PONAS: Progressive One-shot Neural Architecture Search for Very Efficient Deployment

Sian-Yao Huang
National Cheng Kung University
Tainan, Taiwan
P76084245@gs.ncku.edu.tw

Wei-Ta Chu
National Cheng Kung University
Tainan, Taiwan
wtchu@gs.ncku.edu.tw

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533470

***Abstract*—We propose a Progressive One-Shot Neural Architecture Search (PONAS) method to achieve a very efficient model searching for various hardware constraints. Given a constraint, most neural architecture search (NAS) methods either sample a set of sub-networks according to a pre-trained accuracy predictor, or adopt the evolutionary algorithm to evolve specialized networks from the supernet. Both approaches are time consuming. Here our key idea for very efficient deployment is, when searching the architecture space, constructing a table that stores the validation accuracy of all candidate blocks at all layers. For a stricter hardware constraint, the architecture of a specialized network can be efficiently determined based on this table by picking the best candidate blocks that yield the least accuracy loss. To accomplish this idea, we propose the PONAS method to combine advantages of progressive NAS and one-shot methods. A two-stage training scheme, including the meta training stage and the fine-tuning stage, is proposed to make the search process efficient and stable. During search, we evaluate candidate blocks in different layers and construct an accuracy table that is to be used in architecture searching. Comprehensive experiments verify that PONAS is extremely flexible, and is able to find architecture of a specialized network in around 10 seconds. In ImageNet classification, 76.29% top-1 accuracy can be obtained, which is comparable with the state of the arts.**

I. Introduction

Deep neural networks have brought surprising advances in various research fields, such as image classification, image segmentation and object detection. However, designing good neural networks specific to a collection of constraints requires much domain knowledge, rich experience on model training and tuning, and a lot of time on trials and errors. Neural architecture search (NAS) is thus important and urgently-demanded to automate model design. Generally NAS methods can be categorized according to three dimensions [1]: search space, search strategy, and performance estimation strategy. There have been a dozen of studies proposed to search for neural architectures, especially for the task of image recognition. However, they are mostly time-consuming because of the intractable search space or expensive search strategy. Moreover, most of them are not scalable to various hardware constraints so that specialized networks should be determined and trained for each case.

Architecture searching is computationally expensive. For example, typical NAS methods based on reinforcement learning (RL) requires tens of GPU days [2][3]. The recent RL-based NAS methods, e.g., MnasNet [4], were estimated to require about $40K$ GPU hours for one specific network [5].

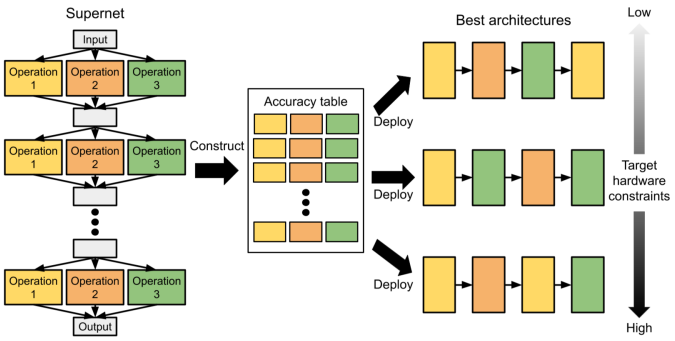


Fig. 1. An overview of the proposed PONAS method.

To reduce search efforts, dfferentiable NAS (DNAS) methods and single-path NAS methods emerge recently. They both can be viewed as one-shot approaches. One-shot NAS methods reduce the search effort by encoding the entire search space into an over-parameterized neural network, called *supernet*. Once the supernet is trained, all subnetworks can be sampled by inheriting the supernet's weights without further training. To make the search space continuous so that gradient-based methods can be adopted, DNAS relaxes categorical choices of a practical operation to a softmax over all possible operations. The *architecture parameters* specify how operations are combined and form the *architecture distribution*. In addition to *architecture optimization*, *weight optimization* is needed to determine the parameters of operations. DNAS jointly optimizes weights of the supernet and architecture parameters by gradient methods. Once the training is done, optimal architectures can be sampled from the trained distribution. However, the sampled network is only suitable to a specific hardware constraint, which makes DNAS less scalable. The FBNet [6] is a DNAS framework that largely speeds up searching for a specific network in 216 GPU hours. But if we need N different networks specific to N different constraints, $216 \times N$ GPU hours are still needed.

In contrast to DNAS, single-path NAS methods [7][8][9] decouple weight optimization from architecture optimization. For finding weights of the supernet, only one block in each layer is activated and optimized in each iteration. After training, the

978-0-7381-3366-9/21/$31.00 ©2021 IEEE

supernet is treated as a performance estimator. With the trained performance estimator, we can incorporate different search strategies, e.g., random search [10] or the evolution algorithm [8][7], to search for the best neural architecture for various hardware constraints without retraining the supernet. However, the time needed to execute the evolutionary algorithm [8][7] or random search is still considerable.

We conclude two problems in previous one-shot NAS approaches:

- High cost of supernet training: Training a supernet needs a lot of computation resource and search time.
- High cost of sub-network specialization: No matter DNAS or single-path NAS, considerable time is needed to do network specialization.

In this work, we propose *Progressive One-shot Neural Architecture Search* (PONAS) to combine the advantages of progressive NAS and the one-shot method. Progressive NASs like [11] and [12] construct convolutional neural networks (CNNs) by stacking predefined numbers of cells. The best structure of the cell is determined by progressively expanding blocks (operations). The determined best cell then acts as a layer. The structure of each layer, therefore, keeps the same. In the proposed PONAS, instead of searching for the best cell, we search for the best block for each layer progressively. In this way, structures of different layers may be different, and richer expressivity may be obtained. To tackle the first problem mentioned above, we propose a two-stage training scheme that separates the searching process into the meta training stage and the fine-tuning stage to make the search process more efficient and stable.

In the progressive search process, we construct an *accuracy table* that stores validation accuracy of each candidate block in each layer. Given a hardware constraint, based on the accuracy table a specialized network can be efficiently evolved by the evolutionary algorithm because only simple table lookup is needed to estimate performance of any specialized network, as illustrated in Fig. 1. In our experiment, architecture of a specialized network can be determined in around 10 seconds. This approach largely resolves the second problem mentioned above.

Notice that, in contrast to FBNet [6] where sub-networks are sampled from a supernet, we directly derive a network specific to the hardware constraint according to the information obtained during the search process of PONAS. This strategy enables us to get multiple specific networks very efficiently based on the accuracy table that only needs to be constructed once.

II. Related Works

Because the formulation of NAS is similar to reinfocement learning (RL), early NAS methods were firstly proposed based on it. However, such approaches are very computationally expensive. A variety of methods were thus proposed based on progressive learning or one-shot learning to reduce computational cost. In the following, we only focus on related works on these two approaches.

A. Progressive Neural Architecture Search

Progressive NAS (PNAS) methods [12][11] search the architecture space in a progressive way. A sequential model-based optimization (SMBO) strategy [13] is adopted to search for architectures in the order of increasing complexity, while a surrogate model is learnt simultaneously to guide the search. Starting from the first cell, all possible block structures are trained and evaluated. Each of them is then expanded by adding all possible blocks, which largely enlarges the search space. To reduce search time, a performance predictor is trained to evaluate all these extensions, and then only the top blocks are retained. According to [11], this approach is about 8 times faster than the RL-based method [14] in terms of total computation.

We are also inspired by the idea of progressive learning. But different from the SMBO strategy, we construct an accuracy table rather than building a surrogate model as a performance predictor.

B. One-Shot Neural Architecture Search

Many recent NAS approaches are conceptually based on the one-shot scheme [5] [15] [6] [16] [17]. The main idea is constructing a supernet to represent the entire search space. Such methods can be generally divided into two categories: differentiable NAS and single-path NAS.

Differentiable NAS [6][5][15] views the search space as an architecture distribution described by architecture parameters. Architecture parameters are optimized jointly with training a supernet. After training, the optimal architectures can be sampled from the trained distribution. Instead of searching over a discrete set of candidate architectures, Liu et al. [15] relaxed the search space from discrete to continuous, and thus the gradient descent algorithm can be adopted to find the optimal architecture. Cai et al. [5] binarized the architecture parameters and forced only one path to be active when training the supernet, which reduces the required GPU memory. Wu et al.[6] used the Gumbel softmax technique [18] to find the optimal distribution of architecture parameters.

Single-path NAS [9][7][8][16] considers the trained supernet as an evaluator to predict performance of all sub-networks. After training a supernet, we can adopt random sampling, evolutionary algorithms, or reinforcement learning to derive multiple specific networks conforming to different constraints without retraining the supernet. Using the trained supernet as the performance evaluator, Guo et al. [7] used an evolutionary algorithm to find the optimal architecture for the given constraint. Cai et al. [8] sampled a set of sub-networks to train an accuracy predictor, which guides architecture search to get a specialized network. Chu et al. [16] pointed out the problem of biased evaluation, which is prone to misjudgments of candidate architectures. They thus proposed to fairly train candidate blocks to get a supernet as a reliable performance evaluator.

III. Progressive One-Shot Neural Architecture Search

Inspired by progressive search, the proposed PONAS also searches the architecture progressively, but PONAS directly searches for the best network layer by layer. More importantly, the accuracy table constructed in PONAS process facilitates efficient network specialization without a surrogate model for accuracy prediction.

A. Overview

1) Previous Network Specialization: Denote the weights of a supernet A as W_A. Each sampled architecture a inherits weights from W_A. Given a constraint C, previous one-shot NAS methods [7][16] achieved network specialization by finding the best sub-network a^* that yields the highest accuracy and fits the constraint C. That is,

$$\begin{aligned} a^* = \underset{a \in A}{\operatorname{argmax}}\, ACC_{val}(W_A, a), \\ \text{s.t.} \quad Cost(a^*) \leq C, \end{aligned} \tag{1}$$

where $ACC_{val}(W_A, a)$ is the validation accuracy yielded by the architecture a. The network searching process mentioned in Eqn. (1) is time consuming because the validation accuracy of sub-networks a's should be calculated case by case.

2) The Proposed Process: Instead of calculating validation accuracy in each searching process, we propose to build an accuracy table when training the supernet. With this table, very efficient network searching can be achieved because only simple table lookup is needed.

Denote the weights of the l-th layer as $W^{(l)}$, and the i-th candidate block in the l-th layer as $B_i^{(l)}$. A candidate block is a set of settings of operations, which may include different kernel sizes, expansion settings, and so on. When training the supernet, we would like to find the architecture as a stack of blocks that yields the highest accuracy. For the l-th layer, the best block is determined by

$$i^* = \underset{i=1,2,...,I}{\operatorname{argmax}}\, ACC_{val}(W_{\hat{A}^{(l)}}, a_{B_i^{(l)}}), \tag{2}$$

where $\hat{A}^{(l)}$ is a supernet containing all candidate blocks in the l-th layer, while containing the default block in all other layers (details will be given in Sec. III-B). The term $a_{B_i^{(l)}}$ is a sub-network taking the block $B_i^{(l)}$ in the l-th layer, which is sampled from $\hat{A}^{(l)}$. The term I is the total number of candidate blocks in a layer. By finding the best blocks $B_{i^*}^{(1)}$, $B_{i^*}^{(2)}$, ..., $B_{i^*}^{(L)}$ in layers 1 to L progressively, we can stack them to construct the best architecture of the supernet, denoted as $[B_{i^*}^{(1)} \rightarrow B_{i^*}^{(2)} \rightarrow \cdots \rightarrow B_{i^*}^{(L)}]$.

When finding the best architecture of the supernet, we simultaneously construct the accuracy table T that stores the validation accuracy of all candidate blocks at all layers:

$$T[l, i] = ACC_{val}(W_{\hat{A}^{(l)}}, a_{B_i^{(l)}}), \quad l = 1, ..., L; i = 1, ..., I. \tag{3}$$

Given a constraint C, we can get the specialized network a^* that maximizes the accuracy and fits the constraint C by checking the accuracy values stored in T.

TABLE I

Macro-architecture of the search space. MBConv E1 3×3 denotes MBConv with kernel size 3 and expansion 1. "Repeat" denotes the number of layers repeat with the corresponding settings.

Input shape	Block	Output channel	Repeat	Stride
$224^2 \times 3$	Conv 3×3	32	1	2
$112^2 \times 32$	MBConv E1 3×3	16	1	1
$112^2 \times 16$	Candidate Block	32	1	2
$56^2 \times 32$	Candidate Block	32	1	1
$56^2 \times 32$	Candidate Block	40	1	2
$28^2 \times 40$	Candidate Block	40	3	1
$28^2 \times 40$	Candidate Block	80	1	2
$14^2 \times 80$	Candidate Block	96	4	1
$14^2 \times 96$	Candidate Block	96	3	1
$14^2 \times 96$	Candidate Block	192	1	2
$7^2 \times 192$	Candidate Block	320	4	1
$7^2 \times 320$	Candidate Block	1280	1	1
$7^2 \times 1280$	Avg pool 7×7	-	1	1
1280	Fully Connected	1000	1	-

$$\begin{aligned} \mathcal{I}^* = \underset{i \in \mathcal{I}}{\operatorname{argmax}}\, T(l, i), \\ \text{s.t.} \quad Cost(a_{\mathcal{I}^*}) \leq C, \end{aligned} \tag{4}$$

where $\mathcal{I}$ is the set of indices of candidate blocks in all layers. The meaning of Eqn. (4) is that we want to find the best sequence of blocks $\mathcal{I}^* = (i^{(1)}, i^{(2)}, ..., i^{(L)})$ among all combinations such that the architecture $a_{\mathcal{I}^*}$ yields the highest accuracy and fits the constraint C simultaneously. The term $i^{(j)}$ denotes the index of the candidate block $B_{i^{(j)}}^{(j)}$ at the j-th layer.

The main difference between Eqn. (4) and Eqn. (1) is that we do not need to calculate or estimate validation accuracy on the fly, but just need to look at the accuracy table. We adopt the genetic algorithm to find the best sequence among all combinations. Because only table lookup is needed, finding the best sub-network architecture usually can be done in 10 seconds.

In this work, the setting of the architecture search space is inspired by [5]. The backbone of candidate block is mobile inverted bottleneck convolution (MBConv) [19] with kernel sizes {3,5,7} and expansion {3,6} in depthwise convolution. The squeeze-and-excite [20] module is considered to be employed or not to expand the search space. Therefore, we have $I = 3 \times 2 \times 2 = 12$ types of candidate blocks for each layer. The number of layers to be constructed is set to $L = 19$. Accordingly, our search space has a size of 12^{19} in total. Particularly, the macro-architecture of the search space is illustrated in Table I.

B. Two-stage Training

One-shot approaches like [6][7][16][15] and our approach mentioned in Sec. III-A need to construct a supernet, and then sub-networks are sampled or derived from it to conform to constraints. However, training a supernet needs a lot of computation resource and search time. In [6], for example, constructing the supernet needs to consider all combinations of candidate blocks in the search space, as illustrated in the

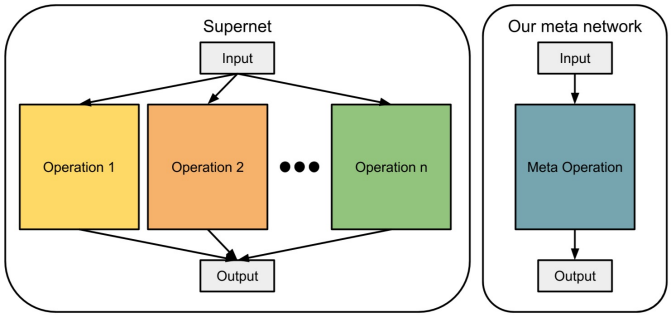


Fig. 2. The main difference between a supernet and the proposed meta network is that the former consists of multiple paths of blocks, and the later is single-path.

left of Fig. 2. This makes structure of the supernet quite complex and thus much resource is required. Some recent works [21][22] use super kernels to encode all candidate blocks into one block, and search the best distribution of architecture parameters to get the best sub-network. But these approaches give rise to the coupling problem mentioned in [7].

To reduce the cost of training a supernet, we propose a two-stage training scheme. This training scheme can be integrated with other one-shot NASs.

This scheme consists of the *meta training stage* and the *fine-tuning stage*. In the meta training stage, we construct a meta network which is the *largest network* in the search space. In each layer of the so-called largest network, only the candidate block with the largest convolution kernel, expansion, and enabled squeeze-and-excite module is used, i.e., kernel size = 7, expansion = 6, with squeeze-and-excite module enabled. Let B_G denote the largest block in the following. Fig. 2 illustrates the difference between a supernet and the proposed meta network. The meta network is a single path consisting of the concatenation of B_G's, and a supernet is a network consisting of multiple paths of various blocks. Training the meta network is thus easier.

Training the meta network acts as finding good initialization parameters based on the largest block. To further elaborate the network, we propose to progressively fine-tune the meta network by finding the *best block* for each layer. Instead of constructing the entire supernet as the left of Fig. 2, we only replace B_G of one layer by 12 candidate blocks each time to construct the supernet $\hat{A}^{(l)}$. For example, to fine-tune the first layer, the default block B_G is replaced by candidate blocks $(B_1, B_2, ..., B_{12})$, and the 2nd layer to the 19th layer keep using the default block B_G (with the parameters discovered in the meta learning stage) to construct $\hat{A}^{(1)}$. The supernet $\hat{A}^{(1)}$ is fine-tuned based on the training data with strict fairness [16]. Let $[B_i \rightarrow B_G \rightarrow \cdots \rightarrow B_G]$ denote the specific network from $\hat{A}^{(1)}$ where the block in the first layer is B_i, followed by B_G's. The validation accuracy of each specific network is evaluated and stored in the *accuracy table* $T[1, 1:12]$, which will play an important role in network specialization described later.

Assume that when B_G replaced by the candidate block

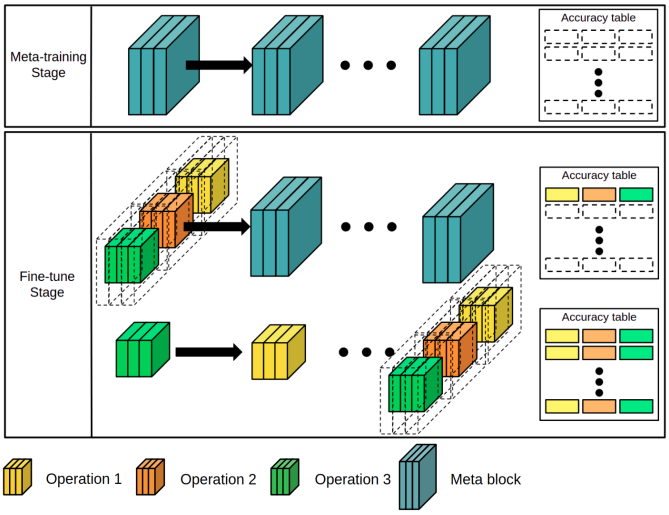


Fig. 3. Our two-stage training scheme. In the meta training stage, we only train meta network which is constructed by the largest candidate block. In the fine-tuning stage, we progressively fine-tune the meta network to construct the accuracy table.

$B_*^{(1)} \in \{B_1, B_2, ..., B_{12}\}$, the specific network yields the highest validation accuracy. This network is denoted as $[B_*^{(1)} \rightarrow B_G^{(2)} \rightarrow \cdots \rightarrow B_G^{(19)}]$. When we try to fine-tune the second layer, parameters of the supernet $\hat{A}^{(2)}$, $[B_*^{(1)} \rightarrow B_1^{(2)} \rightarrow \cdots \rightarrow B_G^{(19)}]$, $[B_*^{(1)} \rightarrow B_2^{(2)} \rightarrow \cdots \rightarrow B_G^{(19)}]$, ..., $[B_*^{(1)} \rightarrow B_{12}^{(2)} \rightarrow \cdots \rightarrow B_G^{(19)}]$, are updated based on the training data. In the same way, the validation accuracy of each specific network from $\hat{A}^{(2)}$ is evaluated and stored in the accuracy table $T[2, 1:12]$.

We progressively find the best candidate block for each layer, and store all validation accuracy values in the accuracy table T. The progressive fine-tuning process is illustrated in Fig. 3. After fine-tuning all layers, we finally get the network $[B_*^{(1)} \rightarrow B_*^{(2)} \rightarrow \cdots \rightarrow B_*^{(19)}]$ and the accuracy table T.

The parameters of smaller candidate blocks $(B_1, B_2, ..., B_{12})$ are initialized by the parameter of the largest block B_G. Fig. 3 illustrates how we crop parameters of the largest block B_G to get the parameters of all smaller blocks. With such initialization, we just need to fine-tune for a few epochs rather than lots of epochs from random initialization. This idea is similar to progressive shrinking mentioned in [8].

C. Network Specialization in the Accuracy Loss Domain

Given a constraint, previous one-shot methods derived a specific network based on the evolutionary algorithm [7][8]. These methods flexibly support different constraints and only need to train the supernet once. However, the evolutionary method proposed in Guo et al.[7] required much time in measuring validation accuracy of the population, and the method proposed in Cai et al.[8] needed to train an accuracy predictor to measure validation accuracy of 16K sub-networks. In our work, we employ the accuracy table T as the performance evaluator in the network specialization process.

A problem from Eqn. (4) arises when comparing with the validation accuracy of candidate blocks in different layers. Candidate blocks in different layers are not directly comparable because different levels of information is learnt. Therefore, we argue that they should be compared in a domain commonly for different layers. Chu et al. [16] pointed out that different choice blocks of the same layer learn similar feature maps on the corresponding channel. Inspired by this observation, we propose to represent performance of each candidate block as *the accuracy loss from the best block at the corresponding layer*. For example, let $t_1^{(l)}, t_2^{(l)}, ..., t_{12}^{(l)}$ be the validation accuracy (evaluated in the training process and stored in the accuracy table T) of the candidate blocks $B_1^{(l)}, B_2^{(l)}, ..., B_{12}^{(l)}$ at the l-th layer, and let $t_*^{(l)}$ denote the best accuracy yielded by $B_*^{(l)}$. The accuracy loss is calculated as $\Delta \boldsymbol{t}^{(l)} = (t_1^{(l)} - t_*^{(l)}, ..., t_{12}^{(l)} - t_*^{(l)}) = (\Delta t_1^{(l)}, \Delta t_2^{(l)}, ..., \Delta t_{12}^{(l)})$. In this way, we calculate $\Delta \boldsymbol{t}^{(1)}, ..., \Delta \boldsymbol{t}^{(19)}$, and transform the performance of candidate blocks into the *accuracy loss domain*.

We then can construct the accuracy loss table $T_d[l,:] = [\Delta \boldsymbol{t}^{(1)}; \Delta \boldsymbol{t}^{(2)}; ...; \Delta \boldsymbol{t}^{(L)}]$. Based on T_d, we assume that *the accuracy loss of candidate blocks can quantify importance of candidate blocks in different layers*. With the accuracy loss table T_d, we can re-formulate Eqn. (4) as

$$\begin{aligned} \mathcal{I}^* &= \underset{i \in \mathcal{I}}{\operatorname{argmin}} T_d(l, i), \\ \text{s.t.} \quad & Cost(a_{\mathcal{I}^*}) \leq C. \end{aligned} \tag{5}$$

Fig. 4 shows the largest accuracy losses $\max_i(\boldsymbol{t}_i^{(j)})$ at different layers. As can be seen, different layers learn different information and yield varied accuracy losses. For the layer with smaller accuracy loss, performances of different candidate blocks in this layer are similar. Therefore, when we do network specialization, we prefer to replace B_G by other smaller blocks so that less resource is needed but the validation accuracy just decreases slightly.

The optimization problem mentioned in Eqn. (5) is solved by the genetic algorithm. We represent an architecture sampled from the supernet as a chromosome of length 19. A chromosome $(g^{(1)}, g^{(2)}, ..., g^{(19)})$ denotes the block indices of the sub-network constituted by $(B_{g^{(1)}}^{(1)}), B_{g^{(2)}}^{(2)}), ..., B_{g^{(19)}}^{(19)})$. We randomly initialize 20 chromosomes in the first population. The cost of each chromosome is calculated based on the accuracy loss table T_d, i.e., $\sum_{l=1}^{19} T_d(l, g^{(l)})$. The top 10 chromosomes that yields the least accuracy loss are selected and grouped into 5 pairs, which are viewed as parent chromosomes. For each pair, a position from 1 to 19 (length of a chromosome) is randomly selected for the crossover operation. Five pairs of children are generated after crossover. For each of this child chromosome, the index of each position may randomly mutate to another index with probability 0.1. After crossover and mutation, these 10 chromosomes and the 10 parent chromosomes form the second-generation population. Notice that, after each crossover and mutation, the resultant chromosome is checked if it fits the constraint C. If not, another crossover or mutation operation is conducted to make new chromosomes. In this work, the same evolution process iterates for 1,000 generations, and the best chromosome in the whole process is picked to represent the architecture of the desired specialized network.

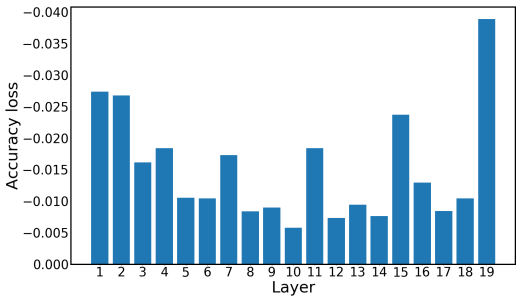


Fig. 4. The maximum accuracy losses in different layers. We argue that they can be the indicators to decide the block to be replaced first.

IV. Experiments

A. Experimental Settings

We perform all experiments based on the ImageNet dataset [24]. Same as the settings in [16][7][5][4], we randomly sample 50,000 images (50 images for each class) from the training set as our validation set, and the rest is kept as our training set. The original validation set is taken as our test set to measure the final performance of each model.

We train the meta-network for 50 epochs using batch size 256 and adopt the stochastic gradient descent optimizer with a momentum of 0.9 and weight decay of 4×10^{-5}, based on data with standard augmentation (random resizing, cropping, and flipping). We set the initial learning rate to be 0.1, and decay 10x after the 20th and the 40th epoch, respectively. For the fine-tuning stage, we follow the same strategy as that for meta training. But we only fine tune each layer for 3 epochs and set the initial learning rate to be 0.001 without decay setting.

After determining the architecture of a specific network that conforms to the given constraint, we adopt the RMSProp optimizer with 0.9 momentum [4] to train the searched architecture from scratch. Learning rate is increased from 0 to 0.16 in the first 5 epochs with batch size 256, and then decays 0.03 every 3 epochs. We use 4 NVIDIA GTX 1080Ti GPUs for training.

B. ImageNet Classification

Three specific models, named as PONAS-A, PONAS-B, and PONAS-C, are specialized from the supernet according to the accuracy loss table to meet different requirements. To ensure a fair comparison, the constraints for getting these three models are set according to the required FLOPS similar to FairNAS-A, -B, and -C, i.e., 330M, 350M, and 380M FLOPs. The result is shown in Table II. As can be seen, the proposed PONAS requires similar search time to the most recent one-shot approaches [16][5][6]. However, thanks to the design of the accuracy loss table, the time for searching can be almost ignored. Overall, the top-1 accuracies are 75.68%, 76.06%,

TABLE II
IMAGENET CLASSIFICATION PERFORMANCE COMPARISON WITH THE SOTA MODELS

Model	Search method	Search space	Search dataset	Train GPU hours	Search GPU hours	Params(M)	FLOPs(M)	Top-1 acc(%)
MobileNetV2 [19]	manual	-	-	-	-	3.4	300	72.00
MobileNetV2(1.4X) [19]	manual	-	-	-	-	6.9	585	74.70
ShuffleNetV2(1.5X) [23]	manual	-	-	-	-	3.5	299	72.60
PNASNet [11]	SMBO	Cell	CIFAR-10	-	-	5.1	588	74.20
DPP-Net-Panacea [12]	SMBO	Cell	CIFAR-10	-	-	4.8	523	74.02
DARTS [15]	gradient	Cell	CIFAR-10	96	$96N$	4.7	574	73.30
FBNet-C [6]	gradient	layer	ImageNet	216	$216N$	5.5	375	74.90
SinglePath OneShot [7]	evolution	layer	ImageNet	288	$24N$	-	328	74.70
ProxylessNAS-R [5]	RL	layer	ImageNet	200	$200N$	4.1	320	74.60
FairNas-A [16]	RL	layer	ImageNet	240	48	4.4	321	74.69
FairNas-B [16]	RL	layer	ImageNet	240	48	4.5	345	75.10
FairNas-C [16]	RL	layer	ImageNet	240	48	4.6	388	75.34
Random search	random	layer	ImageNet	210	$\sim$ **0**	4.5	324	75.36
PONAS-A	evolution	layer	ImageNet	210	$\sim$ **0**	5.1	326	75.68
PONAS-B	evolution	layer	ImageNet	210	$\sim$ **0**	5.1	349	76.06
PONAS-C	evolution	layer	ImageNet	210	$\sim$ **0**	5.6	376	76.29

and 76.29% for PONAS-A, -B, and -C, respectively, which are competitive with the state of the arts (FairNAS [16]).

Li's work in [10] presents that a random search approach usually achieves satisfactory performance. To make comparison, we randomly select 1,000 candidate architectures with FLOPs under 330M from the search space and pick the architecture with the lowest accuracy loss. The performance is named as Random search in Table II. As can be seen, PONAS achieves higher accuracy over random search by a large margin (around 0.35%).

Fig. 5 shows the determined architectures of PONAS-A, PONAS-B, and PONAS-C. Notice that the three models tend to choose high expansion rates and large kernel at more important layers (the layers with larger accuracy losses, see Fig. 4), which tends to improve performance. On the other hand, to reduce computational complexity, the three models tend to choose the block with small expansion rate at less important layers.

C. Ablation Studies

1) Efficiency of Two-stage Training: For fair comparison between the proposed two-stage training and FairNAS [16], which directly trained the entire supernet from random initialization with strict fairness, we train both methods for 50 epochs and set the initial learning rate as 0.1, and decay 10x at the 20th and the 40th epochs, respectively. For two-stage training, we train 30 epochs for the meta training stage and another 20 epochs for the fine-tuning stage. Instead of progressively fine tuning each layer mentioned in Sec. III-B, we fine tune the entire supernet for fair comparison with FairNAS. The reason to compare with FairNAS is that we both train the supernet with the strict fairness property [16].

Fig. 6(a) shows the relationship between top-1 validation accuracy and training epochs. Basically both methods get increasing accuracy as the number of training epoch increases.

TABLE III
IMPACT OF DIFFERENT LAYERS ON PERFORMANCE. THE NETWORK *worst+most importance* ACHIEVES THE HIGHEST ACCURACY WITH THE LEAST FLOPS.

Model	FLOPs	Top-1 accuracy(%)
worst	279	73.4
worst+least importance	305	73.8
worst+most importance	**295**	**73.9**

For the proposed two-stage training, the validation accuracy drops at the beginning of the fine-tuning stage but rises gradually after a few epochs. After 50 epochs, the proposed two-stage training reaches up to 69%, which is 1% higher than FairNAS. Fig. 6(b) shows comparison in terms of total GPU hours required for training a supernet. In the meta training stage, the two-stage training scheme is 2 times faster than the time FairNAS takes to train the supernet because of the simple architecture of the meta network.

2) Analysis of the Accuracy Loss Domain: We randomly sample 8 different architectures, train them from scratch, and then evaluate them. Fig. 7 shows the relationship between an architecture's accuracy loss (from the table T_d) and the real validation accuracy. We see that these two factors positively correlated. Like [25], we calculate the Kendall rank correlation coefficient [26] to measure the correlation, and get the Kendall's τ value as 0.70. This means the architecture with less predicted accuracy loss really yields higher validation accuracy, and verifies the assumption we made in Sec. III-C.

3) Importance of Different Layers: To verify the assumption that *the accuracy loss of candidate blocks can quantify importance of candidate blocks in different layers*, we specialize three models from the supernet. First, for each layer of the supernet, we purposely replace the block with one candidate block that yields the largest accuracy loss. This model is called

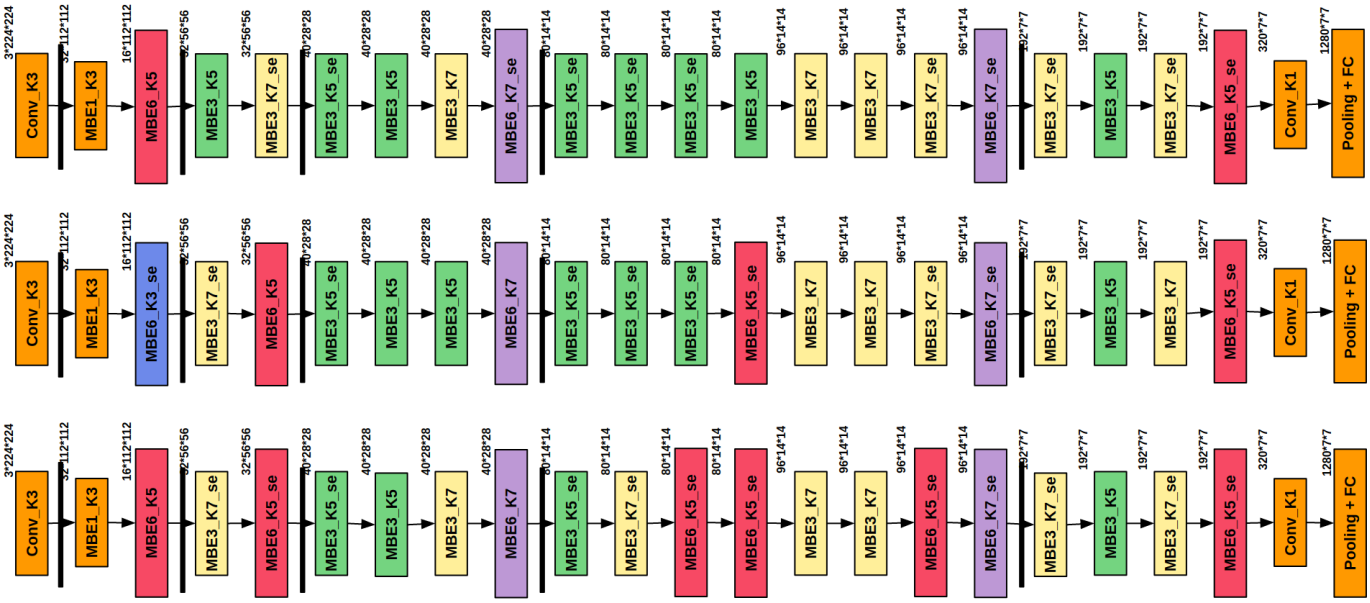
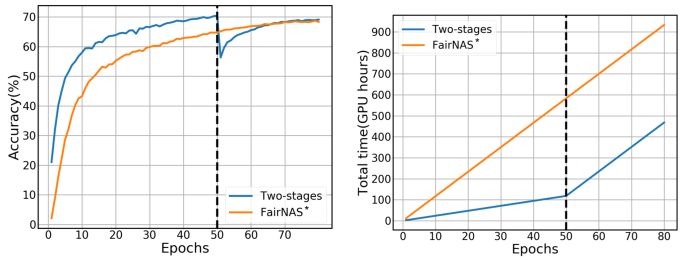


Fig. 5. Architectures of PONAS-A, -B, -C from top to down. MBE3 and MBE6 denote the mobile inverted bottleneck convolution layers with expansion ratios 3 and 6, respectively. KX denotes the kernel size of X. se denotes whether the squeeze-and-excite [20] module is used. The orange blocks are predefined blocks before searching.



(a) (b)

Fig. 6. Comparison between the two-stage training and FairNAS [16]. The black dash line denotes the beginning of the fine-tuning stage in two-stage training. (a) The evolution of validation accuracies of two methods. (b) The total time required to train the supernets in the manners based on two-stage training and random initialization used in FairNAS. We denote FairNAS with the symbol * because these results are based on our re-implementation.

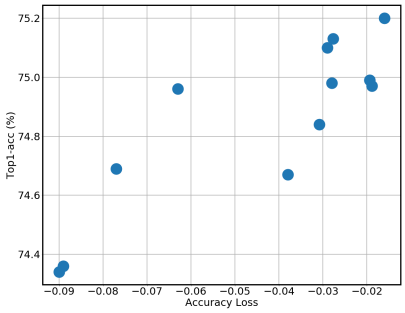


Fig. 7. The relationship between validation accuracies of 12 models and their associated predicted accuracy losses.

the *worst specific network*. Second, we purposely replace the block in the least important layer of the worst specific network with the best candidate block in that layer to construct the network called *worst+least importance*. Third, we also replace the block in the most important layer of the worst specific network with the best candidate block in that layer, and obtain a new specific network called *worst+most importance*. For fair comparison, we actually replace the two least important layers in *worst+least importance* to achieve similar FLOPs with *worst+most importance*.

We train the three models for 150 epochs under the same setting in Sec. IV-A and show the result in Table IV-C2. As expected, the worst specific network obtains the worst performance. When the worst network is changed to *worst+least importance*, top-1 accuracy boosts to 73.8%. When the worst

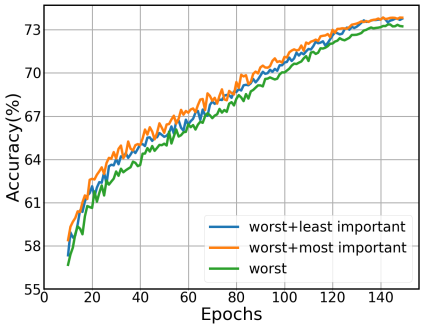


Fig. 8. The top-1 validation accuracies of three different networks.

network is changed to *worst+most importance*, the highest accuracy (73.9%) with less FLOPs can be obtained. The differences on accuracy and FLOPs show the impact of importance of different layers. Fig. 8 shows the evolution of top-1 validation accuracy of these networks. The *worst+most importance* model has higher convergence speed than other networks.

In summary, using better blocks at more important layers is more efficient than that at less important layers. From both Table IV-C2 and Fig. 4, we found that layers with different numbers of channels for input and output have larger accuracy loss and are more impactful to final performance.

V. Conclusion

We present a progressive one-shot neural architecture search method to search the best block in each layer progressively and construct the supernet. When constructing the supernet, we build a table called accuracy table to store the validation accuracy of each candidate block in each layer. By transforming the accuracy table into the accuracy loss domain, candidate blocks in different layers are comparable, and importance of different layers can be measured. Given a constraint, we can simply check the accuracy table to see performance of various specialized architectures and find a specific architecture in around 10 seconds. This is much more efficient than previous one-shot NAS. To speed up and stabilize supernet training, we propose a two-stage training approach, including the meta training stage and the fine-tuning stage. We demonstrate that two-stage training is more stable and converges faster than previous approaches that train the supernet from random initialization. In the evaluation, we show that the proposed PONAS can achieve the state-of-the-art performance with very low-cost architecture searching.

Acknowledgement This work was funded in part by Qualcomm through a Taiwan University Research Collaboration Project and in part by the Ministry of Science and Technology, Taiwan, under grants 108-2221-E-006-227-MY3, 107-2923-E-006-009-MY3, and 109-2218-E-002-015.

References

[1] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 2019.

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *Proceedings of International Conference on Learning Representations*, 2017.

[3] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of International Conference on Learning Representations*, 2017.

[4] 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 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[5] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *Proceedings of International Conference on Learning Representations*, 2019.

[6] 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 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[7] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *Proceedings of European Conference on Computer Vision*, 2020.

[8] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *Proceedings of International Conference on Learning Representations*, 2020.

[9] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *Proceedings of International Conference on Machine Learning*, 2018.

[10] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Proceedings of Machine Learning Research*, 2020.

[11] 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 European Conference on Computer Vision*, 2018.

[12] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *Proceedings of European Conference on Computer Vision*, 2018.

[13] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of International Conference on Learning and Intelligent Optimization*, 2011.

[14] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *Proceedings of International Conference on Learning Representation*, 2019.

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

[17] Shen Yan, Biyi Fang, Faen Zhang, Yu Zheng, Xiao Zeng, Hui Xu, and Mi Zhang. Hm-nas: Efficient neural archi-

tecture search via hierarchical masking. In *Proceedings of IEEE International Conference on Computer Vision Workshops*, 2019.

[18] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *Proceedings of International Conference on Learning Representations*, 2017.

[19] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[20] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[21] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path mobile automl: Efficient convnet design and nas hyperparameter optimization. In *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2019.

[22] Shoufa Chen, Yunpeng Chen, Shuicheng Yan, and Jiashi Feng. Efficient differentiable neural architecture search with meta kernels. *arXiv:1912.04749*, 2019.

[23] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of European Conference on Computer Vision*, 2018.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2009.

[25] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating search phase of neural architecture search. In *Proceedings of International Conference on Learning Representations*, 2020.

[26] M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 1938.