Fast and Practical Neural Architecture Search

Jiequan Cui[1] * Pengguang Chen[1] * Ruiyu Li[2] Shu Liu[2] Xiaoyong Shen[2] Jiaya Jia[1,2]
[1]The Chinese University of Hong Kong [2]YouTu Lab, Tencent
{jqcui, pgchen, leojia}@cse.cuhk.edu.hk, {royryli, shawnshuliu, dylanshen}@tencent.com

Abstract

In this paper, we propose a fast and practical neural architecture search (FPNAS) framework for automatic network design. FPNAS aims to discover extremely efficient networks with less than 300M FLOPs. Different from previous NAS methods, our approach searches for the whole network architecture to guarantee block diversity instead of stacking a set of similar blocks repeatedly. We model the search process as a bi-level optimization problem and propose an approximation solution. On CIFAR-10, our approach is capable of design networks with comparable performance to state-of-the-arts while using orders of magnitude less computational resource with only 20 GPU hours. Experimental results on ImageNet and ADE20K datasets further demonstrate transferability of the searched networks.

1. Introduction

Convolutional Neural Networks (CNN) have achieved remarkable success in many computer vision tasks, including image classification [14, 8], object detection [23, 7], and semantic segmentation [31]. Manually designed networks, such as VGGNet [27], ResNet [8], and DenseNet [11], are very effective to yield top performance along with high computational complexity. Along the other research direction, to pursue decent trade-off between performance and inference speed, mobile architectures, like MobileNet [9, 26] and ShuffleNet [34, 19], were developed to satisfy computation requirements considering constrained computational resource on mobile and embedded devices. In general, design of new network architectures requires expertise and a load of hyper-parameter tunning.

Recent surge of interest in Neural Architecture Search (NAS) [39, 40] aims to construct neural networks automatically. In NAS, candidate networks are selected via reinforcement learning [39, 2, 40, 36, 20, 29] or evolutionary schemes [32, 22, 21, 16, 5]. They are then trained and evaluated on a validation set. The validation performance helps update the network generation process.

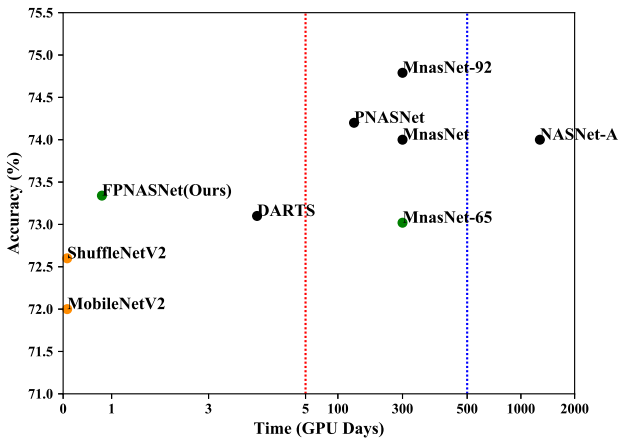

Figure 1. Search time and Top-1 accuracy of different NAS frameworks on ImageNet validation set. Green dots represent networks with less than 300M FLOPs while black ones are those with 300M+ FlOPs. Orange dots are top handcrafted mobile-friendly networks. Our proposed FPNAS is able to find useful networks (FPNASNet) within one GPU day. We note that ENAS [12] is not included in this figure since the performance is not reported on ImageNet.

Although searched architectures yield competitive performance on image recognition and language modeling tasks, most of the search process is computationally expensive – it may take many GPU days, as plotted in Fig. 1. Another bottleneck is that the discovered high-accuracy architectures may be with high FLOPs in Fig. 1, making them not easy to be used on mobile devices. To search for resource-constrained models, MNAS [29] utilized extra mobile devices and considered the inference latency as one of the optimization targets. It may not be widely applicable to general applications since the searched networks are device-specific. It also takes thousands of GPU hours in search process.

In this paper, we propose a Fast and Practical Neural Architecture Search (FPNAS) framework that is capable of

*equal contribution

discovering competitive network architectures using significantly less computational resource. FPNAS focuses on searching for extremely efficient networks with computation complexity less than 300M FLOPs. With constrained resource on mobile devices, we introduce the mobile search space, which consists of computationally efficient building blocks inspired by previously successful practice of designing handcrafted CNNs [26, 19] and NAS [12, 29]. Instead of searching for a general convolutional "cell" and repeatedly stacking it to form the CNN network, FPNAS builds the entire network to ensure block diversity. To accelerate the search process, we model search as a bi-level optimization problem and solve it via iterative approximation. It is achieved by alternatively optimizing blocks while keeping other blocks fixed.

The whole search process only takes 20 GPU hours and the searched network generalizes well on a variety of computer vision tasks. We also empirically show that FPNAS is able to output network architectures with comparable state-of-the-art performance on small network scales and yet with orders of magnitude faster search speed. On CIFAR-10, our FPNAS network (FPNASNet) achieves test error of 3.99%, outperforming MobileNet V2 [26] and ShuffleNet V2 [19] by 0.14% and 1.84% respectively. Directly applying the searched network to ImageNet, our model also works better than MobileNet V2 and ShuffleNet V2 by 1.34% and 0.74%, yielding a Top-1 accuracy of 73.34%. Compared with other NAS methods, our model performs similarly in terms of accuracy under the same computational complexity. The main benefit is that our search process only takes 20 GPU hours, which is *significantly faster* than PNAS[15], NASNet[40], and MNAS[29].

To demonstrate the transferability of the searched network, we utilize our model as the feature extractor in PSPNet [35] framework for the semantic segmentation task. Experimental results on ADE20K [37] show that our model achieves superior performance compared to MobileNet V2 and ShuffleNet V2 based frameworks.

Key contributions of our work are as follows.

- Our Fast and Practical Neural Architecture Search (FPNAS) can construct decent small-scale networks using only 20 GPU hours.
- This strategy involves new mobile search space containing computationally efficient building blocks.
- The networks discovered by FPNAS are applicable to the more challenging semantic segmentation task.

2. Related Work

In this section, we review representative Neutral Architecture Search (NAS) methods, which can be categorized into reinforcement learning based and evolutionary algorithm based approaches.

Reinforcement Learning Schemes Reinforcement learning was first applied to neural architecture search by Zoph *et al*. [39] and Baker *et al*. [2], where controllers are trained to select the neural network architecture from a large space including all possible layer operations. The algorithm is very time-consuming because of the huge search space and the need to train many epochs to get the reward. To alleviate these two problems, Zoph *et al*. [40] and Zhong *et al*. [36] proposed searching for the cell or block structure and then stack them to get the final network, which greatly reduce the complexity of search space. To get the reward as early as possible during search, early stop strategy is applied [36] and the surrogate model was trained in [15]. To further reduce the training steps to get the reward, parameter sharing was developed by ENAS [20], where a controller is trained with policy gradient to select a subgraph by parameter sharing. Other methods [3, 12] used layer transformation or morphism to increase complexity of the network to achieve high performance.

It is found in previous work that the reinforcement learning strategy may be unstable and hard to train. Moreover, work only considering large networks with high accuracy may not be helpful on mobile-level deployment. Recently, MNAS [29] were proposed to search for networks for mobile devices. It needs to get the reward from mobile device cloud, which is difficult in general. Compared with previous methods, ours can search for extremely efficient networks without deploying reinforcement learning schemes.

Evolutionary Methods Evolutionary algorithms give another direction for NAS. In [32, 22], evolutionary algorithms first demonstrated its effectiveness on small datasets, such as CIFAR-10 and MNIST. After that, the method of [21] applied the evolutionary algorithm to ImageNet with search speed faster than reinforcement learning on the same hardware, especially at the early stages of search. Although evolutionary methods work on NAS, it faces the same problem as reinforcement learning – that is, a lot of resource is needed in evolution. To address this problem, Liu et al. [16] proposed a hierarchical genetic representation scheme and an expressive search space to accelerate the evolution progress. Besides, Elsken et al. [5] proposed a LEMONADE evolutionary algorithm along with a Lamarckian inheritance mechanism to much improve the evolution speed. We note it is not always easy to control the evolutionary process due to various hyper parameters, undetermined heredity operations and variation operations. Our search process overcomes these obstacles, and is more controllable.

3. Our Method

3.1. Mobile-efficient Search Space

A convolutional neural network can be defined as an ordered set $\{Block_1, Block_2, ..., Block_N\}$, where each

$Block$ is a directly acyclic graph (DAG). To ensure block diversity, our proposed FPNAS searches for the whole network instead of a single DAG as in [40, 15, 12].

In the following, we first present the basic elements of our search space and then explain the importance of block diversity.

3.1.1 Block Architecture

Following [39, 15], we define a $Block$ as a DAG $G = (V, E)$, where each vertex in V represents a combination operation (*e.g.*, element-wise addition) or split operation, and the edge in E represents an arithmetic operation, such as convolution or pooling. We note that the vertex or edge could be an "empty operation", standing for removing the vertex or edge in the graph. Since our goal is to discover efficient network architectures, we restrict each block to at most three vertices to satisfy the constrained resource on mobile or embedded devices. The candidates of operations of vertexes and edges are as follows.

Vertices:

- Element-wise addition
- Concat operation
- Split operation
- Identity mapping

Edges:

- Convolution: mobile inverted bottleneck convolution with different expansion ratios
- Convolutional kernel size: 3×3 or 5×5
- Identity mapping

We restrict the concat operation as only appearing if it follows a split operation in order to match the channels of input and output. For each block, there is at most one convolutional operation. Fig. 2 illustrates the structure of the block. As for edges, we adopt mobile inverted bottleneck convolution operation in our search space as it has shown good trade-off between speed and accuracy [26]. To enlarge the size of the search space, we search for the expansion ratio and kernel size of depth-wise convolution.

We design our search space based on the following consideration. We keep mobile-inverted bottleneck convolution only, and disregard other common convolutional operations because the search results of [29] show that the search process only favors this type of convolution. The element-wise addition can form a residual connection, which is found useful in previous work [8, 26, 19]. We add the split operation because it is excellent at feature reuse.

Based on the finding of DenseNet[11] and DPN[4], feature reuse is a rather effective technique to improve performance. ShuffleNetV2 [19] demonstrated that the split operation is an efficient operation for mobile level networks. The number of vertices is set to 3 because too many branches cause fragmentation and affect efficiency [19].

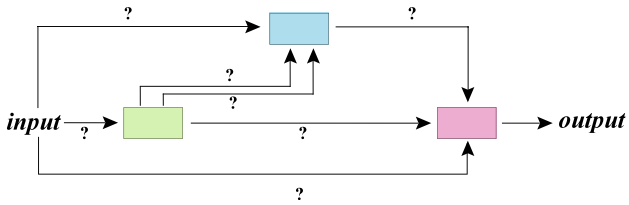

Figure 2. Overview of every block's architecture. Every box represents a vertex and there are at most three vertices, each arrowed line represents an edge. There are two edges from the green vertex to the blue one because the green vertex may be a split operation. Initially, operations on edges and vertices are unknown. We put questions marks for indication.

3.1.2 Block Diversity

Most existing work only searches for one general block and stacks it for predefined times to form a CNN. This makes all blocks have the same topological structure, which means $Block_1 = Block_2 = ... = Block_N$. Such a strategy simplifies the search process and greatly improves search efficiency. The limitation is on the need of repetitive blocks to make them work well. Mobile networks are constrained to limited resource, stacking the same blocks cannot guarantee a good trade-off between speed and accuracy. Recent works [29, 30, 24] also showed that it may be more efficient to have different blocks at various positions in a network. We denote the number of different blocks as *block diversity*.

Generally speaking, the essence of convolutional neural networks is to extract and combine features of images. We consider the importance of block diversity from two aspects. First, there are different kinds of features in images. In [29], it shows that blocks of different topological structures are effective to extract different features. Second, blocks at different positions of CNNs are optimized to conquer their respective difficulties [33]. For instance, blocks in low levels of a network pay more attention to edges and corners while higher-level blocks focus more on semantic information. Therefore, it is intriguing to see if it is beneficial when a CNN has blocks in different topological structures in terms of achieving decent performance.

To answer the above question, we quickly design a system with the following experiments. We train different networks on CIFAR-10 dataset. These networks are very simple and only have 5 blocks followed by a fully-connected layer. The first and last blocks are the same, *i.e.*, 3×3 DW-Conv and 1×1 Conv respectively. The 2nd-4th blocks of networks are chosen with different topological structures listed in Section 3.1.1. These three blocks use a mobile inverted bottleneck convolution and may incorporate short-

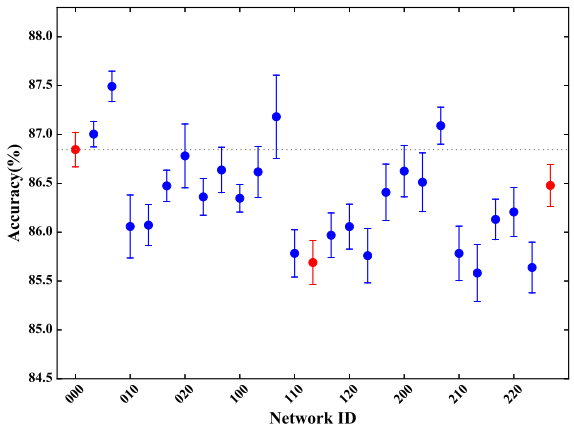

Figure 3. Block diversity verification. The x-axis is the three-digit network ID. Every point denotes the average accuracy and the vertical line is the standard variance for multiple times training. Blue points represent networks with different blocks at different positions. Red points are for networks with only one type of blocks.

cut, split or both along with the convolution. This results in a total of 27 different networks. We use a 3-digit ternary number as the network ID to represent them individually. We set the channels of networks adaptively to ensure their FLOPs are within 10±0.5M. We train these networks five times with the same setting and show the results in Fig. 3.

It is observable from the plot that all top-performing networks are composed of blocks of different topological structures. Moreover, it reveals that the lower-level blocks prefer not to split the channels while blocks at the highest level favor the opposite. This might be because the split operation is better for blocks to combine different features than extracting features.

The above experiments, albeit simple, still show the direction for us to further explore. We then design new efficient networks by making use of different topological structures. We allow each $Block_i$ to have its own structures, which needs to be searched in our method.

Assuming there are a total of K different architectures for one block, the whole search space is naturally K^N. It is too big to get a good architecture with limited computation. Here, we propose a fast neural network architecture search to address this problem.

3.2. Fast Search

In this section, we first define our task as solving an optimization problem. We analyze and define our goal as multiple bi-level optimization and propose a fast search method to efficiently address it. We also compare our fast search with previous reinforcement learning based schemes.

Problem Formulation A network is composed of an ordered set of blocks denoted as $Block_{1:N} = \{Block_1, Block_2, ..., Block_N\}$, where each $Block_i$ is the i-th block's structure. Our NAS is, therefore, formulated as the constrained optimization as

$$\begin{aligned} \max_{Block_{1:N}} \quad & R(Block_{1:N}) \\ s.t. \quad & f(Block_{1:N}) < b. \end{aligned} \tag{1}$$

where $R(Block_{1:N})$ denotes the reward of the network, which is defined as the accuracy on validation set. The constraint written as $f(Block_{1:N}) < b$ is added where $f(x)$ measures the computation complexity of network x in terms of FLOPs. The goal is to search for an efficient network with less than b FLOPs. Similar to that of [1], the hard constraints can be relaxed by redefining the reward function as

$$R(Block_{1:N}) = \begin{cases} Acc, & f(Block_{1:N}) < b \\ -1, & f(Block_{1:N}) \geq b \end{cases} \tag{2}$$

where Acc is the accuracy of the searched network on validation set. Even with the relaxation, the problem is still very challenging since not only the network with different blocks needs to be constructed, but also the structure of each block is to be optimized. Traditional methods or recent reinforcement learning schemes are difficult to yield good results. They may also require a large amount of computation. We instead propose fast search.

Search Algorithm We apply the bi-level optimization [28], which has two levels of optimization formally written as

$$\begin{aligned} \max_{x \in X, y \in Y} \quad & F(x, y) \\ s.t. \quad & x = \arg\max_{x \in X} G(x, y). \end{aligned} \tag{3}$$

This function can be efficiently optimized as discussed in [28].

The reason that we can approximate our problem by bi-level optimization is as follows. Suppose $Block_{1:N/2}$ have been determined, the optimal choice of $Block_{N/2+1:N}$ is related to $Block_{1:N/2}$. Contrarily, when $Block_{N/2+1:N}$ are fixed, optimal $Block_{1:N/2}$ is also related to them. Hence, our problem can be expressed approximately as

$$\begin{aligned} \max_{Block_{1:N/2}, Block_{N/2+1:N}} \quad & R(Block_{1:N}) \\ s.t. \quad & Block_{1:N/2} = \arg\max_{Block_{1:N/2}} R(Block_{1:N}) \end{aligned} \tag{4}$$

If we consider $Block_{1:N/2}$ as x, $Block_{N/2+1:N}$ as y, and $F(\cdot) = G(\cdot) = R(\cdot)$, this problem becomes exactly a bi-level optimization one.

Algorithm 1 Fast Search Algorithm

Initialize $Block_{1:N}$;
repeat
  for $Block_i \in Block_{1:N}$ **do**
    Fix $Block_{1:N} \backslash Block_i$ and optimize $Block_i$;
  end for
until converge
Output $Block_{1:N}$ as the final result.

To solve it, we use the similar idea to that of [17] by optimizing $Block_{1:N/2}$ and $Block_{N/2+1:N}$ alternatively. This makes the complexity of our problem reduces greatly. Besides, when optimizing $Block_{1:N/2}$, we can further depart it into $Block_{1:N/4}$ and $Block_{N/4+1:N/2}$ in a recursive way. The final solution becomes coordinate descent to optimize $Block_i$ alternatively. The detail of the algorithm is given in Algorithm 1.

$\mathbf{Block_i}$ Optimization For each $Block_i$ we observe that the number of possible operation combinations is still large. Exhaustive search like training every possible architecture of $Block_i$ and finding the best one is feasible, and yet could still be time-consuming.

It is noteworthy that when the architecture of $Block_i$ changes, other parts of the network remain the same. It is thus possible to share weights of the unchanged part instead of training from scratch for every architecture. This sharing-weight strategy reduces the search time significantly.

To alleviate possible adverse effect in weight sharing, we develop a two-stage method. In the first stage, we train all architectures with sharing weights and select top architectures. Then in the second stage we train these architectures independently to pick the best. The first stage limits the number of networks into a controllable range. Then we search for the reasonable architectures among these a few candidates.

3.3. Analysis

Complexity of Our Algorithm For the original combinatorial optimization problem, the complexity reaches the size of search space, which is intractable. After breaking it into multiple bilevel optimization problems, the complexity is greatly reduced.

A reasonable measure for comparison is to count the total number of examples used by the search process [15]. From this perspective, suppose we train $\mathcal{P}$ epochs in sharing-weight stage to select $\mathcal{E}$ best ones and train $\mathcal{Q}$ epochs in independent training stage for each of $\mathcal{E}$ networks with $\mathcal{T}$ iterations. We totally use $\mathcal{T} \times (\mathcal{P} + \mathcal{Q} \times \mathcal{E})$ epochs of training data.

In our experiments, we set $\mathcal{P} = 4$, $\mathcal{Q} = 4$, and $\mathcal{E} = 10$. It is found that after $\mathcal{T} = 16$ iterations, FPNAS can converge very well. So, we totally use 31M images, which is significantly smaller than 1 billion images used by [15], 21 billion images used by [40] and 25 billion images in [21].

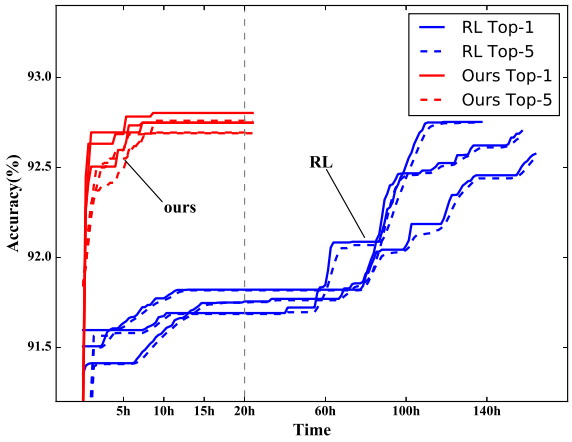

Figure 4. We train our fast search and reinforcement learning for several times, and compare their top-1 and top-5 test accuracy after training 40 epochs on CIFAR-10. It is clear that our method archives comparable accuracy taking much less time.

Comparison with Reinforcement Learning Reinforcement learning (RL) is applied in several previous methods. To compare with it, we designed a reinforcement learning framework as follows. We use a two-layer RNN as the controller. For every different architecture, we encode each property into a number for convenience of controller generation. It makes the controller produce a list of numbers at a time. Then we decode these numbers to an architecture and train it on CIFAR-10 for several epochs. The relative accuracy of moving average on the validation set is taken as the reward.

RL is useful in finding high-performance network architectures in our experiments. It is also well known for its slow process as demonstrated in Fig. 4 . Our method uses much less time because of the special architecture of the proposed search space and the efficient alternative optimization along with sharing-weight strategy. Reinforcement learning does not give the freedom to control output, making weight sharing difficult to apply in the same way.

4. Experiments

During the course of neural architecture search, we conduct experiments on CIFAR-10 [13] given its small scale. After we obtain the target neural network architectures with the best performance on CIFAR-10 [13], we apply them to ImageNet [25] classification and ADE20K [38] semantic segmentation tasks. Experimental results demonstrate the generalization ability of the networks constructed by our

Model	Params	Error (%)	Inference Speed (Images/s)
ShuffleNet V2 (1.5×) [19]*	2.47M	5.83	8,000
MobileNet V2 (1×) [26]*	2.20M	4.13	16,000
FPNASNet (Ours)	1.68M	3.99	13,500
Hier-EA [16]	15.70M	3.75	-
PNASNet-5 [15]	3.20M	3.41	-
AmoebaNet-A [21]	3.20M	3.34	-
DARTS (first order) [17] + CutOut	3.30M	3.00	-
ENAS [20] + CutOut	4.60M	2.89	900
DARTS (second order) + CutOut	3.30M	2.76	-
NASNET-A [40] + CutOut	3.30M	2.65	-
FPNASNet (2×) + CutOut (Ours)	5.76M	3.01	12,800

Table 1. Results of our FPNASNet on CIFAR-10. They are compared to MobileNet V2, ShuffleNet V2 and other NAS methods. For better performance, the first three layers with stride 2 in MobileNet V2 are changed to 1 and the first two layers with stride 2 in ShuffleNet V2 are set to 1. * denotes our implementation. 'FPNASNet (2×)' means we double all channels. The inference time is tested with PyTorch for MobileNet V2, ShuffleNet V2 and FPNASNet. The inference speed is tested with Tensorflow for ENAS.

method. For all the experiments, We implement our method using PyTorch.

4.1. Image Classification on CIFAR-10

Training Details CIFAR-10 comprises of 50,000 training images and 10,000 test images. We use standard data preprocessing and augmentation as those in [15]. Each image is upsampled to 40×40 and a 32×32 patch is randomly cropped from it or its horizontal flip. These patches are subtracted with channel mean and divided by channel standard deviation.

During the search process, we separate the original 50,000 training images into 45,000 for training and 5,000 for validation. We set $N = 16$ for our network and use SGD optimizer with momentum 0.9 and weight decay 0.0005. In the weight-sharing stage, the learning rate is fixed at 0.05. In the independent training stage, we use the cosine annealing schedule [18] with $T_0 = 10$ and only one run to decrease the learning rate until it reaches $1e$-5. We set the batch size to 128.

For full CIFAR-10 training, we use SGD optimizer with momentum 0.9. The initial learning rate is set to 0.02 and a weight decay of 0.0005 is applied. We use cosine annealing schedule [18] with $T_0 = 10$ and $T_{mult} = 2$ to decrease the learning rate until it reaches $1e$-4. Batch size is set to 192.

Comparison with State-of-the-art Classification results on CIFAR-10 are listed in Table 1. Compared with the handcrafted networks, *i.e.*, MobileNet V2 and ShuffleNet V2, our searched model outperforms them by a large margin with a much smaller number of network parameters. For NASNET, Hier-EA, AmoebaNet and PNASNet, they all use thousands of GPU hours to get the final result, while we only use 20 GPU hours to achieve a comparable result.

We also compare other neural architecture search methods for the sake of fairness. ENAS also consumes much GPU resource and takes long time. Our FPNASNet $(2\times)$ net achieves an error rate of 3.01, comparable with ENAS [20], which is also based on parameter sharing. It is noteworthy that the networks searched by ENAS are with much more complex topological structures with several network fragmentations and element-wise operations. According to the analysis of [19], network fragmentation reduces the degree of parallelism and element-wise operations introduce heavy MAC, decreasing network efficiency.

We test the inference speed of networks obtained by our FPNASNet $(2\times)$ and ENAS with image size 32×32 and batch size 100 on one NVIDIA P40 GPU. Our FPNASNet $(2\times)$ handles 12,800 images per second, which is 10 times faster than ENAS. ENAS deals with 900 images per second. Besides, complex topological structures may not be that friendly to hardware deployment. In contrast, our model is easier to implement with limited components, thus becomes more applicable to mobile devices.

Analysis of Search Efficiency Our method only takes 20 GPU hours to achieve decent performance, far more efficient than other methods. We summarize here that NAS and NASNet take thousands of GPU hours to construct a reasonable result. PNAS trains networks for 25,600 epochs in total on CIFAR-10. Although much time is saved compared with NAS, still thousands of GPU hours are needed. MNAS conducts network search directly on the large-scale ImageNet with gradient-based reinforce learning. At least 40,000 epochs in training are conducted during the entire search process – it takes much longer time than ours.

4.2. Image Classification on ImageNet

Training Details We follow the common practice for training networks on ImageNet [8]. We first pre-process

Model	Type	GPU Hours	DataSet for NAS	Parameters	Mult-Adds	Top-1 (%)
SqueezeNext [6]	manual	-	-	3.20M	708M	67.50
MobileNet V1 [9]	manual	-	-	4.20M	575M	70.60
CondenseNet (G=C=8) [10]	manual	-	-	4.80M	529M	73.80
MobileNet V2 (1×) [26]	manual	-	-	3.47M	300M	72.00
ShuffleNet V2 (1.5×) [19]	manual	-	-	3.50M	299M	72.60
ShuffleNet (1.5×) [34]	manual	-	-	3.40M	292M	71.50
CondenseNet (G=C=4) [10]	manual	-	-	2.90M	274M	71.00
MobileNet V2 (0.75×) [26]	manual	-	-	2.61M	209M	69.80
MobileNet V1 (0.5×) [9]	manual	-	-	1.30M	149M	63.70
ShuffleNet V2 (1×) [19]	manual	-	-	2.30M	146M	69.40
NASNet-A [40]	auto	32400	CIFAR-10	5.30M	564M	74.00
MnasNet-92 [29]	auto	7000*	ImageNet	4.40M	388M	74.79
MnasNet [29]	auto	7000*	ImageNet	4.20M	317M	74.00
MnasNet-65 [29]	auto	7000*	ImageNet	3.60M	270M	73.02
PNASNet [15]	auto	3600	CIFAR-10	5.10M	588M	74.20
DARTS [17]	auto	96	CIFAR-10	4.90M	595M	73.10
FPNASNet-C (Ours)	auto	**20**	CIFAR-10	1.91M	149M	69.91
FPNASNet-B (Ours)	auto	**20**	CIFAR-10	3.07M	216M	70.67
FPNASNet-A (Ours)	auto	**20**	CIFAR-10	2.95M	245M	72.01
FPNASNet (Ours)	auto	**20**	CIFAR-10	3.41M	300M	73.34

Table 2. Results of image classification on ImageNet. We compare our FPNASNet models with both handcrafted mobile models and other automated approaches. FPNASNet, FPNASNet-A, FPNASNet-B and FPNASNet-C are the searched models (for comparison) with different FLOPs and parameters. #Parameters: number of trainable parameters; #Mult-Adds: number of multiplication-add operations per image; Top-1 Acc: top-1 accuracy on ImageNet validation set. * is our estimation according to the number of models reported in [29] (with about 8K different models).

each image and obtain a random crop of the image with size 224×224. This is accomplished using the *RandomResizedCrop* function in PyTorch with the default setting. Then the cropped image patch is randomly horizontal flipped, followed by subtraction of channel mean and division by channel standard deviation. We use slightly less aggressive scale augmentation for a small model, where similar modifications are also utilized in [9]. In the test process, we resized the input image to 256×256 and the central crop of size 224×224 is used as network input.

We use SGD optimizer with momentum 0.9. The initial learning rate is set to 0.45, which decreases to $1e$-5 over the training process according to cosine annealing schedule [18] with $T_0 = 10$ and $T_{mult} = 2$. We use 4 GPUs for training with batch size 1,024. To compare the performance of our FPNASNet to other networks, we adjust channels of each layer to limit the model size and computational cost when necessary. Part of the structure of our network (FPNASNet) is shown in figure 5. The full structure is longer and is presented in the supplementary files.

Comparison with State-of-the-art Table 2 summarizes results of variants of our model and other handcrafted or automatically searched mobile networks on ImageNet validation set. We first observe that compared with MobileNet V2 and ShuffleNet V2, our model (FPNASNet) achieves the best performance with 300M FLOPs, outperforming them by 1.34% and 0.74% respectively. This manifests that our search algorithm is capable of discovering generic and effective neural network architecture. Compared with CondenseNet (G=C=4), our FPNASNet-A outperforms it by 1.01% with less FLOPs and comparable number of parameters. Besides, in the extreme setting with FLOPs less than 150M, our model (FPNASNet-C) improves ShuffleNet V2 (1×) by 0.51% and significantly outperforms MobileNet V1 (0.5×).

Second, most networks discovered by other NAS methods [15, 40] require a lot of search time or computation resource. They may focus on discovering large-scale networks. For models with around 300M FLOPs, MNAS performs slightly better than ours. We note MNAS directly searches networks on ImageNet and requires at least 350 times more GPU hours than ours. The major advantage of our work is the more efficient search process since it relies on CIFAR-10. The contemporary work DARTS [17] uses the same dataset for NAS and takes comparable GPU hours. Our searched models perform significantly better.

4.3. Semantic Segmentation

To evaluate the generalization ability of our FPNASNet, we also apply it to the semantic segmentation task with

Model	MultiScale Testing	Mean IoU (%)	Pixel Accuracy (%)
ShuffleNet V2*	No	35.64	77.51
MobileNet V2*	No	35.75	77.77
FPNASNet (Ours)	No	**36.76**	**77.91**
ShuffleNet V2*	Yes	35.95	77.99
MobileNet V2*	Yes	36.28	78.26
FPNASNet (Ours)	Yes	**37.40**	**78.42**

Table 3. Results of semantic segmentation on ADE20K. We compare our FPNASNet with MobileNet V2 and ShuffleNet V2. Result of MobileNet V2 is extracted from [38] with the same experimental setting. The result of Shufflenet V2 is from our own implementation.

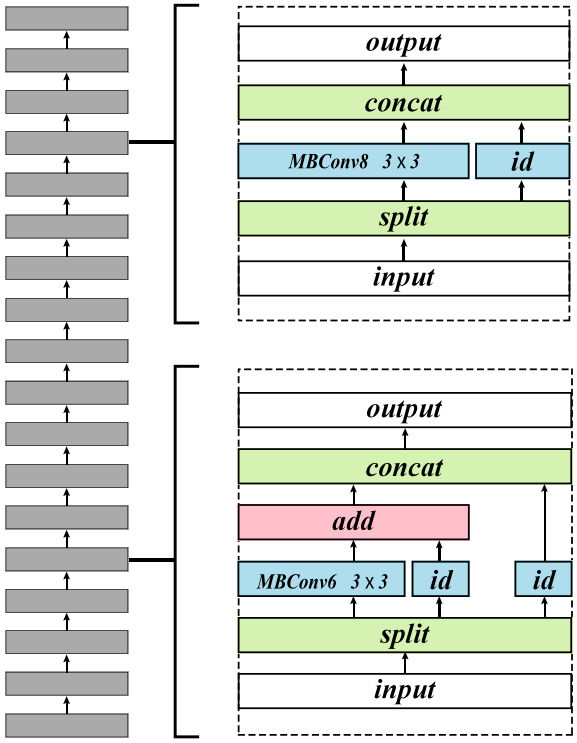

Figure 5. More details of our searched network structure. Blocks at different positions are with their special topological architectures. $MBConv6$ and $MBConv8$ are mobile inverted bottleneck convolution with expansion ratios 6 and 8 respectively.

ADE20K dataset. We use PSPNet with deep supervision trick in the open-source code from [38]. We use the default setting except for replacing the encoder network by ours.

Our results are listed in Table 3. Our FPNASNet reaches 36.76% Mean IoU and 77.91% Pixel Accuracy without multiscale testing, which is significantly higher than both MobileNet V2 and ShuffleNet V2. With multiscale testing, our FPNASNet further increases Mean Iou to 37.40% and Pixel Accuracy to 78.42%. The performance on ADE20K shows that our FPNASNet possesses a good generalization ability on other challenging computer vision tasks.

4.4. Ablation Study

We conduct experiments on CIFAR10 to exam the different choices of $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{E}$. As shown in Table 4, with the increase of $\mathcal{Q}$ and $\mathcal{E}$, performance of searched networks increases. However, when $\mathcal{Q} > 4$ and $\mathcal{E} > 10$, performance does not change significantly. Moreover we need to select a proper $\mathcal{P}$ to balance performance and complexity. Our choice of $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{E}$ is made upon these experiments.

Parameters			Top-1(%)
$\mathcal{P}$	$\mathcal{Q}$	$\mathcal{E}$	
2	4	10	95.59
8	4	10	95.48
4	2	10	95.80
4	8	10	95.93
4	4	5	95.38
4	4	20	96.01
4	**4**	**10**	**96.01**

Table 4. FPNAS results with different parameter setting.

5. Conclusion

In this paper, we have proposed a Fast and Practical Neural Architecture Search (FPNAS) framework for mobile-level network architecture design. We first formulate NAS as a mathematical optimization problem and break down the original combinatorial optimization into multiple bilevel optimization tasks, which greatly reduces complexity of the problem. We also introduce a new search space, which targets at light-weight and efficient network search. Our FPNAS only takes 20 GPU hours, considered as extremely fast compared with other NAS methods. Finally, FPNASNet found by FPNAS shows its great generalization ability on ImageNet and ADE20K datasets for classification and semantic segmentation tasks respectively.

There are many possible directions for future work. First, channel numbers are important factors that can be considered in the search process. Second, considering dynamically changing the depth of networks in the search process is possible. Finally, applying FPNAS directly to semantic segmentation or object detection tasks is also an intriguing and promising direction to explore.

References

[1] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*, 2017. 4

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016. 1, 2

[3] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In *ICML*, 2018. 2

[4] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. *CoRR*, abs/1707.01629, 2017. 3

[5] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Multi-objective architecture search for cnns. *arXiv preprint arXiv:1804.09081*, 2018. 1, 2

[6] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. Squeezenext: Hardware-aware neural network design. In *CVPR*, 2018. 7

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 3, 6

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 1, 7

[10] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. *CoRR*, abs/1711.09224, 2017. 7

[11] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1, 3

[12] Haifeng Jin, Qingquan Song, and Xia Hu. Efficient neural architecture search with network morphism. *CoRR*, abs/1806.10282, 2018. 1, 2, 3

[13] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 5

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[15] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 2, 3, 5, 6, 7

[16] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *CoRR*, abs/1711.00436, 2017. 1, 2, 6

[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. 5, 6, 7

[18] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *ICLR*, 2017. 6, 7

[19] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018. 1, 2, 3, 6, 7

[20] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018. 1, 2, 6

[21] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. 1, 2, 5, 6

[22] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017. 1, 2

[23] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1

[24] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017. 3

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. 5

[26] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 1, 2, 3, 6, 7

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1

[28] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Trans. Evolutionary Computation*, 22(2):276–295, 2018. 4

[29] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. 1, 2, 3, 7

[30] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *ECCV*, 2018. 3

[31] Xundong Wu. Fully convolutional networks for semantic segmentation. *Computer Science*, 2015. 1

[32] Lingxi Xie and Alan L. Yuille. Genetic CNN. In *ICCV*, 2017. 1, 2

[33] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 3

[34] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018. 1, 7

[35] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017. 2

[36] Zhao Zhong, Junjie Yan, and Cheng-Lin Liu. Practical network blocks design with q-learning. *CoRR*, abs/1708.05552, 2017. 1, 2

[37] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017. 2

[38] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3), 2019. 5, 8

[39] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. 1, 2, 3

[40] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. 1, 2, 3, 5, 6, 7