# ModuleNet: Knowledge-Inherited Neural Architecture Search

Yaran Chen, *Member, IEEE*, Ruiyuan Gao, Fenggang Liu, and Dongbin Zhao, *Fellow, IEEE*

***Abstract*—Although neural the architecture search (NAS) can bring improvement to deep models, it always neglects precious knowledge of existing models. The computation and time costing property in NAS also means that we should not start from scratch to search, but make every attempt to reuse the existing knowledge. In this article, we discuss what kind of knowledge in a model can and should be used for a new architecture design. Then, we propose a new NAS algorithm, namely, ModuleNet, which can fully inherit knowledge from the existing convolutional neural networks. To make full use of the existing models, we decompose existing models into different *module*s, which also keep their weights, consisting of a knowledge base. Then, we sample and search for a new architecture according to the knowledge base. Unlike previous search algorithms, and benefiting from inherited knowledge, our method is able to directly search for architectures in the macrospace by the NSGA-II algorithm without tuning parameters in these *module*s. Experiments show that our strategy can efficiently evaluate the performance of a new architecture even without tuning weights in convolutional layers. With the help of knowledge we inherited, our search results can always achieve better performance on various datasets (CIFAR10, CIFAR100, and ImageNet) over original architectures.**

***Index Terms*—Evaluation algorithm, knowledge inherited, neural architecture search (NAS).**

## I. Introduction

DUE TO the powerful capabilities of representation learning and data modeling, deep learning has widely been applied in many fields [1], and achieved state-of-the-art performances in many tasks, even outperforming human, such as image classification [2]–[5], object detection [6]–[8], and natural language processing. However, these impressive successes usually are at the cost of human experts' huge efforts and many attempts in the exquisite design of the network architecture and tedious training techniques. These days, the manual design has hardly satisfied the increasing needs of various applications. Therefore, it is urgent and promising to automatically design an excellent network for a given task.

Auto machine learning (AutoML) provides a new paradigm to design a new neural architecture [9], [10]. Neural architecture search (NAS) is a promising technique to implement automatic design and search of neural architectures, which has achieved remarkable performances, even surpassing hand-designed models in some tasks. NAS usually uses a search algorithm to sample a candidate model architecture, such as a child model. The candidate model needs to be trained to convergence for evaluating its performance that is used as feedback to guide the controller to find more promising model architectures at the next step. In addition, sampling and training candidates are needed to be repeated many times, leading to time-consuming and high computational costs. Zoph and Le [11] first used a recurrent neural network as the controller to search the entire chain structure and found a competitive architecture using 800 GPU for about 30 days. To improve this, Zoph *et al.* [12] searched a cell structure first and then stacked it to generate the entire architecture while it also costs 450 GPUs for 3–4 days. After that, based on parameter sharing, ENAS [13] is proposed to find an architecture that outperforms the human-designed ones only using 0.5 GPU days. The one-shot paradigm is proposed, such as FairNAS [14] and Single-path [15], [16], which build a supernet, share weights among child architectures in the supernet, and find excellent architectures within a few GPU days. In these parameter sharing methods, a supernet is built and all the candidate architectures inherit weights from the supernet directly. The supernet is initialized with random weights and trained from scratch. However, training supernet to convergence is difficult with high costs. In fact, various deep neural networks have been designed by experts for different tasks and trained on datasets in recent years, in which there are at least two aspects of knowledge: 1) architecture and 2) trained parameters. The knowledge ignored by recent NAS methods can be used extensively, thereby reducing the search cost.

*First, Architecture:* The convolutional neural network (CNN), a well-known deep learning architecture, contains precious knowledge, which reflects scientists' comprehension of CNN and inspirations from that. In the early years, inspired by the visual perception mechanism of the creatures, Kunihiko proposes the predecessor of CNN: Neocognition [17] in 1970,

Manuscript received 15 October 2020; revised 31 January 2021; accepted 30 April 2021. Date of publication 7 June 2021; date of current version 17 October 2022. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62006226; in part by the Youth Research Fund of the State Key Laboratory of Management and Control for Complex Systems under Grant 20190213; and in part by the Program of the Huawei Technologies Company Ltd. under Grant FA2018111061SOW12. This article was recommended by Associate Editor L. Rutkowski. *(Yaran Chen and Ruiyuan Gao contributed equally to this work.) (Corresponding author: Dongbin Zhao.)*

Yaran Chen and Dongbin Zhao are with The State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with the College of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: chenyaran2013@ia.ac.cn; dongbin.zhao@ia.ac.cn).

Ruiyuan Gao is with the the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong (e-mail: rygao@cse.cuhk.edu.hk).

Fenggang Liu is with the College of Automation, Beijing Institute of Technology, Beijing 100811, China (e-mail: liufgtech@bit.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCYB.2021.3078573.

Digital Object Identifier 10.1109/TCYB.2021.3078573

2168-2267 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

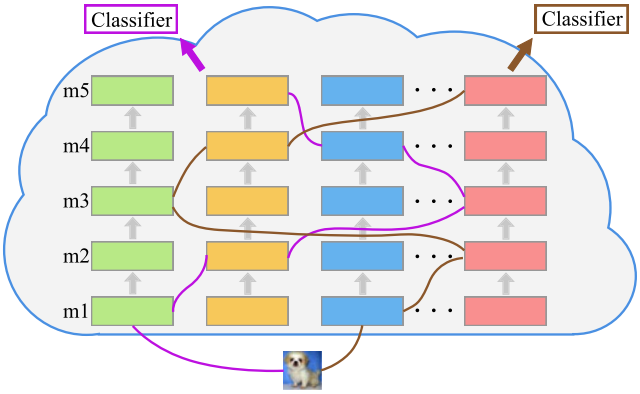


Fig. 1. Example of architecture generation of ModuleNet. We decompose some existing architectures (shown with the same color) into five cells and keep their parameters to form *module*s ("m" in the figure). Arrows in gray show original inference paths. Purple lines and brown lines separately show two possible architectures.

then LeCun establishes the modern architecture of CNN in 1997 [18], and later designs LeNet-5 [19] for the digit classification task. LeNet-5 uses backpropagation to train and learn a representation of original images, but not effective for complex tasks. Krizhevsky expands the architecture depth for a complex task, designing AlexNet [2] and achieving the best performance in ILSVRC2012. Experts study that filter size should be smaller [20] and the architecture depth needs to be expanded [4], [21]. The success of VGG [4] confirms the significance of depth in visual representations and while the architecture depth increases, gradient diffusion and overfit occur. Kaiming He's introduction of shortcut connection in ResNet [3] saves CNNs from the degradation problem when going deeper. GoogleNet [21], UNet [22], and FPN [23] separately show great importance of features in multiscale and multiresolution. All of this expert knowledge has great potential for rediscovery and reorganization. However, current AutoML methods constrain themselves in searching from scratch, turning a blind eye to this knowledge.

*Second, Trained Parameters:* By training on a given dataset, CNN can learn and distill knowledge contained implicitly inside massive data. For one thing, transfer learning through weight-sharing is widely accepted in various computer vision tasks, such as backbones in object detection models [6], [24]. Besides, weight-sharing is used as a basic method in NAS after [13]. Therefore, trained weights have great *transferability*. For the other, within a specific architecture, different trained parameters can extract features from different aspects. Since different features are clearly helpful to separability among inputs, parameters are important for *module*s to possess *diversity*. *Transferability* makes trained parameters usable for reorganization. *Diversity* can introduce more knowledge into consideration when searching. Therefore, trained parameters in *module*s are helpful for NAS.

From the evolutionary algorithm (EA) [25], [26] to reinforcement learning (RL) [12], [13], [27], [28] and gradient-based methods [29], scientists overrate optimization in the scenario of starting from the very stage to search, but overlook precious knowledge in the existing architecture and trained parameters. Actually, we should make progress by "standing on the shoulders of giants."

Therefore, we propose a new NAS algorithm, namely, ModuleNet, to solve the problems above. As shown in Fig. 1, we build a knowledge base for the existing architectures with their trained parameters. By searching over different *module*s for the entire architecture, ModuleNet can inherit all knowledge from the knowledge base. Specifically, we first acquire the knowledge base by decomposing various architectures with their trained weights into different *module*s to keep their integrity. Then, we iteratively search for some best architectures according to the knowledge base using nondominated sorting genetic algorithm II (NSGA-II) algorithm [30], without tuning parameters in convolutional layers. In each iteration, new architectures will be generated by reorganizing *module*s, which keep their weights as the origin to inherit from the knowledge base. In this way, we can make full use of the existing architecture and trained parameters, and rediscover and reorganize them for better results. In ModuleNet, the knowledge base, regarded as the supernet in NAS, inherits these weights from existing architectures, which avoids the burden of training the supernet. In our experiments, the effectiveness of ModuleNet is verified on various visual datasets, showing improvements over the original architectures it inherits.

To summarize, our contributions in this article are mainly as follows.

1) We propose a modular network architecture search scheme that effectively exploits the empirical knowledge and data knowledge of the existing artificially designed models. Systematic experiments show that the proposed method is always able to search for better network architectures than existing models by inheriting *module*s of existing different models.
2) We propose an ingenious way to connect different modules with different channels and sizes in various existing networks. In addition, the proposed *module* connection does not contain trainable parameters, which significantly reduces the computational consumption of search.
3) We propose a performance evaluation method to search for differentiated network architectures to avoid falling into existing and pretrained network architectures during the search and invent equal opportunity among these existing architectures and new reassembled architectures, which consider loss changing rate, error rate, and architecture similarity.

The remainder of this article is organized as follows. In Section II, we give a review on the architecture design and EA for NAS. Section III introduces the proposed approaches, including the overview for ModuleNet, knowledge base, encoding, and the performance evaluation. Section IV presents our experiments and results in CIFAR10, CIFAR100, and ImageNet. Finally, we conclude the study in Section V.

## II. Related Work

### A. CNN Architecture Design

From the very step of the CNN architecture design, scientists use trial-and-error to discover better architecture for

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

target tasks. In this stage, though laborious, various successful contributions are made, such as VGG [4], ResNet [3], GoogleNet [21], and EfficientNet [31]. Besides, new operators and principles are also introduced for different targets. For example, batch normalization [32] helps us to solve the internal covariate shift. Dense connection [33] extends thinking in skip connection to every layer in macro space. The underlying mechanism is discussed further in [34], through which the preactivation architecture is discovered. To another end, depthwise separable convolution is extendedly used to shrink the barrier between accuracy and latency [35]. Due to the incomplete comprehension of the underlying mechanisms of CNN, however, these works can only pay attention to few aspects of CNN design. Actually, both their inspirations of the architecture design and trained parameters contain very meaningful knowledge. We should consider every part of them from a more general view.

For another, with the boom of computing power by accelerating hardware, AutoML has become usable to search for promising architecture automatically. With the help of parameter sharing and performance prediction, ENAS [13] sets a good example in this area. Although based only on one design principle from manual experience—similar cell repeating, AutoML has been broadly developed. BlockQNN [36] searches the connection among blocks. DARTS [29] relaxes the search space to be continuous and makes the architecture generation optimizable using a gradient. BNAS [37] proposes a method for searching abroad neural architecture. Besides, various algorithms are proposed to better search for the optimal architecture. Progressive shrinking makes it possible to train once-for-all weights before searching [38]. The prediction with experts advice (PEA) theory is introduced in [39] to optimize regret for a better architecture search. However, all these works need to train supernet or child model from scratch, costing lots of computation. Moreover, none of these works can efficiently consider previous experts' effort in architecture design, causing a huge waste.

### B. Evolutionary Algorithm for NAS

RL [12], [13]; EA [25], [26]; and gradient-based algorithm [29] are always used for NAS. Among them, EA has been used for the neural-network design for some time. Reference [40] is an early example using EA, and then the NeuroEvolution of Augmenting Topologies (NEAT) algorithm [41], can only search well for small networks. From them, various works tend to extend the usage of EA in NAS, such as CoDeepNEAT [42] or AmoebaNet [25]. Loni *et al.* [43] uses fixed-length combinations of binary code to represent the network structure, which facilitates population operations (such as crossover operation). In order to represent more details and more freedom expression of architecture, Sun *et al.* [44] proposed a variable-length encoding strategy. The initial population starts from the random initialization in the designed search space [44], some initial conditions [45], and the rich initialization (called the well-designed architecture) [46]. Generally, only one population is generated during the evolution. Kotyan and Vargas [47] though created three subpopulations, which have different population operations. Rawal and Miikkulainen [48] and Behjat *et al.* [49] also divide the population into several subpopulations, in order to maintain diversity during the evolutionary processing.

Conceptually, the search back end of the proposed ModuleNet is inspired by NSGA-Net [26], which also uses NSGA-II [30] for searching. NSGA-II is an evolutionary multiobjective optimization algorithm. By extending NSGA [50], NSGA-II solves the problem of the nonelitism approach and lowers its computational complexity, making it suitable for NAS.

**Algorithm 1:** Search Algorithm for ModuleNet

**Input**: $n$ architectures $arch_1 \cdots arch_n$, cell number $c$, evolutionary generation $gen$, population size $p\_size$
**Output**: population $pop$

decompose each $arch_i$ into $c$ cells, $arch_i$-$cell_j$ stands for $j^{\text{th}}$ cell in $arch_i$;
/* Initialize knowledge base */
**for** $j$ from 1 to $c$ **do**
  **for** $i$ from 1 to $n$ **do**
    $module_j^i = f_m(arch_i\text{-}cell_j)^{\ddagger}$;
    $knowledge\_base[j][i] = module_j^i$;
/* Initialize population */
**for** $i$ from 1 to $p\_size$ **do**
  **for** $j$ from 1 to $c$ **do**
    $pop[i][j] = module_j^*$ sampled from $knowledge\_base[j][:]$;
evaluate$^{\ddagger}$ individuals in $pop$;
/* Do evolutionary search */
**for** $g$ from 2 to $gen$ **do**
  $new\_individual$ = mate$^{\dagger}$ and generate$^{\dagger}$ according to $pop$ and encoding method$^{\ddagger}$;
  assemble new architecture with connections$^{\ddagger}$;
  evaluate$^{\ddagger}$ individuals in $new\_individual$;
  compare$^{\dagger}$ over ($pop + new\_individual$) and sort$^{\dagger}$;
  $pop$ = select$^{\dagger}$ from sort results;

$^{\dagger}$ Evolutionary procedures show respect to NSGA-II.
$^{\ddagger}$ Our proposed methods are seen in the following sections.

## III. Method

### A. Overview for ModuleNet

An overview of our method can be seen in Algorithm 1. In general, we first decompose some existing architectures to different cells according to the downsampling positions of architectures. Then, we extract the architectures and weights of cells to form different *module*s and add these *module*s to the knowledge base. Finally, we make use of the NSGA-II algorithm as a search algorithm for finding the optimal architecture from the built knowledge base. In the following parts, we will focus on four important details in our proposed method.

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

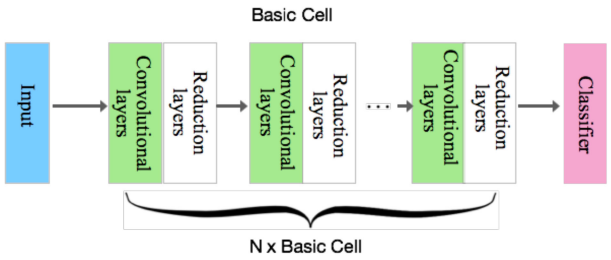


Fig. 2. Universal view for most CNN architectures. We combine convolutional layers and their following reduction layer as a basic cell.

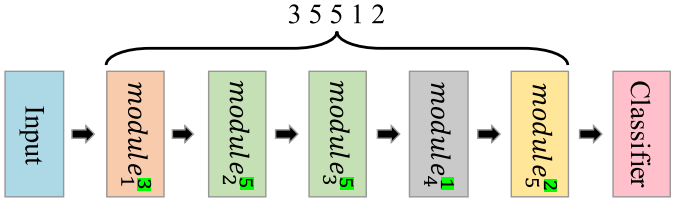


Fig. 3. Example of decoding string to architecture. Cells from the same architectures are in the same color. The reassembled architecture consists of the first cell of the third architecture, following by the second cell of the fifth architecture, the third cell of the fifth architecture, the fourth cell of the first architecture, and the fifth cell of the second architecture.

In Section III-B, we will illustrate how we decompose an existing architecture to make it suitable for reassembling and compatible with other *module*s in the new architectures. We fix parameters from our knowledge base of each *module* to both save on high computing costs for gradient backward and effectively inherit knowledge from the existing *module*s. In Section III-C, our encoding method for different *module*s, together with the definition of search space, will be illustrated. These two parts can be considered as preprocessing for search.

After that, Sections III-D and III-E serve as the key points in our method. To relieve the pressure of parameter-tuning when searching, we design new operators as connections, namely, the Channel Pool and Channel DePool (ChP and ChDP). Together with the fixed parameters, we use these operators to completely eliminate trainable parameters before linear layers when searching. In Section III-E, we will present a new function to better evaluate performance with restriction from fixed parameters. Experiments show the fine correlative relationship between our function and test error obtained with all parameters trainable.

To be acknowledged, we only use NSGA-II with one objective in the article. However, taking advantage of the basic design target of NSGA-II, the proposed ModuleNet can be easily extended for multiobjective search.

### B. Knowledge Base

Existing CNN architectures, regardless if they were discovered by experts or AutoML, are all precious knowledge that should be used to the fullest extent. We first decompose some existing architectures into uniform cells and then build a knowledge base to hold. As shown in Fig. 2, inspired by [13], we consider a CNN architecture as a stack of convolutional layers and reduction layers between the input and the classifier (which always has softmax and linear layers). Considering the continuity of the layers, we combine convolutional layers and their following reduction layer as a basic cell.

Not merely architectures of CNNs, we also consider weights in cells to avoid the burden of retraining. Keeping weights can also help us better inherit knowledge not only from architectures but also from the training procedure. By extracting weights to form the entire architecture, we can finally obtain different *module*s for search, referred to as

$$\text{module}_j^i = f_m\left(\text{arch}_i - \text{cell}_j\right)$$

for the $j$th cell in $arch_i$. To make it clear, we have the following.

*Definition 1 (Module):* This pertains to a cell that has decomposed from an existing CNN architecture, and has kept its trained weights in the original architecture.

Paying attention to that in our method, we consider CNNs as multilayer filters, and each layer can process information from different semantic aspects. For example, layers that are in a relatively shallow stage of CNNs may process information at a local level; however, deeper layers, which have a larger receptive field, are fit for extracting information with a global view or at a high semantic level. Therefore, we have to keep some settings unchanged when reassembling new architectures, to make weights in *module*s usable. First, we keep the *module*'s position of the order in the new architecture as the origin. Second, we keep the resolution of input unchanged by adjusting the reduction in the preceding block. Only in this way can we make each *module* take effect of its original semantic level.

### C. Encoding

Considering that we have $n$ architectures in total, and a decomposition of $c$ cells for each architecture, by assigning different architectures $\text{arch}_i$ to an integer $i$, each string of integers in $\{i_1 i_2 \cdots i_c | i \in N^+, i \leq n\}$ can be decoded as an architecture. Specifically, $i_j$ represents $\text{arch}_i - \text{cell}_j$, namely, the $j$th cell of the $i$th architecture. We use $\text{module}_j^i$, $\{i, j \in N^+ | i \leq n, j \leq c\}$ as representations.

An example with $c = 5$ and $n \geq 5$, shown in Fig. 3, means that there are more than five architectures and five cells of each architecture in the knowledge base. The reassembled architecture shown in this figure consists of the first cell of the third architecture, followed by the second cell of the fifth architecture, the third cell of the fifth architecture, the fourth cell of the first architecture, and the fifth cell of the second architecture. Besides, we can obtain the size of search space ($\Omega$) through

$$|\Omega| = n^c$$

which is much smaller than those in previous works [13], [26], [29] when searching in macro space, but, benefiting from existing fine design, is powerful enough to make progress for tasks.

### D. Module Connection

Since we are using *module*s from a different architecture, which are separately designed, neighboring *module*s in

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

a new architecture may have different channels. Following Definition 1, we need not and should not change weights in *module*s. However, if we use trainable parameters in connection, we still need to backpropagate loss through a gradient to the front stage. This can cause a huge computing cost and may cause instability due to weight-fixing in each *module*. To solve the problems above, in this section, we will introduce two new operators as connections, namely: 1) ChP and 2) ChDP. These two operators do not contain trainable parameters, thus solving those problems elegantly.

*Channel Pool* performs a standard 1-D average pooling on the channel dimension. It is used to decrease the number of channels. Given the architecture *input* with size $(N, C, H, W)$, which means dimensions of the *Number of batch, Channel, Height,* and *Width*, respectively, the expected output with size $(N, C_{\text{out}_P}, H, W)$ is denoted as $\text{out}_P$. In ModuleNet, the numbers of batch, height, and width of feature maps do not change through the processing of every *module* in the knowledge base. But the number of channels usually changes for extracting more features. Assuming that the output channel number is smaller than the input ones ($C_{\text{out}_P} < C$) and that $C_{\text{out}_P}$ is divided by $C$ with no remainder ($C_{\text{out}_P}|C$), we have

$$\text{out}_P = \text{ChP}(\text{input}; \{\boldsymbol{k_P}\}) \tag{1}$$

where $\boldsymbol{k_P}$ is the kernel size, $\boldsymbol{k_P} = [C/C_{\text{out}_P}]$. The calculation for the $l$th channel dimension of $\text{out}_P$, or $\text{out}_P(*, l, *, *)$, is

$$\text{out}_P(N, l, H, W) = \frac{1}{\boldsymbol{k_P}} \sum_{m=0}^{\boldsymbol{k_P}-1} \text{input}(N, \boldsymbol{k_P} \times l + m, H, W)$$

for $l \in \{l \in N^+ | 0 \le l < [C/\boldsymbol{k_P}]\}$.

*Channel DePool* performs a duplication and connection on the channel dimension. It is used to increase the number of channels. Considering *input* with size $(N, C, H, W)$, the expected size of $\text{out}_{DP}$ is $(N, C_{\text{out}_{DP}}, H, W)$. Assuming that $C_{\text{out}_{DP}} > C$, and $C$ is divided by $C_{\text{out}_{DP}}$ with no remainder ($C|C_{\text{out}_{DP}}$), we have

$$\text{out}_{DP} = \text{ChDP}(\text{input}; \{\boldsymbol{k_{DP}}\}) \tag{2}$$

where $\{\boldsymbol{k_{DP}}\}$ is the duplication times, $\boldsymbol{k_{DP}} = (C_{\text{out}_{DP}}/C)$. The calculation for the $l$th channel dimension of $\text{out}_{DP}$, or $\text{out}_{DP}(*, l, *, *)$, is

$$\text{out}_{DP}(N, l, H, W) = \text{input}(N, \text{mod}(l, C), H, W)$$

where $\text{mod}(l, C)$ means $l$ performs the remainder operation on $C$ and $l \in \{l \in N^+ | 0 \le l < C \times \boldsymbol{k_{DP}}\}$.

However, the actual situation may break our assumptions in definitions easily. Therefore, we should consider more complex situations.

For neighboring *module*s that need a connection from $C$ channels to $C_{\text{out}}$ channels, if $C_{\text{out}}|C$ or $C|C_{\text{out}}$, we just use a ChP or ChDP to make connections. Otherwise, we will extend these two operators as follows. In this part, since we only focus on the channel dimension of *input*, we use $C^{\text{In}}$ as the representation. $C^{\text{In}}$ can be considered as a 1-D array.

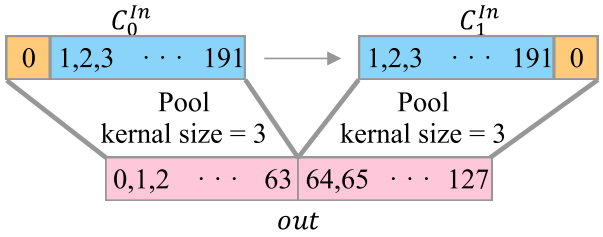


Fig. 4. Example for using ChP with $C^{\text{In}} = 192$, and$C_{\text{out}} = 128$.

*1) $C > C_{out}$:* We first find the greatest common divisor of $C$ and $C_{\text{out}}$ as $\eta$

$$\eta = \text{gcd}(C, C_{\text{out}})$$

where $\text{gcd}(C, C_{\text{out}})$ means the operation to find the greatest common divisor of $C$ and $C_{\text{out}}$.

Then, we use a ChP as

$$\begin{cases} \text{out} = \underset{i=0}{\overset{C_{\text{out}}/\eta-1}{\text{Concat}}}\left[\text{ChP}\left(C_i^{\text{In}}; \{\boldsymbol{k'_P}\}\right)\right] \\ C_i^{\text{In}} = \text{Concat}\left(C^{\text{In}}[i : \text{end}], C^{\text{In}}[\text{begin} : i]\right) \end{cases} \tag{3}$$

where $\boldsymbol{k'_P} = (C/\eta)$ and [:] denotes the slice operation. Note that $C^{\text{In}}$[0:end] contains all elements in $C^{\text{In}}$, and $C^{\text{In}}$[begin:0] is empty. An example of this part can be seen in Fig. 4.

*2) $C < C_{out}$:* We use a slice of output from ChDP as

$$\text{out} = \text{ChDP}\left(C^{\text{In}}; \left\{\boldsymbol{k'_{DP}}\right\}\right)\left[\text{begin} : C_{\text{out}}\right]$$

where $\boldsymbol{k'_{DP}} = \lceil (C_{\text{out}}/C) \rceil$.

Finally, through ChP and ChDP, we can make *module* connections between neighbors without parameters. In this way, the gradient can be avoided from being backpropagated deeply and a lot of computing costs can be saved.

Experiments also show that using nonparameter connections has a non-negative, even sometimes positive, effect on performance evaluation when searching.

### *E. Performance Evaluation*

Admittedly, although fixing parameters in *module*s can largely reduce both computing costs in a single iterator and the total epoch needed to convergence, the accuracy on the validation set ($\text{acc}_{\text{val}}$) may not fully represent the real performance for an architecture. Since the parameters in use are extracted from some pretrained models, and these models or architectures are still reachable through our search algorithm, $\text{acc}_{\text{val}}$ on these models can be much higher than the others. Besides, for those architectures that are very similar to those pretrained models (only have a few *module*s changed), original parameters may fit them better compared to others with more different *module*s. To avoid this problem of unfair accuracy, we propose a new metric by taking the loss changing rate ($l_{\text{rate}}$), error rate ($\text{err}_{\text{val}}$), and architecture similarity (sim) into consideration to better evaluate the real performance when searching, as defined by

$$\text{score} = \text{err}_{\text{val}} - \alpha \cdot l_{\text{rate}} + \beta \cdot \text{sim} \tag{4}$$

where $\alpha$ and $\beta$ are parameters to balance different items, and determined through experiments. $\text{err}_{\text{val}}$ is the evaluation error

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

on the validation set, which can be obtained by

$$\text{err}_{\text{val}} = 1 - \text{acc}_{\text{val}}. \tag{5}$$

With a basic consideration that fixing parameters may lead to a decrease in the architecture's ability for generalization, but not convergence, the changing rate of loss can be suitable to evaluate the convergence ability of an architecture. By defining $l_{\text{rate}}$ as the loss changing rate when training on the training set for $n$ epochs, it can be obtained by

$$l_{\text{rate}} = \frac{\text{loss}_{\text{epoch}=1} - \text{loss}_{\text{epoch}=n}}{\text{loss}_{\text{epoch}=1}}. \tag{6}$$

With $l_{\text{rate}}$ normalized in [0, 1], it can work well together with $\text{err}_{\text{val}}$.

The last item in (4), *sim*, represents the degree of similarity between a given architecture and each architecture used for pretraining. Moreover, we discover that given the same number of different *module*s between two architectures, the place where the different *module*s lie is also one of the important factors for performance evaluation. Besides, we also find that if we break up the continuity in relatively shallow places of the entire architecture, $\text{acc}_{\text{val}}$ may decrease slightly, but not that much if in relatively deep places. Therefore, we define *sim* through

$$\text{sim} = \frac{1}{c} f_{\text{sim}}(\text{code}, 2, c) \tag{7}$$

where

$$f_{\text{sim}}(a, u, c) = \begin{cases} f_{\text{sim}}(a, u+1, c) + 1, & u \le c \text{ and } a[u] = a[u-1] \\ 0, & u > c \text{ or } a[u] \neq a[u-1]. \end{cases} \tag{8}$$

*Code* is the architecture encoding $i_1 i_2 \cdots i_c$, for $c$ architectures in our knowledge base.

Our experiments show that the scoring function (4) has enough correlative relationship to help us accurately evaluate the performance for a given architecture when searching.

## IV. Experiments

Our experiments for the proposed ModuleNet are conducted with two stages: 1) the searching stage and 2) the evaluation stage. The first is the searching stage. As defined by Algorithm 1, we use $c = 5$, gen $= 30$, and $p_{\text{size}} = 40$ for each experiment. As for $n$ (Algorithm 1), $\alpha$, and $\beta$ (4), we will illustrate in the section of each experiment. After the searching stage, we can obtain the population of the final generation $\text{pop}_{\text{final}}$ (Algorithm 1). Then, in the second stage, we make all parameters in the architectures trainable and fine tune the parameters. Through this evaluation stage, we can determine the best final architecture for a given task.

As for the classifier (depicted in Fig. 2) in each experiment, we use three fully connected layers, with a feature size of *input size* $-$ 4096 $-$ 4096 $-$ *class number*, of which the *input size* is determined by the last convolutional layer and the *class number* is determined by the dataset. We use the standard *Cross-Entropy Loss* as

$$\text{loss}(x, \text{class}) = -\log\left(\frac{\exp x[\text{class}]}{\sum_j \exp x[j]}\right)$$

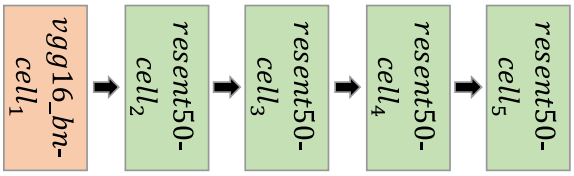


Fig. 5. Searched architecture based on craft design for CIFAR10: this architecture is searched on CIFAR10 with the knowledge base containing only VGGs and ResNets.

TABLE I
Result Comparison Between Our Searched Architecture and Its Original Architectures in the Knowledge Base on CIFAR10. Our Dataset Splitting for Train/Evaluation Follows 40K/10K on Training Set for Each Architecture

| Method | Test Error (%) |
|---|---|
| Best for ResNets [3] | 6.43 |
| Best for VGGs [52] | 6.27 |
| ModuleNet (ours) | **5.81** |

where $x$ is the array of network output, indicating possibilities to each class, and *class* is the class label for input.

We train for 20 epochs to make parameters in the classifier convergent in the searching stage, and 50 epochs for fine tuning in the evaluation stage. The original architectures used for comparison are trained or fine tuned with about 350 epochs to achieve the best performance. They have achieved equivalent or even better performance than the original papers. More specifically, the feature extractor is composed of convolution-based modules, which extract feature representations for the following fully connected classification layer, as shown in Fig. 3. We mention that the backbone of the feature extractor is exactly the architecture we search from the existing modules. As for training when searching, we only optimize the last classifier weights with 20 epochs but fix weights of the feature extractor.

### A. Results on CIFAR10

CIFAR10 [51] is a highly used dataset for image classification. In this section, we conduct the experiment based on the hand-designed *modules* on CIFAR10 and show our results. Then, we add some *module*s, whose architectures are searched by some state-of-the-art search algorithms to our knowledge base and show that our algorithm can still make steady improvements even for these already-perform-well modules. In addition, we also compare the search time between the proposed method and the common method.

*1) Search With Craft Design:* We use a configuration with $n = 7$ architectures designed by craft for searching, containing ResNet34, ResNet50, and ResNet101 [3] and VGG13, VGG16, VGG13bn, and VGG16bn [4]. We use the implements from *torchvision*[1] with their pretrained weights accordingly.

After two stages of searching, the best architecture we obtain is shown in Fig. 5 and the result comparison can be

[1]Torchvision can be found in https://github.com/pytorch/vision, we are using version v0.3.0.

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

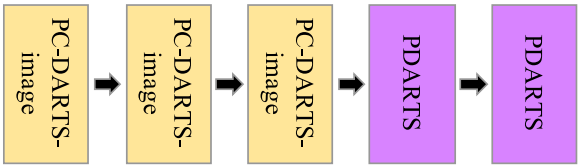


Fig. 6. Searched architecture based on other NAS results for CIFAR10.

TABLE II
SEARCH TIME COMPARISON OF THE INHERITED AND THE NONINHERITED WEIGHTS IN THE SAME CONFIGURATION

| Methods | Test Error(%) |
|---|---|
| DARTS + cutout [29] | 2.86 |
| PDARTS + cutout [53] | 2.91 |
| PC-DARTS-cifar† + cutout[54] | 2.87 |
| PC-DARTS-ImageNet ‡ + cutout [54] | 2.8 |
| ModuleNet (ours) + cutout | **2.77** |

† the architecture searched by PC-DARTS for CIFAR10 task.
‡ the architecture searched by PC-DARTS for ImageNet task.

seen in Table I. From the results, we may notice that although our searched architecture contains only one *module* different from the original architecture, it can bring a huge increase in performance.

*2) Search With NAS Design:* To extend our experiments, we then add some *module*s searched by other NAS algorithms in our knowledge base to enlarge our search space. DARTS [29] is the first to introduce continuous relaxation to architecture representation, and famous work in the NAS area. By extending DARTS, PDARTS [53] and PC-DARTS [54] also make progress in searching for better architectures and follow the same configuration. Therefore, we introduce *module*s searched by these three algorithms[2] into our knowledge base, referred to as DARTS, PDARTS, PC-DARTS-cifar (searched by PC-DARTS for CIFAR10), and PC-DARTS-ImageNet (searched by PC-DARTS for ImageNet [55]). Since we are using $c = 5$ in other experiments, we also change their stacking strategy with a reduction in 5, 10, 15, and 20 layers in total. In this way, we have five cells in each *module*, together with a stem cell as the first *module*, making up five *module*s.

Keeping other setting the same as the previous experiments except for $\alpha = 10$ and $\beta = 30$, after two stages of the search, the architecture we obtain is shown in Fig. 6 and the result comparison can be seen in Table II. From the results, we may note that even for NAS searched architectures, stacking cells is not the best way. Besides, our result does not contain those cells that perform better when stacked, from which we can conclude that there is no strong correlation on performance between a single cell and entire architecture.

*3) Comparison on GPU Cost:* In this part, we compare the GPU costs during the search phase. All experiments are run on a server computer configured with Intel Xeon E5-2620 CPUs, 128-GB RAM, and 4 Nvidia 2080 GPUs. Table III reports the GPU days. Under the same experimental configuration, the proposed knowledge-inherited search method takes 0.22 min to train the searched architecture an epoch, while the common search method without knowledge inheritance takes 1.01 min to train the searched architecture an epoch. Therefore, the proposed knowledge-inherited NAS method can significantly reduce the search time and bring the acceleration of 4.59× per epoch.

[2]Implements can be found in https://github.com/flymin/darts.

TABLE III
SEARCH TIME COMPARISON OF THE INHERITED AND THE NONINHERITED WEIGHTS IN THE SAME CONFIGURATION

| Methods | 1 Epoch Time* (minutes) | Search Time (days) |
|---|---|---|
| One-shot NAS † | 1.01 | 5.5 |
| ModuleNet ‡ | **0.22** | **2.0** |

† One-shot NAS is the common NAS methods without knowledge-inheritance.
‡ ModuleNet is the proposed knowledge-inherited search method.
* 1 epoch time is the average time of 10 epochs during smooth phase of the network training process.

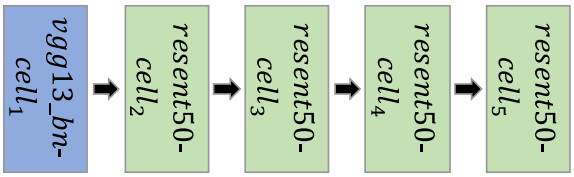


Fig. 7. Searched architecture based on craft design for CIFAR100: this architecture is searched on CIFAR100 with the knowledge base containing only VGGs and ResNets.

TABLE IV
RESULT COMPARISON BETWEEN OUR SEARCHED ARCHITECTURE AND ITS ORIGINAL ARCHITECTURES IN THE KNOWLEDGE BASE ON CIFAR100. IN THE FINE-TUNE STAGE, WE USE CUTOUT LENGTH = 16. OUR DATASET SPLITTING FOR TRAIN/EVALUATION FOLLOWS 40K/10K ON TRAINING SET FOR EACH ARCHITECTURE

| Method | Test Error (%) |
|---|---|
| Best for ResNets + cutout [3] | 22.97 |
| Best for VGGs + cutout [52] | 28.27 |
| ModuleNet (ours) + cutout | **17.99** |

## B. Results on CIFAR100

CIFAR100 [51] is also a highly used dataset for image classification. Because it has a small image size, a deep model [20] can be trained on this dataset in a short time. Since its class number is more than CIFAR10 [51], it is broadly used in the NAS scenario.

In this section, we go one step further in this section to verify the efficiency of our ModuleNet. First, as a standard experiment, similar to those on CIFAR10, we first fine tune those architectures given in Section IV-A for new *module*s in the knowledge base. Then, we apply searching with the same configurations as above.

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

TABLE V
RESULT COMPARISON AMONG OUR SEARCHED ARCHITECTURE, ITS ORIGINAL ARCHITECTURES

| Methods | Test Error (%) | |
|---|---|---|
| | top 1 | top 5 |
| VGG13bn [56] | 28.45 | 9.63 |
| ResNet50 [56] | 23.85 | 7.13 |
| ModuleNet-Exp$_1$ (ours)$^{\ddagger}$ | **22.74** | **6.96** |
| ResNet101 [56] | 22.63 | 6.44 |
| FairDARTS [57] | 22.80 | 6.50 |
| ModuleNet-Exp$_2$ (ours)$^{\dagger}$ | **21.31** | **5.80** |

$^{\ddagger}$ ModuleNet-Exp$_1$ is reassembled by VGG and ResNet50
$^{\dagger}$ ModuleNet-Exp$_2$ is reassembled by VGG and ResNet101.

After two stages of searching, the best architecture we obtain is shown in Fig. 7. The evaluation result comparison can be seen in Table IV. The best architecture of ResNets (ResNet34, ResNet50, and ResNet101) has 22.97% test error, and the test error of the best architecture of VGGs (VGG13, VGG16, VGG13bn, and VGG16bn) is 28.27%. While the new architecture searched by the proposed ModuleNet has only 17.99% test error, which reduced by at least 4.98% and 10.28% compared to ResNets and VGGs, respectively. It may due to that the proposed ModuleNet finds the novel reassembled architecture from the knowledge base with the existing models.

However, compared to the experiments on CIFAR10, we also note that different datasets need different architectures to guarantee better performance, and simply stacking the same cells or transferring architecture designed for other datasets may not bring the best results.

### C. Results on ImageNet

To evaluate the architecture searched by our approach on larger datasets, we evaluate the architecture searched in CIFAR10 or CIFAR100 directly on ImageNet [55] using the DGX station. From Figs. 5 and 7, we can see that the best architectures searched by ModuleNet replace the first *module* in ResNet with VGG's. We suppose a possible explanation that in shallow layers, modules need to rule out more useless information, whereas in deep layers, with the loss of useless information, modules need to be more careful when filtering. Therefore, VGG modules, which are better at ruling out information, are used as shallow layers. Whereas, ResNet modules, which are better at identifying and keeping useful information, are used as deep layers. In order to evaluate the explanation, we conduct two experiments on ImageNet: the first *module* of VGG13bn replacing ResNet50 (ModuleNet-Exp$_1$) and ResNet101 (ModuleNet-Exp$_2$), and results are shown in Table V.

We inherit the knowledge of VGG13bn, ResNet50, and ResNet101 models, which are trained in the ImageNet dataset [56]. The test errors (top 1) of VGG13bn, ResNet50, and ResNet101 are 28.45%, 23.85%, and 22.63%. From the first four lines of Table V, we can see that ModuleNet-Exp$_1$ reassembled by VGG and ResNet50 improves its performance by at least 1.11% compared to ResNet50 and VGG in the top 5. Compared to ResNet101, ModuleNet-Exp$_2$ reassembled by VGG and ResNet101 has a slight improvement.

Compared to the last two lines, the architecture searched by the proposed ModuleNet outperforms the one searched by FairDARTS [57] by 1.49% in the top 1 error and 0.64% in the top 5 error. We can see that our algorithm can still make improvements even compared to the state-of-the-art method FairDARTS [57].

These results show that the searched architecture replacing the first *module* of ResNet with VGG's is also robust for the large-scale dataset ImageNet. Compared to the ModuleNet architectures of Tables I, IV, and V, we also see that a small and simple architecture is fit for a simple task, such as CIFAR10 only with ten categories, and the deeper and larger architecture is suitable to solve difficult task such as ImageNet with 1000 categories.

We also search the architectures in the modular space composed of both hand-designed modules and NAS-optimized modules. However, the searched architectures combining the different modules both in hand-designed modules and NAS-optimized modules do not achieve good performance. We observe the instability of the ModuleNet searching algorithm when mixing hand-designed and NAS-optimized modules. It could be due to that the NAS-optimized modules are more complex and impressible to the architecture than the hand-designed ones. There are also many differences between them, such as the channel size, the feature map size, and the skip connection. In order to connect these modules with great differences, we need to introduce many extra pooling/depooling (ChP, ChDP) connection channels. Fortunately, with the help of the proposed connection channels, we can obtain better model architectures under the spaces of both hand-designed and NAS-optimized modules.

## V. ABLATION STUDIES

In this section, we will show some additional experiments to verify the effectiveness of three core parts (evaluation strategy, evolutionary search, and nonparameters) in our proposed method separately.

### A. Efficiency of Performance Evaluation

As a core part of our search algorithm, *score* in (4) performs an important role to make a comparison between different architectures during evolution. Therefore, the efficiency of the evaluation function is very important and directly related to the final results we obtain. In this section, we conduct some more experiments to verify the efficiency of our evaluation function *score*.

As the NAS algorithm aiming to improve the performance on image classification tasks, the basic evaluation metric should be the *Test Error*. However, directly calculating *Test Error* requires a fully trained architecture on the training set, which demands high computing costs and time. Therefore, we introduce a new strategy and new function to evaluate. Accordingly, the best way to show the efficiency of our strategy and function is to make a comparison with *Test Error* for architectures after being fully trained.

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

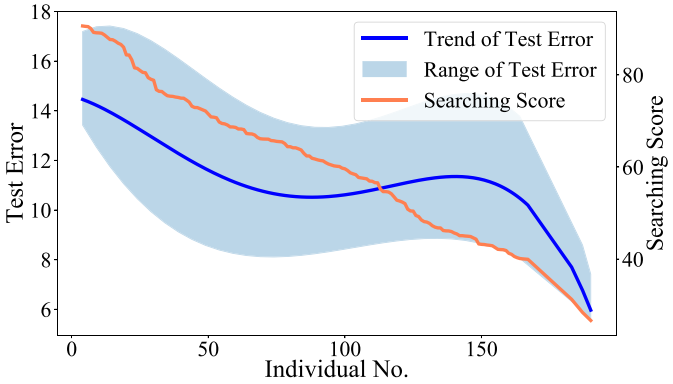


Fig. 8. Comparison between results generated through our searching strategy by *score* (donated by *searching score*), and *test error* after retraining. Both metrics are the lower the better. Architectures are obtained from all 30 generations of populations in the experiment of Section IV-B1, descendingly sorted according to *searching score*.

TABLE VI
CORRELATION COEFFICIENT (COE) BETWEEN EACH ITEM IN (4) AND *Test Error*, WHERE *Test Error* MEANS THE PERFORMANCE OF ARCHITECTURES AFTER FULLY TRAINED

| Item | *coe* to *Test Error* |
|---|---|
| $err_{val}$ | 0.79 |
| $l_{rate}$ | -0.72 |
| $sim$ | -0.26 |
| $score$ | 0.82 |

As shown in Fig. 8, descending of the *searching score* indicates the descending of *Test Error*. Such a result can verify that there is a correlative relationship between our evaluation strategy and *Test Error*, which proves that our strategy is usable to search for better architectures. Furthermore, architectures used in Fig. 8 are from the same searching path of experiment in Section IV-B1, which can be a side proof that our evaluation strategy *searching score* is fit for the EA we used.

Moreover, we analyze each item in complicate *score* in (4) to make the effect of *score* convincing. Using the data in Fig. 8, we calculate the correlation coefficient (coe) between each item and *Test Error*, shown in Table VI. $l_{\text{rate}}$ and $err_{\text{val}}$ have a strong correlationship with *Test Error*, which makes sense because of their definitions. Although *sim* itself has no strong correlationship with *Test Error*, by combining all these three items as a *score*, we can note an increase in the correlationship.

### B. Efficiency of Evolutionary Search

In general, the EA NSGA-II is a multiobjective optimization algorithm. We choose NSGA-II as the back end to bring more scalability to our search algorithm. In the future, other objectives, such as the number of parameters, latency, or amount of floating-point operations, can be easily extended into the current search framework. However, for now, we only consider *score* as the only objective.

To evaluate the efficiency of EA, one side is judging the final result searched by this algorithm, which has already shown in Section IV; the other side is the convergence of the algorithm. Although *mutation* and *crossover* are made between

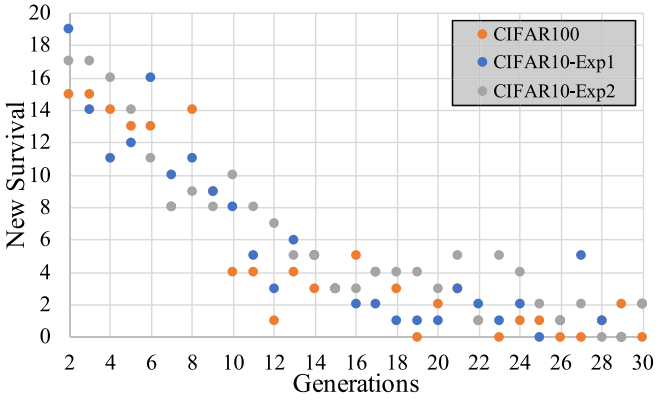


Fig. 9. New survival between generations: For each experiment in Section IV, new survived genotypes for $\text{pop}_i$ (population of generation $i$) compared to $\text{pop}_{i-1}$. CIFAR100 refers to the experiments in Section IV-A, and CIFAR10-Exp1 and CIFAR10-Exp2 refer to the experiments in Section IV-B1 and Section IV-B2, respectively.

generations, we still expect that genotypes of living generation may become relatively stable after generations of evolution. Therefore, we calculate genotype changes between (indicated by *new survival*) two consecutive generations, as shown in Fig. 9.

We can note that in each separate experiment, genotypes in populations will always converge to be stable after generations of evolution. From such results, we can conclude that the EA, NSGA-II, is fit for the NAS task within the scenario of our proposed ModuleNet.

### C. Nonparameter Connection

As illustrated in Section III-D, using nonparameter connections (Channel Pool and Channel DePool) can slightly reduce trainable parameters when searching and, thus, reduce the searching time. Although we theoretically keep enough diversity when connecting the preceding *module* with the following *module*, these operations, however, may leave a question. Do nonparameter connections result in a negative effect on the performance judgment? To verify the efficiency of our proposed nonparameter connections, we do some further experiments.

In our experiments, we apply a $1 \times 1$ convolution between each *module* to transfer between different channels. $1 \times 1$ convolution can be the simplest way to change channel dimensions and keep others. Besides, it contains trainable parameters to make it adjustable by the gradient.

We use architectures searched from all 30 generations of populations in the experiment of Section IV-B1, and make a comparison between the nonparameter architectures and architectures connecting with $1 \times 1$ convolution. All these architectures are trained under search setting (fix parameters in each *module* and tune others). We obtain the validation accuracy ($\text{acc}_{\text{val}}$) of each architecture and $\text{acc}_{\text{val}}$ comparison is shown in Fig. 10. $\text{acc}_{\text{val}}$ w/o params means the validation accuracy of the architecture with the proposed nonparameter connection and $\text{acc}_{\text{val}}$ w/ params means the validation accuracy of the architecture with $1 \times 1$ convolution.

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

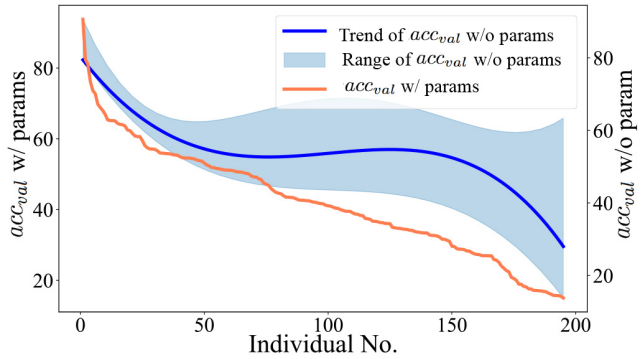


Fig. 10. Comparison between validation accuracy ($acc_{\text{val}}$ acc) generated through nonparameter connections ($\text{acc}_{\text{val}}$ acc w/o params, described in Section III-D) and connections using $1 \times 1$ convolutions ($\text{acc}_{\text{val}}$ w/ params). Architectures are obtained from all 30 generations of populations in experiment of Section IV-B1, descendingly sorted by $\text{acc}_{\text{val}}$ w/ params.

As shown in Fig. 10, using nonparameter connections can keep and even strengthen the differential ability of our algorithm. Specifically, for those better architectures (left end in the $x$ axis), both metrics ($\text{acc}_{\text{val}}$ w/o params and $\text{acc}_{\text{val}}$ w/ params) indicate better results, and vice versa. Besides, for those architectures in the middle of the $x$ axis, nonparameter connections ($\text{acc}_{\text{val}}$ w/o params) could lead to a better differentiated status. Therefore, our nonparameter connection is not only usable but a better fit for our search algorithm.

## VI. Conclusion

This article presented ModuleNet, a novel NAS algorithm to fully inherit existing knowledge and explore the new architecture design. We proposed that both the architecture and trained parameters of an existing model should be used for further exploration. By decomposing the existing architectures into *module*s, we can use a uniform view to reorganize and rediscover them. In order to connect different *module*s, we proposed nonparameter connections (Channel Pool and Channel DePool), reducing the computational consumption of search significantly. Moreover, we proposed an effective performance evaluation method to avoid falling into existing network architectures and invent equal opportunity among architectures. In this way, we can make CNNs transfer quickly among different tasks and datasets and always guarantee a performance improvement.

In our experiments, we showed that the search architectures by ModuleNet have the best performances not only in human-designed existing models but also in NAS searched models for CIFAR10, CIFAR100, and ImageNet classification tasks. Moreover, we also conducted experiments to show the efficiency of our *score* equation, EA, and connections between *module*s. All of these show that existing knowledge is of great importance, and ModuleNet has set up a new NAS scheme for using them. Actually, there are also many directions to improve ModuleNet further. For example, the *score* equation can only indicate a relevance relationship, which can be better if a linear relationship is reached. Some extensions may be added to the search algorithm to fulfill other constrictions. These interesting topics could be potential directions for future studies.

## References

[1] E. Yang, T. Liu, C. Deng, and D. Tao, "Adversarial examples for hamming space search," *IEEE Trans. Cybern.*, vol. 50, no. 4, pp. 1473–1488, Apr. 2020.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2012, pp. 1097–1105.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. Eur. Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.

[5] D. Zhao, Y. Chen, and L. Lv, "Deep reinforcement learning with visual attention for vehicle classification," *IEEE Trans. Cogn. Develop. Syst.*, vol. 9, no. 4, pp. 356–367, Dec. 2017.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[7] Y. Chen, D. Zhao, and H. Li, "Deep Kalman filter with optical flow for multiple object tracking," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, 2019, pp. 3036–3041.

[8] C.-Q. Huang, J.-K. Chen, P. Yan, H.-J. Lai, Y. Jian, and Q.-H. Huang, "Clothing landmark detection using deep networks with prior of key point associations," *IEEE Trans. Cybern.*, vol. 49, no. 10, pp. 3744–3754, Oct. 2019.

[9] A. Brankovic, A. Falsone, M. Prandini, and L. Piroddi, "A feature selection and classification algorithm based on randomized extraction of model populations," *IEEE Trans. Cybern.*, vol. 48, no. 4, pp. 1151–1162, Apr. 2018.

[10] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent.*, Toulon, France, Apr. 2017.

[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[13] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4092–4101.

[14] X. Chu, B. Zhang, R. Xu, and J. Li, "FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search," 2019. [Online]. Available: arXiv: 1907.01845.

[15] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," 2019. [Online]. Available: arXiv: 1904.00420.

[16] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. V. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 549–558.

[17] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 826–834, Sep./Oct. 1983.

[18] L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, vol. 2. Cambridge, MA, USA: MIT Press, 1997, pp. 396–404.

[19] R. Hechtnielsen, "Theory of the backpropagation neural network," *Neural Netw.*, vol. 1, no. s1, pp. 445–448, 1988.

[20] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 818–833.

[21] S. Christian *et al.*, "Going deeper with convolutions," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1–9.

[22] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Cham, Switzerland: Springer, 2015, pp. 234–241.

[23] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 936–944.

[24] Y. Chen, D. Zhao, L. Lv, and Q. Zhang, "Multi-task learning for dangerous object detection in autonomous driving," *Inf. Sci.*, vol. 432, pp. 559–571, Mar. 2018.

[25] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4780–4789.

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.

[26] Z. Lu *et al.*, "NSGA-NET: A multi-objective genetic algorithm for neural architecture search," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 419–427.
[27] Z. Ding, Y. Chen, N. Li, and D. Zhao, "Simplified space based neural architecture search," in *Proc. IEEE Symp. Series Comput. Intell. (SSCI)*, 2019, pp. 43–49.
[28] N. Li, Y. Chen, Z. Ding, and D. Zhao, "Multi-objective neural architecture search for light-weight model," in *Proc. Chin. Autom. Congr. (CAC)*, Hangzhou, China, 2019, pp. 3794–3799.
[29] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Comput. Vis.*, 2019, pp. 1294–1303.
[30] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[31] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.
[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[33] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 2261–2269.
[34] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.
[35] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: arXiv: 1704.04861.
[36] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2423–2432.
[37] Z. Ding, Y. Chen, N. Li, D. Zhao, Z. Sun, and C. L. P. Chen, "BNAS: Efficient neural architecture search using broad scalable architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 31, 2021, doi: 10.1109/TNNLS.2021.3067028.
[38] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," 2019. [Online]. Available: arXiv: 1908.09791.
[39] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik-Manor, "XNAS: Neural architecture search with expert advice," 2019. [Online]. Available: arXiv: 1906.08031.
[40] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[41] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002.
[42] R. Miikkulainen *et al.*, "Evolving deep neural networks," in *Proc. Artif. Intell. Age Neural Netw. Brain Comput.*, 2019, pp. 293–312.
[43] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjdin, "DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems," *Microprocess. Microsyst.*, vol. 73, Mar. 2020, Art. no. 102989.
[44] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Apr. 2020.
[45] E. Real, S. Moore, A. Selle, S. Saxena, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.
[46] S. Fujino, N. Mori, and K. Matsumoto, "Deep convolutional networks for human sketches by means of the evolutionary deep learning," in *Proc. Joint 17th World Congr. Int. Fuzzy Syst. Assoc. 9th Int. Conf. Soft Comput. Intell. Syst. (IFSA-SCIS)*, 2017, pp. 1–5.
[47] S. Kotyan and D. V. Vargas, "Evolving robust neural architectures to defend from adversarial attacks," 2019. [Online]. Available: arXiv:1906.11667.
[48] A. Rawal and R. Miikkulainen, "From nodes to networks: Evolving recurrent neural networks," 2018. [Online]. Available: arXiv:1803.04439.
[49] A. Behjat, S. Chidambaran, and S. Chowdhury, "Adaptive genomic evolution of neural network topologies (agent) for state-to-action mapping in autonomous agents," in *Proc. Int. Conf. Learn. Represent.*, Mar. 2019, pp. 9638–9644.
[50] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, Sep. 1994.
[51] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, USA, Rep. TR-2009, 2009.
[52] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *Proc. IAPR Asian Conf. Pattern Recognit. (ACPR)*, 2015, pp. 730–734.
[53] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," 2019. [Online]. Available: arXiv:1904.12760.
[54] Y. Xu *et al.*, "PC-DARTS: Partial channel connections for memory-efficient architecture search," 2019. [Online]. Available: arXiv:1907.05737.
[55] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Miami, FL, USA, 2009, pp. 248–255.
[56] *Torchvision.Models*. Accessed: May 21, 2021. [Online]. Available: https://pytorch.org/docs/1.1.0/torchvision/models.html
[57] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair DARTS: Eliminating unfair advantages in differentiable architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 465–480.

**Yaran Chen** (Member, IEEE) received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2018.

She is currently an Associate Professor with The State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences and also with the College of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. Her research interests include deep learning, neural architecture search, deep reinforcement learning, and autonomous driving.

**Ruiyuan Gao** received the B.E. degree in computer science from Shenyuan Honors College, Beihang University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree in computer science and engineering with the Chinese University of Hong Kong, Hong Kong.

His research interests include computer vision, robustness, and security in artificial intelligence.

**Fenggang Liu** received the B.S. degree in vehicle engineering from Harbin Institute of Technology, Shandong, China, in 2014. He is currently pursuing the M.S. degree with the School of Mechanical Engineering, Beijing Institute of Technology, Beijing, China.

His current research interests include AutoML, computer vision, and deep learning.

**Dongbin Zhao** (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees from Harbin Institute of Technology, Harbin, China, in 1994, 1996, and 2000, respectively.

He was a Postdoctoral Fellow with Tsinghua University, Beijing, China, from 2000 to 2002. He has been a Professor with the Institute of Automation, Chinese Academy of Sciences, Beijing, since 2002, and also a Professor with the University of Chinese Academy of Sciences, Beijing. From 2007 to 2008, he was also a Visiting Scholar with the University of Arizona, Tucson, AZ, USA. He has published six books and over 90 international journal papers. His current research interests are in the area of deep reinforcement learning, computational intelligence, autonomous driving, game artificial intelligence, and robotics.

Prof. Zhao serves as the Associate Editor for IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON CYBERNETICS, and IEEE TRANSACTIONS ON ARTIFICIAL INTELLIGENCE. He is the Chair of Distinguished Lecture Program, and the Chair of Technical Activities Strategic Planning Sub-Committee in 2019, Beijing Chapter from 2017 to 2018, Adaptive Dynamic Programming and Reinforcement Learning Technical Committee from 2015 to 2016, Multimedia Subcommittee from 2015 to 2016, and Newsletter from 2013 to 2014 of IEEE Computational Intelligence Society. He works as a guest editor of renowned international journals. He is involved in organizing many international conferences.

Authorized licensed use limited to: INSTITUTE OF AUTOMATION CAS. Downloaded on June 12,2023 at 01:30:59 UTC from IEEE Xplore. Restrictions apply.