RelativeNAS: Relative Neural Architecture Search via Slow-Fast Learning

Hao Tan, Ran Cheng, *Senior Member, IEEE*, Shihua Huang, Cheng He, *Member, IEEE*, Changxiao Qiu, Fan Yang, Ping Luo

arXiv:2009.06193v3 [cs.CV] 13 Jul 2021

Abstract—Despite the remarkable successes of Convolutional Neural Networks (CNNs) in computer vision, it is time-consuming and error-prone to manually design a CNN. Among various Neural Architecture Search (NAS) methods that are motivated to automate designs of high-performance CNNs, the differentiable NAS and population-based NAS are attracting increasing interests due to their unique characters. To benefit from the merits while overcoming the deficiencies of both, this work proposes a novel NAS method, RelativeNAS. As the key to efficient search, RelativeNAS performs joint learning between fast-learners (*i.e.* decoded networks with relatively lower loss value) and slow-learners in a pairwise manner. Moreover, since RelativeNAS only requires low-fidelity performance estimation to distinguish each pair of fast-learner and slow-learner, it saves certain computation costs for training the candidate architectures. The proposed RelativeNAS brings several unique advantages: (1) it achieves state-of-the-art performances on ImageNet with top-1 error rate of 24.88%, *i.e.* outperforming DARTS and AmoebaNet-B by 1.82% and 1.12% respectively; (2) it spends only nine hours with a single 1080Ti GPU to obtain the discovered cells, *i.e.* $3.75\times$ and $7875\times$ faster than DARTS and AmoebaNet respectively; (3) it provides that the discovered cells obtained on CIFAR-10 can be directly transferred to object detection, semantic segmentation, and keypoint detection, yielding competitive results of 73.1% mAP on PASCAL VOC, 78.7% mIoU on Cityscapes, and 68.5% AP on MSCOCO, respectively. The implementation of Relative-NAS is available at https://github.com/EMI-Group/RelativeNAS.

Index Terms—AutoML, Convolutional Neural Network, Neural Architecture Search, Population-Based Search, Slow-Fast Learning.

I. INTRODUCTION

DEEP Convolutional Neural Networks (CNNs) have achieved remarkable results in various computer vision tasks (*e.g.*, image classification [1], [2], [3], object detection [4], and semantic segmentation [5], [6]), and a number of state-of-the-art networks have been designed by experts since 2012 [7], [8], [9]. Since the manual designs of CNNs heavily rely on expert knowledge and experience, it is usually time-consuming and error-prone. To this end, researchers have turned to automatic generation of high-performance network

H. Tan, R. Cheng, S. Huang, and C. He are with University Key Laboratory of Evolving Intelligent Systems of Guangdong Province, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, P.R. China. E-mails: tanbox@live.com, ranchengcn@gmail.com, shihuahuang95@gmail.com, chenghehust@gmail.com. (*Corresponding author: Ran Cheng*)

C. Qiu and F. Yang are with Hisilicon Research Department, Huawei Technologies Co.,Ltd. Shenzhen 518055, P.R. China. E-mails: {qiuchangxiao, yangfan74}@huawei.com.

P. Luo is with Department of Computer Science, The University of Hong Kong, Hong Kong, P.R. China. E-mail: pluo.lhi@gmail.com.

architectures for any given tasks, a.k.a. Neural Architecture Search (NAS) [10]. Without loss of generality, the problem of NAS for a target dataset $\mathcal{D} = \{\mathcal{D}_{trn}, \mathcal{D}_{vld}, \mathcal{D}_{tst}\}$ and a search space $\mathcal{A}$ can be formulated as [11]:

$$\begin{aligned} \text{minimize} \quad & \mathcal{L}_{vld}(\boldsymbol{\alpha}, \boldsymbol{\omega}^*) \quad\quad\quad (1) \\ \text{subject to} \quad & \boldsymbol{\omega}^* \in \arg\min_{\boldsymbol{\omega}} \mathcal{L}_{trn}(\boldsymbol{\alpha}, \boldsymbol{\omega}), \end{aligned}$$

where $\boldsymbol{\alpha} \in \mathcal{A}$ defines the model architecture, $\boldsymbol{\omega}$ defines the associated weights, and $\boldsymbol{\omega}^*$ defines the corresponding optimal weights. Besides, $\mathcal{D}_{trn}$, $\mathcal{D}_{vld}$, and $\mathcal{D}_{tst}$ are the training data, validation data, and test data, respectively. $\mathcal{L}_{trn}$ and $\mathcal{L}_{vld}$ denote the loss on the training data and validation data respectively.

Despite the promising performance of recent NAS methods on image classification [12], [13], object detection [14], [15], semantic segmentation [16], [17], and designing generative adversarial networks [18], there are two major challenges: (1) NAS is treated as a black-box optimization problem due to the lack of a priori knowledge adopted with the exact functional relationship between architectures and their performances; (2) NAS suffers from high computation costs due to a large number of performance estimations of candidate architectures.

Among various NAS methods, the differentiable NAS (*i.e.* DARTS) [19] and population-based NAS [20] are among the most popular ones due to their unique merits for tackling each challenge: DARTS mainly benefits from the merit of high search efficiency due to relaxing the search space to be continuous; population-based NAS mainly benefits from the merit of diversified candidate architectures in the population and involving genetic operators (*e.g.*, crossover/mutation) to drive the search process. Nevertheless, they also suffer from some deficiencies: since DARTS jointly trains a supernet and search for an optimal solution merely by gradient, it suffers from low robustness in terms of flexibility and versatility; population-based NAS mainly relies on stochastic crossover/mutation for search, and it usually requires a large amount of computation cost for performance evaluations.

One research question is: *can we benefit from the merits of both differentiable NAS and population-based NAS while overcoming their deficiencies?* To answer it, this work proposes a novel RelativeNAS method, as shown in Fig. 1 (right).

Particularly, this work proposes a novel continuous encoding scheme for cell-based search space by considering connections between pairwise nodes and the corresponding operations, inspired by [19]. However, in contrast to the encoding method as given in [19], the proposed one has no requirement of
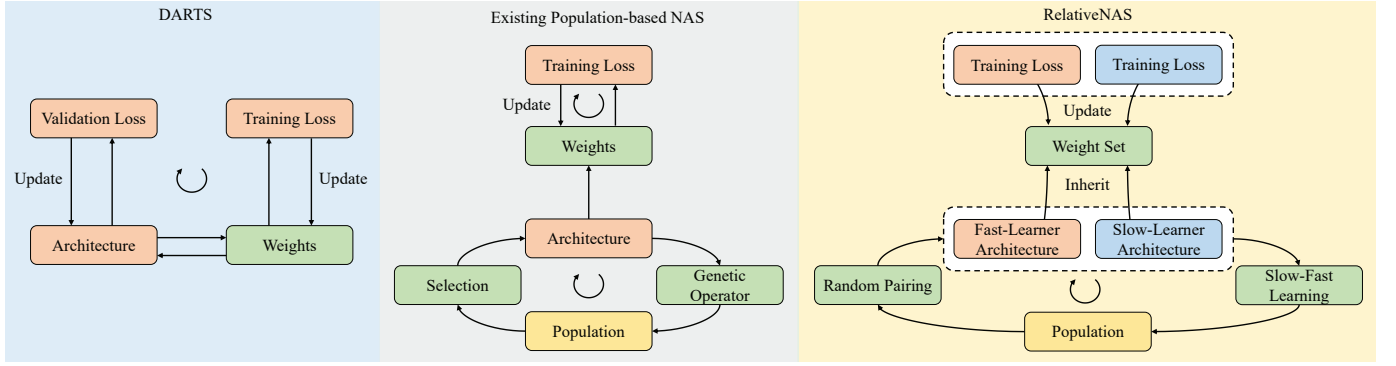
Fig. 1: Illustration of the three general frameworks of different NAS methods. LEFT: DARTS updates the solo candidate architecture and its weights simultaneously by gradients generated by validation loss and training loss respectively. MIDDLE: existing population-based NAS updates a population of architectures by stochastic crossover/mutation. In order to obtain the ranking of the architectures in each generation, the newly generated architectures need to be trained on the training set for a certain number of epochs. RIGHT: RelativeNAS updates a population of architectures by the proposed *slow-fast learning paradigm* (instead of gradient or crossover/mutation). For each pair of architectures, slow-learner is distinguished from fast-learner by their relative performances, where for such low-fidelity performance estimation, each architecture only needs to inherit its weights from a weight set, and is trained by only one epoch to update the weights.

differentiability, neither does it consider the probability/weight of choosing an operation; instead, it directly encodes the operations between pairwise nodes into real values in a naive manner (as shown in Fig. 2). The main advantages of the proposed continuous encoding method are: (1) it provides more flexibility and versatility; (2) the enlarged search space encourages the search for diverse architectures when applied to population-based NAS [21].

With the proposed continuous encoding scheme, this work further proposes a slow-fast learning paradigm for efficient search in the encoded space, inspired by [22]. In this paradigm, a population of architecture vectors is iteratively paired and updated. Specifically, in each iteration of the proposed slow-fast learning, the architecture vectors are randomly paired; for each pair of architecture vectors, the one with worse performance (denoted as slow-learner) is updated by learning from the one with better performance (denoted as fast-learner). In contrast to the population-based NAS methods such as large-scale evolution [23] and AmoebaNet [20], the proposed slow-fast learning paradigm does not involve any genetic operator (*e.g.*, crossover/mutation), but essentially, the architecture vectors are updated using a pseudo-gradient mechanism which aims to learn the joint distribution between each pair of slow-learner and fast-learner implicitly. Specifically, the pseudo-gradient is determined by the pairwise learning between the fast-learner and slow-learner. The main advantages of the proposed slow-fast learning paradigm are: (1) it provides a scheme to perform NAS in generic continuous search space without considering its specific properties (*e.g.* differentiability); (2) it suggests a way to learn joint distributions of multiple architectures.

To improve the computation efficiency of RelativeNAS, this work further adopts a weight set as a knowledge base to estimate the performances when comparing the architectures in each pair, where the weight set is an operation collection of all the candidate architectures, as well as a gathering of the promising knowledge in the population. Since the slow-
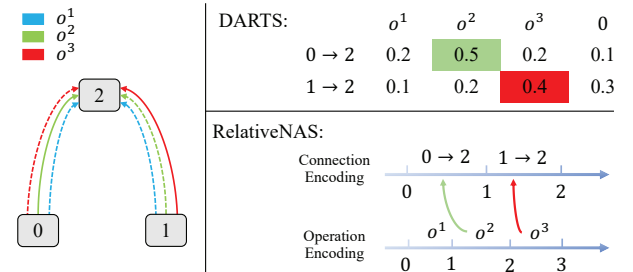

Fig. 2: Examples of the encoding schemes of DARTS [19] and RelativeNAS. LEFT: an architecture to be encoded. Boxes, dashed lines, and solid lines represent different nodes, the candidate operations, and chosen operations respectively. Node 2 needs to choose two operations (*i.e.* two out of the six lines). RIGHT: the two different continuous encoding schemes. $a \rightarrow b$ represents the connection between Node a and Node b while o^1, o^2, and o^3 denote the candidate operations. DARTS encodes every operation by its weight and uses 0, *i.e. zero* operation, to denote that there is no connection between two nodes; each real number means the probability/weight of choosing the corresponding operation. RelativeNAS uses four continuous variables, including two connections and their corresponding operations, to encode the architecture; the real number falling into each interval means that the corresponding connection exists or the corresponding operation is chosen. Note that, since each node in RelativeNAS must connect to two predecessor nodes, $zero$ operation is not considered in the search space.

fast learning only requires low-fidelity performance estimation when distinguishing slow-learner and fast-learner in each pair, a newly discovered network is trained for only one epoch to obtain the estimated performance, thus saving substantial computation cost for performance evaluations. It is worth noting that the weight set is not directly trained but updated

by the trained weights of each paired networks in an online manner. The contributions of this paper can be summarized as follows:

1) We propose a novel continuous encoding scheme and a slow-fast learning paradigm for NAS. Specifically, a population of architecture vectors is iteratively updated by pseudo-gradients, towards optima.

2) We propose a novel performance estimation strategy to expedite the evaluation of architectures, where the weight set is not directly trained but updated using other trained weights. It couples weight set updating and searching into a single integrated online process.

3) The proposed RelativeNAS significantly reduces the general computation cost of NAS, spending only nine hours with a single 1080Ti GPU to obtain the discovered architecture on CIFAR-10. Furthermore, we show that the searched architecture achieves state-of-the-art performance (97.66% accuracy) on CIFAR-10.

4) We transfer the searched architecture to multiple intra- and inter-domain tasks, including image classification, object detection, image segmentation, and keypoint detection. The consistently better performances across these tasks demonstrate that the improvement on CIFAR-10 is an actual advancement from the architecture as obtained by RelativeNAS.

The rest of the paper is organized as follows. The background knowledge, including some related works and the motivation of this work, is given in Section II. The details of the proposed RelativeNAS are elaborated in Section III. Experimental studies are presented in Section IV. Finally, the conclusions are drawn in Section V.

II. BACKGROUND

In this section, we first present some related works, including those in differentiable NAS and population-based NAS. Then, we briefly summarize the motivation of this work.

A. Differentiable Neural Architecture Search

Differentiable NAS is motivated to relax the architecture into continuous and then use gradient-based approaches for optimization. DARTS [19] is a pioneering work in differentiable NAS, where three components have significantly improved the computation efficiency: the cell-based search space, the continuous relaxation approach, and the approximation technique. Specifically, the cell-based search space modularizes the entire CNN into a stack of several cells to reduce the number of parameters to be optimized; the continuous relaxation schema transforms the choices of discrete operations into a differentiable learning objective for the joint optimization of the architecture and its weights; moreover, the approximation technique approximates the architecture gradient for reducing the expensive inner optimization.

Following DARTS, SNAS [24] has proposed to optimize the architecture distribution for the operations during the backpropagation. Specifically, the search space is differentiable by relaxing a set of one-hot random variables, which is used to select the corresponding operations in the graph. Since SNAS

Algorithm 1 RelativeNAS Framework

Require: Training set $\mathcal{L}_{trn}$, validation set $\mathcal{L}_{vld}$, population size N, generation number G

1: Initialize a population $\{\boldsymbol{\alpha}^0\}_{n=1}^N$ and a weight set Ω;
2: **for** $g = 1, 2, \ldots, G$ **do**
3:    $\{\boldsymbol{\alpha^g}\}_{n=1}^N$ is randomly divided into $N/2$ pairs;
4:    **for** $p = 1, 2, \ldots, N/2$ **do**
5:       The p-th pair of encoded vectors $\{\boldsymbol{\alpha}_{p,j}^g\}_{j=1,2}$ are decoded into networks $\{C_{\boldsymbol{\alpha}_{p,j}^g}\}_{j=1,2}$;
6:       $\{C_{\boldsymbol{\alpha}_{p,j}^g}\}_{j=1,2}$ inherit weights $\{\boldsymbol{\omega}_{\boldsymbol{\alpha}_{p,j}^g}\}_{j=1,2}$ from weight set Ω;
      /* train each network with one epoch */
7:       **for** $j = 1, 2$ **do**
8:          $\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,j}^g} = step(\boldsymbol{\omega}_{\boldsymbol{\alpha}_{p,j}^g} | C_{\boldsymbol{\alpha}_{p,j}^g})$;
9:          $\mathcal{L}_{\boldsymbol{\alpha}_{p,j}^g} = \mathcal{L}_{vld}(C_{\boldsymbol{\alpha}_{p,j}^g}, \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,j}^g})$;
10:       **end for**
      /* distinguish slow-learner and fast-learner */
11:       $\boldsymbol{\alpha}_{p,f}^g = \arg\min_{\boldsymbol{\alpha}}(\mathcal{L}_{\boldsymbol{\alpha}_{p,1}^g}, \mathcal{L}_{\boldsymbol{\alpha}_{p,2}^g})$;
12:       $\boldsymbol{\alpha}_{p,s}^g = \arg\max_{\boldsymbol{\alpha}}(\mathcal{L}_{\boldsymbol{\alpha}_{p,1}^g}, \mathcal{L}_{\boldsymbol{\alpha}_{p,2}^g})$;
      /* update weight set Ω */
13:       Ω is updated with (6);
      /* slow-learner learns from fast-learner */
14:       $\boldsymbol{\alpha}_{p,s}^g$ is updated with (4);
15:    **end for**
16: **end for**

---

has used *search gradient* as the reward instead of *training loss* for reinforcement learning, the objective is not changed but the process is more efficient. ProxylessNAS [25] has used the binarized architecture parameters to reduce the GPU memory. Besides, it makes latency differentiable and adds the expected latency into the loss function as a regularization term.

However, a recent work [26] has discovered that the bi-level optimization of weights and architecture in DARTS will collapse when the search epochs become larger. Since the number of weight parameters is larger than the number of architecture parameters, weight optimization will restrain the architecture optimization, and more *identity* operations are chosen to deteriorate the performance. Essentially, it is mainly attributed to the lack of diversity when the solo candidate architecture in DARTS is optimized by the gradient.

B. Population-Based Neural Architecture Search

As indicated by the term itself, population-based NAS maintains a population where each individual inside it represents a candidate architecture. Cooperation among candidate architectures modifies their attributes, and competition for removing the worse and retaining the better pushes the population towards optima. Neuroevolution is a traditional approach that evolves the neural network topologies and their weights simultaneously [27], [28]. Along with the deeper layers of the neural networks and increasing parameters, evolving the weights becomes prohibitively time-consuming. Hence, population-based NAS merely evolves the neural network topologies but train the neural networks using the back-propagation method.
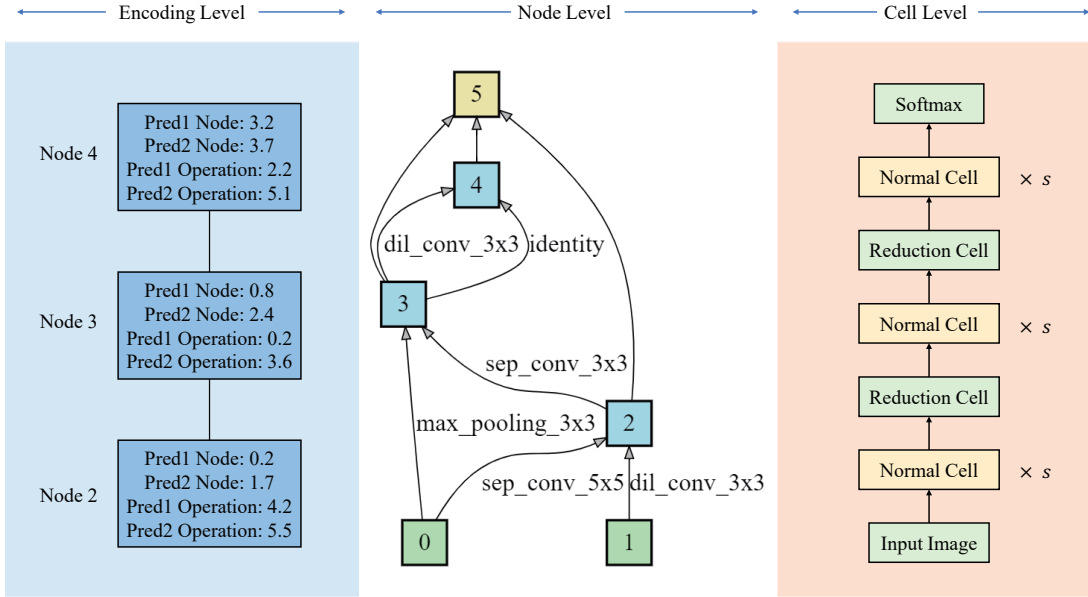
Fig. 3: An example of an encoded vector which maps to the intermediate nodes of the cell-based structure. LEFT: An encoded vector with three blocks. MIDDLE: the corresponding architecture by the encoded vector with three intermediate nodes. RIGHT: the overall cell-based architecture.

Most existing population-based NAS methods have adopted the genetic algorithm or genetic programming [29], [30], [31] to mimic the process of natural evolution, which requires the elaborate designs of stochastic crossover/mutation operators. The research in [23] is a pioneering work in population-based NAS, where 11 mutation operators are designed to modify the attributes of the networks, including altering learning rate, resetting weights, inserting convolution, removing convolution, etc. This research has indicated good feasibility in designing different genetic operators, showing that a neural network could be evolved starting from a very simple form and growing into complex architecture. Afterwards, AmoebaNet [20] discovered an architecture that surpassed the human-craft architectures by using NAS for the first time. In AmoebaNet, a macro architecture is predefined to comprise a number of identical *cells*, such that the search space is reduced to the cell architecture instead of the entire one. On top of such a cell-based framework, two different kinds of mutation operators are designed to change the operation types and the connections among different nodes.

Despite the promising performance of population-based NAS, the existing methods mainly suffer from low computation efficiency. This can be attributed to two factors. First, the stochastic crossover/mutation, as commonly adopted in existing population-based NAS methods, can be inefficient in generating high-performance candidate architectures. Second, the performance evaluations of newly generated candidate architectures can be computationally expensive.

Several methods have been proposed in order to improve the efficiency of the population-based NAS. Lower fidelity estimates are common strategies for speeding up NAS, where the performance is estimated on the basis of lower fidelities of the actual performance, including a reduced number of training epochs, models with smaller sizes, or training on a subset of the datasets. For example, AmoebaNet [20] uses each model trained for 25 epochs with a fewer number of cells. Another strategy to speed up the estimation of architectures is the one-shot approach [32], [33], where only the weights of a single supernet are trained. The searchable architectures can then be evaluated using the weights inherited from the supernet without further training.

C. Motivation

While differential NAS has opened a new dimension in the literature, its development is still in its infancy. One pivotal issue is how to make the best of the continuously encoded search space. Due to the limitations of back-propagation, searching over one solo architecture by gradient can be ineffective due to the lack of proper diversity. By contrast, population-based NAS is intrinsically advanced in maintaining diversity when searching over multiple candidate architectures, but its search efficiency is poor due to the stochastic crossover/mutation and a large number of performance evaluations. Therefore, this work is essentially motivated to design an effective and efficient NAS method, which benefits from the merits of both differentiable NAS and population-based NAS while overcoming their deficiencies.

III. METHODOLOGY

We present the pseudo-code of the proposed slow-fast learning paradigm in Algorithm 1. The proposed RelativeNAS mainly involves three pivotal components: encoding/decoding of the search space (Line 1 and Line 5), performance estimation of the candidate architectures (Line 6 to Line 13), and slow-fast learning among the architecture vectors (Line 14). The computational complexity of performance estimation

strategy mainly depends on the task (*i.e.*, classification, detection, *etc.*) and the scale of dataset. Apart from the performance estimation strategy, another major computational complexity in RelativeNAS is associated with the update of the slow-learner, which is an inevitable operation for population-based NAS. Accordingly, the computational complexity of updating the population is $\mathcal{O}(N * |\boldsymbol{\alpha}|)$, where N is the population size and $|\boldsymbol{\alpha}|$ is the dimensionality of the searched architecture vector. In the following subsections, we will elaborate the three components respectively.

A. Search Space

To limit the scale of search space, this work adopts the cell-based architecture inspired by [19]. Specifically, There are two types of cells: the normal cell $\mathcal{G}^n$ and the reduction cell $\mathcal{G}^r$. The only difference between the two types of cells is the output feature size. The normal cell does not change the size of the feature, while the reduction cell is served as down-sampling to reduce the size of the feature by stride operation. The internal structure of the two types of cells is represented by a directed acyclic graph. As shown in Fig. 3 (middle), there are two input nodes from the two previous cells. Every intermediate node contains two predecessor nodes and applies operations on them. Consequently, the edge in the graph is denoted as the possible operation and information flow between different nodes. Edges are only allowed to point from low indexed nodes to higher ones. On the other hand, a cell only contains one output and all the intermediate nodes are concatenated to the output node. Besides, the reduction cell is connected after s normal cells as shown in Fig. 3 (right). In detail, the cell-based search space can be formulated as:

$$
\begin{aligned}
\mathcal{G}_c = \{ & \mathcal{G}_1^n, ..., \mathcal{G}_s^n, \mathcal{G}_{s+1}^r, \\
& \mathcal{G}_{s+2}^n, ..., \mathcal{G}_{2s+1}^n, \mathcal{G}_{2s+2}^r, \\
& \mathcal{G}_{2s+3}^n, ..., \mathcal{G}_{3s+2}^n \}.
\end{aligned}
$$

Since there are $3s + 2$ different cells in total, the search space can be too large for efficient NAS. To address this issue, all the normal cells are identified, so are the reduction cells. Therefore, the cell-based search space is constrained to a normal cell $\mathcal{G}^n$ and a reduction cell $\mathcal{G}^r$ as below:

$$
\mathcal{G}_c = \{\mathcal{G}^n, \mathcal{G}^r\}.
$$

Specifically, such a cell-based architecture has the following advantages. First, it maintains diversity inside the cells. While all of them share the same macro architecture, the different connections among nodes and operations inside the two types of cells diversify the structure of the network. Second, under the cell-based design, networks can achieve promising performance and high transferability for different tasks by adjusting the total number of cells in the final architecture.

To transform a directed acyclic graph into uniform representation in the continuous space, this work proposes a novel continuous encoding scheme, as illustrated in Table I. Since the input nodes and the output node are fixed, this work only needs to encode every intermediate node along with its two predecessor nodes and the corresponding operations. To be more specific, this work encodes the node and the operation

TABLE I: Illustration of the encoding scheme in RelativeNAS. An operation is determined by the operation type and its kernel size. Each node or operation corresponds to a unique range in real number space.

Node		Operation		
Range	Index	Range	Type	Kernel Size
$[0, 1)$	0	$[0, 1)$	Max Pooling	3
$[1, 2)$	1	$[1, 2)$	Avg Pooling	3
$[2, 3)$	2	$[2, 3)$	Identity	0
$[3, 4)$	3	$[3, 4)$	Sep Conv	3
$[4, 5)$	4	$[4, 5)$	Sep Conv	5
$[5, 6)$	5	$[5, 6)$	Dil Conv	3
$[6, 7)$	6	$[6, 7)$	Dil Conv	5

separately. Each node or operation is represented by a real value interval. Each interval is left-closed and all intervals added together are continuous. To guarantee uniqueness, there is no overlap among different intervals of the nodes or operations. Without bias, each interval of a node or an operation has the same length.

Specifically, there are seven different operations, including 3×3 max pooling, 3×3 average pooling, two depth-wise separable convolutions [34] (Sep Conv 3×3, 5×5), two dilated separable convolutions (Dil Conv 3×3 and 5×5), and identity. Unless specified, each convolutional layer in the network is fronted by ReLU activation and followed by batch normalization [35], and each separable convolution is applied twice.

With the proposed encoding scheme, the cell-based search space $\mathcal{G}_c$ is encoded into a new one $\mathcal{A}$, *i.e.* the encoded search space. In this way, every architecture vector $\boldsymbol{\alpha} \in \mathcal{A}$ can be decoded into its corresponding cell architecture $C_{\boldsymbol{\alpha}} \subset \mathcal{G}_c$ by mapping the vector into the connections and operations of the intermediate nodes in the cell.

An illustrative example of the above encoding process is given in Fig. 3. This work uses a list of blocks to represent an architecture vector as shown in Fig. 3 (left). Each block represents an intermediate node in the cell and needs to be specified by four variables, including Pre1 Node (the first predecessor node) and Pre2 Node (the second predecessor node), and their corresponding operations. Fig. 3 (middle) shows a cell architecture decoded from Fig. 3 (left) using the mapping rules in Table I.

B. Slow-Fast Learning

The general target of NAS is to search for an architecture vector $\boldsymbol{\alpha}^* \in \mathcal{A}$ such that the decoded architecture $C_{\boldsymbol{\alpha}^*}$ minimizes validation loss $\mathcal{L}_{vld}(C_{\boldsymbol{\alpha}^*}, \boldsymbol{\omega}_{\boldsymbol{\alpha}^*}^*)$, where the weights $\boldsymbol{\omega}_{\boldsymbol{\alpha}^*}^*$ associated with the architecture are obtained by minimizing the training loss $\mathcal{L}_{trn}(C_{\boldsymbol{\alpha}^*}, \boldsymbol{\omega}_{\boldsymbol{\alpha}^*})$. To this end, the slow-fast learning paradigm in the proposed RelativeNAS is essentially an optimizer to approximate $\boldsymbol{\alpha}^*$.

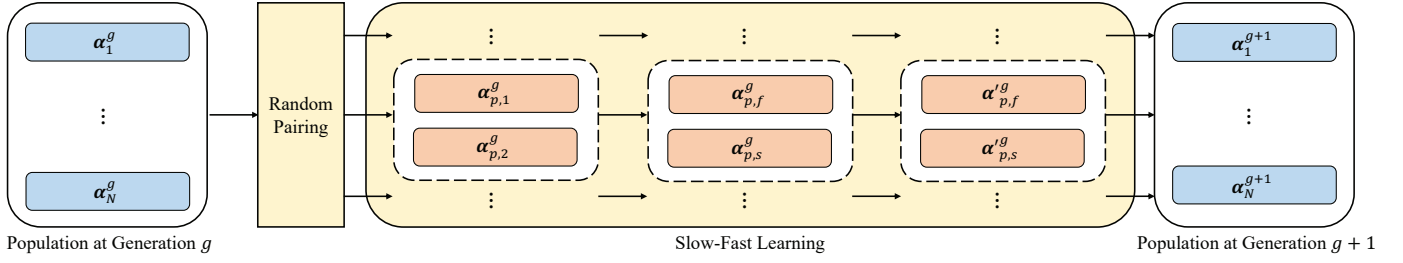Specifically, given an architecture vector $\boldsymbol{\alpha}^g$ obtained at the

Fig. 4: Illustration of the slow-fast learning process at generation g. A population of N architecture vectors is randomly divided into $N/2$ pairs. For each pair p, slow-learner $\boldsymbol{\alpha}_{p,s}^g$ updates its vector by learning from a fast-learner $\boldsymbol{\alpha}_{p,f}^g$ using (2) and (4), while fast-learner $\boldsymbol{\alpha}_{p,f}^g$ itself remains unchanged. After the slow-fast learning process, all fast-learners and updated slow-learners are re-merged to become the new population of the next generation $g+1$.

g-th generation[1] of slow-fast learning, it is iteratively updated by:

$$\boldsymbol{\alpha}^{g+1} = \boldsymbol{\alpha}^g + \Delta\boldsymbol{\alpha}^g, \qquad (2)$$

where $\mathcal{L}_{vld}(C_{\boldsymbol{\alpha}^{g+1}}, \boldsymbol{\omega}_{\boldsymbol{\alpha}^{g+1}}^*) < \mathcal{L}_{vld}(C_{\boldsymbol{\alpha}^g}, \boldsymbol{\omega}_{\boldsymbol{\alpha}^g}^*)$ holds, such that:

$$\lim_{g \to +\infty} \boldsymbol{\alpha}^g = \boldsymbol{\alpha}^*. \qquad (3)$$

To efficiently generate the pseudo gradient $\Delta\boldsymbol{\alpha}^g$, this work proposes to use a population of N architecture vectors $\{\boldsymbol{\alpha}\}_{n=1}^N$. As illustrated in Fig. 4, at each generation, the population is randomly divided into $N/2$ pairs. Then, for each pair p, a fast-learner $\boldsymbol{\alpha}_{p,f}^g$ and a slow-learner $\boldsymbol{\alpha}_{p,s}^g$ are specified by the parital ordering of validation loss values, where the one having smaller loss is the fast-learner and the other is the slow-learner (*i.e.* $\mathcal{L}_{vld}(C_{\boldsymbol{\alpha}_{p,f}^g}, \boldsymbol{\omega}_{\boldsymbol{\alpha}_{p,f}^g}^*) < \mathcal{L}_{vld}(C_{\boldsymbol{\alpha}_{p,s}^g}, \boldsymbol{\omega}_{\boldsymbol{\alpha}_{p,s}^g}^*)$). Then, $\boldsymbol{\alpha}_{p,s}^g$ is updated by learning from $\boldsymbol{\alpha}_{p,f}^g$ with:

$$\Delta\boldsymbol{\alpha}_{p,s}^g = \lambda_1(\boldsymbol{\alpha}_{p,f}^g - \boldsymbol{\alpha}_{p,s}^g) + \lambda_2\Delta\boldsymbol{\alpha}_{p,s}^{g-1}, \qquad (4)$$

where $\lambda_1, \lambda_2 \in [0,1]$ are randomly generated values by uniform distribution. Specifically, λ_1 determines the step size that $\boldsymbol{\alpha}_{p,s}^g$ learns from $\boldsymbol{\alpha}_{p,f}^g$, and λ_2 determines impact of the momentum term $\Delta\boldsymbol{\alpha}_{p,s}^{g-1}$. Such a pseudo gradient is inspired by the second derivatives in the gradient descent of the back propagation [36]. Thanks to such a pseudo-gradient based mechanism, the proposed RelativeNAS is applicable not only to the search space in this work, but also to any other generic continuously encoded search space.

Eventually, all fast-learners and updated slow-learners are re-merged to become the new population of the next generation $g+1$. By such an iterative process of slow-fast learning, each architecture vector in the population is expected to move towards optima by learning from those converging faster than them.

C. Performance Estimation

As described above, the proposed RelativeNAS needs to evaluate the performances of candidate architectures thus decoded from the architecture vectors in the population of each generation, such that for each pair of architecture vectors, fast-learner can be distinguished from slow-learner by their
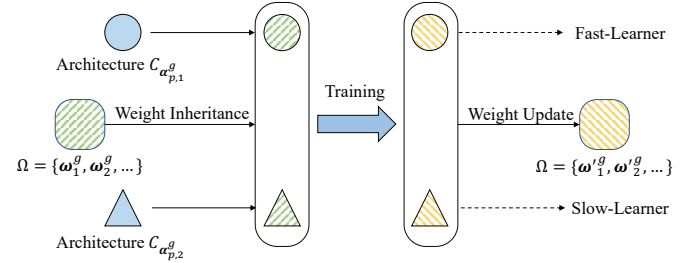
[1]To distinguish the *iteration* in network training, this work uses the term *generation* to denote the iteration in slow-fast learning

Fig. 5: Illustration of the architecture evaluation process. Given two paired architectures $C_{\boldsymbol{\alpha}_{p,1}^g}$ and $C_{\boldsymbol{\alpha}_{p,2}^g}$, they inherit weights from the weight set Ω firstly. After that, we train each network only for one epoch and differ them to fast-learner and slow-learner. Finally, the weight set is updated by all the trained weights $\boldsymbol{\omega}'^g$.

validation losses. Ideally, the performance of a candidate architecture $C_{\boldsymbol{\alpha}^g}$ can by evaluated by solving the following optimization problem:

$$\boldsymbol{\omega}_{\boldsymbol{\alpha}^g}^* = optimize(\boldsymbol{\omega}_{\boldsymbol{\alpha}^g} | C_{\boldsymbol{\alpha}^g})$$
$$= step(step(...step(\boldsymbol{\omega}_{\boldsymbol{\alpha}^g} | C_{\boldsymbol{\alpha}^g})...|C_{\boldsymbol{\alpha}^g})|C_{\boldsymbol{\alpha}^g}), \qquad (5)$$

where $\boldsymbol{\omega}_{\boldsymbol{\alpha}^g}^*$ is the optimal weights of the candidate architecture, and function $step(\cdot)$ is one step[2] of the iterative optimization procedure to update the weights of the neural network. In practice, however, solving such an optimization (*i.e.* training the candidate architecture) can be quite computationally expensive, especially when there are a large number of candidate architectures obtained during the iterative slow-fast learning process. Hence, to reduce the computation cost of performance evaluations in RelativeNAS, this work proposes a performance estimation strategy.

In contrast to existing differentiable NAS methods (*e.g.* DARTS [19]), the validation losses in RelativeNAS are not directly involved in updating the candidate architectures; instead, they are merely used to determine the *partial orders* among each pair of candidate architectures (*i.e.* to distinguish fast-learner and slow-learner). Therefore, in RelativeNAS, it is intuitively feasible to use performance estimations (instead of exact performance evaluations) to obtain the approximate validation losses of the candidate architectures.

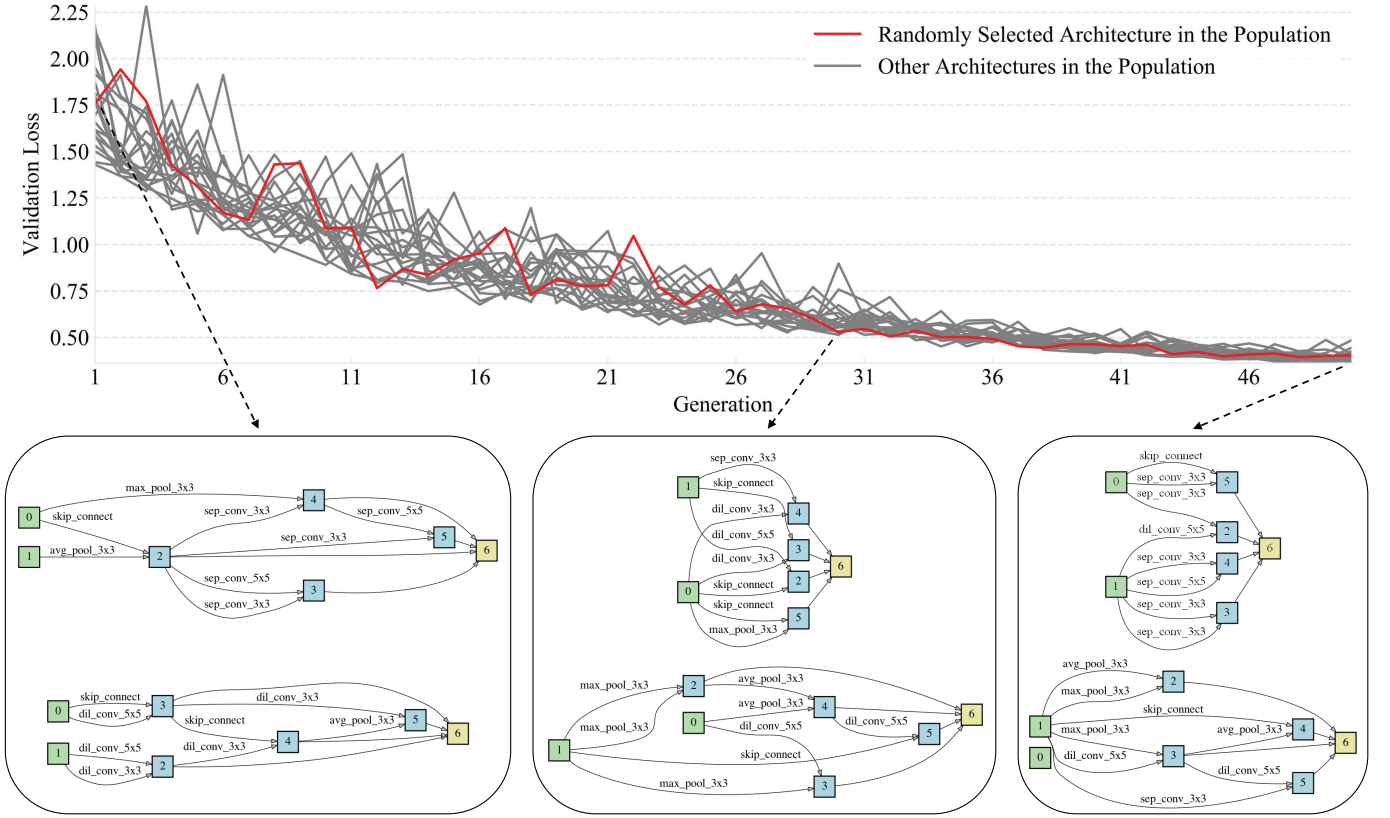[2]In this work, one *step* is specified as one *epoch* in the training process.

Fig. 6: Searching process on CIFAR-10. TOP: trajectories of validation losses using our performance estimation method for all candidate architectures decoded from the architecture vectors in the population. The red line indicates a randomly selected architecture in the population. BOTTOM: architectures of a randomly selected candidate architecture in the population at generation 1, 30, and 50 respectively. Each box contains a normal cell (top) and reduction cell (bottom).

Specifically, the proposed performance estimation strategy first randomly initializes a weight set Ω to contain the weights of all possible operations in $\mathcal{G}_c$. During the search process, given the p-th pair of candidate architecture $\{C_{\boldsymbol{\alpha}_{p,j}^g}\}_{j=1,2}$ at generation g, they inherit the corresponding weights $\boldsymbol{\omega}_{\boldsymbol{\alpha}_{p,1}^g}$ and $\boldsymbol{\omega}_{\boldsymbol{\alpha}_{p,2}^g}$ from Ω according the their own operations respectively. With the inherited weights as warm-up, the weights of $\{C_{\boldsymbol{\alpha}_{p,j}^g}\}_{j=1,2}$ only need to be updated on training set $\mathcal{D}_{trn}$ by one step of optimization:

$$\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,j}^g} = step(\boldsymbol{\omega}_{\boldsymbol{\alpha}_{p,j}^g}|C_{\boldsymbol{\alpha}_{p,j}^g}).$$

Then, the updated weights $\{\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,j}^g}\}_{j=1,2}$ are used to estimate the validation losses $\{\mathcal{L}_{vld}(C_{\boldsymbol{\alpha}_{p,j}^g}, \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,j}^g})\}_{j=1,2}$ of $\{C_{\boldsymbol{\alpha}_{p,j}^g}\}_{j=1,2}$ on validation set $\mathcal{D}_{vld}$. Finally, Ω is updated by $\{\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,j}^g}\}_{j=1,2}$ as:

$$\Omega = \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,f}^g} \cup \{\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,s}^g} - \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,f}^g} \cap \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,s}^g}\} \cup \{\Omega - \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,f}^g} \cup \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,s}^g}\},$$
(6)

where $\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,f}^g}$ and $\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,s}^g}$ are weights from fast-learner and slow-learner (refer to Section III-B), respectively. The first term $\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,f}^g}$ means that Ω receives all the weights from $\omega'^g_{p,f}$ as it is assumed that $\omega'^g_{p,f}$, as the weights of fast-learner, is generally more valuable than $\omega'^g_{p,s}$. The second term $\{\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,s}^g} - \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,f}^g} \cap \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,s}^g}\}$ means that Ω receives the weights in

$\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,s}^g}$ but not in $\boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,f}^g}$. The third term $\{\Omega - \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,f}^g} \cup \boldsymbol{\omega}'_{\boldsymbol{\alpha}_{p,s}^g}\}$ mean that Ω keeps those unused weights unchanged.

With the above procedure as further illustrated in Fig. 5, it is expected that the weight set Ω becomes increasingly knowledgeable by co-evolving with the candidate architectures during the iterative slow-fast learning process, such that the performance estimation strategy is able to save substantial computation costs.

IV. EXPERIMENTS

In this section, we first provide the basic experiment on architecture search using RelativeNAS. Then, we elaborate two analytical experiments to investigate some core properties of RelativeNAS. Afterwards, we present experimental results on CIFAR-10 to evaluate the discovered cell architectures and comparisons with other state-of-the-art networks. Finally, we show the transferability of our discovered cell architectures in both intra- and inter-tasks.

A. Architecture Search

Basically, this work performs NAS on CIFAR-10 [37] which is widely used for benchmarking image classification. Specifically, CIFAR-10 contains 60K images with a spatial solution of 32×32 and these images are equally divided into 10 classes, where the training set and the testing set are 50K
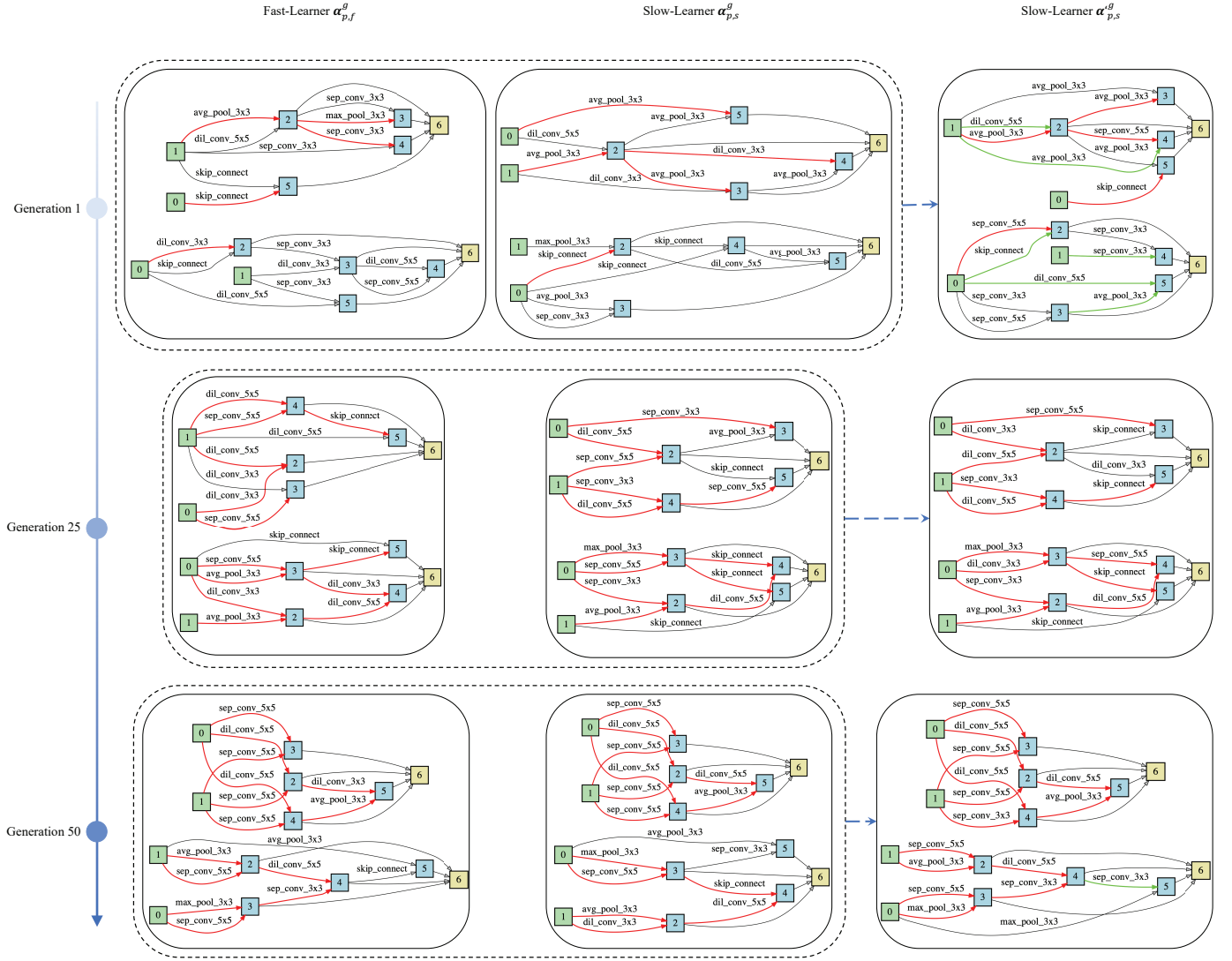
Fig. 7: Illustration of the slow-fast learning with decoded architectures at generation 1, 25, and 50 respectively. Slow-learner updates its connections and operations by learning from fast-learner. The red lines denote the common connections between fast-learner and slow-learner, and the green lines denote the new connections after learning.

and 10K, respectively. Half of the training images of CIFAR-10 are randomly taken out as the search validation set.

In RelativeNAS, the population size N and the generation number are set to 20 and 50, respectively. To evaluate the discovered architectures, every architecture vector α is first decoded into a small network with 8 cells (*i.e.* $s = 2$) and initial channels set to 16. Then, this work trains those networks on the training set for one epoch using SGD with the weight decay and batch size set to 3×10^{-4} and 256 respectively. In addition, the initial learning rate lr is 0.1 which decays to zero following a cosine annealing schedule with T_{max} set to the generation number, and Cutout [38] of length 16 and the path dropout [39] with the probability of 0.3 are both applied for regularization. The trained networks are assessed on the validation set to distinguish fast-learners and slow-learner by comparing the validation losses. All in all, it takes about **nine hours** with **a unique 1080Ti** or **seven hours** with **a Tesla V100** to complete the above search procedure.

B. Population Searching Analysis

Fig. 6 (top) provides the validation losses of all the 20 decoded candidate architectures in a population over the searching process. Initially, the losses differ widely among the candidate architectures due to the randomly initialized architectures; as the search proceeds, the differences of the losses gradually decrease towards a stable value, indicating convergence of the population.

For more insightful observations, this work randomly selects one decoded candidate architecture in the population to trace its architectures obtained over the searching process. As shown in Fig. 6 (bottom), at the initial stage (generation 1) of the searching process, the normal and reduction cells are randomly generated, without any specified property; at the middle stage (generation 30), however, the normal cell becomes denser while the reduction cell remains flat; at the final stage (generation 50), the topologies of the two types of cells remain stable, despite the changes in the detailed

TABLE II: Comparisons with other state-of-the-art methods on CIFAR-10.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	GPU Device	Search Method
FractalNet [40]	5.22	38.6	-	-	manual
Wide-ResNet [41]	4.17	36.5	-	-	manual
DenseNet-BC [9]	3.46	25.6	-	-	manual
PNAS [42]	3.41	3.2	225	-	SMBO
NAONet + Cutout [43]	2.48	10.6	200	V100	NAO
Random search + Cutout [19]	3.29	3.2	4	1080Ti	random
DARTS(first) + Cutout [19]	3.0	3.3	1.5	1080Ti	gradient-based
DARTS(second) + Cutout [19]	2.76	3.3	4.0	1080Ti	gradient-based
SNAS+mild constraint + Cutout [24]	2.98	2.9	1.5	TITAN Xp	gradient-based
SNAS+moderate constraint + Cutout [24]	2.85	2.8	1.5	TITAN Xp	gradient-based
SNAS+aggressive constraint + Cutout [24]	3.10	2.3	1.5	TITAN Xp	gradient-based
ProxylessNAS + Cutout [25]	2.08	5.7	4.0	V100	gradient-based
MetaQNN [44]	6.92	11.8	100	-	RL
NASNet-A + Cutout [45]	2.65	3.41	2000	K40	RL
BlockQNN + Cutout [12]	2.80	39.8	32	1080Ti	RL
ENAS + Cutout [46]	2.89	4.6	0.5	1080Ti	RL
AmoebaNet-A [20]	3.34	3.2	3150	K40	population-based
AmoebaNet-B + Cutout [20]	2.55	2.8	3150	K40	population-based
Large-Scale Evolution [23]	5.4	5.4	2600	-	population-based
Hierarchical Evolution [47]	3.75	15.7	300	P100	population-based
EffPnet [48]	3.49	2.54	<3	RTX 2070	population-based
RelativeNAS + Cutout	**2.34**	**3.93**	**0.4**	**1080Ti**	**population-based**

connections inside them. Such observations indicate that the population searching process would generate expected candidate architectures towards converged optima, in terms of both connections and operations. Indeed, this work further trains the architecture shown at the final stage on the different datasets to validate its performance.

C. Slow-Fast Learning Analysis

To empirically analyze the slow-fast learning process, this work randomly picks up three pairs of architectures obtained at generation 1, generation 25, and generation 50, respectively, to provide the illustrative example in Fig. 7. At generation 1, the architectures of fast-learner and slow-learner both are randomly initialized at the beginning. Hence, there exist substantial differences between fast-learner and slow-learner, such that slow-learner substantially changes its connections as well as operations after learning from fast-learner. At generation 25, there are some common connection patterns between fast-learner and slow-learner, *e.g.* the two predecessor nodes of Node 4 are both Node 1 in the normal cells, and the two predecessor nodes of Node 3 are both Node 0 in the reduction cells. Therefore, slow-learner will not change these patterns during slow-fast learning. By contrast, due to the differences between operations Dil Conv 3×3 and Sep Conv 5×5 in the connection between Node 2 and Node 0 for fast-learner and slow-learner, slow-learner learns from fast-learner and changes to Dil Conv 3×3. At generation 50, the connections of the normal cells are exactly the same, and thus slow-learner only makes some minor adjustments in its operations by learning from fast-learner. Besides, despite that there is still a gap between fast-learner and slow-learner in the reduction cells, while the overall architectures become quite similar after 50 generations of slow-fast learning. Based on the above observations, this work can conclude that the slow-fast learning paradigm is generally effective over the search process, showing different functionalities at different stages.

D. Results on CIFAR-10

In this work, we follow the training settings of DARTS [19] for image classification tasks to train networks from randomly initialized weights, for fair comparisons between the searched architectures. A large network of 20 cells (*i.e. s* is set to 6) is built with the selected normal and reduction cells while the initial number of channels is set to 36. Most hyperparameters are the same as the ones used in the above search process except lr, path dropout, and batch size which are set to 0.025, 0.2, and 128, respectively. For further enhancement, an auxiliary head with weight 0.4 is added into the network. Instead of half training images, this work trains the network from scratch over the whole training set for 600 epochs and evaluate it over the test set.

The results and comparisons with other state-of-the-art networks (including manual and NAS) on CIFAR-10 are summarized in Table II. Besides, we label all the GPU devices for different NAS methods. Compared with the manual networks, our RelativeNAS has fewer parameters while outperforms them by a large margin. The proposed RelativeNAS gains an encouraging improvement to DARTS [19] and a random search baseline [19] in terms of test error and search cost. Compared with other NAS networks, it can be observed that ours needs the least cost on time while gets superior results in terms of test error and parameters. Although ProxylessNAS [25] achieves less test error than ours (2.08% vs 2.34%), it has much more parameters (5.7M vs 3.9M) and costs $10\times$ longer search time than ours (4 vs 0.4). While EffPnet [48] has fewer parameters (2.54M vs 3.9M), the test error and search cost are larger than ours. Furthermore, RelativeNAS is the only one involving the pseudo gradient between architecture vectors among those population-based NAS. To the best of our knowledge, our RelativeNAS is the most efficient search method among those population-based methods. With ENAS [46] and RelativeNAS proposed, RL-based and population-based methods are no longer time-

TABLE III: Comparison with other state-of-the-art networks on CIFAR-100. † denotes directly searching on CIFAR-100, while others are searched on CIFAR-10.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
FractalNet [40]	23.30	38.6	-	manual
Wide-ResNet [41]	20.50	36.5	-	manual
DenseNet-BC [9]	17.18	25.6	-	manual
NAONet + Cutout [43]	15.67	10.8	200	NAO
MetaQNN [44]	27.14	11.8	100	RL
Large-Scale Evolution† [23]	23.0	40.4	2600	population-based
EffPnet [48]	18.49	2.54	<3	population-based
RelativeNAS + Cutout	**15.86**	**3.98**	**0.4**	**population-based**

TABLE IV: Comparison with other state-of-the-art methods on ImageNet. † denotes directly searching over ImageNet, while others are searched on CIFAR-10.

Architecture	Test Error (%)		Params (M)	× + (M)	Search Cost (GPU days)	Search Method
	top-1	top-5				
Inception-V1 [7]	30.2	10.1	6.6	1448	-	manual
MobileNet-V1 (1x) [49]	29.4	10.5	4.2	575	-	manual
MobileNet-V2 (1.4) [50]	25.3	-	6.9	585	-	manual
ShuffleNet-V1 (2x) [51]	26.4	10.2	5	524	-	manual
ShuffleNet-V2 (2x) [52]	25.1	-	5	591	-	manual
PNAS [42]	25.8	8.1	5.1	588	225	SMBO
NAONet [43]	25.7	8.2	11.35	584	200	NAO
DARTS (second order) [19]	26.7	8.7	4.7	574	4.0	gradient-based
SNAS (mild constraint) [24]	27.3	9.2	4.3	533	1.5	gradient-based
ProxylessNAS (GPU)† [25]	24.9	7.5	7.1	465	8.3	gradient-based
NASNet-A [45]	26.0	8.4	5.3	564	1800	RL
NASNet-B [45]	27.2	8.7	5.3	488	1800	RL
NASNet-C [45]	27.5	9.0	4.9	558	1800	RL
AmoebaNet-A [20]	25.5	8.0	5.1	555	3150	population-based
AmoebaNet-B [20]	26.0	8.5	5.3	555	3150	population-based
AmoebaNet-C [20]	24.3	7.6	6.4	570	3150	population-based
EffPnet [48]	27.01	9.25	2.54	-	<3	population-based
RelativeNAS	**24.88**	**7.7**	**5.05**	**563**	**0.4**	**population-based**

consuming and even $2\times$ faster than gradient-based methods. Moreover, the proposed RelativeNAS outperforms ENAS in all aspects.

E. Transferability Analyses

In this subsection, we will validate the transferability of the normal and reduction cells discovered by the proposed RelativeNAS on CIFAR-10. We first validate their generality in other image classification tasks (*i.e.* intra-tasks), and then demonstrate their transferability in inter-tasks, including object detection, semantic segmentation, and keypoint detection.

1) Intra-task Transferability: The discovered normal cell and reduction cell on CIFAR-10 both are directly transferred to CIFAR-100 and ImageNet without further search. Since the architecture is transferred, the overall search cost is the same as on CIRAR-10.

CIFAR-100. CIFAR-100 contains 60K images with a spatial resolution of 32×32, where 50K images are used as the training set and the left 10K images are used as the testing set. Moreover, these images are distributed equally for 100 classes. The network used in CIFAR-10 is directly transferred to CIFAR-100 with a small modification in the last classification layer to adapt to the different number of classes. The training details are the same as CIFAR-10 except the weight decay and batch size which is set to 5×10^{-4} and 96, respectively.

Table III shows the experimental results and comparisons with other state-of-the-art networks. Surprisingly, our direct transferred network achieves a test rate of 15.86% and still outperforms most networks. In particular, RelativeNAS outperforms Large-Scale Evolution [23] by about 7 points which is searching on CIFAR-100 instead of transferring from CIFAR-10. It can be concluded that our RelativeNAS derived from CIFAR-10 is indeed transferable to a more complicated task (*i.e.* CIFAR-100) while maintains its superiority.

ImageNet. ImageNet 2012 dataset [53] is one of the most challenging benchmarks for image classification, which is consisted of 1.28M and 50K images for training and validation, respectively. Those images are unevenly distributed in the 1000 different classes, and they do not have unified spatial resolution but usually much larger than 32×32. In order to fit such a difficult dataset, this work follows the common practice [19] to modify the network structure used in CIFAR-10/100. To be more concrete, the macro-architecture starts with three convolutional layers with stride set to 2, which can reduce the spatial resolution of input images 8 times. In the following, 14 cells (*i.e.* s is set to 4) are stacked. With the consideration of the mobile setting (*i.e.* the number of multiply-add operations should be less than 600M), the initial channel is set to 46. This work trains the model over the train set while reporting the results on the validation set. During training, this work adopts some common data augmentation strategies, including randomly resize and crop, random horizon flip, and color jitter. There are 1024 examples in each training batch and the size of each image is equal to 224×224. The

model is optimized with SGD for 250 epochs, where the initial learning rate, momentum, and weight decay are set to 0.5, 0.9, and 3×10^{-5}, respectively. This work applies the warm-up strategy over the first 5 epochs, where the learning rate is gradually increasing linearly from 0 to the initial value. During the left 245 epochs, the learning rate decays linearly from 0.5 to 1×10^{-5}. In addition, this work also uses the label smoothing [54] to regularize our model, and the smoothing parameter ϵ is equal to 0.1.

The results of RelativeNAS compared with other state-of-the-art networks on the ImageNet are presented in Table IV. It is worth noticing that RelativeNAS achieves competitive performance, *i.e.* top-1 and top-5 test error rate of 24.88% and 7.7%, respectively. Interestingly, our transferred RelativeNAS performs a little better than ProxylessNAS (GPU) [25] which is searching on ImageNet directly. The results further demonstrate that the proposed method enables the transformation of simple cells to complex macro architectures for solving more complicated tasks with low cost but high performance.

2) Inter-task Transferability: We will further demonstrate the transferability by transferring our network pretrained on ImageNet to other tasks instead of image classification. To be more specific, we will train and evaluate SSD [55], BiSeNet [6], and SimpleBaseline [56] with different mobile-setting backbones under the same training settings for object detection, semantic segmentation, and keypoint detection, respectively. We note that all compared models (network structures as well as the ImageNet pretrained weights) in this part are from PyTorch repository except DARTS [19], which is from the official released GitHub repository.

Object Detection. For object detection, this work compares our network with other counterparts on PASCAL VOC, in which thousands of images over 20 object classes are annotated with bounding boxes. Among those object classes, bottles and plants are both small objects. Following [50], this work adopts the SSDLite as our object detection framework, which is a mobile-friendly variant of Single Shot Detector (SSD) [55]. Specifically, all the regular convolutions are replaced with separable convolutions in SSD extra layers and prediction layers, with which SSDLite is slighter and more efficient than the original SSD. This work trains all models over the combined trainval sets of VOC 2007 and 2012 using SGD with a batch size of 32, the momentum of 0.9, and weight decay of 5×10^{-4}. Besides, input images are resized to 320×320 and the learning rate is set to 0.01 which will decay to zero in 200 epochs with cosine annealing scheduler without a restart. Table V presents the performance achieved by those models on PASCAL VOC 2007 test set. This work can conclude that our RelativeNAS achieves the best performance while keeps comparable in terms of parameters and FLOPs under the same settings. Moreover, the discovered model outperforms others in small objects by a large margin, which can be attributed to the fact that our model has a strong ability to retain spatial details while extracting abstract semantic information. In addition to the quantitative comparison, this work also provides some qualitative results in Fig. 8. From it, this work can see that our model indeed surpasses others in detecting the bottle (first row)

and bird (second row). Furthermore, it seems our model can well exploit the surrounding context to improve performance, as it can identify this is a boat instead of a bird in the last row.

Semantic Segmentation. Cityscapes [58] is a large-scale dataset containing pixel-level annotations of 5000 images (2975, 500, and 1525 for the training, validation, and test sets respectively) and about 20000 coarsely annotated images. Following the evaluation protocol [58], 19 semantic labels are used for evaluation without considering the void label. This work evaluates the BiSeNet [6] with different mobile-setting backbones on the validation set when training with only 2975 images (*i.e.* train fine set). All models are trained for 80K iterations with the initial learning rate and batch size set to 1×10^{-2} and 16, respectively. Similar to [6], this work decays the lr with the "poly" learning rate strategy. To be more concrete, the initial lr is multiplied by $(1 - \frac{iter}{max\_iter})^{0.9}$. This work follows the BiSeNet to augment our training images. Specifically, this work employs the color jitter, random scale ($scales = \{0.75, 1, 1.25, 1.5, 1.75, 2.0\}$), and random horizontal flip. After that, this work randomly crops the augmented images into a fixed size that is 1024×1024 for training. Note that, the multi-crop testing is adopted during the test phase, and the test crop size is equal to 1024×1024, too. Table VI provides the comparison with several representative mobile-setting backbones on the Cityscapes val set in terms of the parameter, computation complexity (*i.e.* FLOPs) and mIoU. It can be seen that our RelativeNAS has fewer parameters and FLOPs than the NASNet while outperforming NASNet by 0.8 points in terms of mIoU. Furthermore, when compared with BiSeNet that adopts ResNet101 as the backbone [6], our RelativeNAS achieves a better result over val set (80.4 vs 80.3) when adopting the same multi scales ($scales = \{0.5, 0.75, 1, 1.25, 1.5, 1.75\}$) as well as flipping during inference. Some visual examples are displayed in Fig. 9, where it can be seen that the BiSeNet paired with our RelativeNAS can better segment the boundaries of objects.

Keypoint Detection. Keypoint detection aims to detect the locations of k human parts (*e.g.*, ankle, shoulder, etc) from an image. The MSCOCO [59] is a widely used benchmark dataset for keypoint detection which includes over 250k person instances labelled with 17 keypoints. SimpleBaseline [56] is adopted as our general keypoint detection framework, and this work assesses it when paired with different backbones. This work trains all models on the MSCOCO train2017 set and evaluate them on the val2017 set, containing 57K and 5K images, respectively. Following the SimpleBaseline [56], this work crops the human detection boxes from the images which are then resized to 256×192. In addition, the random rotation, scale, and flipping are all applied to data augmentation. Each model is trained with the Adam optimizer [60] for 140 epochs and the initial learning rate is set to 1×10^{-3} which will drop to 1×10^{-4} and 1×10^{-5} at the 90th and 120th epoch, respectively. Moreover, each batch contains 128 examples. Similar as [56], a two-stage top-down paradigm is adopted during inference. To be more specific, an independent person detector is applied to detect the person instances and then those instances are input to the trained keypoint detector for

TABLE V: Results of SSDLite [50] with different mobile-setting backbones on PASCAL VOC 2007 test set.

Backbone	# Params (M)	# FLOPs (M)	Small Objets (AP (%))		mAP (%)
			Bottle	Plant	
ShuffleNet-V2 (1x) [52]	2.17	355.76	29.9	38.1	65.4
MobileNet-V2 (1x) [50]	3.30	680.88	37.9	43.9	69.4
NASNet [45]	5.22	1238.92	41.5	46.1	71.6
MnasNet [57]	4.18	708.72	37.7	44.4	69.6
DARTS [19]	4.73	1138.16	38.3	49.3	71.2
RelativeNAS	**5.07**	**1202.97**	**45.9**	**50.3**	**73.1**

Fig. 8: Visual examples achieved by SSDLite with different backbones. From left to right are groudtruth, ShuffleNetV2, MobileNetV2, NASNet, MnasNet, DARTS, and RelativeNAS, respectively. The confidence threshold is 0.5. Different colors represent different classes.
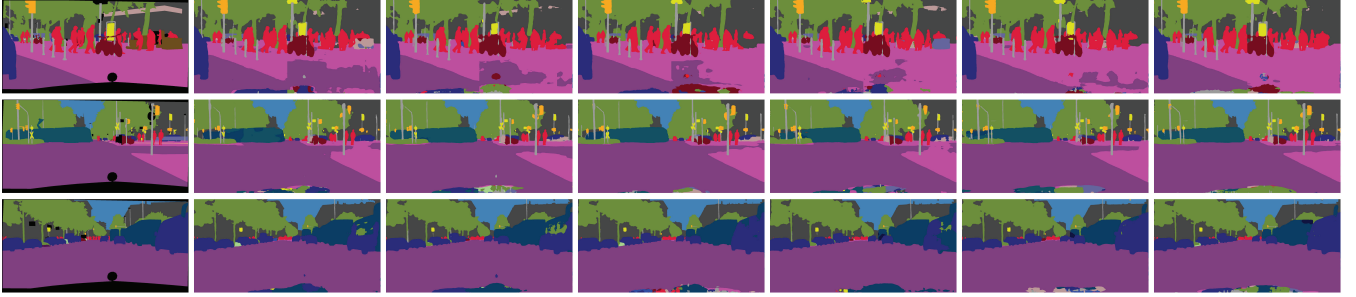
Fig. 9: Visual examples achieved by BiSeNet with different backbones. From left to right are ground truth, ShuffleNetV2, MobileNetV2, NASNet, MnasNet, DARTS, and RelativeNAS, respectively. Different colors denote different classes.
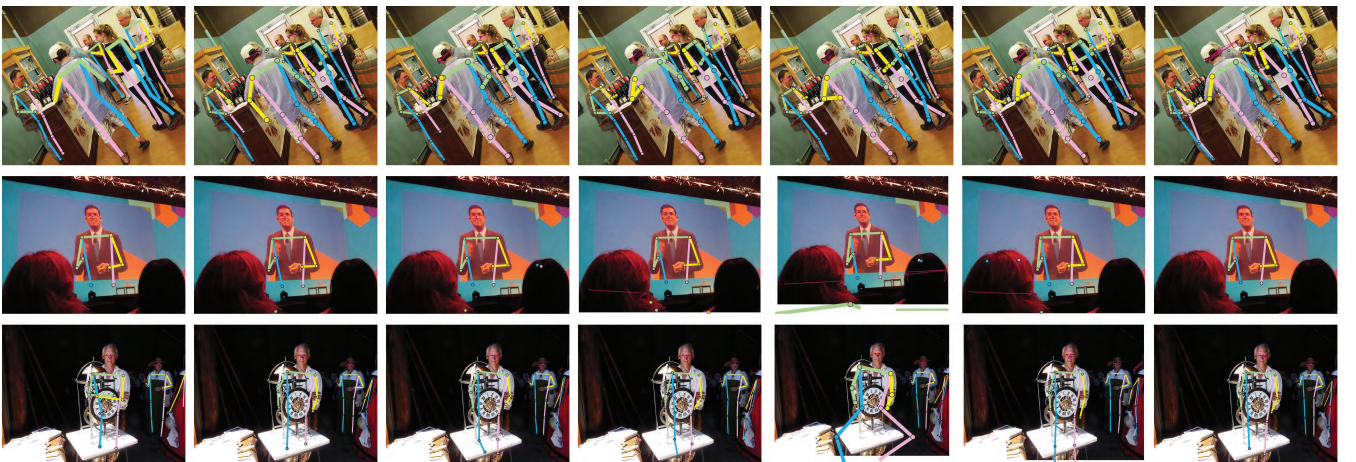

Fig. 10: Visual examples achieved by SimpleBaseline with different backbones. From left to right are ground truth, ShuffleNetV2, MobileNetV2, NASNet, MnasNet, DARTS, and RelativeNAS, respectively. Different colors represent different keypoints.

TABLE VI: Results of BiSeNet [6] with different mobile-setting backbones on Cityscapes val set. (single scale and no flipping).

Backbone	# Params (M)	# FLOPs (G)	mIoU (%)
ShuffleNet-V2 (1x) [52]	4.10	26.30	73.0
MobileNet-V2 (1x) [50]	5.24	29.21	77.1
NASNet [45]	7.46	36.51	77.9
MnasNet [57]	6.12	29.50	76.8
DARTS [19]	6.64	34.77	77.5
RelativeNAS	**6.94**	**35.35**	**78.7**

predicting human keypoints. This work reports the experimental results of SimpleBaseline with different mobile-setting backbones in Table VII. Our RelativeNAS still performs better than others in terms of AP while is comparable in the other two aspects. Moreover, our claims are supported by visual examples in Fig. 10.

TABLE VII: Results of SimpleBaseline [56] with different mobile-setting backbones on MS COCO2017 val set. Flip is used during validation.

Backbone	# Params (M)	# FLOPs (M)	AP (%)
ShuffleNet-V2 (1x) [52]	7.55	154.37	60.4
MobileNet-V2 (1x) [50]	9.57	306.80	649
NASNet [45]	10.66	569.11	67.9
MnasNet [57]	10.45	320.17	62.5
DARTS [19]	9.20	531.77	66.9
RelativeNAS	**9.43**	**564.19**	**68.3**

V. Conclusions

This paper has presented a framework, called RelativeNAS, for the effective and efficient automatic design of high-performance networks. Within RelativeNAS, a novel continuous encoding scheme for cell-based search space has been proposed firstly. To further utilize the continuously encoded search space, a slow-fast learning paradigm has been applied as an optimizer to iteratively update the architecture vectors. In contrast to existing learning/optimization methods in NAS, the proposed one does not directly use *loss-based knowledge* to update the architectures. Instead, the candidate architectures are made to learning from each other by the pariwisely generated pseudo-gradients, *i.e.* slow-learner learning from fast-learner in each pair of candidate architectures. In addition, a performance estimation strategy has been proposed to reduce the cost of evaluating candidate architectures. The effectiveness of such a strategy can be largely attributed to the fact that the validation loss is merely used for distinguishing slow-learner and fast-learner by partial ordering, which only requires estimated (instead of exact) loss values.

With the proposed RelativeNAS as above, consequently, it takes about nine 1080Ti GPU hours (*i.e.* 0.4 GPU Day) for our RelativeNAS to search on CIFAR-10. Furthermore, our discovered network has been able to outperform or match other state-of-the-art manual and NAS networks on CIFAR-10 while showing promising transferability in other intra- and inter-tasks, such as ImageNet, object detection. In particular, our transferred network has yielded the best performance on PASCAL VOC, Cityscapes, and MS COCO. In conclusion, this work highlights the merits of differentiable NAS and

combining population-based NAS, to be more effective and more efficient. Moreover, the proposed slow-fast learning paradigm can be also potentially applicable to other generic learning/optimization tasks.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[2] Y. Kong, X. Kong, C. He, C. Liu, L. Wang, L. Su, J. Gao, Q. Guo, and R. Cheng, "Constructing an automatic diagnosis and severity-classification model for acromegaly using facial photographs by deep learning," *Journal of Hematology & Oncology*, vol. 13, no. 1, pp. 1–4, 2020.

[3] Z. Lu, K. Deb, and V. N. Boddeti, "MUXConv: Information multiplexing in convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[5] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[6] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "BiSeNet: Bilateral segmentation network for real-time semantic segmentation," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 325–341.

[7] 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.

[8] 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.

[9] 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.

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

[11] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "Multiobjective evolutionary design of deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 2, pp. 277–291, 2021.

[12] 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.

[13] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural architecture transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[14] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "NAS-FPN: Learning scalable feature pyramid architecture for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7036–7045.

[15] H. Xu, L. Yao, W. Zhang, X. Liang, and Z. Li, "Auto-FPN: Automatic network architecture adaptation for object detection beyond classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6649–6658.

[16] 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.

[17] W. Chen, X. Gong, X. Liu, Q. Zhang, Y. Li, and Z. Wang, "FasterSeg: Searching for faster real-time semantic segmentation," in *Proceedings of the International Conference on Learning Representations*, 2020.

[18] X. Gong, S. Chang, Y. Jiang, and Z. Wang, "AutoGAN: Neural architecture search for generative adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3224–3234.

[19] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proceedings of the International Conference on Learning Representations*, 2019.

[20] 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, no. 01, 2019, pp. 4780–4789.

[21] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[22] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6202–6211.

[23] 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 International Conference on Machine Learning*, 2017, pp. 2902–2911.

[24] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *Proceedings of the International Conference on Learning Representations*, 2019.

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

[26] H. Liang, S. Zhang, J. Sun, X. He, W. Huang, K. Zhuang, and Z. Li, "DARTS+: Improved differentiable architecture search with early stopping," *arXiv preprint arXiv:1909.06035*, 2019.

[27] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[28] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[29] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 497–504.

[30] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-Net: neural architecture search using multiobjective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 419–427.

[31] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search," in *European Conference on Computer Vision*, 2020.

[32] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: one-shot model architecture search through hypernetworks," in *Proceedings of the International Conference on Learning Representations*, 2018.

[33] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proceedings of the International Conference on Machine Learning*, 2018, pp. 549–558.

[34] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1251–1258.

[35] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning*, vol. 37, 2015, pp. 448–456.

[36] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[37] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Tech Report*, 2009.

[38] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[39] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Dropblock: A regularization method for convolutional networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2018, pp. 10 727–10 737.

[40] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-deep neural networks without residuals," in *Proceedings of the International Conference on Learning Representations*, 2017.

[41] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *In Proceedings of the British Machine Vision Conference (BMVC)*, 2016.

[42] 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*, 2018, pp. 19–34.

[43] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Proceedings of the Advances in Neural Information Processing Systems*, 2018, pp. 7816–7827.

[44] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proceedings of the International Conference on Learning Representations*, 2017.

[45] 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.

[46] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proceedings of the International Conference on Machine Learning*, vol. 80, 2018, pp. 4095–4104.

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

[48] B. Wang, B. Xue, and M. Zhang, "Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2021.

[49] 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.

[50] 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.

[51] 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.

[52] 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*, 2018, pp. 116–131.

[53] 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 the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[54] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[55] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision*, 2016, pp. 21–37.

[56] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 466–481.

[57] 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.

[58] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

[59] 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 *Proceedings of the European Conference on Computer Vision*, 2014, pp. 740–755.

[60] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2015.

Changxiao Qiu received the B.S. degree in 2012, and the M.S. degree in 2015 from the University of Electronic Science and Technology of China(UESTC), Chengdu, China. He is currently a Senior Engineer with the Hisilicon Research Department, Huawei Technologies Co., Ltd., Shenzhen, China. His research interests include deep learning and computer vision.

Hao Tan received the B.Eng. degree in computer science and technology from the Southern University of Science and Technology, Shenzhen, China, in 2020. He is currently a Research Assistant with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. His current research interests include computer vision, neural architecture search, and swarm intelligence.

Ran Cheng (Senior Member, IEEE) received the B.Sc. degree in computer science and technology from the Northeastern University, Shenyang, China, in 2010, and the Ph.D. degree in computer science from the University of Surrey, Guildford, U.K., in 2016. He is currently an Associate Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. Dr. Cheng was a recipient of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award in 2018 and 2021, the IEEE Computational Intelligence Society Outstanding Ph.D. Dissertation Award in 2019, and the IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE Outstanding Paper Award in 2020. He is an Associate Editor of the IEEE TRANSACTIONS ON ARTIFICIAL INTELLIGENCE.

Fan Yang received the electronic bachelor and master degree from the Paris-Sud University (University of Paris XI), 91400 Orsay, France, and the Ph.D. degree in informatics from the Paris-Saclay University, 91400 Orsay, France, in 2015. He is currently a Principal Engineer and Project Manager with the Hisilicon Research Department, Huawei Technologies Co., Ltd., Shenzhen, China. His current research interests include neural network compression and acceleration.

Shihua Huang received the B.Eng. degree from Northeastern University, Shenyang, China, in 2018. He is currently a Research Assistant with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. His current research interests include representation learning, multiobjective optimization, and their applications.

Ping Luo is an Assistant Professor in the department of computer science, The University of Hong Kong (HKU). He received his PhD degree in 2014 from Information Engineering, the Chinese University of Hong Kong (CUHK), supervised by Prof. Xiaoou Tang and Prof. Xiaogang Wang. He was a Postdoctoral Fellow in CUHK from 2014 to 2016. He joined SenseTime Research as a Principal Research Scientist from 2017 to 2018. His research interests are machine learning and computer vision. He has published 100+ peer-reviewed articles in top-tier conferences and journals such as TPAMI, IJCV, ICML, ICLR, CVPR, and NIPS. His work has high impact with 13000 citations according to Google Scholar. He has won a number of competitions and awards such as the first runner up in 2014 ImageNet ILSVRC Challenge, the first place in 2017 DAVIS Challenge on Video Object Segmentation, Gold medal in 2017 Youtube 8M Video Classification Challenge, the first place in 2018 Drivable Area Segmentation Challenge for Autonomous Driving, 2011 HK PhD Fellow Award, and 2013 Microsoft Research Fellow Award (ten PhDs in Asia).

Cheng He (Member, IEEE) received the B.Eng. degree from the Wuhan University of Science and Technology, Wuhan, China, in 2012, and the Ph.D. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2018. He is currently a Research Assistant Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. His current research interests include model-based evolutionary algorithms, multiobjective optimization, la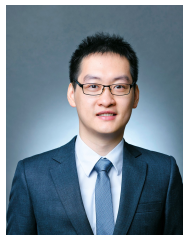rge-scale optimization, deep learning, and their applications. He is a recipient of the SUSTech Presidential Outstanding Postdoctoral Award from Southern University of Science and Technology, and the leading guest editor for "SPECIAL ISSUE: Emerging Topics in Evolutionary Multiobjective Optimization" of the Complex & Intelligent Systems.