Single Path One-Shot Neural Architecture Search with Uniform Sampling

Zichao Guo*, Xiangyu Zhang*, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, Jian Sun

Megvii Technology

Tsuinghua University, *Hong Kong University of Science and Technology*

{*guozichao, zhangxiangyu, hengwen, weiyichen, sunjian*}*@megvii.com*

muhy17@mails.tsinghua.edu.cn, *zliubq@connect.ust.hk*

Abstract

One-shot method [2] is a powerful Neural Architecture Search (NAS) framework, but its training is non-trivial and it is difficult to achieve competitive results on large scale datasets like ImageNet. In this work, we propose a Single Path One-Shot model to address its main challenge in the training. Our central idea is to construct a simplified supernet, Single Path Supernet, which is trained by an uniform path sampling method. All underlying architectures (and their weights) get trained fully and equally. Once we have a trained supernet, we apply an evolutionary algorithm to efficiently search the best-performing architectures without any fine tuning.

Comprehensive experiments verify that our approach is flexible and effective. It is easy to train and fast to search. It effortlessly supports complex search spaces (e.g., building blocks, channel, mixed-precision quantization) and different search constraints (e.g., FLOPs, latency). It is thus convenient to use for various needs. It achieves start-of-the-art performance on the large dataset ImageNet.

1. Introduction

Deep learning automates *feature engineering* and solves the *weight optimization* problems. Neural Architecture Search (NAS) aims to automate *architecture engineering* by solving one more problem, *architecture search*. Early NAS approaches [38, 34, 35, 13, 20, 24] use *nested* optimization. Architectures are sampled from the search space. Their weights are then trained from scratch. The computation is too intensive for large datasets.

Recent approaches [26, 4, 15, 29, 19, 32, 3, 2] adopt a *weight sharing* strategy to reduce the computation. A supernet is defined to subsume all architectures and is trained only once. All architectures inherit their weights directly from the supernet without training from scratch. Therefore, the computation is more efficient and affordable on large datasets.

Most weight sharing approaches use a continuous relaxation of the search space [26, 4, 15, 29, 32]. The architecture distribution is continuously parameterized. Such parameters are *jointly* optimized during the supernet training via gradient based methods. Thus, architecture search is performed during optimization, and the best architecture is sampled from the distribution after optimization. The formulation is elegant and theoretically sound. However, there are two issues. *First*, the weights in the supernet are deeply coupled. It is unclear why inherited weights for a specific architecture are still effective. *Second*, joint optimization introduces further coupling between the architecture parameters and supernet weights. The greedy nature of the gradient based methods inevitably introduces bias in both architecture distribution and supernet weights. This easily misleads the architecture search. Careful tuning of hyper parameters and optimization process are used in previous approaches.

One-shot approaches [3, 2] belong to a new paradigm. It defines the supernet and performs weight inheritance in a similar way. However, there is no architecture relaxation and distribution parametization. The architecture search problem is decoupled from the supernet training and addressed in a separate step. Thus, a one-shot approach is *sequential*. It combines the merits of both *nested* and *joint* optimization approaches above. The architecture search is both efficient and flexible.

While the second issue above is addressed, existing one-shot approaches [3, 2] do not address the first issue well. The weights in the supernet are still coupled. The optimization is complicated and involves sensitive hyper parameters. The performance is not yet competitive on large datasets.

This work is motivated by the excellence of the one-shot paradigm and aims to overcome its drawback. As firstly observed in [2], the key to the success of one-shot is that the accuracy of an architecture using inherited weights should be predictive for the accuracy using opti-

*Equal contribution. This work is done when Haoyuan Mu and Zechun Liu are interns at Megvii Inc.

mized weights. Thus, we propose that the supernet training should be *stochastic*, in that *all* architectures can have their weights optimized simultaneously. To reduce the weight coupling in the supernet, we propose using a simple search space, *single path supernet*, that only contains single path architectures. For training, we use a hyperparameter-free method, *uniform sampling*, to treat all architectures equally.

Our approach is simple and flexible. There is no hyper parameter in supernet training. The simplicity allows us to design a rich search space, including novel designs for channel size and bit width, all addressed in a unified manner. Architecture search is efficient as only inference using inherited weights is performed. Evolutionary algorithm is used to support multiple latency constraints easily.

Comprehensive ablation experiments and comparison to previous works on a large dataset (ImageNet) verify that the proposed approach is state-of-the-art in terms of accuracy, memory consumption, training time, architecture search efficiency and flexibility.

2. Review of NAS Approaches

Without loss of generality, the architecture search space $\mathcal{A}$ is represented by a directed acyclic graph (DAG). A network architecture is a subgraph $a \in \mathcal{A}$, denoted as $\mathcal{N}(a, w)$ with weights w.

Neural architecture search aims to solve two related problems. The first is *weight optimization* of a given network architecture as in standard deep learning,

$$w_a = \underset{w}{\operatorname{argmin}} \, \mathcal{L}_{\text{train}} \left(\mathcal{N}(a, w) \right), \tag{1}$$

where $\mathcal{L}_{\text{train}}(\cdot)$ is the loss function on the training set.

The second is *architecture optimization*. In a general sense, it finds the architecture that is trained on the training set and has best accuracy on the validation set, as

$$a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \text{ACC}_{\text{val}} \left(\mathcal{N}(a, w_a) \right), \tag{2}$$

where $\text{ACC}_{\text{val}}(\cdot)$ is the accuracy on the validation set.

Real world tasks usually have additional requirements on a network's memory consumption, FLOPs, latency, energy consumption, etc. These requirements are up to the architecture a, software and hardware platforms, but irrelevant to the weights w_a. Thus, they are called *architecture constraints* in this work. A typical constraint is that the network's latency is no more than a preset budget, as

$$\text{Latency}(a^*) \le \text{Lat}_{\text{max}}. \tag{3}$$

Note that it is challenging to satisfy Eq. (2) and Eq. (3) simultaneously for most previous approaches.

Early NAS approaches perform weight optimization and architecture optimization in a *nested* manner [37, 38, 34, 35, 1]. Numerous architectures are sampled from $\mathcal{A}$ and trained from scratch as in Eq. (1). Efficient architecture search algorithms are critical to making Eq. (1) affordable. Previous works use reinforcement learning [19, 24, 1, 37, 38] and evolution [21, 20, 18, 28, 14]. The computation cost is still high. Only the small dataset (e.g., CIFAR 10) and small search space (e.g, a single block) are affordable.

Weight Sharing Approaches Recent NAS approaches adopt a common *weight sharing* strategy [4, 15, 26, 29, 2, 3, 32, 19]. The architecture search space $\mathcal{A}$ is encoded in a *supernet*[1], denoted as $\mathcal{N}(\mathcal{A}, W)$, where W is the weights in the supernet. The supernet is trained once. All architectures inherit their weights directly from W. Thus, they share the weights in their common graph nodes. Fine tuning of an architecture is performed in need, (e.g., [19]) but no training from scratch is incurred. Therefore, architecture search is fast and suitable for large datasets like ImageNet.

Most weight sharing approaches convert the discrete architecture search space into a continuous one [26, 4, 15, 29, 32]. Formally, space $\mathcal{A}$ is relaxed to $\mathcal{A}(\theta)$, where θ denotes the continuous parameters that represent the *distribution* of the architectures in the space. Note that the new space subsumes the original one, $\mathcal{A} \subseteq \mathcal{A}(\theta)$. An architecture sampled from $\mathcal{A}(\theta)$ could be invalid in $\mathcal{A}$.

An evident advantage of the continuous search space is that gradient based methods [15, 4, 26, 25, 29, 32] become feasible for the *joint* optimization of both weights and architecture distribution parameters. This is in contrast to the nested optimization methods above, and expressed as

$$(\theta^*, W_{\theta^*}) = \underset{\theta, W}{\operatorname{argmin}} \, \mathcal{L}_{train}(\mathcal{N}(\mathcal{A}(\theta), W)). \tag{4}$$

After optimization, the best architecture a^* could be sampled from $\mathcal{A}(\theta^*)$. Note that it could be invalid in $\mathcal{A}$. If so, it is validated (e.g., by binarization of θ [15]). It then inherits the weights from W_{θ^*} and is fine-tuned.

While this is theoretically sound, the optimization of Eq. (4) is challenging. *First*, the weights of the graph nodes in the supernet depend on each other and become *deeply coupled* during optimization. For a specific architecture, the inherited weights from W are decoupled from the dependents. It is unclear why such weights are effective, although they could be better than random (training from scratch).

Second, joint optimization of architecture parameter θ and weights W introduces further coupling. The parameter space is also huge. All these make the optimization extremely difficult. Specifically, solving Eq. (4) inevitably introduces bias to certain areas in θ and certain nodes in W during the progress of optimization. The bias would leave

[1] "Supernet" is used as a general concept in this work. It has different names and implementation in previous approaches.

certain nodes in the graph poorly trained. Different architectures quickly become non-comparable. Yet, their prediction accuracy is still used as guidance for sampling in $\mathcal{A}(\theta)$ (e.g., used as reward in policy gradient [4]). This could mislead the search. This problem is in analogy to the notable "dilemma of exploitation and exploration" problem in reinforcement learning. To alleviate such problems, existing approaches adopt complicated optimization processes (see Table 1 for a summary). Nevertheless, their effectiveness is not fully verified [12].

It is also hard to impose architecture constraints. Some works augment the loss function $\mathcal{L}_{train}$ in Eq. (4) with carefully designed *soft* loss terms that consider the architecture latency [4, 26, 29, 25]. However, it is hard, if not impossible, to guarantee a hard constraint like Eq. (3).

3. Our One-Shot NAS Approach

3.1. Revisiting One-Shot NAS

As analyzed above, the *coupled* architecture search and weight optimization is challenging and could be problematic. Recall that the earlier NAS approaches solve the two problems using nested optimization, as in Eq. (1) and Eq. (2). This raises the question of whether we can have the merits of both *problem decoupling* and *weight sharing*.

This consideration leads to the so called *one-shot* approaches [3, 2]. Such approaches still train a supernet once and allow architectures to share the weights therein later. However, the supernet training and architecture search are decoupled, in two *sequential* steps. Note that this is different from either the *nested* optimization [37, 38, 34, 35, 1] or the *joint* optimization [26, 4, 15, 29, 32].

Firstly, the supernet weight is optimized as

$$W_{\mathcal{A}} = \underset{W}{\operatorname{argmin}}\, \mathcal{L}_{\text{train}}\left(\mathcal{N}(\mathcal{A}, W)\right). \qquad (5)$$

Compared to Eq. (4), the continuous parameterization of search space is gone. Only weights are optimized.

Secondly, architecture searched is performed as

$$a^* = \underset{a\in\mathcal{A}}{\operatorname{argmax}} \text{ACC}_{\text{val}}\left(\mathcal{N}(a, W_{\mathcal{A}}(a))\right). \qquad (6)$$

During search, each sampled architecture a inherits its weights from $W_{\mathcal{A}}$ as $W_{\mathcal{A}}(a)$. The key difference of Eq. (6) from Eq. (1) and (2) is that architecture weights are properly initialized. Evaluation of $\text{ACC}_{val}(\cdot)$ only requires inference. There is no fine tuning (e.g., [19]) or retraining (e.g., [38, 34, 35, 24]). Thus, the search is very *efficient*. The found optimal a^* is then fine tuned to obtain w_{a^*}.

The search is also *flexible*. Any adequate search algorithm is feasible. This work uses evolutionary search (Sec. 3.4). The architecture constraint like Eq. (3) can be exactly satisfied. Search can be repeated many times on the same supernet once trained, using different constraints (e.g., 100ms latency and 200ms latency). These properties are absent in previous approaches. They make One-Shot NAS attractive for real world tasks.

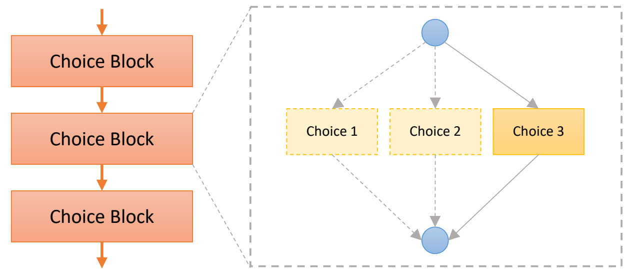


Figure 1. Architecture of a single path supernet. It consists of a series of *choice blocks*. Each has several *choices*. Only one choice is invoked at the same time.

One problem in Sec. 2 still remains. The weights of the graph nodes in the supernet training Eq. 5) are coupled. It is unclear why the inherited weight $W_{\mathcal{A}}(a)$ is good for an arbitrary architecture a.

The recent one-shot approach [2] attempts to decouple the weights using a "path dropout" strategy. During an SGD step in Eq. (5), each edge in the supernet graph is randomly dropped. The chance is controlled via a dropout rate parameter. In this way, the co-adaptation of the node weights is reduced during training. Experiments in [2] indicate that the performance is very sensitive to the dropout rate parameter. Careful tuning is necessary. In fact, a heating-up like strategy is used. The effect of the parameter and the strategy is also likely related to the search space[2]. In our experiment, we find tuning this parameter and strategy very difficult. The validation accuracy is very sensitive to the dropout rate parameter.

3.2. Single Path Supernet and Uniform Sampling

Let us restart to derive the fundamental principle to make the One-Shot paradigm effective. The key to the success of architecture search in Eq. (6) is that, the accuracy of *any* architecture a on a validation set using inherited weight $W_{\mathcal{A}}(a)$ (without extra fine tuning) is highly predictive. As Eq. (1) is the ideal case, this requires that the weight $W_{\mathcal{A}}(a)$ to approximate the optimal weight w_a. The quality of the approximation depends on how well the training loss $\mathcal{L}_{\text{train}}\left(\mathcal{N}(a, W_{\mathcal{A}}(a))\right)$ is minimized. This gives rise to the principle that the supernet weights $W_{\mathcal{A}}$ should be optimized in a way that *all* architectures in the search space are optimized *simultaneously*. This is expressed as

$$W_{\mathcal{A}} = \underset{W}{\operatorname{argmin}}\, \mathbb{E}_{a\sim\Gamma(\mathcal{A})}\left[\mathcal{L}_{\text{train}}(\mathcal{N}(a, W(a)))\right], \qquad (7)$$

[2]The search space of each block in [2] contains an identity path. This greatly eases the training.

where $\Gamma(\mathcal{A})$ is a prior distribution of $a \in \mathcal{A}$. Note that Eq. (7) is an implementation of Eq. (5). In each step of optimization, an architecture a is randomly sampled. Only weights $W(a)$ are activated and updated. This is memory saving and efficient. In this sense, the supernet itself is no longer a valid network. It behaves as a *stochastic supernet* [25]. This is different from [2].

To reduce the co-adaptation between node weights, we propose simplifying the search space $\mathcal{A}$ to the extreme. It only contains *single path* architectures as Fig.1 shows. This strategy can be thought as the extreme opposite to the path dropout strategy in [2], with all but one path are dropped in each training round. There is no dropout rate parameter or any tuning. Training converges well in our experiments.

The prior distribution $\Gamma(\mathcal{A})$ could be important. In this work, we empirically find that *uniform sampling* is good enough. This is not much of a surprise. A recent work also finds that purely random search is competitive to several SOTA NAS approaches [12]. We also experimented with a variant that samples the architectures uniformly according to their constraints. This is because a real task usually expects to find multiple architectures satisfying different constraints (e.g., 100 ms latency of 300M FLOPs). Fig. 2 shows that both sampling methods work well. The *uniform constraint sampling* method is slightly better. It is used by default in the paper.

We note that sampling a path according to architecture distribution during optimization is already used in previous weight sharing approaches [19, 25, 26, 4, 29, 32]. The difference is that, the distribution $\Gamma(\mathcal{A})$ is a *fixed* priori during our training (Eq. (7)), while it is *learnable and updated* (Eq. (4)) in previous approaches (e.g. RL [19], policy gradient [25, 4], Gumbel Softmax [26, 29], APG [32]). As analyzed in Sec. 2, the latter makes the supernet weight and architecture optimization highly correlated and difficult.

Comprehensive experiments in Sec. 4 show that our simple approach achieves better results than the SOTA methods [4, 26]. Note that we do not claim that using a fixed prior distribution is *inherently* better than optimizing the distribution during training. There is no such theoretical guarantee. Our better result likely benefits from the fact that the joint optimization in Eq. (4) is too difficult with the current maturity of optimization techniques.

3.3. Supernet Architecture and Choice Blocks

Following the name in [2], *choice blocks* are used to build a *stochastic* architecture. Fig. 1 illustrates an example case. A choice block consists of multiple architecture choices. For our single path supernet (Sec. 3.2), each choice block only has one choice invoked at the same time. A path is obtained by sampling all the choice blocks.

The simplicity of our approach enables us to define different types of choice blocks to search various architecture

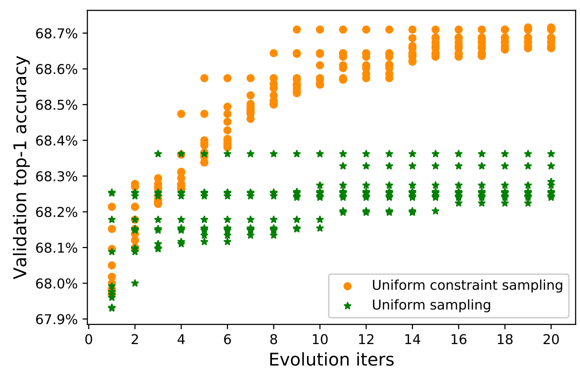


Figure 2. Evolutionary architecture search (see Sec. 3.4) on the single path supernets with different sampling strategies.

variables. Specifically, we propose two novel choice blocks to support complex search spaces.

Channel Number Search. The choice block aims to search the number of channels of a convolutional layer. The main idea is to preallocate a weight tensor with maximum number of channels. During supernet training, the system randomly selects the channel number and slices out the corresponding subtensor for convolution. Please refer to Sec. 4 and Fig. 4 for details.

Mixed-Precision Quantization Search. The choice block is designed to search the quantization precision of weights and features for a convolutional layer. In supernet training, bit widths of feature maps and kernel weights are randomly chosen. See Sec. 4 and Fig. 5 for details.

Sec. 4 demonstrates that the composition of all our choice blocks forms a huge and rich search space for various needs.

3.4. Evolutionary Architecture Search

For architecture search in Eq. (6), previous one-shot works [3, 2] use random search. This is not effective for a large search space. This work uses an evolutionary algorithm. Note that evolutionary search in NAS is used in [20], but it is costly as each architecture is trained from scratch. In our search, each architecture only performs inference. This is very efficient.

The algorithm is elaborated in Algorithm 1. For all experiments, population size $P = 50$, max iterations $\mathcal{T} = 20$ and $k = 10$. For crossover, two randomly selected candidates are crossed to produce a new one. For mutation, a randomly selected candidate mutates its every choice block with probability 0.1 to produce a new candidate. Crossover and mutation are repeated to generate enough new candidates that meet the given architecture constraints. Be-

Algorithm 1 Evolutionary Architecture Search

Input: *supernet weights* $\mathrm{W}_{\mathcal{A}}$*, population size P, architecture constraints* $\mathcal{C}$*, max iteration* $\mathcal{T}$*, validation dataset* D_{val}

Output: *the architecture with highest validation accuracy under architecture constraints*

$\mathrm{P}_0 := Initialize\_population(P, \mathcal{C})$;
$n := P/2$; # Crossover number
$m := P/2$; # Mutation number
$prob := 0.1$; # Probability to mutate
$\mathrm{Topk} := \emptyset$;
for $i = 1 : \mathcal{T}$ **do**
  $\mathrm{ACC}_{i-1} := Inference(\mathrm{W}_{\mathcal{A}}, \mathrm{D}_{val}, \mathrm{P}_{i-1})$;
  $\mathrm{Topk} := Update\_Topk(\mathrm{Topk}, \mathrm{P}_{i-1}, \mathrm{ACC}_{i-1})$;
  $\mathrm{P}_{crossover} := Crossover(\mathrm{Topk}, n, \mathcal{C})$;
  $\mathrm{P}_{mutation} := Mutation(\mathrm{Topk}, m, prob, \mathcal{C})$;
  $\mathrm{P}_i := \mathrm{P}_{crossover} \cup \mathrm{P}_{mutation}$;
end for
return the entry with highest accuracy in Topk;

fore the inference of an architecture, the statistics of all the *Batch Normalization (BN)* [11] operations are recalculated on a random subset of training data (20000 images on ImageNet). It takes a few seconds. This is because the BN statistics from the supernet are usually not applicable to the candidate nets. This is also referred in [2].

Fig. 3 plots the accuracy on the validation set over generations during evolution, using both evolutionary and random search methods. It is clear that evolutionary search is more effective. Experiment details are in Sec. 4.

The evolutionary algorithm is flexible in dealing with different constraints in Eq. (3), because the mutation and crossover processes can be directly controlled to generate proper candidates to satisfy the constraints. Previous RL-based [24] and gradient-based [4, 26, 25] methods design tricky rewards or loss functions to deal with such constraints. For example, [26] uses a loss function $\mathrm{CE}(a, w_a) \cdot \alpha \log(\mathrm{LAT}(a))^{\beta}$ to balance the accuracy and the latency. It is hard to tune the hyper parameter β to satisfy a hard constraint like Eq. (3).

3.5. Summary

The combination of single path supernet, uniform sampling training strategy, evolutionary architecture search, and rich search space design makes our approach simple, efficient and flexible. Table 1 performs a comprehensive comparison of our approach against previous weight sharing approaches on various aspects. Ours is the easiest to train, occupies the smallest memory, best satisfies the architecture (latency) constraint, and easily supports large datasets. Extensive results in Sec. 4 verify that our approach is the state-of-the-art.

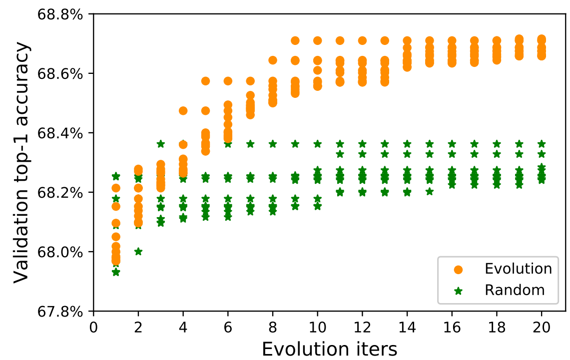


Figure 3. Evolutionary vs. random architecture search. See Sec. 3.4 and 4 for details.

4. Experiment Results

Dataset. All experiments are performed on *ImageNet* [22]. We randomly split the original training set into two parts: 50000 images for validation (50 images for each class exactly) and the rest as the training set. The original validation set is used for testing, on which all the evaluation results are reported, following [4].

Training. For the training of the supernet and retraining of the best architecture (after evolutionary search) from scratch, we use the same settings (including data augmentation strategy, learning rate schedule, etc.) as [17]. The batch size is 1024. Supernet is trained for 120 epochs (150000 iterations) and the best architecture for 240 epochs (300000 iterations). Training uses 8 *NVIDIA GTX 1080Ti* GPUs.

Search Space: Building Blocks First, we evaluate our method on the task of *building block selection*, i.e. to find the optimal combination of building blocks under a certain complexity constraint. Such search space has been exploited in recent state-of-the-art NAS systems like [24, 4, 26], which achieves outstanding performance on ImageNet.

Our basic building block design is inspired by a state-of-the-art manually-designed network – *ShuffleNet v2* [17]. Table 2 shows the overall architecture of the supernet. There are 20 *choice blocks* in total. Each choice block has 4 candidates, namely "choice_3", "choice_5", "choice_7" and "choice_x" respectively (see Appendix 1 for details). They differ in kernel sizes and the number of depthwise convolutions. The size of the search space is 4^{20}.

We use FLOPs $\leq 330M$ as the complexity constraint, as the FLOPs of a plenty of previous networks lies in [300,330], including manually-designed networks [10, 23, 33, 17] and those obtained in NAS [4, 26, 24].

Table 3 shows the results. For comparison, we set up a series of baselines as follows: 1) select a certain block

Approach	Supernet optimization	Architecture search	Hyper parameters in supernet Training	Memory consumption in supernet training	How to satisfy constraint	Experiment on ImageNet
ENAS [19]	Alternative RL and fine tuning		Short-time fine tuning setting	Single path + RL system	None	No
BSN [25]	Stochastic super networks + policy gradient		Weight of cost penalty	Single path	Soft constraint in training. Not guaranteed	No
DARTS [15]	Gradient-based, path dropout		Path dropout rate. Weight of auxiliary loss	Whole supernet	None	Transfer
Proxyless [4]	Stochastic relaxation of the discrete search + policy gradient		Scaling factor of latency loss	Two paths	Soft constraint in training. Not guaranteed.	Yes
FBNet [26]	Stochastic relaxation of the discrete search to differentiable optimization via Gumbel softmax		Temperature parameter in Gumbel softmax. Coefficient in constraint loss	Whole supernet	Soft constraint in training. Not guaranteed.	Yes
SNAS [29]	Same as FBNet		Same as FBNet	Whole supernet	Soft constraint in training. Not guaranteed.	Transfer
SMASH [3]	Hypernetwork	Random	None	Hypernet+single Path	None	No
One-Shot [2]	Path dropout	Random	Drop rate	Whole supernet	Not investigated	Yes
Ours	Uniform path sampling	Evolution	None	Single path	Guaranteed in searching. Support multiple constraints.	Yes

Table 1. Overview and comparison of SOTA *weight sharing* approaches. Ours is the easiest to train, occupies the smallest memory, best satisfy the architecture (latency) constraint, and easily supports the large dataset. Note that those approaches belonging to the joint optimization category (Eq. (4)) have "Supernet optimization" and "Architecture search" columns merged.

input shape	block	channels	repeat	stride
$224^2 \times 3$	3×3 conv	16	1	2
$112^2 \times 16$	CB	64	4	2
$56^2 \times 64$	CB	160	4	2
$28^2 \times 160$	CB	320	8	2
$14^2 \times 320$	CB	640	4	2
$7^2 \times 640$	1×1 conv	1024	1	1
$7^2 \times 1024$	GAP	-	1	-
1024	fc	1000	1	-

Table 2. Supernet architecture. *CB* – choice block. *GAP* – global average pooling. The "stride" column represents the stride of the first block in each repeated group.

model	FLOPs	top-1 acc(%)
all choice_3	324M	73.4
all choice_5	321M	73.5
all choice_7	327M	73.6
all choice_x	326M	73.5
random select (5 times)	~320M	~73.7
SPS + random search	323M	73.8
ours (fully-equipped)	319M	**74.3**

Table 3. Results of building block search. *SPS* – single path supernet (Sec. 3.2).

choice only (denoted by "all choice_*" entries); note that different choices have different complexity, thus we rescale the channels uniformly to adjust the FLOPs. 2) Randomly select some candidates from the search space. 3) Replace our evolutionary architecture optimization with random search used in [3, 2]. Results show that random search equipped with our single path supernet finds an architecture only slightly better that random select (73.8 vs. 73.7). It does no mean that our single path supernet is less effective. This is just because the random search is too naive to pick good candidates from the large search space. Using evolutionary search, our approach finds out an architecture that achieves superior accuracy (74.3) over all the baselines.

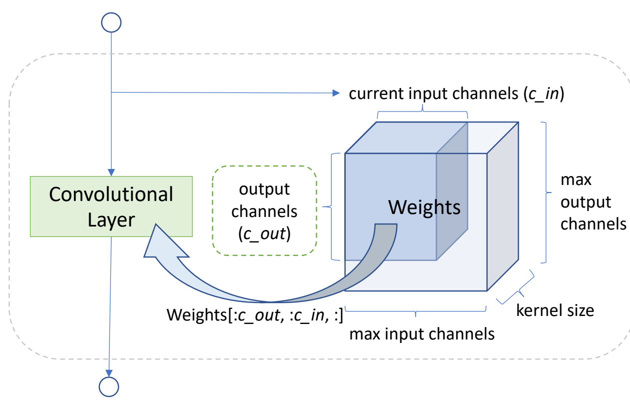


Figure 4. *Choice block* for channel number search.

Search Space: Channels Searching the number of channels in convolutional layers is more challenging, because the number of output channels in the current layer is *correlated* with the number of input channels in the next layer. Previous works usually deal with the problem by *channel pruning* [8, 9] or *model adaption* [30, 6].

We propose a new choice block for channel number search, as shown in Fig. 4. Weights of dimensions (max_c_out, max_c_in, ksize) for the convolutional kernels are preallocated. For each batch in supernet training, the number of current output channels c_out is randomly sampled. Then, we slice out the weights for current batch with the form Weights[: c_out, : c_in, :], which is used to produce the output. The optimal number of channels is determined in the evolutionary step.

Model	FLOPs	Top-1 acc(%)
all choice_3	324M	73.4
rand sel. channels (5 times)	$\sim$ 323M	$\sim$ 73.1
choice_3 + channel search	329M	**73.9**
rand sel. blocks + channels	$\sim$ 325M	$\sim$ 73.4
block search	319M	74.3
block search + channel search	328M	**74.7**
MobileNet V1 (0.75x) [10]	325M	68.4
MobileNet V2 (1.0x) [23]	300M	72.0
ShuffleNet V2 (1.5x) [17]	299M	72.6
NASNET-A [38]	564M	74.0
PNASNET [13]	588M	74.2
MnasNet [24]	317M	74.0
DARTS [15]	595M	73.1
Proxyless-R (mobile)* [4]	320M	74.2 (74.6)
FBNet-B* [26]	295M	74.1 (74.1)

Table 4. Results of channel search. * Performances are reported in the form "x (y)", where "x" means the accuracy retrained by us and "y" means accuracy reported by the original paper.

We first evaluate channel search on the baseline structure "all choice_3" (refer to Table 3): for each building block, we search the number of "mid-channels" (output channels of the first 1x1 conv in each building block) varying from 0.2x to 1.6x (with stride 0.2), where "k-x" means k times the number of default channels. Same as Sec. 4, we set the complexity constraint FLOPs $\leq 330M$. Table 4 (first part) shows the result. Our channel search method has higher accuracy (73.9) than the baselines.

To further boost the accuracy, we search building blocks and channels jointly. There are two alternatives: 1) running channel search on the best building block search result of Sec. 4; or 2) searching on the combined search space directly. In our experiments, we find the results of the first pipeline is slightly better. As shown in Table 4, searching in the joint space achieves the best accuracy (**74.7%** acc.), surpassing all the previous state-of-the-art manually-designed [17, 23] or automatically-searched models [24, 38, 13, 15, 4, 26] under the complexity of $\sim$ 300M FLOPs.

Comparison with State-of-the-arts Results in Table 4 shows our method is superior. Nevertheless, the comparisons could be unfair because different search spaces and training methods are used in previous works [4]. To make *direct* comparisons, we benchmark our approach to the *same* search space of [4, 26]. In addition, we retrain the searched models reported in [4, 26] under the same settings to guarantee the fair comparison.

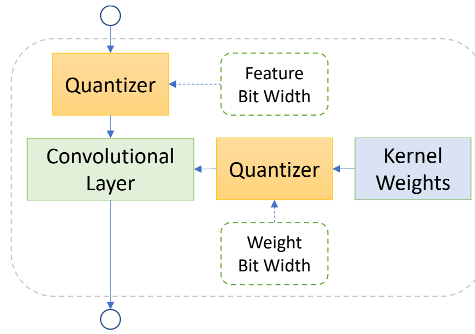


Figure 5. *Choice block* for mixed-precision quantization search.

The search space and supernet architecture in *ProxylessNAS* [4] is inspired by *MobileNet v2* [23] and *MnasNet* [24]. It contains 21 *choice blocks*; each choice block has 7 choices (6 different building blocks and one skip layer). The size of the search space is 7^{21}. *FBNet* [26] also uses a similar search space.

Table 5 reports the accuracy and complexities (FLOPs and latency on our device) of 5 models searched by [4, 26], as the baselines. Then, for each baseline, our search method runs under the constraints of same FLOPs or same latency, respectively. Results shows that for all the cases our method achieves comparable or higher accuracy than the counterpart baselines. We also point out that since the target devices in [4, 26] are different from ours, the reported results may be sub-optimal on our platform.

Furthermore, it is worth noting that all our 10 architectures in Table 5 are searched on the *same* supernet, justifying the flexibility and efficiency of our approach to deal with different complexity constraints: supernet is trained once and searched multiple times. In contrast, previous methods [26, 4] have to train multiple supernets under various constraints. According to Table 7, searching is much cheaper than supernet training.

Application: Mixed-Precision Quantization Quantization is required to deploy models on low-power devices. Given a structure with the overall complexity constraint (usually measured in BitOps), it is difficult to determine the optimal bit widths for the weights and the activations in different layers. Finding a good trade-off between channel numbers and bit width is also challenging. We demonstrate our framework can deal with these problems conveniently.

The search space here consists of the *channel search space* discussed earlier, and a *mixed-precision quantization search space*. The latter uses a novel *choice block* to search the bit widths of the weights and feature maps, as shown in Fig. 5. During supernet training, for each choice block fea-

baseline network	FLOPs	latency	top-1 acc(%) baseline	top-1 acc(%) ours (same FLOPs)	top-1 acc(%) ours (same latency)
FBNet-A [26]	249M	13ms	73.0 (73.0)	**73.2**	**73.3**
FBNet-B [26]	295M	17ms	74.1 (74.1)	**74.2**	**74.8**
FBNet-C [26]	375M	19ms	74.9 (74.9)	**75.0**	**75.1**
Proxyless-R (mobile) [4]	320M	17ms	74.2 (74.6)	**74.5**	**74.8**
Proxyless (GPU) [4]	465M	22ms	74.7 (75.1)	**74.8**	**75.3**

Table 5. Compared with state-of-the-art *NAS* methods [26, 4] *using the same search space*. The latency is evaluated on a single *NVIDIA Titan XP* GPU, with $batchsize = 32$. Accuracy numbers in the brackets are reported by the original papers; others are trained by us. All our architectures are searched from the **same** supernet via evolutionary architecture optimization (see Sec. 3.4).

method	BitOps	top-1 acc(%)
ResNet-18	float point	70.9
2W2A	6.32G	65.6
ours	**6.21G**	**66.4**
3W3A	14.21G	68.3
DNAS [27]	15.62G	68.7
ours	**13.49G**	**69.4**
4W4A	25.27G	69.3
DNAS [27]	25.70G	**70.6**
ours	**24.31G**	70.5
ResNet-34	float point	75.0
2W2A	13.21G	70.8
ours	**13.11G**	**71.5**
3W3A	29.72G	72.5
DNAS [27]	38.64G	73.2
ours	**28.78G**	**73.9**
4W4A	52.83G	73.5
DNAS [27]	57.31G	74.0
ours	**51.92G**	**74.6**

Table 6. Results of mixed-precision quantization search. "kWkA" means k-bit quantization for all the weights and activations.

Method	Proxyless	FBNet	Ours
GPU memory cost (8 GPUs in total)	37G	63G	24G
Training time	15 Gds	20 Gds	12 Gds
Search time	0	0	<1 Gds
Retrain time	16 Gds	16 Gds	16 Gds
Total time	31 Gds	36 Gds	29 Gds

Table 7. Search Cost. *Gds* - GPU days

ture bit width and weight bit width are randomly sampled. They are determined in the evolutionary step.

Evaluation is on *ResNet-18* and *ResNet-34* [7], as common practice in previous quantization works (e.g. [5, 27, 16, 36, 31]). Following [36, 5, 27], we only search and quantize the *res-blocks*, excluding the first convolutional layer and the last fully-connected layer. In the search space, choices of weight and feature bit widths include $\{(1,2),(2,2),(1,4),(2,4),(3,4),(4,4)\}$. As for channel search, we search the number of "bottleneck channels" (i.e. the output channels of the first convolutional layer in each residual block) in $\{0.5x, 1.0x, 1.5x\}$, where "k-x" means k times the number of original channels. The size of the search space is $(3\times 6)^N = 18^N$, where N is the number of choice blocks ($N = 8$ for ResNet-18 and $N = 16$ for ResNet-34). Note that for each building block we use the same bit widths for the two convolutions. We use *PACT* [5] as the quantization algorithm.

Table 6 reports the results. The baselines are denoted as kWkA ($k = 2, 3, 4$), which means uniform quantization of weights and activations with k-bits. Then, our search method runs under the constraints of the corresponding BitOps. We also compare with a recent mixed-precision quantization search approach [27]. Results shows that our method achieves superior accuracy in most cases. Also note that all our results for ResNet-18 and ResNet-34 are searched on the **same** supernet. This is very efficient.

Search Cost Analysis The search cost is a matter of concern in NAS methods. In this section, we analyze the search cost of our method and previous methods [26, 4] (reimplemented by us). We use the search space of our *building blocks* to measure the memory cost of training supernet and overall time cost. All the supernets are trained for 120 epochs with a batch size of 256. All models are trained with 8 GPUs. The results are reported in Table 7. For GPU memory, our approach clearly uses less memory than other two methods because of the single path supernet. Although we have an extra search step that costs less than 1 GPU day, our approach is much more efficient overall. Note Table 7 only compares a single run. Our approach is more advantageous when multiple searches are needed.

In practice, our approach is also more convenient to use. As summarized in Table 1, it guarantees to find out the architecture satisfying constraints within one search. Repeated search is easily supported. In contrast, *FBNet* [26] samples 6 architectures from the final distribution to be trained from scratch after supernet training. To satisfy a hard constraint, the two methods [26, 4] need to adjust their soft loss term coefficient and train the supernet repeatedly.

References

[1] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. 2, 3

[2] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 549–558, 2018. 1, 2, 3, 4, 5, 6

[3] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017. 1, 2, 3, 4, 6

[4] H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 1, 2, 3, 4, 5, 6, 7, 8

[5] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018. 8

[6] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. *arXiv preprint arXiv:1812.08934*, 2018. 6

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 8

[8] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018. 6

[9] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017. 6

[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 5, 7

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 5

[12] L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019. 3, 4

[13] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 1, 7

[14] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017. 2

[15] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1, 2, 3, 6, 7

[16] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 722–737, 2018. 8

[17] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018. 5, 7

[18] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019. 2

[19] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 1, 2, 3, 4, 6

[20] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018. 1, 2, 4

[21] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017. 2

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 5

[23] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 5, 7

[24] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018. 1, 2, 3, 5, 7

[25] T. Véniat and L. Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3492–3500, 2018. 2, 3, 4, 5, 6

[26] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *arXiv preprint arXiv:1812.03443*, 2018. 1, 2, 3, 4, 5, 6, 7, 8

[27] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018. 8

[28] L. Xie and A. Yuille. Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388, 2017. 2

[29] S. Xie, H. Zheng, C. Liu, and L. Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018. 1, 2, 3, 4, 6

[30] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018. 6

[31] D. Zhang, J. Yang, D. Ye, and G. Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 365–382, 2018. 8

[32] X. Zhang, Z. Huang, and N. Wang. You only search once: Single shot neural architecture search via direct sparse optimization. *arXiv preprint arXiv:1811.01567*, 2018. 1, 2, 3, 4

[33] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018. 5

[34] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2423–2432, 2018. 1, 2, 3

[35] Z. Zhong, Z. Yang, B. Deng, J. Yan, W. Wu, J. Shao, and C.-L. Liu. Blockqnn: Efficient block-wise neural network architecture generation. *arXiv preprint arXiv:1808.05584*, 2018. 1, 2, 3

[36] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 8

[37] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 2, 3

[38] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 1, 2, 3, 7

Appendix 1

Structures of choice blocks

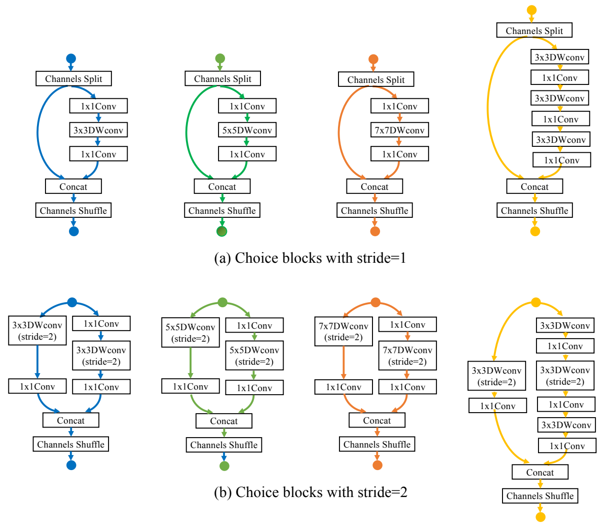


Figure 1. *Choice blocks* used in Sec. 4. From left to right : *Choice_3, Choice_5, Choice_7, Choice_x*.

Structures of searched architectures

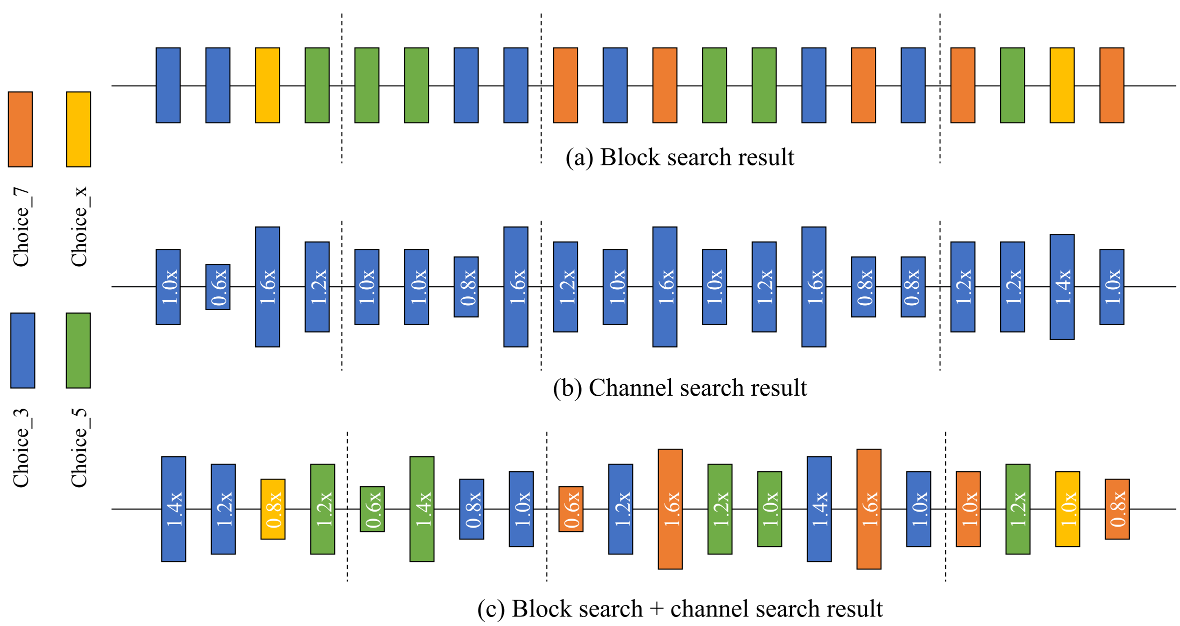


Figure 2. Structures of searched architectures in Sec. 4. (a) Result of building block search. (b) Result of channel search on all_choice_3 structure. (c) Result of channel search on best building block search structure.

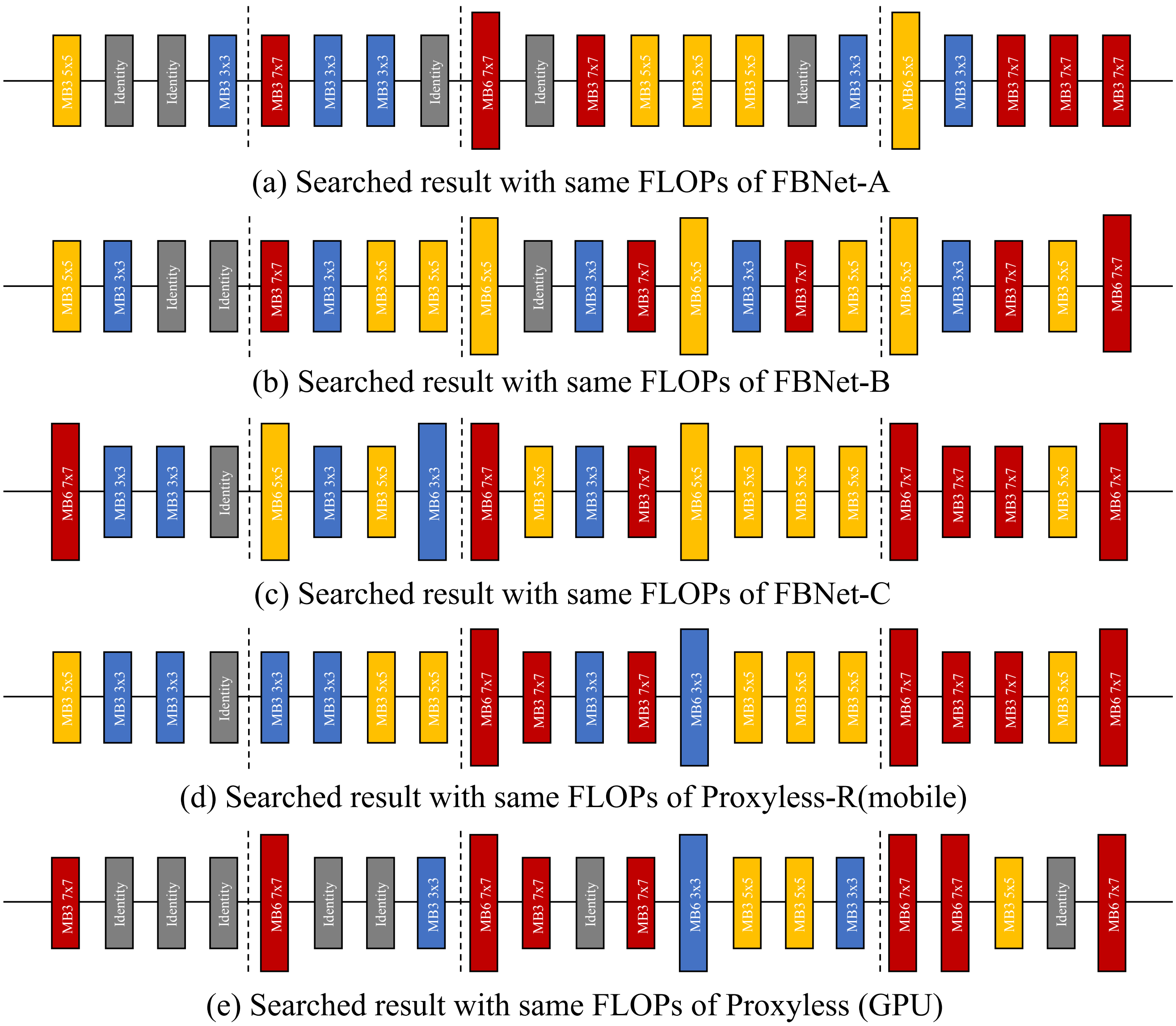
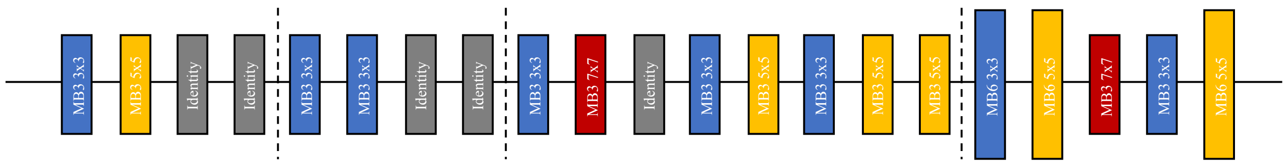
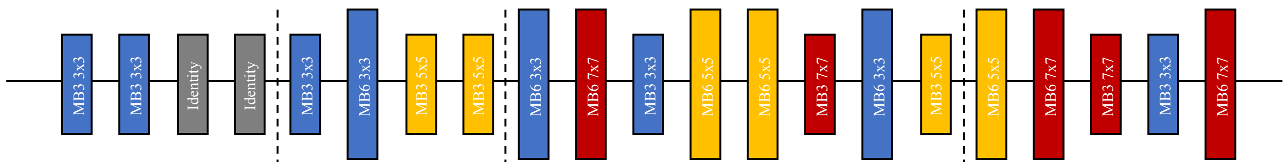


Figure 3. Structures of searched architectures under FLOPs constraints by using *ProxylessNAS* search space, see Table 5 for details.

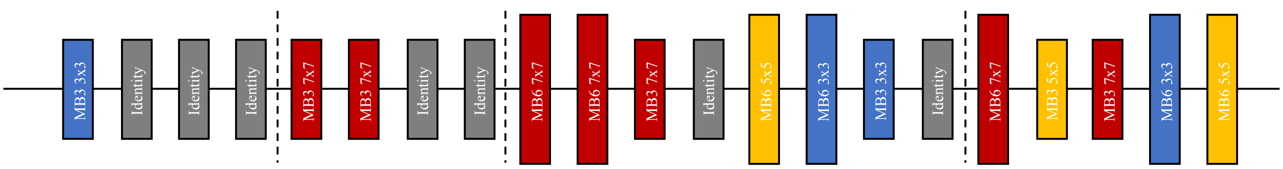


(a) Searched result with same latency of FBNet-A



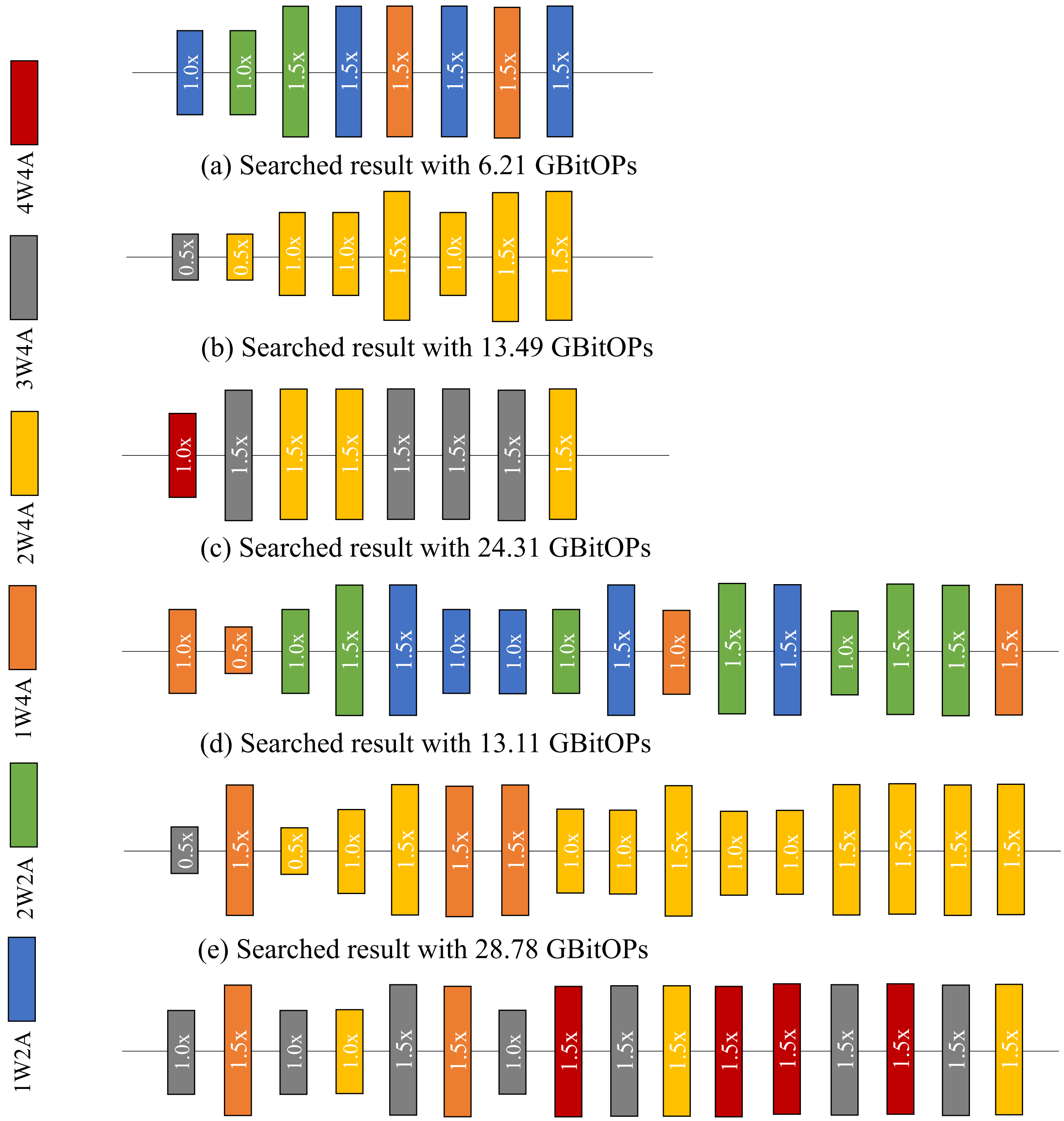
(b) Searched result with same latency of FBNet-B and Proxyless-R(mobile)

(c) Searched result with same latency of FBNet-C



(d) Searched result with same latency of Proxyless(GPU)

Figure 4. Structures of searched architectures under GPU latency constraints by using *ProxylessNAS* search space, see Table 5 for details.



(a) Searched result with 6.21 GBitOPs

(b) Searched result with 13.49 GBitOPs

(c) Searched result with 24.31 GBitOPs

(d) Searched result with 13.11 GBitOPs

(e) Searched result with 28.78 GBitOPs

(f) Searched result with 51.92 GBitOPs

Figure 5. Searched architectures of joint searching channel size and bit width under BitOPs constraints, see Table 6 for details. (a) - (c) are searched based on Resnet18. (d) - (f) are searched based on Resnet34.