially-good paths involved, the candidate pool can be maintained by a min-heap structure in real time.

As a result, we can conservatively implement *local search* by sampling from the candidate pool since it consists of a smaller number (but promising) of paths. However, this greedy *exploitation* brings in the risks of losing path diversity for the training. In this way, we also favor a *global search* with the hope of probing other promising paths that are yet to be sampled and get trained, which can be easily fulfilled by uniform sampling from $\mathcal{A}$. For achieve a balanced trade-off of exploration and exploitation, we adopt a typical ϵ-sampling policy, *i.e.*, implementing uniform sampling both from $\mathcal{A}$ and pool $\mathcal{P}$ (line 4 of Algorithm 1),

$$\boldsymbol{a} \sim (1-\epsilon) \cdot U(\mathcal{A}) + \epsilon \cdot U(\mathcal{P}), \tag{8}$$

where $\epsilon \in [0, 1]$ indicates the probability of sampling from the pool $\mathcal{P}$. Note that candidate pool runs through the training process of supernet; however, it might be not reliable at first since the priority ℓ is calculated based on a much less-trained supernet. In this case, we propose to actively anneal the pool sampling probability ϵ from 0 to a pre-defined level. In our experiment, we find $\epsilon = 0.8$ will be a good option.

Training with exploration and exploitation encourages the supernet to refine the already-found good paths as well as probing new territory for more better paths. Besides, it actually also contributes to our greedy path filtering by improving our filtering confidence. Basically, the collected potentially-good paths can be regarded as a subset of $\mathcal{A}_{good}$, then sampling from $\mathcal{P}$ amounts to increasing the probability q of Theorem 1 into

$$q = \epsilon + (1-\epsilon)|\mathcal{A}_{good}|/|\mathcal{A}|, \tag{9}$$

which refers to the proportion of potentially-good paths. For example, assume we evenly sample from $\mathcal{P}$ or $\mathcal{A}$ ($\epsilon = 0.5$), then the probability of sampling at least 5 good paths out of 10 paths will rise from 83.38% to 99.36% according to Theorem 1. Comparing reducing $r = k/m$ to increase the sampling confidence, sampling with $\mathcal{P}$ is almost cost-neglectable since we only need to maintain a min-heap. The supernet thus gradually shifts its training from $\mathcal{A}_{good}$ more to $\mathcal{P}$, and the training efficiency will be further improved accordingly.

4.1.2 Stopping principle via candidate pool

Different to conventional networks, a supernet serves as a performance estimator and it is difficult to judge when it is trained well. Current single-path NAS methods control its training by manually specifying an maximum epoch number. In our GreedyNAS, however, we propose an adaptive stopping principle based on the candidate pool.

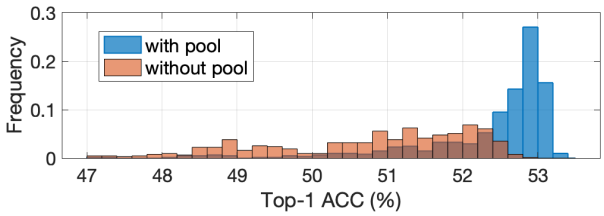


Figure 3: Histogram of accuracy of searched paths on supernet by evolutionary searching method (with or without candidate pool).

Candidate pool $\mathcal{P}$ indicates a bunch of best empirical paths, and it is updated dynamically during the supernet training. In consequence, if a supernet is trained well, the pool $\mathcal{P}$ should tend to be steady. This *steadiness* can be measured by the update frequency π of candidate pool $\mathcal{P}$, *i.e.*,

$$\pi := \frac{|\mathcal{P}_t \bigcap \mathcal{P}|}{|\mathcal{P}|} \leq \alpha, \tag{10}$$

where $\mathcal{P}_t$ refers to the old $\mathcal{P}$ in previous t iterations. Thus smaller π implies that fewer new paths are involved in the pool $\mathcal{P}$ within t iterations, and $\mathcal{P}$ is more steady. Given a certain tolerance level α[3], when the update frequency π is less than α, we believe the supernet has been trained enough, and its training can be stopped accordingly.

4.2. Searching with candidate pool

After the supernet is trained, we can use supernet to evaluate the quality (ACC) of each path on validation dataset $\mathcal{D}_{val}$, and search the optimal path $\boldsymbol{a}^*$ as Eq.(2). However, enumerating all paths in $\mathcal{A}$ is prohibitively computation-intensive. One remedy is by dint of evolutionary algorithms [11] or reinforced version (*e.g.*, MoreMNAS [5]), which takes the supernet as an off-the-shelf evaluator. In our paper, we adopt the multi-objective NSGA-II [6] algorithm for searching, where the hardware constraint can be easily integrated in the evolution process. If a path violates the pre-defined hardware constraint (*e.g.*, under 330 FLOPs), we just ditch it for good.

Besides, evolutionary algorithms need to initialize population with size N_{pop} before implementing iterative mutation and crossover. Current methods usually random sample N_{pop} paths under the constraint as initial population. In contrast, our method makes the initialization with the help of candidate pool $\mathcal{P}$, and select its Top-$\mathcal{N}_{pop}$ paths instead. As Figure 3 shows, searching with candidate pool can boost the evolutionary performance for supplying a good initial population. The ACC of searched paths using candidate pool is on average higher than that using random initialization. More details of our searching algorithm refer to the supplementary materials.

[3] $\alpha = 0.08$ suffices in our experiment.

Table 1: Comparison of classification performance and supernet training efficiency w.r.t. different searching methods on ImageNet dataset under same search space. #optimization means the accumulated #examples calculated for a whole optimization step, while #evaluation is for that of forward evaluation. corrected #optimization is based on our statistics that cost of a whole optimization step is 3.33 times larger than that of forward evaluation. Details of calculation refer to supplementary materials.

Methods	performance				supernet training efficiency		
	Top-1 (%)	FLOPs	latency	Params	#optimization	#evaluation	corrected #optimization
Proxyless-R (mobile) [1]	74.60	320M	79 ms	4.0M	-	-	-
Random Search	74.07	321M	69 ms	3.6M	1.23M×120	-	147.6M
Uniform Sampling [11]	74.50	326M	72 ms	3.8M	1.23M×120	-	147.6M
FairNAS-C [4]	74.69	321M	75 ms	4.4M	1.23M×150	-	184.5M
Random Search-E	73.88	320M	91 ms	3.7M	1.23M×73	-	89.8M
Uniform Sampling [11]-E	74.17	320M	94 ms	3.6M	1.23M×73	-	89.8M
GreedyNAS (FLOPs$\leq$ 322M)	**74.85**	320M	89 ms	3.8M	1.23M×46	2.40M×46	89.7M
GreedyNAS (latency$\leq$ 80ms)	**74.93**	324M	78 ms	4.1M	1.23M×46	2.40M×46	89.7M

5. Experimental Results

5.1. Configuration and settings

Dataset. We conduct the architecture search on the challenging ImageNet dataset [27]. As [1], we randomly sample 50,000 images (50 images per class) from training dateset as the validation dataset ($|\mathcal{D}_{val}| = 50$K), and the rest of training images are used for training. Moreover, we use the original validation dataset as the test dataset to report the accuracy performance.

Search space. Following [1, 4], we adopt the same macro-structure of supernet for fair comparison as shown in Table 5 (see supplementary materials). Moreover, we use MobileNetV2 inverted bottleneck [28] as the basic building block. For each building block, the convolutional kernel size is within $\{3, 5, 7\}$ and expansion ratio is selected in $\{3, 6\}$. An identity block is also attached for flexible depth search. As a result, with 21 building blocks, the search space is of size $(3 \times 2 + 1)^{21} = 7^{21}$. In addition, we also implement searching on a larger space by augmenting each building block with an squeeze-and-excitation (SE) option. The size of the larger search space is thus 13^{21}.

Supernet training. For training the supernet, Algorithm 1 is adopted to sample 10 paths and filter 5 paths. We randomly sample 1000 images (1 image per class) from the validation dataset for evaluating paths in Algorithm 1. For training each path, we use a stochastic gradient descent (SGD) optimizer with momentum 0.9 and Nesterov acceleration. The learning rate is decayed with cosine annealing strategy from initial value 0.12. The batch size is 1024. As for candidate pool, we empirically find 1000 is a good option for pool size $|\mathcal{P}|$, which approximates the amount of paths involved in one epoch. The candidate sampling probability ϵ is linearly increased from 0 to 0.8. Instead of specifying an epoch number [11, 4], we use the proposed principle to stop the supernet training with tolerance $\alpha = 0.08$.

Evolutionary searching. For searching with NSGA-II [6] algorithm, we set the population size as 50 and the number of generations as 20. The population is initialized by the candidate pool $\mathcal{P}$ while other comparison methods use random initialization. During searching, we use constraint of FLOPs or latency. All our experiments use Qualcomm® Snapdragon™ 855 mobile hardware development kit (HDK) to measure the latency.

Retraining. To train the obtained architecture, we use the same strategy as [1] for search space without SE. As for the augmented search space, we adopt a RMSProp optimizer with 0.9 momentum as Mnasnet [30]. Learning rate is increased from 0 to 0.064 in the first 5 epochs with batch size 512, and then decays 0.03 every 3 epochs. Besides, exponential moving average is also adopted with decay 0.999.

5.2. Performance comparison with state-of-the-art methods

Searching on same search space. For fair comparison, we first benchmark our GreedyNAS to the same search space as [1] to evaluate our superiority to other Single-path One-shot NAS methods. We also cover a baseline method Random Search, which shares the same supernet training strategy with Uniform Sampling [11]; but during search, instead of using evolutionary algorithms it randomly samples 1000 paths, and retrains the rank-1 path according to Top-1 ACC on the supernet. As Table 1 shows, when searching with similar 320 FLOPs, our GreedyNAS achieves the highest Top-1 ACC. We further align our searched constraint to latency of 80 ms. Table 1 indicates that with similar latency, GreedyNAS is still consistently superior to other comparison methods. For example, GreedyNAS can search an architecture with 74.93% Top-1 ACC, enjoying a 0.43% improvement over uniform sampling, which in a way illustrates the superiority of our greedy supernet to a uniform supernet.

Besides advantages on the classification performance of searched models, we also evaluate our superiority in terms of supernet training efficiency. Since the main differences of our GreedyNAS and other Single-path One-shot NAS methods lie in the supernet training, we report in Table 1 the supernet training cost. To eliminate the efficiency gap

Table 2: Comparison of searched architectures w.r.t. different state-of-the-art NAS methods. †: searched on CIFAR-10, ‡: TPU, ⋆: reported by [11].

Methods	Top-1 (%)	Top-5 (%)	FLOPs (M)	latency (ms)	Params (M)	Memory cost	training cost (GPU days)	search cost (GPU days)
SCARLET-C [4]	75.6	92.6	280	67	6.0	single path	10	12
MobileNetV2 1.0 [28]	72.0	91.0	300	38	3.4	-	-	-
MnasNet-A1 [30]	75.2	92.5	312	55	3.9	single path + RL	$288^{\ddagger}$	-
GreedyNAS-C	**76.2**	92.5	284	70	4.7	single path	7	< 1
Proxyless-R (mobile) [1]	74.6	92.2	320	79	4.0	two paths	$15^{\star}$	-
FairNAS-C [4]	74.7	92.1	321	75	4.4	single path	10	2
Uniform Sampling [11]	74.7	-	328	-	-	single path	12	< 1
SCARLET-B [4]	76.3	93.0	329	104	6.5	single path	10	12
GreedyNAS-B	**76.8**	93.0	324	110	5.2	single path	7	< 1
SCARLET-A [4]	76.9	93.4	365	118	6.7	single path	10	12
EfficientNet-B0 [31]	76.3	93.2	390	82	5.3	single path	-	-
DARTS [21]	73.3	91.3	574	-	4.7	a whole supernet	$4^{\dagger}$	-
GreedyNAS-A	**77.1**	93.3	366	77	6.5	single path	7	< 1

due to different implementation tools (*e.g.*, GPU types, dataloader wrappers), we calculated the accumulated number of images involved in a whole gradient-based optimization step, *i.e.*, #optimization in Table 1. Our GreedyNAS has an additional evaluation process during training, thus we also report the accumulated number of images for forward evaluation, *i.e.*, #evaluation. For overall efficiency comparison, we empirically find the cost of a whole optimization step is approximately 3.33 times larger than that of a forward evaluation. The corresponding corrected #optimization is covered accordingly.

From Table 1, we can see that the training cost of our GreedyNAS is much smaller than that of other comparison methods, which indicates GreedyNAS enjoys significant efficiency in supernet training since it greedily shrinks its training space into those potentially-good paths. Besides, we also implement Random Search and Uniform Sampling using same training cost of GreedyNAS, denoted as *Random Search-E* and *Uniform Sampling-E*, respectively. The results show that with decreased iterations of supernet training, the searched architectures are inferior to those of larger iterations. In contrast, our method can achieve higher accuracy by a large margin (almost 1%). This implies that GreedyNAS is capable of learning a decent supernet with much less iterations.

Searching on augmented search space. To comprehensively illustrate our superiority to various state-of-the-art NAS methods, we implement searching by augmenting the current space with an SE option. Moreover, we search the architectures under different FLOPs constraint. But we also report the corresponding latency and parameter capacity to comprehensively analyze the statistics of searched models. As Table 2 shows, our GreedyNAS achieves new state-of-the-art performance with respect to different FLOPs and latency levels. For example, with similar FLOPs and latency, GreedyNAS-C has higher Top-1 ACC than the competing SCARLET method by a margin of 0.6%. Our searched models are visualized in Figure 6 (see supplementary materials). It shows smaller network (GreedyNAS-C) tends to select more identity blocks to reduce the FLOPs while larger networks will exploit more SE modules (GreedyNAS-A&B) to further improve the accuracy performance. Moreover, GreedyNAS-A adopts more 3×3 kernels to have smaller latency since 3×3 kernels are optimized more maturely in mobile inference framework. We also report our real training cost in Table 2 based on Tesla V100 GPU. It shows that GreedyNAS can significantly reduce the training time compared to other NAS methods, which empirically validates the efficiency of GreedyNAS.

5.3. Ablation studies

5.3.1 Effect of evaluation in path filtering

To filter the weak paths, GreedyNAS evaluates each path by a small portion (1000) of validation images as a surrogate for the whole validation dataset (50K images). We first investigate whether this approximation suffices in our experiment. By random sampling 1000 paths from supernet, we examine the correlation of two path orderings, which are generated by ranking the evaluation results using 1000 and 50K validation images, respectively. In Table 3, we report the widely-used Spearman rho [25] and Kendall tau [13] rank correlation coefficient, which are in the range $[0, 1]$ and larger values mean stronger correlation. We also cover three types of supernets, *i.e.*, randomly initialized, trained by uniform sampling and our greedy sampling.

From Table 3, we can see that our greedy supernet achieves fairly high rank correlation coefficient (0.997 and 0.961), which indicates that the ranking of greedy supernet using 1000 validation images is significantly consistent with that of all validation dataset. Moreover, supernet trained with uniform sampling has smaller correlation coefficient, even with different evaluation images (see left

Table 3: Rank correlation coefficient of 1000 paths measured by the loss (ACC) of 1K validation images and ACC of 50K validation images w.r.t. different types of supernets.

Spearman rho			Kendall tau		
random	uniform(ACC)	greedy	random	uniform(ACC)	greedy
0.155	0.968(0.869)	**0.997**	0.113	0.851(0.699)	**0.961**

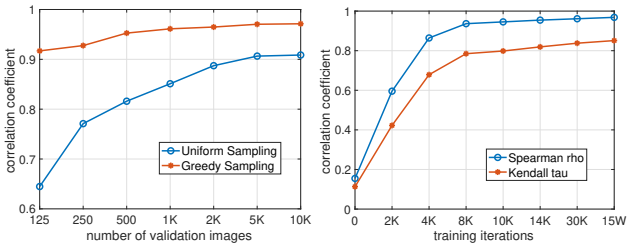


Figure 4: Rank correlation coefficient of 1000 paths measured by the loss of N validation images and ACC of the whole 50K validation images. Left: Comparison (Kendall tau) of supernet by uniform and greedy sampling w.r.t. different number N of evaluation images. Right: $N = 1$K w.r.t. different training iterations of supernet by uniform sampling.

Figure 4). This implies in a sense that our greedy supernet is more discriminative since it can use less images to identify whether a path is good or weak. Nevertheless, as left Figure 4 shows, too few evaluation images might have weak correlation while too many evaluation images mean greater evaluation cost. But 1000 evaluation images enjoy a balanced trade-off between the rank correlation and evaluation cost. Note that we also report the results w.r.t. ranking using the ACC of 1000 images, which is smaller than that using loss. This results from that during training the value of loss might be more informative than that of ACC.

As for the random supernet, the correlation coefficient is fairly small (0.155 and 0.113). This makes sense since the ranking is based on the classification performance; however, a random supernet fails to learn sufficient domain-related information but gives disordered ranking of paths. This smaller correlation coefficient implies that it might be not sensible to implement greedy sampling from a random supernet since the ranking evaluated by 1000 validation images will be rather noisy. In this way, we record the trend of rank correlation coefficients with uniform sampling in right Figure 4. It shows that with more iterations, the correlation coefficients increase and at 10K iteration, they tend to be steady at a high level (*e.g.*, 0.81 for Kendall Tau). As a result, in our GreedyNAS we propose to have a *warm-up* stage by uniform sampling for 10K iterations, so that we can safely use 1000 validation images to evaluate paths.

5.3.2 Effect of path filtering and candidate pool

To study the effect of our proposed path filtering and the candidate pool, we implement experiments on the search space without SE. In our GreedyNAS, path filtering is to block the training of weak paths. In contrast, the use of candidate pool is mainly three-fold as shown in Table 4. First, we can sample from it as the exploitation process; second, we can initialize the evolutionary searching with the pool for better paths; third, we can use it to adaptively stop the supernet training. Then we control each factor and obtain 6 variants of GreedyNAS as well as 6 corresponding searched architectures Net1∼Net6. For fair comparison, we search all nets under 330M FLOPs. Besides, if the candidate pool is not used for stopping training, we specify a maximum epoch 60 as [3].

Table 4: Comparison of accuracy performance of searched paths by GreedyNAS w.r.t. different usage of path filtering and candidate pool.

	path filtering	candidate pool			Top-1 (%)
		sampling (exploitation)	evolutionary initialization	training stopping	
Net1	-	-	-	-	74.31
Net2	✓	-	-	-	74.59
Net3	-	✓	✓	✓	74.48
Net4	✓	✓	-	✓	74.71
Net5	✓	✓	✓	-	74.84
Net6	✓	✓	✓	✓	74.89

As Table 4 shows, comparing with the baseline Net1 (Net3), Net2 (Net6) achieves 0.28% (0.41%) better Top-1 ACC, which indicates that path filtering does contribute to the supernet training, and thus improves the searching results. By involving the candidate pool, Net6 can increase its accuracy from 74.59% (Net2) to 74.89%. In specific, initialization with candidate pool in evolutionary algorithms enables to have a 0.18% gain on Top-1 ACC since it helps to search paths with higher ACC on supernet (also see Figure 3). Note that stopping by candidate pool usually saves training cost; however, full training with candidate pool (Net5) seems to drop the accuracy a bit (0.05% w.r.t. Net6). It might result from that extreme greedy exploitation on the candidate pool harms the supernet training instead. Then the stopping in a sense brings benefits for a more balanced trade-off between exploration and exploitation.

6. Conclusion

Training a supernet is a key issue for Single-path One-shot NAS methods. In stead of treating all paths equally, we propose to greedily focus on training those potentially-good ones. This greedy path filtering can be efficiently implemented by our proposed multi-path sampling strategy with rejection. Besides, we also adopt an exploration and exploitation policy and introduce a candidate pool to further boost the supernet training efficiency. Our proposed method GreedyNAS shows significant superiority in terms of both accuracy performance and training efficiency.

References

[1] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[2] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. Detnas: Neural architecture search on object detection. *arXiv preprint arXiv:1903.10979*, 2019.

[3] Xiangxiang Chu, Bo Zhang, Jixiang Li, Qingyuan Li, and Ruijun Xu. Scarletnas: Bridging the gap between scalability and fairness in neural architecture search. *arXiv preprint arXiv:1908.06022*, 2019.

[4] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019.

[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.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[7] Cheng Deng, Erkun Yang, Tongliang Liu, Jie Li, Wei Liu, and Dacheng Tao. Unsupervised semantic-preserving adversarial hashing for image search. *IEEE Transactions on Image Processing*, 28(8):4032–4044, 2019.

[8] Cheng Deng, Erkun Yang, Tongliang Liu, and Dacheng Tao. Two-stream deep hashing with class-specific centers for supervised image search. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[9] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[10] Minghao Guo, Zhao Zhong, Wei Wu, Dahua Lin, and Junjie Yan. Irlas: Inverse reinforcement learning for architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9021–9029, 2019.

[11] 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.

[12] Kai Han, Jianyuan Guo, Chao Zhang, and Mingjian Zhu. Attribute-aware attention model for fine-grained representation learning. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 2040–2048. ACM, 2018.

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

[14] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[15] Shumin Kong, Tianyu Guo, Shan You, and Chang Xu. Learning student networks with few data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[16] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[17] Yue Liao, Si Liu, Guanbin Li, Fei Wang, Yanjie Chen, Chen Qian, and Bo Li. A real-time cross-modality correlation filtering method for referring expression comprehension. *arXiv preprint arXiv:1909.07072*, 2019.

[18] Yue Liao, Si Liu, Fei Wang, Yanjie Chen, and Jiashi Feng. Ppdm: Parallel point detection and matching for real-time human-object interaction detection. *arXiv preprint arXiv:1912.12898*, 2019.

[19] Peiwen Lin, Peng Sun, Guangliang Cheng, Sirui Xie, Xi Li, and Jianping Shi. Graph-guided architecture search for real-time semantic segmentation. *arXiv preprint arXiv:1909.06793*, 2019.

[20] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR (Poster)*, 2019.

[22] 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.

[23] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[25] W Pirie. S pearman rank correlation coefficient. *Encyclopedia of statistical sciences*, 2004.

[26] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[27] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[28] 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.

[29] 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.

[30] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[31] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.

[32] Yehui Tang, Shan You, Chang Xu, Jin Han, Chen Qian, Boxin Shi, Chao Xu, and Changshui Zhang. Reborn filters: Pruning convolutional neural networks with limited data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[33] Yehui Tang, Shan You, Chang Xu, Boxin Shi, and Chao Xu. Bringing giant neural networks down to earth with unlabeled data. *arXiv preprint arXiv:1907.06065*, 2019.

[34] Chaoyue Wang, Chang Xu, Chaohui Wang, and Dacheng Tao. Perceptual adversarial networks for image-to-image transformation. *IEEE Transactions on Image Processing*, 27(8):4066–4079, 2018.

[35] Chaoyue Wang, Chang Xu, Xin Yao, and Dacheng Tao. Evolutionary generative adversarial networks. *IEEE Transactions on Evolutionary Computation*, 23(6):921–934, 2019.

[36] Fei Wang, Liren Chen, Cheng Li, Shiyao Huang, Yanjie Chen, Chen Qian, and Chen Change Loy. The devil of face recognition is in the noise. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 765–780, 2018.

[37] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2017.

[38] 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*, pages 10734–10742, 2019.

[39] Jianlong Wu, Keyu Long, Fei Wang, Chen Qian, Cheng Li, Zhouchen Lin, and Hongbin Zha. Deep comprehensive correlation mining for image clustering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8150–8159, 2019.

[40] Erkun Yang, Cheng Deng, Chao Li, Wei Liu, Jie Li, and Dacheng Tao. Shared predictive cross-modal deep quantization. *IEEE transactions on neural networks and learning systems*, 29(11):5292–5303, 2018.

[41] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1285–1294, 2017.

[42] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial distribution learning for effective neural architecture search. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[43] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

A. Details of evolutionary searching in Section 4.2

We present the details of our adopted NSGA-II [6] evolutionary algorithm in the following Algorithm 3. In our experiment, population size $N_{pop} = 50$ and number of generations $T = 20$.

Algorithm 3 Evolutionary Architecture Search

Input: supernet $\mathcal{N}$, candidate pool $\mathcal{P}$, population size N_{pop}, number of generations T, validation data $\mathcal{D}_{val}$, constraints $\mathcal{C}$.
Output: architecture with highest validation accuracy under constraints.

Initialize populations P_0 with $\mathcal{P}$ so that $|P_0| = N_{pop}$ and P_0 satisfies constraints $\mathcal{C}$.
$E = \emptyset$; # evaluation set E which stores all evaluated architectures with accuracy
for $i = 0, 1, ..., T-1$ **do**
  $Q_i = make\text{–}new\text{–}pop(P_i)$;
    # generate offspring population Q_i using binary tournament selection, recombination, and mutation operators
  $R_i = P_i \cup Q_i$;
  $F_i = fast\text{–}non\text{–}dominated\text{–}sort(R_i)$; # generate all nondominated fronts of R_i
  $P_{i+1} = \emptyset$ and $j = 0$;
  while $|P_{i+1}| + |F_i| \leq N_{pop}$ **do**
    $crowding\text{–}distance\text{–}assignment(F_i)$; # calculate crowding-distance in F_j
    $P_{i+1} = P_{i+1} \cup F_j$;
    $j = j + 1$;
  end while
  $E_i = evaluation\text{–}architecture(F_j, D_{val}, \mathcal{C})$; # evaluate architecture with constraints and validation data
  $E = E \cup E_i$ # extend E_i to E
  $Sort(F_j, E_i)$; # sort in descending order using E_i
  $P_{i+1} = P_{i+1} \cup F_j[1 : (N_{pop} - |P_{i+1}|)]$; # choose the first $(N_{pop} - |P_{i+1}|)$ elements of F_j
  $Q_{i+1} = make\text{–}new\text{–}pop(P_{i+1})$; # make new population with constraints
end for
return architecture with highest accuracy in E

B. More Experimental Results

B.1. Details of (augmented) search space

The macro-structure of supernet is presented in Table 5, where each operation choice for Choice Block is attached in Table 6.

Table 5: Macro-structure of supernet. "input" indicates the size of feature maps for each layer, and "channels" means for the number of output channels. "repeat" is for the number of stacking same blocks, and "stride" is for the stride of first block when stacked for multiple times. "MB1_K3" refers to Table 6.

input	block	channels	repeat	stride
$224^2 \times 3$	3×3 conv	32	1	2
$112^2 \times 32$	MB1_K3	16	1	1
$112^2 \times 16$	Choice Block	32	4	2
$56^2 \times 32$	Choice Block	40	4	2
$28^2 \times 40$	Choice Block	80	4	2
$14^2 \times 80$	Choice Block	96	4	1
$14^2 \times 96$	Choice Block	192	4	2
$7^2 \times 192$	Choice Block	320	1	1
$7^2 \times 320$	1×1 conv	1280	1	1
$7^2 \times 1280$	global avgpool	-	1	-
1280	FC	1000	1	-

Table 6: Operation choices for each MobileNetV2-based Choice Block in Table 5, where ID means for an identity mapping.

block type	expansion ratio	kernel	SE
MB1_K3	1	3	no
ID	-	-	-
MB3_K3	3	3	no
MB3_K5	3	5	no
MB3_K7	3	7	no
MB6_K3	6	3	no
MB6_K5	6	5	no
MB6_K7	6	7	no
MB3_K3_SE	3	3	yes
MB3_K5_SE	3	5	yes
MB3_K7_SE	3	7	yes
MB6_K3_SE	6	3	yes
MB6_K5_SE	6	5	yes
MB6_K7_SE	6	7	yes

B.2. Calculating corrected #optimization in Table 1

In our GreedyNAS, when equipped with the stopping principle of candidate pool, the supernet training is stopped at approximately 46-th epoch. Thus the accumulated number of examples calculated for a whole optimization step is equal to

$$\text{\#optimization=1.23M} \times 46,$$

where 1.23M refers to the quantity of training dataset. As for the path filtering, we evaluate 10 paths based on 1000 validation images, and select 5 paths for training, whose batch size is 1024. In this way, the number of images for evaluation amounts to

$$\begin{aligned}\text{\#evaluation} &= \text{1.23M} \times \frac{1000}{1024} \times \frac{10}{5} \times 46, \\ &= \text{2.40M} \times 46.\end{aligned}$$

Given our empirical findings that the cost of a whole optimization step is approximately 3.33 times larger than that of a forward evaluation, the corrected #optimization is thus

$$\begin{aligned}\text{corrected \#optimization} &= \text{\#optimization} + \text{\#evaluation}/3.33, \\ &= \text{1.23M} \times 46 + \text{2.40M} \times 46/3.33, \\ &= \text{89.7M}.\end{aligned}$$

B.3. Details of rank correlation coefficient

In ablation study 5.3.1, we use two Spearman rho [25] and Kendall tau [13] rank correlation coefficient to evaluate the correlation of two path orderings, which are generated by ranking the evaluation results using 1000 and 50K validation images, denoted as $\boldsymbol{r}$ and $\boldsymbol{s}$, respectively. Basically, we aim to calculate the correlation of $\boldsymbol{r}$ and $\boldsymbol{s}$.

For Spearman rho rank correlation coefficient, it is simply the Pearson correlation coefficient between random variable r and s, if we regard $\boldsymbol{r}$ and $\boldsymbol{s}$ as two observation vectors of random variable r and s, *i.e.*,

$$\rho_S = \frac{cov(r, s)}{\sigma_r \sigma_s},$$

where $cov(\cdot, \cdot)$ is the covariance of two variables, and σ_r (σ_s) is the standard deviations of r (s). Based on observation vectors, it can be more efficiently calculated as

$$\rho_S = 1 - \frac{6 \sum_{i=1}^{n} (\boldsymbol{r}_i - \boldsymbol{s}_i)^2}{n(n^2 - 1)},$$

where $n = 1000$ in our experiment.

For Kendall tau rank correlation coefficient, it focuses on the pairwise ranking performance. For any pair $(\boldsymbol{r}_i, \boldsymbol{r}_j)$ and $(\boldsymbol{s}_i, \boldsymbol{s}_j)$, it is said to be *concordant* if $\boldsymbol{r}_i > \boldsymbol{r}_j$ and $\boldsymbol{s}_i > \boldsymbol{s}_j$ hold simultaneously, or also for $\boldsymbol{r}_i < \boldsymbol{r}_j$ and $\boldsymbol{s}_i < \boldsymbol{s}_j$. Otherwise, it will be called *disconcordant*. Then the coefficient is calculated as

$$\rho_K = \frac{\text{\#concordant pairs - \#disconcordant pairs}}{\text{\#all pairs}},$$

where #all pairs $= \mathbb{C}_n^2$ refers to the total number of pairs. In this way, if two rankings $\boldsymbol{r}$ and $\boldsymbol{s}$ are exactly the same, ρ_K will be 1 while if the two are completely different (*i.e.*, one ranking is the reverse of the other), ρ_K will be -1. According to the definition, it can also be calculated in a closed-form as

$$\rho_K = \frac{2}{n(n-1)} \sum_{i<j} \text{sign}(\boldsymbol{r}_i - \boldsymbol{r}_j)\text{sign}(\boldsymbol{s}_i - \boldsymbol{s}_j),$$

where $\text{sign}(\cdot)$ is the sign function.

B.4. More ablation studies

B.4.1 Performance of trained supernet

To further investigate the performance of the trained supernet, we implement two different searching methods (random search and evolutionary search) on various trained supernet, *i.e.*, greedy supernet, uniform supernet (full training) and uniform supernet-E (same training cost with GreedyNAS). The results can be regarded as supplementary for Table 1.

Table 7: Comparison of performance on ImageNet dataset of searched architectures w.r.t. different supernets under same search space.

supernet	searcher	Top-1 (%)	FLOPs
uniform	random	74.07	321M
uniform-E	random	73.88	320M
greedy	random	74.22	321M
uniform	evolutionary	74.50	326M
uniform-E	evolutionary	74.17	320M
greedy	evolutionary	**74.85**	320M

From Table 7, we can see that a greedy supernet improves consistently the classification accuracy in terms of different searchers. This validates the superiority of our greedy supernet since it helps searchers to probe better architectures. Moreover, to comprehensively investigate the effect of supernets, we implement systematic sampling [4] to sample 30 paths from $50 \times 20 = 1000$ paths, which are discovered by the evolutionary algorithms and ranked according to the accuracy on supernet. Then we retrain these 30 paths from scratch, and report their distribution histogram in Figure 5.

As shown in Figure 5, we can see that on average, paths searched with our greedy supernet have higher retraining Top-1 accuracy than that with uniform supernet. This implies that our greedy supernet serves as a better performance estimator, so that those good paths can be eventually identified and searched.

[4] https://en.wikipedia.org/wiki/Systematic_sampling

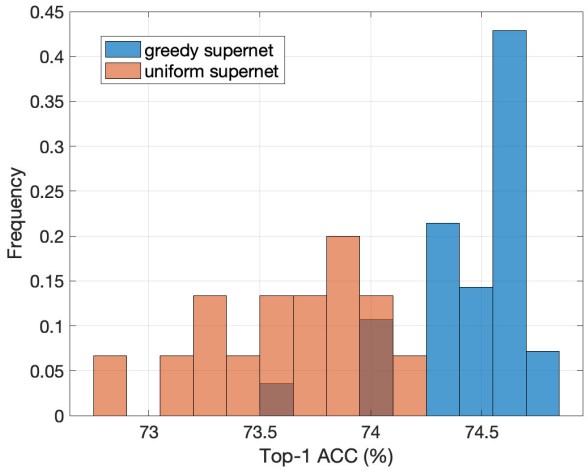
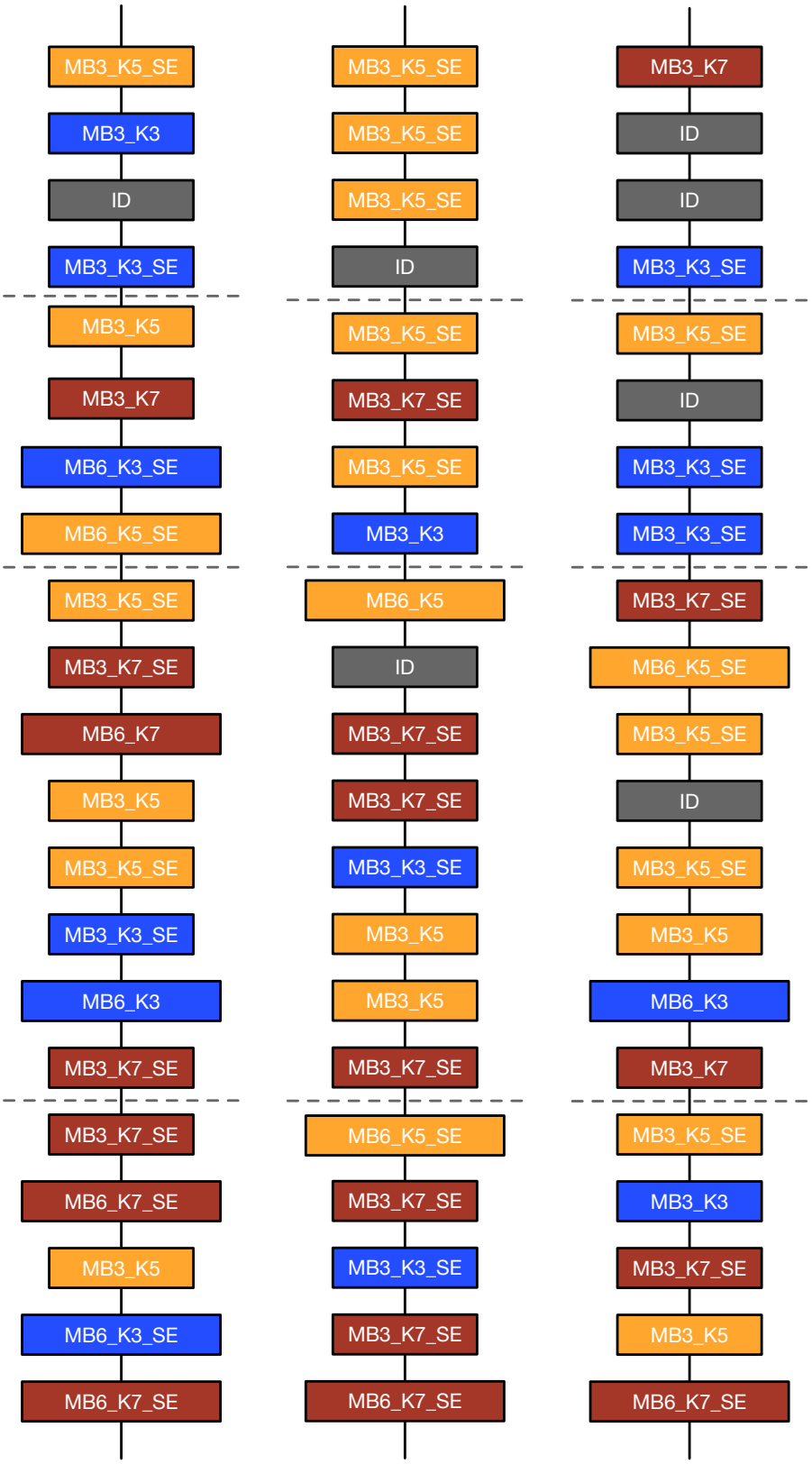


Figure 5: Top-1 accuracy histogram of 30 systematically sampled paths from 1000 paths searched by evolutionary algorithm after trained from scratch.

B.5. Visualization of searched architectures

We visualize the searched architectures by our GreedyNAS method in Figure 6.



(a) GreedyNAS-A (b) GreedyNAS-B (c) GreedyNAS-C

Figure 6: Visualization of searched architectures by GreedyNAS in Table 2.