Block Proposal Neural Architecture Search

Jiaheng Liu, Shunfeng Zhou, Yichao Wu, Ken Chen, Wanli Ouyang, Dong Xu

***Abstract*—Typical neural architecture search methods usually restrict the search space to the pre-defined block types for a fixed macro-architecture. However, this strategy will affect the search space and architecture flexibility if block proposal search (BPS) is not considered for NAS. As a result, block structure search is the bottleneck in many previous NAS works. In this paper, we propose a new evolutionary algorithm referred as latency EvoNAS (LEvoNAS) for block structure search, and also introduce it to the NAS framework by developing a novel two-stage framework called Block Proposal NAS (BP-NAS). Comprehensive experimental results across multiple computer vision tasks demonstrate the superiority of our approaches over the state-of-the-art lightweight models. For the classification task on the ImageNet dataset, our BPN-A is better than 1.0-MobileNetV2 [1] with similar latency, and our BPN-B saves 23.7% latency when compared with 1.4-MobileNetV2 with higher top-1 accuracy. Furthermore, for the object detection task on the COCO dataset, the face identification task on the MegaFace dataset, and the re-identification task on Market-1501 dataset, our methods outperform MobileNetV2, which demonstrates the generalization capability of our newly proposed framework.**

***Index Terms*—neural architecture search, neural network design, image classification**

I. Introduction

Neural architecture search (NAS) was proposed to relieve the burden from the hand-crafted network architecture design process, and its effectiveness has been demonstrated in various computer vision tasks, such as image classification [2]–[28], object detection [29], [30] and image segmentation [31]. Despite encouraging performance has been achieved, the conventional NAS algorithms [5], [13] usually require us to train and evaluate thousands of network architectures, leading to huge computational costs. Furthermore, the requirement from many resource-constrained platforms also restrict their practical industry impact, as the latency information is barely considered except in a few recent works [2], [4], [7]. Among them, the differentiable NAS (DNAS) framework [2], [4] allows efficient search of small, fast and accurate model architecture, and it thus becomes a common approach for designing the platform-aware networks.

The DNAS approaches [2]–[4], [19], [20] consider all potential architectures as the sub-networks of an over-parameterized network. Thus the search architectures can be quickly evaluated by inheriting the learned weights from the supernet. The typical DNAS methods [2], [4] explore a layer-wise search space and obtain the actual connections between the layers, which consist of the pre-defined blocks. Since they neglect the process for searching proper candidate blocks, the search space is usually limited, which constraints architecture flexibility and generalization ability.

Jiaheng Liu is with the School of Computer Science and Engineering, Beihang University, Beijing, China (e-mail: liujiaheng@buaa.edu.cn)

Shunfeng Zhou, Yichao Wu and Ken Chen are with the Research Institute of SenseTime Group Limited, Beijing, China (e-mail:wuyichao@sensetime.com, zhoushunfeng@sensetime.com, kenchen1024@gmail.com)

Wanli Ouyang, Dong Xu are with the School of Electrical and Information Engineering, The University of Sydney, Sydney, NSW 2006, Australia (e-mail: wanli.ouyang@sydney.edu.au, dong.xu@sydney.edu.au)

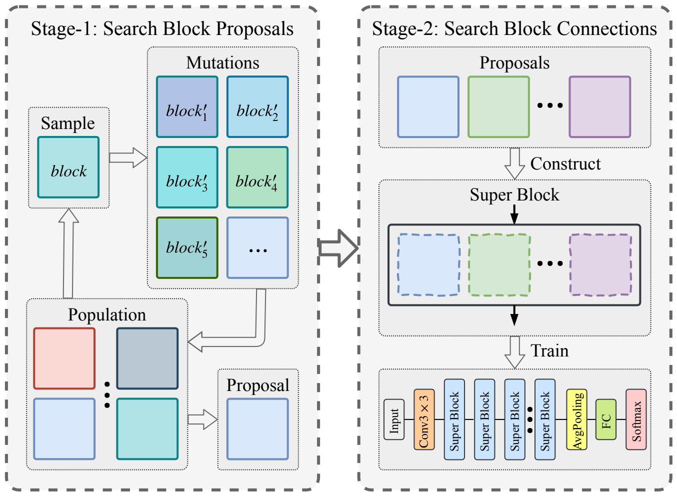


Fig. 1: Our BP-NAS framework. In Stage-1, we use an improved evolutionary algorithm to search the block proposals under various latency constraints. In Stage-2, we use these searched block proposals to construct a super block and supernet, and use DNAS on this network to search the block connections.

While it is often helpful to enlarge the layer-wise search space for DNAS, it will substantially increase the computational costs and GPU memory overhead. Specifically, if we directly search the block structure in each layer with larger search space, the over-parameterized network would require much more computation costs and GPU memory.

This work aims to develop an efficient optimization method to conduct both block structure search and connection search. Inspired from the coarse-to-fine strategy [32]–[39], we propose a two-stage framework named Block Proposal Neural Architecture Search (BP-NAS). As illustrated in Fig. 1, our framework can be decomposed into two subproblems: Block Proposal Search (BPS) and Block Connection Search (BCS). BPS could be conducted on a proxy dataset, which aims at finding the candidate blocks with various latency constraints. To this end, we propose a new evolutionary algorithm with latency constraints, named Latency EvoNAS (LEvoNAS). The LEvoNAS method can generate block proposals with different latency constraints, while preserving its simplicity and effectiveness. On the other hand, our BCS is implemented using DNAS for determining the connections between different layers.

Comprehensive experimental results across a broad range of tasks and datasets demonstrate the networks searched by our approach outperform the state-of-the-art lightweight methods. We name our searched models as BPN. For the classification task on the ImageNet dataset, our BPN-A is better than 1.0-MobileNetV2 [1] ($+2.0\%$) with similar latency, and our BPN-B saves 23.7% latency when compared with 1.4-MobileNetV2 with higher ($+0.5\%$) top-1 accuracy. Furthermore, for the object detection task on COCO dataset, our BPN-A model achieves significant improvement when compared with 1.0-MobileNetV2, and our BPN-B even surpasses ResNet-50 [40], which is a standard heavy backbone for object detection.

Our contributions are in two folds:

- To acquire the optimal block proposals with the expected computational complexity, we propose LEvoNAS by additionally introducing multiple latency constraints, which makes this method more useful in the practical applications.
- We propose a two-stage NAS framework called BP-NAS, which aims to further enlarge the search space while maintaining efficiency. In this framework, the proposed LEvoNAS and DNAS are seamlessly integrated into a unified framework.

Extensive experimental results across multiple computer vision tasks, including the classification task on ImageNet [41], the object detection task on COCO [42], the face identification task on MegaFace [43] and the re-identification task on Market-1501 [44], demonstrate the effectiveness of our framework.

II. Related Work

Efficient hand-crafted Convolutional Network. Efficiency is often crucial for many real-world applications [45]–[51]. Recently, many researchers have proposed new approaches to design fast and high-performance networks. SqueezeNet [52] uses the fire module to compress the model with equivalent accuracy. MobileNet [53] introduces pointwise convolution operations and depthwise convolution operations to replace spatial convolution operations. MobileNetV2 [1] further builds the inverted residual blocks and linear bottleneck, and achieves remarkable improvement in terms of both speed and accuracy. ShuffleNet [54] adopts the bottleneck structure of ResNet [40], and additionally introduces the group convolution operations and channel shuffle operations. Furthurmore, in ShuffleNetV2 [55], the authors pointed out that the FLOPs count cannot precisely reflect the lantency of the model and thus introduced a series of principles to design the networks manually. Moreover, based on DenseNet [56], CondenseNet [57] uses fewer parameters to achieve higher efficiency by employing group convolution operations and pruning operations during the training process. However, manual development of ConvNet by human experts is a time-consuming and error-prone process, which often requires extensive experience and expertise.

Neural Architecture Search. The works [5], [18] opened up the wave of automatic network architecture search by using reinforcement learning (RL). Although they can achieve competitive performance, the direct use of RL for network architecture design on large-scale datasets is often computationally expensive, which makes NAS less practical for various applications. To solve this problem, the efficient NAS methods, such as the progressive search methods [8], parameter sharing [11] and network transfer [58], were recently proposed. Since the platform constraints are not incorporated into the search process or the search space, these works are not appropriate for the platform-aware architecture design purpose. MnasNet [7] formulates the search process as a multi-objective optimization problem, which considers both accuracy and inference latency. However, it still suffers from the high search cost (9.1×10^4 GPU hours for a complete search as reported in [4]). Furthermore, MobileNetV3 [21] applies the automated search algorithm and the network design strategy together to obtain the lightweight network for better hardware implementation. EfficientNet [59] adopts neural architecture search based on reinforcement learning to design a new baseline network, which can scale it up to obtain a family of networks by balancing the network depth, width and resolution carefully.

Recently, several NAS approaches based on evolutionary algorithms [6], [12], [14]–[16], [60] were proposed. The recent work [6] modifies tournament selection of the evolutionary algorithm to select younger genotypes, which improves the efficiency of NAS and also enhances the performance of the CNN models. However, as far as we know, the existing EvoNAS methods do not consider the latency as an optimization objective, as it is challenging to generate fast and accurate models on the target device.

The differentiable neural architecture search (DNAS) method [2]–[4], [17], [19], [20] treats NAS as a single training process of a full-parameterized model, which contains all candidate operations and connections. Thus, the DNAS method is more efficient than the previous works. The work One-Shot [17], [61] generates new architectures by sampling the network operations and connections with some fixed probabilities. The work DARTS [3] introduces a simple continuous relaxation approach to jointly train the weight parameters and the architecture parameters by using the SGD algorithm. The work P-DARTS [20] allows the depth of the searched architectures to gradually grow in the training procedure, while the work PC-DARTS [19] proposed to sample a small part of super-network to reduce the redundancy when exploring the network space. The work ProxylessNAS [2] solves the GPU memory consumption issue through path binarization, thus it can search the network architectures on the large-scale datasets. The work FBNet [4] builds different candidate blocks according to some manually designed networks [1], [62], and considers a layer-wise search space. Both ProxylessNAS and FBNet employ the latency loss to make the CNN models hardware-friendly. However, the search space is usually restricted to the pre-defined block types for DNAS, which largely limits architecture flexibility. To address the aforementioned issues, in this work, we propose a new two-stage BP-NAS framework to conduct block structure search and connection search while maintaining high efficiency.

III. Methodology

In this section, we introduce a two-stage pipeline named BP-NAS to design a convolutional neural network automatically, shown in Fig. 1. In the first stage, which is named as Block Proposal Search (BPS), we use the proposed LEvoNAS to find several superior block proposals with different latency constraints. In the second stage, which is named as Block Connection Search (BCS), we use DNAS method to choose an appropriate block from block proposals in each layer. In this work, we consider the real-world latency constraint for NAS, and formulate the problem as:

$$\begin{aligned} &\min_{a\in\mathcal{A}} \min_{w_a} \mathcal{L}(a, w_a) \\ &\text{s.t. } \textsc{Latency}(a) \le T_{target}, \end{aligned} \tag{1}$$

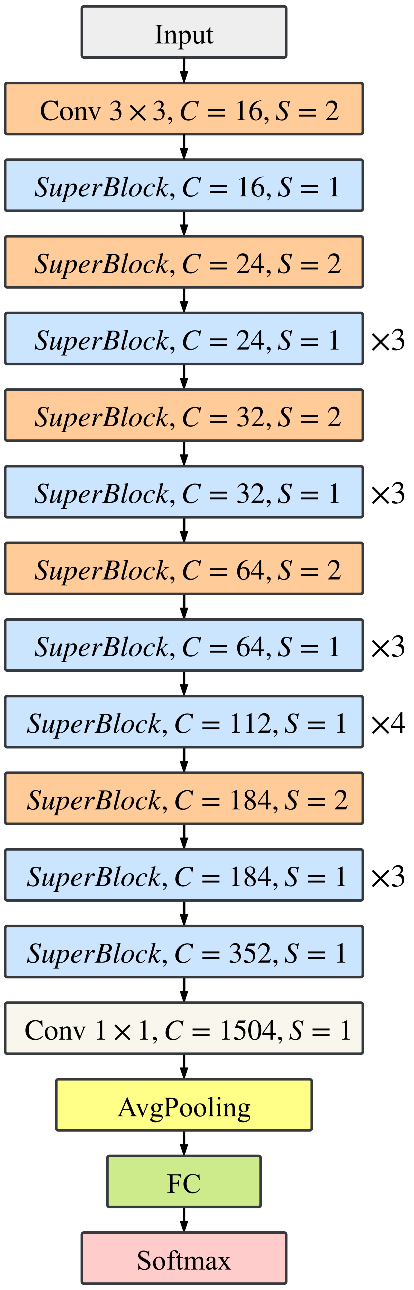
where $\mathcal{L}(a, w_a)$ represents the loss function, $\mathcal{A}$ denotes architecture space, T_{target} denotes the target latency constraint on specific device, a denotes a possible architecture and w_a denotes its weights.

A. The Search Space

Following existing DNAS approaches [2]–[4], we simplify the problem of network architecture search by choosing one block for each layer and stacking the blocks to obtain the whole network. Based on MobileNetV2 [1], we specify the size of the input feature map, and the number of filters in each layer to establish a backbone network, as shown in Fig. 2.

To enlarge the search space, we redesign the block framework, as shown in Fig. 3. The proposed block framework adopts multi-branch design, which is inspired by inverted residual block [1] and inception module [63]. The block contains at most three paths. One is called *simple* path, and consists of a normal convolution and a ReLU transformation. Motivated by ResNeXt [64], the other two paths, called *complex path*, have the same potential structure by increasing cardinality. By integrating simple and complex paths, the proposed block can achieve a better trade-off between accuracy and efficiency. Compared with the inverted residual block with a single branch, it has better representation ability; while compared with the potential cell structure proposed by DARTS [3] or BlockQNN [9], the proposed one can achieve hardware-friendly inference speed with a simpler design.

In order to encode the block structure, we design a coding strategy, The encoding format takes 23 bits in total, which is shown by an example in Fig. 4. The first three bits, denoted as u, represent whether the three paths in each block exist or not. When the value of one bit of u is 1, we set the corresponding branch to select the searched cell. Otherwise, we set the corresponding branch not to select the searched cell. Kernel size k_1 and group size g_1 control the variability of the simple path. For the complex paths, we assign k_j to control the kernel size of the lightweight depthwise convolution, and adopt expansion ratio e_j [1] and group size g_j to control the variability and computation complexity of different blocks, where $j \in \{2, 3\}$. In details, for the i-th path, where $i \in \{1, 2, 3\}$, the kernel size k_i has four options for assignment, which are 1, 3, 5 or 7. These four options can be encoded as two bits, {00, 01, 10, 11}. Similarly, the group size g_i also has four possible values, which is 1, 2, 4, 8, and can be encoded as two bits, too. The expansion ratio e_j is from 1 to 16, can be encoded as four bits. Therefore, a block structure can be represented as $3+(2+2)+(2+2+4)\times 2 = 23$ bits in total. It is obvious that the choices of block structures are extremely large.


Input
Conv 3×3, C = 16, S = 2
SuperBlock, C = 16, S = 1
SuperBlock, C = 24, S = 2
SuperBlock, C = 24, S = 1 ×3
SuperBlock, C = 32, S = 2
SuperBlock, C = 32, S = 1 ×3
SuperBlock, C = 64, S = 2
SuperBlock, C = 64, S = 1 ×3
SuperBlock, C = 112, S = 1 ×4
SuperBlock, C = 184, S = 2
SuperBlock, C = 184, S = 1 ×3
SuperBlock, C = 352, S = 1
Conv 1×1, C = 1504, S = 1
AvgPooling
FC
Softmax
Fig. 2: The backbone network for DNAS.

DNAS methods [2]–[4] usually need to construct an over-parameterized super block for each layer. In our settings, if we use all possible block structures as our candidate blocks to construct a supernet, the searching costs of both time and GPU memory cannot meet our need. For a backbone network containing 22 layers described in Fig. 2, the search space would be very large. Thus, we propose a two-stage architecture search method to solve this problem. For example,

1057-7149 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Auckland University of Technology. Downloaded on November 03,2020 at 23:30:45 UTC from IEEE Xplore. Restrictions apply.

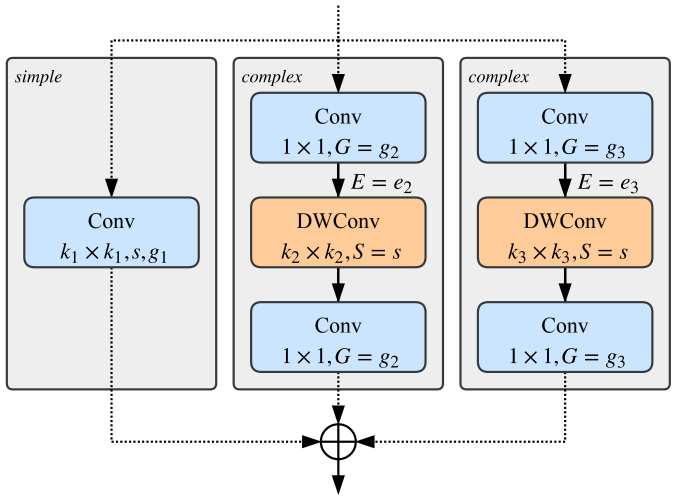


Fig. 3: The block architecture.

0 0 1	1 0	0 1	1 0	0 1	1 0 0 1	1 0	0 1	1 0 0 1
u	k_1	g_1	k_2	g_2	e_2	k_3	g_3	e_3

Fig. 4: Block encoding for the architecture in Fig. 3 and an example. u is used for representing the choice of the path. k_j for the kernel size of path $j \in \{1, 2, 3\}$, g_j for the group size, and e_j for the expansion ratio.

we search 5 block proposals from possible blocks in the first stage, which means that our layer-wise search space will contain 6[1] candidate blocks in total. In the second stage, the block connection search is conducted using the searched block proposals. Thus, our proposed two-stage BP-NAS framework would reduce the computation complexity greatly.

B. Block Proposal Search

The stage of Block Proposal Search (BPS) dominates the computational complexity in BP-NAS, thus an efficient method is expected for BPS. We apply the proposed LEvoNAS to BPS. In order to increase the diversity of blocks, we search the block proposals on multiple latency constraints, as shown in Algorithm 1.

Algorithm 1 SEARCHBLOCKPROPOSALS(*contraints*)

```
$proposals = \varnothing$
**for** $contraint \in constraints$ **do**
    $block \leftarrow$ SEARCHBLOCK($contraint$)
    add $block$ to $proposals$
**end for**
**return** $proposals$
```

The evolutionary algorithm for searching the block is summarized in Algorithm 2. It keeps a population of P measured blocks throughout the experiment. The population is initialized by blocks with random architectures. Our model evolves for C cycles. At every cycle, we will sample S blocks from the population. The CONSTRUCTMODEL function will stack the specific block repeatedly to construct a model based on the backbone network described in Fig. 2. Besides, to cooperate with a size of 32×32, we set the stride of the first downsampling convolution to 1. We use the performance of the model to approximate the performance of the block. The efficiency of the model is measured by the latency of the model. To speed up the evaluation, we use the CIFAR-10 [65] as a proxy dataset and train 30 epochs, and then take the final top-1 accuracy as a performance evaluation of the block.

[1]We add a "Skip" block to deliver the input feature map to the output without change, which can decrease the number of layers and increase the variousness of the network.

Algorithm 2 SEARCHBLOCK(*contraint*)

```
$population \leftarrow$ empty queue
$history \leftarrow \varnothing$
**while** $|population| < P$ **do**
    $block \leftarrow$ RANDOMBLOCK($constraint$)
    $model \leftarrow$ CONSTRUCTMODEL($block$)
    $block.accuracy \leftarrow$ TRAINANDEVAL($model$)
    add $block$ to right of $population$
    add $block$ to history
**end while**
**while** $|history| < C$ **do**
    $sample \leftarrow$ sample $S$ elements from $population$
    $parent \leftarrow$ highest-accuracy block in $sample$
    $child \leftarrow$ MUTATE($parent$, $constraint$)
    $model \leftarrow$ CONSTRUCTMODEL($child$)
    $child.accuracy \leftarrow$ TRAINANDEVAL($model$)
    add $child$ to right of $population$
    add $child$ to $history$
    remove the left of $population$
**end while**
**return** highest-accuracy $block$ in $history$
```

On the other hand, each block proposal needs to comply with the latency constraint during the search process. Since the block search space is too large, we cannot measure the latencies of all possible blocks. To solve this problem, we use the sum of latencies of all the paths in the block to estimate the latency of block. This approximation is valid for most devices. We enumerate all possible input sizes and channels, measure the latencies of each possible path separately, and store them in a lookup table. Thus, we can use the sum of latencies to check whether the block satisfies the given latency constraint. This method is used in the RANDOMBLOCK and MUTATE functions in Algorithm 2.

C. Block Connection Search

In this section, we discuss the details of choosing appropriate blocks from the block proposals searched by the BPS in the first stage. In DNAS, choosing appropriate blocks from block proposals in each layer is equivalent to finding the actual connections block connections. Therefore, we call the second stage Block Connection Search (BCS), as shown in Fig. 5.

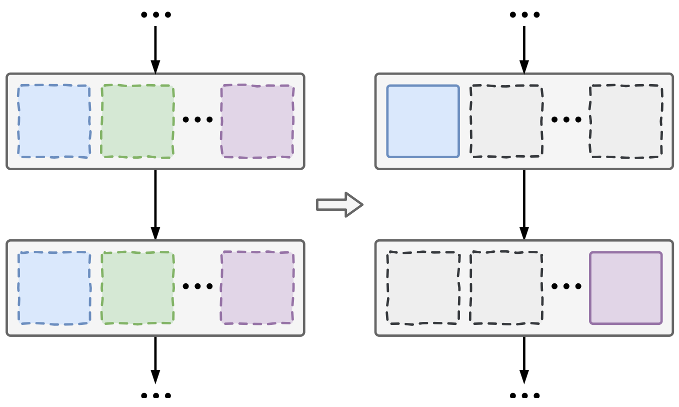


Fig. 5: The procedure of BCS. The left part shows all candidate blocks (highlighted with the dashed rectangles) in each layer of BCS. The right part shows the selected blocks (highlighted with the solid rectangles). Before the training process, each block proposal in the super block has an equal sampling probability. After the training process, the sampling probability of the super block converges to a certain block, and the block connection is obtained.

1) Super Block Optimization: A super block consisting of all the proposals by BPS is constructed for each layer, as shown in Fig. 5, and it takes the weighted output of all blocks as the final output:

$$B^l(x) = \sum_i P(i;l) \cdot b_i^l(x), \tag{2}$$

where $P(i;l)$ denotes the sampling possibility of the i-th block at the l-th layer, $b_i^l(x)$ is the output of the i-th block for the input x, $B^l(x)$ is the output of the super block at the l-th layer, $P(i;l)$ is calculated by SOFTMAX function.

In order to choose a certain block, the sampling probability of block proposals in l-th layer, denoted $\boldsymbol{P}(l)$, needs to converge to a one-hot vector as close as possible. We add random noise and temperature parameter in SOFTMAX function:

$$P(i;l) = \frac{\exp\left((\alpha_i^l + g_i^l)/\tau\right)}{\sum_j \exp\left((\alpha_j^l + g_j^l)/\tau\right)}, \tag{3}$$

where $\boldsymbol{\alpha^l}$ denotes the architecture parameter in l-th layer, and $g_i^l \sim \text{GUMBEL}(0,1)$ denotes the random noise following Gumbel distribution. τ denotes the temperature, making the $P(i;l)$ in Equation 3 a uniform distribution when $\tau \to \infty$. $P(i;l)$ approaches the ARGMAX function and becomes equivalent to the discrete sampler when $\tau \to 0$. During the training, we decrease the τ with fixed decay, so that the network architecture gradually converges to a deterministic topology. The random noise g_i^l can improve the diversity of block selection in the training process and avoid situations where some blocks are rarely trained.

2) Latency Loss: As illustrated in Equation 1, it is required to optimize both the accuracy and latency for real-world applications. To optimize this multi-objective optimization problem directly by SGD, we convert the loss function to:

$$\min_{a \in \mathcal{A}} \min_{w_a} \mathcal{L}(a, w_a) \cdot \mathcal{L}_{latency}(a). \tag{4}$$

$\mathcal{L}(a, w_a)$ denotes the cross-entropy loss of architecture a with parameter w_a, and $\mathcal{L}_{latency}(a)$ denotes the loss of architecture latency, which is defined as:

$$\mathcal{L}_{latency}(a) = \lambda \left(\frac{\text{LATENCY}(a)}{T_{target}} - 1 \right) + 1. \tag{5}$$

LATENCY(a) represents the inference time of the structure a on the device, meanwhile T_{target} denotes the target latency constraint in millisecond. The parameter λ controls the degree of penalty. When the time exceeds the constraint, the degree is greater. Otherwise, when the time satisfies the constraint, a small reward is given. In our design, the parameter λ is assigned with different values:

$$\lambda = \begin{cases} 1, & \text{LATENCY}(a) \geq T_{target} \\ 0.01, & \text{LATENCY}(a) < T_{target} \end{cases}. \tag{6}$$

However, we are unable to make accurate latency measurements for each structure in such a vast space. Similar to the Section III-B, we use the sum of the latency of each layer in the architecture to approximate the latency of the whole architecture, which could be illustrated as:

$$\text{LATENCY}(a) = \sum_i \text{LATENCY}(block_i, a), \tag{7}$$

where $block_i$ denotes the i-th block in architecture a.

3) Training of Supernet: We stack the super block described in Section III-C1 to construct the supernet using the backbone network shown in Fig. 2. We adopt the lookup table to obtain the LATENCY$(block_i^l)$ and use sampling probability $P(i;l)$ to estimate the latency of the whole network, which can be described as follows:

$$\text{LATENCY}(a) = \sum_l \sum_i P(i;l) \cdot \text{LATENCY}(block_i^l, a). \tag{8}$$

With the LATENCY(a) designed above, the loss function in Equation 4 can be differentiated. We can use SGD optimizer to solve this optimization problem, and acquire the network that meets our target latency constraint on the device.

Overall, the search space of our proposed method is very large, which is also described in Section III-A. In the block search stage, when one block is obtained, we compose the whole network based on the backbone network shown in Fig. 2. If we search the whole structure with the latency constraint, the search cost of the evolutionary algorithm would be higher than our proposed block search scheme. In Mnas-Net [7], the search process is to search the whole network structure dynamically, and the search cost is very high. As a result, in our BP-NAS pipeline, our coarse-to-fine framework can enlarge search space and still requires relatively low search cost.

IV. Experiments

A. Implementation details

In this section, we introduce the implementation details of our BP-NAS, a two-stage architecture search method with latency constraint. Taking Samsung Galaxy S9 with Qualcomm

Snapdragon 845 as the target device, we aim to learn effective architecture which has high accuracy and low latency. In this paper, all our architecture search processes are implemented on the PyTorch [66] platform. Meanwhile, the latency of a given model is evaluated on our float32 inference engine with a single image.

Configuration of BPS. During BPS, population size and sample size are set as $P = 100$ and $S = 25$, respectively. To measure the latency, a latency lookup table is developed on CIFAR-10 [65] dataset with 32×32 input resolution. Meanwhile, we will select the architecture with the highest accuracy on validation dataset from the history of evolution algorithm. In our implementation, we design five latency constraints, resulting in five block proposals. For model training, we use SGD to optimize weights of the architecture generated by evolution algorithm with learning rate, momentum and L2 weight decay as 0.06, 0.9 and 5×10^{-4}, respectively. Moreover, the model is trained for 30 epochs on 1 GPU with batch size as 400.

Configuration of BCS. In the period of BCS, we randomly choose 100 classes from ImageNet dataset, which we cal ImageNet-100, to train the supernet for 120 epochs with a mini-batch of 640. Specifically, a lookup table is designed for ImageNet [41] classification dataset with 224×224 input resolution to calculate the latency. In the first ten epochs, we only update the weights w_a while the architecture probability parameter $\boldsymbol{\alpha}$ remains unchanged. Next, both w_a and $\boldsymbol{\alpha}$ are optimized in an iterative manner. To be specific, we first use 80% of the training set to train $\boldsymbol{\alpha}$, and then use the remaining 20% of the training set to train $\boldsymbol{\alpha}$ in each epoch. Besides, the value of the initial temperature parameter τ in Equation 3 is set as 5, and it decays by $exp(-0.045) \approx 0.956$ every epoch. During optimization of w_a, we adopt the standard Nesterov SGD optimizer with the cosine learning rate scheduler, where the initial learning rate is set to be 0.2 and the final learning rate is decreased to 0.0001. Besides, the momentum is set as 0.9, and weight decay is set as 10^{-4}. When optimizing $\boldsymbol{\alpha}$, we use the Adam optimizer with weight decay as 5×10^{-4} and learning rate as 0.02.

Architecture Analysis. Under the constraints of different latencies, we have obtained three architectures. All these architectures, named as BPN-A, BPN-B and BPN-C, are illustrated in Fig. 6. It is clear that all networks only constructed by one or two path blocks, which is attributed to limited latency constraints. In addition to the limitation, advantages of our method are also remarkable. Naturally, it can be seen that blocks in shallow layers prefer to adopt relatively simple architectures, which lies in two aspects. On the one hand, as the layer becomes deeper, the expand ratio has a tendency to become bigger. On the other hand, blocks with only one path are concentrated in the bottom layers. Moreover, an impressive result has been observed. There are two forms of the simple path in networks. If it constructed a layer individually, its kernel size would be 3. On the contrary, when it is parallel with a complex path, the kernel size is always 1. The construction mode is similar to inception modules, which demonstrates its rationality from a human perspective. Besides, the certain collocation can reduce the search space to some degree.

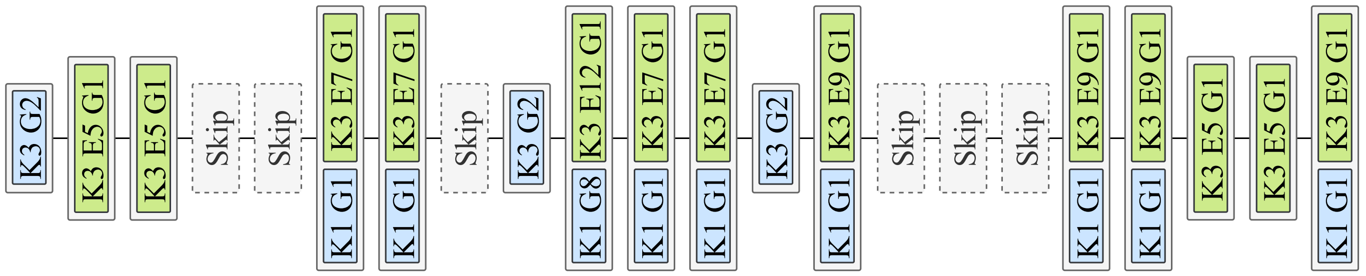
Model	Top-1 (%)	Mobile Latency	Type[†]	Cost[‡]
1.0-MobileNetV1 [53]	70.6	70.9ms	M	-
1.0-MobileNetV2 [1]	72.0	52.7ms	M	-
1.2-MobileNetV2 [1]	73.7	70.6ms	M	-
1.3-MobileNetV2 [1]	74.4	80.7ms	M	-
1.4-MobileNetV2 [1]	74.7	91.0ms	M	-
1.5-ShuffleNetV2 [54]	72.6	49.4ms	M	-
2.0-ShuffleNetV2 [54]	74.9	84.7ms	M	-
CondenseNet [57] (G=C=8)	71.0	-	M	-
CondenseNet [57] (G=C=4)	73.8	-	M	-
MnasNet [7]	74.0	82.6ms	RL	91000
PNASNet [8]	74.2	-	S	6000
AmoebaNet-A [6]	74.5	-	E	75600
NASNet-A [5]	74.0	-	RL	48000
V3-Large 0.75 [21]	73.3	60.8ms	RL	>91000
V3-Large 1.0 [21]	75.2	65.6ms	RL	>91000
DARTS [3]	73.3	-	G	96
P-DARTS [20]	75.3	-	G	7.2
PC-DARTS [19]	74.9	-	G	2.4
ProxylessNAS [2] (mobile)	74.6	-	G	200
FBNet-A [4]	73.0	58.7ms	G	216
FBNet-B [4]	74.1	70.3ms	G	216
FBNet-C [4]	74.9	86.8ms	G	216
BPN[♯](ours) (first stage)	-	-	E	3600
BPN-A(ours)	74.0	**52.2ms**	G	288
BPN-B(ours)	75.2	69.4ms	G	288
BPN-C(ours)	**75.6**	83.2ms	G	288

TABLE I: The classification results on the ImageNet dataset. [†] M, G, RL, S and E represent "manual", "gradient", "reinforcement learning", "SMBO" and "evolution", respectively. [‡] Cost is measured by GPU hours. [♯] We only need to perform the evolution algorithm only once to obtain the block proposals, and we employ these proposals to obtain the diverse models under different latency constraints.

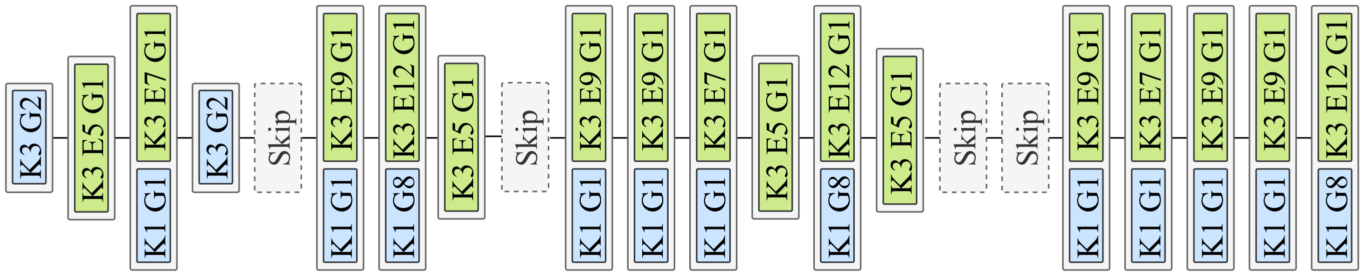
B. ImageNet Classification

To evaluate the superiority of BP-NAS, we apply it to the task of ImageNet classification. We train the models searched by our BP-NAS framework on the ImageNet classification dataset. The training configuration of our models on ImageNet is as follows. We adopt standard Nesterov SGD optimizer with the momentum of 0.9 and weight decay of 4×10^{-5}. Each model is trained with a mini batch of 2048. Meanwhile, cosine learning rate scheduler is employed. The results are shown in Table I.

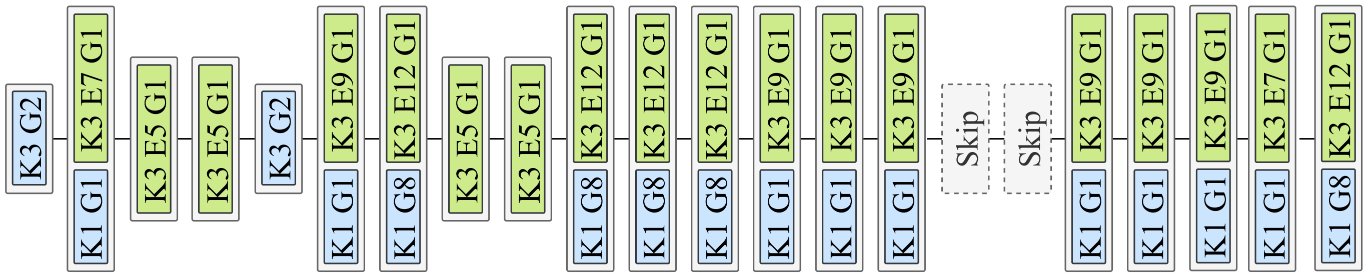
First of all, we compare our models with some manually designed models. BPN-A improves the top-1 accuracy by 2.0% and 1.4% while preserving similar latency with 1.0-MobileNetV2 [1] and 1.5-ShuffleNetV2 [54], respectively. BPN-B reaches 75.2% top-1 accuracy, which is better than 1.0-MobileNetV1 (+4.6%), 1.2-MobileNetV2 (+1.5%) with similar latency. Moreover, BPN-C achieves 75.6% top-1 accuracy which is superior to 1.4-MobileNetV2 and 2.0-



(a) BPN-A



(b) BPN-B



(c) BPN-C

Fig. 6: The block structures of our BPN-A, BPN-B and BPN-C. The blue block represents the simple path and the green block represents the complex path. Our searched models are based on the backbone network shown in Fig. 2. "K3E7G1" means the kernel size is 3, the expansion ratio is 7 and the group size is 1, which defines the hyperparameters of the convolutional layer.

ShuffleNetV2 [54] while preserving lower latency.

Then, we make a comparison with automated models. For the comparion with differentiable NAS methods, our searched models are better than the models searched by FBNet [4]. Besides, our BPN-C obtains higher top-1 accuracy than FBNet-C(+0.4%), DARTS(+2.3%), P-DARTS(+0.3%), PC-DARTS(+0.7%). At the same time, FBNet [4], DARTS [3], P-DARTS [20] and PC-DARTS [19] have lower search cost. For the comparison with indifferentiable NAS methods, it can be seen that our BPN-C wins higher top-1 accuracy than PNASNet [8], AmoebaNet-A [6], NASNet-A [5], and MnasNet [7]. On the other hand, the search cost of BPN-C is far less than these methods. Notably, our cost in the first stage is only 0.04 times of MnasNet, which is estimated based on FBNet [4]. Moreover, as the candidate blocks are reusable, our BCS would be over 300 times faster than MnasNet for repeated usage. Besides, when comparing current state-of-the-art lightweight neural network MobileNetV3 [21], our searched models also obtain comparable performance. Specifically, our BPN-B achieves same accuracy with V3-Large 0.1, and is a little slower than V3-Large 0.1. Our BPN-A is superior (+0.7% top-1 accuracy) and is faster than V3-Large 0.75 apparently. At the same time, the method of

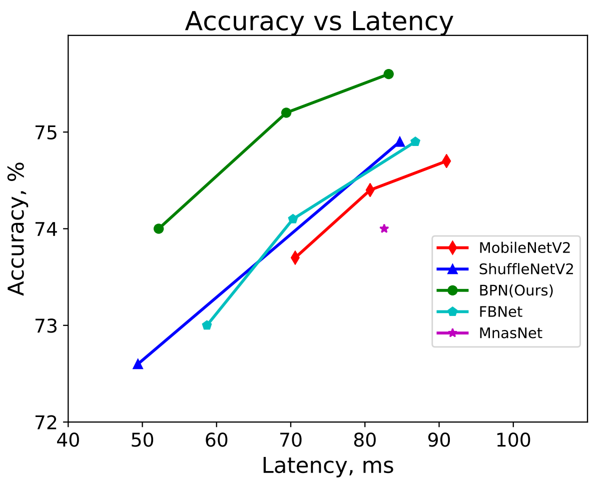


Fig. 7: Performance comparison of accuracy and latency on ImageNet.

MobileNetV3 [21] costs too much GPU hours, and contains some manual designed strategies including activation function, large squeeze-and-excite, etc. We observe that MobileNetV3 is based on the MnasNet [7], so we think that the search cost of MobileNetV3 is higher than MnasNet.

In general, it can be seen from Fig. 7 that our BPN is significantly superior to other methods, whether manually designed models or automatically searched models. Compared to other architecture search methods based on the gradient, ours do not need manually designed block proposals. In addition, we should run evolutionary algorithm only one time to search block proposals and obtain models restricted to different latency constraints, which means that our search cost is much lower than other NAS methods.

C. Results for Other Computer Vision Tasks

In order to prove the generalization of our BPNs, we transfer BPNs as a backbone network to other vision tasks directly. In our settings, we evaluate the performance of BPNs on COCO object detection task, MegaFace one-to-million face identification task and Market-1501 person re-identification task.

Object detection results. We adopt BPN and MobileNetV2 [1] as feature extractors on object detection with Faster-RCNN [67] + FPN [68] on COCO dataset [42]. We reimplement Faster-RCNN + FPN with the same configuration for all backbones as that in [68]. Backbones are pre-trained on ImageNet classification task. After that, all backbones are trained on `trainval35k` and evaluated on a 5k subset of val images (`minival`). During training, input images are resized such that its shorter side has 800 pixels. We train all models on 8 GPUs and Cross-GPU synchronized batch normalization is adopted with a mini-batch of 4 per GPU. Meanwhile, we use the SGD optimizer with weight decay of 0.0001 and moment of 0.9. The initial learning rate is set as 0.04 and decreased by a factor of 0.1 after 60k and 80k iterations, which is same as the "2x" schedule in the public-available Detectron [69]. Detailed results are reported in Table II. At first, it should be noted that the pairs of BPN-A versus 1.0-MobileNetV2, BPN-B versus 1.2-MobileNetV2 both have similar latency on the target device. Naturally, BPN-A achieves 2.5% improvement on mAP compared with 1.0-MobileNetV2 while BPN-B is 2% better than 1.2-MobileNetV2. Notably, the AP^{50} of BPN-B is 0.9% higher than that of ResNet-50 [40] while AP of small objects is even 2.6% higher. Moreover, BPN-B is very lighter compared to ResNet-50.

Face recognition results. We also perform the experiments on the face recognition task. Specifically, BPN-A and BPN-B are compared with MobileNetV2 [1], which has achieved promising results in many tasks. All the experiments are performed based on the criterion of the ArcFace [70] loss fuction with $m = 0.5$. The learning rate is initialized as 0.1, which is divided by 10 at the 100k-th, 140k-th, and 160k-th iterations. The momentum is set as 0.9 and the weight decay is set as 5×10^{-4}. The training process is terminated after reaching the 200k-th iterations. Based on the above settings, we perform the experiments using MS-Celeb-1M [71] and VGG2-Face [72] as the training datasets and evaluate different methods on the MegaFace [43] benchmark dataset. When compared with 1.0-MobileNetV2, our BPN-A improves the identification rate from 91.57% to 92.40% at the distractors of 10^6 . Meanwhile, our BPN-B achieves the accuracy of 93.87% , which outperforms 1.2-MobileNetV2 with the accuracy of 92.74%.

Backbone model	mAP	AP^{50}	AP^{S}	AP^{M}	AP^{L}
1.0-MobileNetV2 [1]	31.8	53.5	18.4	34.5	41.0
BPN-A	**34.3**	56.3	20.0	37.0	44.9
1.2-MobileNetV2 [1]	34.1	56.2	19.6	37.7	43.7
BPN-B	**36.1**	58.2	21.0	39.1	47.8
ResNet-50 [40]	36.5	57.3	18.4	40.6	50.6

TABLE II: Performanece comparison of our BPN and other backbones with Faster-RCNN + FPN for the object detection task on the COCO dataset. AP at the IoU threshold of 0.5 ($AP^{0.5}$) and APs based on small, medium and large objects (AP^{S}, AP^{M}, AP^{L}) as defined in [42].

Person re-identification results. We also use our BPNs for the person re-identification task. The experiments are conducted on the Market-1501 [44] dataset. We set the batch size as 32 and train the whole network in 90 epochs. The initial learning rate is set as 0.0003, which is divided by 10 at the 45th and the 60th epoches. Meanwhile, SGD is adopted with the weight decay as 5×10^{-4} and the momentum as 0.9. We compare our BPN-A with 1.0-MobileNetV2. Our BPN-A model achieves the Rank-1 accuracy of 67.2% and the mAP of 45.4%, which outperforms 1.0-MobileNetV2 with the Rank-1 accuracy of 64.3% and the mAP of 42.3%. Simultaneously, our BPN-B model achieves the Rank-1 accuracy of 67.4% and the mAP of 46.3%, while 1.2-MobileNetV2 achieves the Rank-1 accuracy of 66.1% and the mAP of 44.8%. Overall, our searched models are much better than MobileNetV2.

The above experimental results demonstrate our BPN models outperform the baseline methods for different computer vision tasks, which demonstrates our proposed BP-NAS method has the advantages for different computer vision tasks in terms of generalization capability and robustness.

D. Ablation Study

In order to demonstrate the effectiveness of our proposed BP-NAS framework, we perform ablation studies in this section.

1) Effectiveness of each stage: We also introduce randomness at each stage in our training procedure as randomness is shown to be also competitive baseline in several recent NAS works [3], [73]. In this section, we perform new experiments to evaluate two alternative methods by introducing randomness at each training stage. In the first alternative method, we replace the first BPS stage with randomly selected block proposals, and apply the BCS stage described in Section III-C to obtain the connections of each layer. The detailed process is described as follows. We first randomly generate 100 block proposals, and select the best block proposals according to the accuracy on the CIFAR-10 dataset (See Section IV-A for the details of the training process on the CIFAR-10 dataset). After that, we utilize the BCS stage to obtain the final model, which is named as BPN-Rand-Stage-1. In the second alternative method, we first obtain the block proposals from the BPS stage described in Section III-B, and then randomly select the connections of each layer to obtain the final searched model, which is named as BPN-Rand-Stage-2. As shown in Table III, we compare our BPN-A method with two alternative methods. It is clear that our BPN-A is better than both alternative methods by a large margin, which verifies the effectiveness of our proposed two-stage BP-NAS framework.

Moreover, we also utilize the predefined candidate blocks in FBNet [4] as our block proposals in the second BCS stage. The searched model is named as BCS-A. As shown in Table III, our BPN-A outperforms BCS-A while preserving lower latency, which indicates our searched block proposals are better than the manually designed proposals.

Model	Top-1(%)	Mobile Latency
BPN-Rand-Stage-1	69.2	60.7ms
BPN-Rand-Stage-2	72.5	66.7ms
BCS-A	73.0	55.1ms
BPN-A	**74.0**	**52.2ms**

TABLE III: Performance for the classification task on ImageNet dataset when comparing the effectiveness at each stage.

2) Effectiveness of different λ values of the latency loss: The latency loss defined in Equation 5 is a regularization term for Equation 4. The λ value in Equation 5 is used to control the weight of the penalty. We have performed the detailed experiments to choose the appropriate value in this section. Specifically, we set the value of λ as 1 when $\text{LATENCY}(a)$ is larger than T_{target}. When the $\text{LATENCY}(a)$ is smaller than T_{target}, we have tried many values (*i.e.*, 0, 0.001, 0.01, 0.05, 0.5, 1) as shown in Table IV. In addition, we empirically set the parameter T_{target} in Equation 5 as 55ms in the second BCS stage. We benchmark the mobile latency of all the searched models and retrain them on the ImageNet dataset. Besides, in Equation 5 , when the latency of the network architecture satisfies the latency constraint and the λ value is higher, the reward of the latency loss is higher.

In Table IV, with regard to the latency value of the searched model, when the value of λ is set as 0, 0.5, 1, we observe that the mobile latency of the searched model is not well suitable for the target latency value, which cannot meet the requirement of application scenarios. When we set the value of λ as 0.001, 0.01, 0.05, the latency of the searched model is close to our designed target latency T_{target}. For the top-1 accuracy on ImageNet, the searched model with λ = 0.01 is better than other searched models.

3) Analysis of the evolutionary algorithm of the block proposal stage: As we all know, appropriate randomness is important to avoid converging to suboptimal results for the evolutionary algorithms. At the same time, there is an exploration-exploitation trade-off [74] in the NAS field, in which the exploration process aims to avoid premature convergence to suboptimal architectures, while the exploitation process aims to discover well-performing architectures quickly.

λ ($\text{LATENCY}(a) < T_{target}$)	Top-1(%)	Mobile Latency
0	63.6	26.0ms
0.001	73.6	56.2ms
0.01	**74.0**	**52.2ms**
0.05	73.2	48.3ms
0.5	62.3	25.7ms
1	52.4	16.7ms

TABLE IV: Performance for the classification task on ImageNet dataset when using different λ values.

So in our work, our coding strategy can help the exploration process by introducing randomness in the mutation process. Specifically, the kernel size 3 can be changed to 7 directly, which can help us find more architectures and avoid local suboptimal architectures.

Moreover, we have conducted new experiment by changing the encoding method with another pattern. (0,0), (0,1), (1,1), (1,0) denote the kernel size of 1, 3, 5, 7, respectively. (0,0,0,0), (0,0,0,1), (0,0,1,1), (0,0,1,0), (0,1,1,0), (0,1,1,1), (0,1,0,1), (0,1,0,0), (1,1,0,0), (1,1,0,1), (1,1,1,1), (1,1,1,0), (1,0,1,0), (1,0,1,1), (1,0,0,1), (1,0,0,0) represent the expansion ratio of 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, respectively. We have learnt one new model with the latency of 56.3 ms and the top-1 accuracy of 74.0% on ImageNet, which is close to the performance of our BPN-A model. The result demonstrates the effectiveness of our designed evolutionary algorithm, which is also not sensitive to the coding order.

V. Conclusion

In this paper, we have proposed a two-stage framework named BP-NAS for effective platform-aware lightweight neural architecture search. We first search for the micro-scale proposals by using a new evolutionary approach with latency constraints and then search for the connections between the selected proposals, in which an improved latency loss function is proposed. This simple coarse-to-fine search strategy is quite efficient. Extensive experimental results across multiple computer vision tasks demonstrate the superiority of our approaches over the existing platform-aware NAS methods. Furthermore, how to extend the search space is an important research problem in NAS. Our coarse-to-fine approach aims to extend the search space while maintaining low search costs. In the future, we will continue to work along this direction and study better search space and search strategies.

References

[1] 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*, 2018, pp. 4510–4520.

[2] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019.

[3] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2018.

[4] 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," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[5] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *International Conference on Machine Learning*, 2017.

[6] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 2019, pp. 4780–4789.

[7] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[8] 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)*, 2018, pp. 19–34.

[9] Z. Zhong, Z. Yang, B. Deng, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Blockqnn: Efficient block-wise neural network architecture generation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[10] 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*, 2018, pp. 2423–2432.

[11] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*, 2018, pp. 4095–4104.

[12] 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*. Elsevier, 2019, pp. 293–312.

[13] 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*, 2018, pp. 8697–8710.

[14] 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*. JMLR. org, 2017, pp. 2902–2911.

[15] L. Xie and A. Yuille, "Genetic cnn," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1379–1388.

[16] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *International Conference on Learning Representations*, 2018.

[17] 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*, 2018, pp. 549–558.

[18] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *International Conference on Machine Learning*, 2017.

[19] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, "Pc-darts: Partial channel connections for memory-efficient architecture search," in *International Conference on Learning Representations*, 2019.

[20] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1294–1303.

[21] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1314–1324.

[22] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1761–1770.

[23] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Advances in neural information processing systems*, 2018, pp. 2016–2025.

[24] B. Baker*, O. Gupta*, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," 2018. [Online]. Available: https://openreview.net/forum?id=BJypUGZ0Z

[25] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1946–1956.

[26] D. Zhou, X. Zhou, W. Zhang, C. C. Loy, S. Yi, X. Zhang, and W. Ouyang, "Econas: Finding proxies for economical neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 396–11 404.

[27] X. Li, C. Lin, C. Li, M. Sun, W. Wu, J. Yan, and W. Ouyang, "Improving one-shot nas by suppressing the posterior fading," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 836–13 845.

[28] R. Guo, C. Lin, C. Li, K. Tian, M. Sun, L. Sheng, and J. Yan, "Powering one-shot topological nas with stabilized share-parameter proxy," *arXiv preprint arXiv:2005.10511*, 2020.

[29] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun, "Detnas: Backbone search for object detection," in *Advances in Neural Information Processing Systems*, 2019, pp. 6642–6652.

[30] F. Liang, C. Lin, R. Guo, M. Sun, W. Wu, J. Yan, and W. Ouyang, "Computation reallocation for object detection," in *International Conference on Learning Representations*, 2019.

[31] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 82–92.

[32] H. Sahbi and D. Geman, "A hierarchy of support vector machines for pattern detection," *Journal of Machine Learning Research*, vol. 7, no. Oct, pp. 2087–2123, 2006.

[33] H. Sahbi, "Coarse-to-fine deep kernel networks," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017, pp. 1131–1139.

[34] H. Sahbi, D. Geman, and N. Boujemaa, "Face detection using coarse-to-fine support vector classifiers," in *Proceedings. International Conference on Image Processing*, vol. 3. IEEE, 2002, pp. 925–928.

[35] S. Z. Li and Z. Zhang, "Floatboost learning and statistical face detection," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 26, no. 9, pp. 1112–1123, 2004.

[36] S. C. Brubaker, J. Wu, J. Sun, M. D. Mullin, and J. M. Rehg, "On the design of cascades of boosted ensembles for face detection," *International journal of computer vision*, vol. 77, no. 1-3, pp. 65–86, 2008.

[37] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5325–5334.

[38] W. Ouyang, K. Wang, X. Zhu, and X. Wang, "Chained cascade network for object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1938–1946.

[39] K. Chen, J. Pang, J. Wang, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Shi, W. Ouyang *et al.*, "Hybrid task cascade for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 4974–4983.

[40] 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*, 2016, pp. 770–778.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[42] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[43] I. Kemelmacher-Shlizerman, S. M. Seitz, D. Miller, and E. Brossard, "The megaface benchmark: 1 million faces for recognition at scale," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4873–4882.

[44] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification: A benchmark," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1116–1124.

[45] Y. Amit and D. Geman, "A computational model for visual selection," *Neural computation*, vol. 11, no. 7, pp. 1691–1715, 1999.

[46] W. Kienzle, M. O. Franz, B. Schölkopf, and G. H. Bakir, "Face detection—efficient and rank deficient," in *Advances in Neural Information Processing Systems*, 2005, pp. 673–680.

[47] C. J. Burges and B. Schölkopf, "Improving the accuracy and speed of support vector machines," in *Advances in neural information processing systems*, 1997, pp. 375–381.

[48] S. Changpinyo, M. Sandler, and A. Zhmoginov, "The power of sparsity in convolutional neural networks," *arXiv preprint arXiv:1702.06257*, 2017.

[49] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[50] N. Yu, S. Qiu, X. Hu, and J. Li, "Accelerating convolutional neural networks by group-wise 2d-filter pruning," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2502–2509.

[51] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *International Conference on Learning Representations (ICLR)*, 2016.

[52] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[53] 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.

[54] 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*, 2018, pp. 6848–6856.

[55] 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)*, 2018, pp. 116–131.

[56] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[57] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2752–2761.

[58] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[59] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019, pp. 6105–6114.

[60] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *International Conference on Learning Representations*, 2018.

[61] Y. Akimoto, S. Shirakawa, N. Yoshinari, K. Uchida, S. Saito, and K. Nishida, "Adaptive stochastic natural gradient method for one-shot neural architecture search," in *International Conference on Machine Learning*, 2019, pp. 171–180.

[62] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, "Shift: A zero flop, zero parameter alternative to spatial convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9127–9135.

[63] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[64] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1492–1500.

[65] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, p. 4, 2014.

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[67] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[68] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2117–2125.

[69] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, "Detectron," https://github.com/facebookresearch/detectron, 2018.

[70] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4690–4699.

[71] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao, "Ms-celeb-1m: A dataset and benchmark for large-scale face recognition," in *European Conference on Computer Vision*, 2016, pp. 87–102.

[72] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age," in *IEEE International Conference on Automatic Face & Gesture Recognition*. IEEE, 2018, pp. 67–74.

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

[74] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

RESPONSES TO COMMENTS

We thank the associate editor (AE) and two reviewers for their comments. We have revised this manuscript substantially based on the reviewers' comments.

To the Associate Editor: Thank you very much. We have carefully addressed the comments from all reviewers and revised the manuscript accordingly. Please see below for the detailed responses to the comments from each reviewer.

Reviewer 1

Comment 1: The authors carefully addressed my concerns in the revision. I think this paper indeed provides new insights to the NAS community and meets the standard of the journal, so I suggest acceptance this time.

Response: Thanks for recognizing our contributions. We have revised our manuscript based on your comments.

Comment 2: the way to extend the search space is indeed important and the current paper is not a perfect solution (by copying the block for 20+ times). I shall encourage the authors to continue research in this direction to make this method even stronger.

Response: Thanks for your kind suggestions. Extending search space is very important in the NAS field. Our coarse-to-fine approach aims to extend search space while maintaining low search costs. In the future, we will continue to work in this direction to design better search space and search strategy.

The newly added discussion is provided at the last of the Conclusion Section (See Section V, Page 10), which is also shown as follows,

> " ...Furthermore, how to extend the search space is an important research problem in NAS. Our coarse-to-fine approach aims to extend the search space while maintaining low search costs. In the future, we will continue to work along this direction and study better search space and search strategies. "

Comment 3: BTW, please try to clean up the reference. For example, [5] should be ICLR 2017, [18] should be ICLR 2017, [19] should be ICLR 2020, etc.

Response: We have cleaned up the reference section in the revised manuscript.

Reviewer 2

Comment 1: The paper addresses the issues raised in the previous review. Some parts of the paper require a minor revision to correct some typos, and improve english usage...

Response: Thanks for your kind review. We have corrected some typos and improved English usage in our revised manuscript.

Comment 2: The reference section should be made more comprehensive, with more papers about (i) NAS and other efficient deep learning models, AND (ii) early works on coarse to fine and latency-driven models in general machine learning (including but not limited to neural nets)...

Response: As suggested by the reviewer, we have introduced more related works in our revised manuscript. The references [R1]-[R17] from the reviewer are denoted as [23], [24], [25], [29], [51], [50], [49], [48], [47], [32], [37], [46], [33], [36], [45], [34], [35] in the revised manuscript.

(See Section I, First Paragraph, Page 1)

> " Neural architecture search (NAS) was proposed to relieve the burden from the hand-crafted network architecture engineering process, and its effectiveness has been demonstrated in various computer vision tasks, such as image classification [2]-[28], object detection [29],[30] and image segmentation [31]."

(See Section I, Fifth Paragraph, Page 1)

> " This work aims to develop an efficient optimization method to conduct both block structure search and connection search. Inspired from the coarse-to-fine strategy [32]-[39], we propose a two-stage framework named Block Proposal Neural Architecture Search (BP-NAS). "

(See Section II, First Paragraph, Page 2)

> " Efficiency is often crucial for many real-world applications[45]-[51]."

[R1] Neural architecture search with bayesian optimisation and optimal transport
[R2] Accelerating neural architecture search using performance prediction
[R3] Auto-keras: An efficient neural architecture search system
[R4] Detnas: Neural architecture search on object detection
[R5] Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding
[R6] Accelerating convolutional neural networks by group-wise 2d-filter pruning
[R7] Learning both weights and connections for efficient neural network
[R8] The power of sparsity in convolutional neural networks
[R9] Improving the accuracy and speed of support vector machines
[R10] A Hierarchy of Support Vector Machines for Pattern Detection
[R11] A convolutional neural network cascade for face detection
[R12] Face Detection — Efficient and Rank Deficient

[R13] Coarse-To-Fine Deep Kernel Networks
[R14] On the design of cascades of boosted ensembles for face detection
[R15] A computational model for visual selection
[R16] Face detection using coarse-to-fine support vector classifiers
[R17] Floatboost learning and statistical face detection