Stacked BNAS: Rethinking Broad Convolutional Neural Network for Neural Architecture Search

Zixiang Ding, Yaran Chen, Member, IEEE, Nannan Li, Graduate Student Member, IEEE, Dongbin Zhao, Fellow, IEEE, and C. L. Philip Chen, Fellow, IEEE

Abstract—Different from other deep scalable architecture-based neural architecture search (NAS) approaches, broad NAS (BNAS) proposes a broad scalable architecture which consists of convolution and enhancement blocks, dubbed broad convolutional neural network (BCNN), as the search space for amazing efficiency improvement. BCNN reuses the topologies of cells in the convolution block so that BNAS can employ few cells for efficient search. Moreover, multiscale feature fusion and knowledge embedding are proposed to improve the performance of BCNN with shallow topology. However, BNAS suffers some drawbacks: 1) insufficient representation diversity for feature fusion and enhancement and 2) time consumption of knowledge embedding design by human experts. This article proposes Stacked BNAS, whose search space is a developed broad scalable architecture named Stacked BCNN, with better performance than BNAS. On the one hand, Stacked BCNN treats mini BCNN as a basic block to preserve comprehensive representation and deliver powerful feature extraction ability. For multiscale feature enhancement, each mini BCNN feeds the outputs of deep and broad cells to the enhancement cell. For multiscale feature fusion, each mini BCNN feeds the outputs of deep, broad and enhancement cells to the output node. On the other hand, knowledge embedding search (KES) is proposed to learn appropriate knowledge embeddings in a differentiable way. Moreover, the basic unit of KES is an overparameterized knowledge embedding module that consists of all possible candidate knowledge embeddings. Experimental results show that: 1) Stacked BNAS obtains better performance than BNAS-v2 on both CIFAR-10 and ImageNet; 2) the proposed KES algorithm contributes to reducing the parameters of the learned architecture with satisfactory performance; and 3) Stacked BNAS delivers a state-of-the-art efficiency of 0.02 GPU days.

Index Terms—Broad neural architecture search (BNAS), knowledge embedding search (KES), stacked broad convolutional neural network (BCNN).

I. INTRODUCTION

A

RTIFICIAL neural networks have shown powerful capability of feature extraction in various fields, e.g., computer vision [1], [2], natural language processing [3], etc. To design high-performance neural networks automatically, neural architecture search (NAS) is proposed. NAS has achieved unprecedented accomplishments for various structures design, e.g., convolutional neural network [4], tensor ring [5], language model [6], etc. However, it needs enormous computational requirements, e.g., more than 22 000 GPU days for vanilla NAS [7]. NASNet [8] proposes cell search space to alleviate the time-consuming issue and delivers a higher efficiency of 1800 GPU days. However, the search cost is unbearable yet. Subsequently, a weight-sharing NAS pipeline is proposed to deliver novel efficiency. Reinforcement learning (RL)-based ENAS [6] needs only 0.45 GPU days via parameter sharing. Gradient-based DARTS [9] employs a continuous relaxation strategy to transfer the search space from discrete to continuous, and delivers an efficiency of 0.45 GPU days. Furthermore, based on the DARTS framework, partial channel connection (PC)-DARTS [10] delivers a state-of-the-art search efficiency of 0.1 GPU days via PCs.

Different from deep search space-based approaches, broad NAS (BNAS) [11] proposed a broad search space named broad convolutional neural network (BCNN). Benefiting of BCNN, RL-based BNAS delivers an efficiency of 0.2 GPU days, which is 2.2× faster than ENAS [6]. However, the training mechanism of BNAS following ENAS leads to terrible unfair training issue [12]. BNAS-v2 [13] was proposed to solve the above issue by a differentiable broad search space with an efficiency of 0.05 GPU days. Admittedly, BNASs achieve satisfactory performance, especially in terms of efficiency. Nevertheless, BCNN suffers two drawbacks as shown in Fig. 1: 1) insufficient representation diversity for feature fusion and enhancement and 2) time-consuming knowledge embedding design.

This article proposes Stacked BNAS whose scalable architecture is named Stacked BCNN, which treats mini BCNN as its basic block. Moreover, each mini BCNN can feed sufficient representations to the GAP layer and enhancement block.
for feature fusion and enhancement, respectively. As a new paradigm of neural networks, this article proves also the universal approximation ability of Stacked BCNN. Furthermore, the knowledge embedding design task is transferred from discrete to continuous space, and learn appropriate knowledge embeddings in a differentiable way to solve the second issue. Our contributions can be summarized as follows.

1) **Stacked BNAS**: Stacked BNAS is proposed to further improve the performance of NAS via Stacked BCNN.

2) **Stacked BCNN**: This article does not only propose a new broad search space dubbed Stacked BCNN for NAS, but also mathematically analyze the universal approximation ability of the proposed Stacked BCNN.

3) **Knowledge Embedding Search (KES)**: An algorithm is also proposed for knowledge embedding design.

4) **Efficiency**: Contributing to the proposed Stacked BCNN and optimization algorithm, Stacked BNAS delivers a state-of-the-art efficiency of 0.02 days on a single NVIDIA GTX 1080Ti GPU.

### II. RELATED WORK

#### A. Neural Architecture Search

Elsken et al. [14] claimed that NAS consists of three components: 1) search space; 2) optimization strategy; and 3) estimation strategy. The search space referred to not only primitive operators, but also the combination paradigm of those candidate operators [15]. As described in [7], there were mainly five types of search spaces: 1) entire structured; 2) cell-based [16]; 3) hierarchical [17]; 4) morphism based [18]; and 5) broad [13]. These search spaces are briefly introduced as below.

1) **Entire-Structured Search Space**: In the entire-structured search space, each layer with a specified operation (e.g., various convolutions and average pooling) was stacked one after another [7]. Beyond that, the skip connection was used in the above search space to explore more complex neural architectures. This search space had two disadvantages:

   1) computationally expensive and 2) insufficient transferability [15].

2) **Cell-Based Search Space**: To mitigate the above issues of the entire-structured search space, Zoph et al. [8] proposed a cell-based search space where each cell with a list of operations was stacked to construct a deep search space. The cell-based search space consists of normal and reduction cells that have different architectures and strides. Moreover, each cell treats the outputs of two predecessors as its inputs. Due to the effectiveness of cell-based search space in terms of efficiency and transferability, many cell-based NAS approaches were proposed, e.g., weight-sharing ENAS [6] and differentiable DARTS [9].

3) **Hierarchical Search Space**: Most cell-based NAS approaches [6], [9] followed a two-level hierarchy: 1) the inner cell level and 2) the outer network level. On the one hand, the inner level chose operation and connection of each intermediate node in the cell. On the other hand, the outer level controlled the spatial-resolution changes. A general formulation [17] was proposed to learn the network-level structure. Liu et al. [19] proposed a hierarchical architecture representation to avoid manually predefining the network block number.

4) **Morphism-Based Search Space**: The morphism-based search space directly modified the existing architecture. Net2Net [20] employed identity morphism (IdMorph) to design architecture based on the existing model. He et al. [21] claimed that there were several issues in IdMorph: 1) limited width and depth changes and 2) identity layer drawback. To solve the above issue, a developed approach named network morphism [18] was proposed. Network morphism adopted a parameter sharing strategy [6] to inherit the knowledge from the parent network to the child network, which grew into a more robust model. Furthermore, network morphism was improved to deliver better performance in terms of optimization algorithm [22] and its level [23]. Furthermore, Chen et al. [24] used hand-crafted and learned blocks to discover novel architecture via parameter inheriting.

5) **Broad Search Space**: Ding et al. [11] proposed a broad search space named BCNN, which employs broad topology to obtain extreme fast search speed and satisfactory performance. BCNN belongs to the cell-based network-level search space. There are three broad search spaces: 1) BCNN [see Fig. 2(a)]; 2) BCNN-CCLE [see Fig. 2(b)]; and 3) BCNN-CCE (see Fig. 1). BCNNs consist of convolution and enhancement blocks which densely connect with the GAP layer for multiscale feature fusion. The main difference among BCNNs is the connection between convolution and enhancement blocks. Similarly, Fang et al. [25] proposed a network-level deep search space named dense search space where the MBConv [26] is treated as its basic block rather than cell. In contrast, each block of dense search space connects to several subsequent blocks. Based on the broad search space, the efficiency of BNAS [11] was 0.2 GPU days via RL. However, BNAS suffers from unfair training issue, so its optimization mechanism does not take full advantage of the BCNN, i.e., the efficiency improvement is limited. Furthermore, a differentiable over-parameterized broad search...
space was proposed to solve the unfair training issue in BNAS-v2 [13]. Experimental results show that BNAS-v2 can deliver 2× faster search efficiency than BNAS when BCNN’s all advantages works.

B. Broad Learning System

Inspired by random vector functional link neural network (RVFLNN) [27] and incremental learning strategy [28], [29], proposed broad learning system (BLS) and several variants [30]. Subsequently, BLSs were widely used in many fields, e.g., image classification [31], noisy data regression [32], wind speed prediction [33], privacy preserving [34], emotion recognition [35], and NAS [36], [37], [38].

In [32], a self-paced learning strategy was proposed to incorporate with BLS to readjust the importance of each sample. Cao et al. [34] proposed a novel method named multiparty secure BLS to solve the task of privacy-preserving machine learning. To combine the superiority of deep model and BLS, Liu et al. [39] proposed Stacked BLS, whose structure is shown in Fig. 3. BLS was the basic block of Stacked BLS whose output was the combination of all BLSs’ outputs. Liu et al. [39] claimed that Stacked BLS not only efficiently optimized trainable weights via an incremental learning algorithm, but also extracted deep representation using multiple BLSs.

III. STACKED BROAD NEURAL ARCHITECTURE SEARCH

In this section, a developed broad search space named Stacked BCNN is first proposed to solve the first issue of vanilla BCNN, i.e., insufficient presentation diversity for feature fusion and enhancement. Then, the KES algorithm is proposed to solve the second issue of vanilla BCNN, i.e., the time consumption of knowledge embedding design by human experts. Finally, the optimization algorithm of Stacked BNAS is given.

A. Search Space: Stacked BCNN

To improve the feature diversity, mini BCNN-based stacked BCNN is proposed. The structures of Stacked BCNN and mini BCNN are shown in Fig. 4.

1) Stacked BCNN: As shown in Fig. 4(a), the proposed Stacked BCNN consists of \(u\) mini BCNNs, where \(u\) is determined by the input size of the first mini BCNN (mini BCNN1).

To preserve the multiscale feature fusion ability of the vanilla BCNN, the Stacked BCNN feeds the output of each mini BCNN into the GAP layer.

2) Mini BCNN: Fig. 4(b) shows the structure of mini BCNN. Mini BCNN consists of \(k + 1\) convolution layers (\(k\) 1-stride deep cells, single 2-stride broad cell) and 1 enhancement cell with stride 1. Deep and broad cells are used for deep and broad feature extraction, respectively. The enhancement cell treats both the outputs of
and knowledge embeddings with respect to mini BCNN that can be obtained by deep and broad cells as inputs to obtain a comprehensive enhancement representation. Furthermore, a family of $1 \times 1$ convolutions is inserted into fixed locations as the knowledge embeddings.

There are two main differences between BCNN and Stacked BCNN: 1) the output of each mini BCNN is the combination of all outputs of three cells rather than the outputs of deep and enhancement cells and 2) the enhancement cell treats the outputs of convolution cells as inputs. The above two differences provide sufficient feature diversity to the GAP layer and enhancement block for representation fusion and enhancement so that better performance can be obtained.

3) **Mathematical Information Flow:** The Stacked BCNN employs a $3 \times 3$ convolution as the stem layer, and its output is treated as the two inputs of the first deep cell of mini BCNN$_1$, denoted as $y^{(1)}_1$ and $y^{(1)}_0$. Similarly, we treat the output of mini BCNN$_i$ as the two inputs of mini BCNN$_{i+1}$.

For mini BCNN$_i$ ($i = 1, 2, \ldots, u$), its output $y^{(i)}$ can be obtained by the outputs of three cells

$$y^{(i)} = \phi \left( \delta^{(i)}_{de}(y^{(i)}_k), \delta^{(i)}_{be}(y^{(i)}_{k+1}), \delta^{(i)}_{eo}(y^{(i)}_{k+2}) \right)$$

where $\phi$ and $\delta^{(i)}_{eo}$ are concatenation of the channel dimension and knowledge embeddings with respect to mini BCNN$_i$’s output, respectively. Moreover, $y^{(i)}_k$ is the output of the last deep cell with a list of operations $\varphi_d$, and can be computed by

$$y^{(i)}_k = \varphi_d(y^{(i)}_{k-2}, y^{(i)}_{k-1}; W^{(i)}_d, \theta_d)$$

and $y^{(i)}_{k+1}$ is the output of the broad cell with a list of operations $\varphi_b$ that can be calculated by

$$y^{(i)}_{k+1} = \varphi_b(y^{(i)}_{k-1}, y^{(i)}_{k}; W^{(i)}_b, \theta_b)$$

and $y^{(i)}_{k+2}$ is the output of the enhancement cell with a list of operations $\varphi_e$ that can be calculated by

$$y^{(i)}_{k+2} = \varphi_e(y^{(i)}_k, \delta^{(i)}_{be}(y^{(i)}_{k+1}); W^{(i)}_e, \theta_e)$$

where $\delta^{(i)}_{be}$ represents knowledge embeddings with respect to the enhancement cell of mini BCNN$_i$ and $W^{(i)}_e$ and $\theta^{(i)}_e$ are the weight and bias matrices of corresponding cells in mini BCNN$_i$, respectively.

The output of Stacked BCNN can be computed by

$$y = \phi \left( y^{(1)}, y^{(2)}, \ldots, y^{(u)} \right).$$

4) **Channel Flow Graph:** As shown in Fig. 4(a), for mini BCNN$_i$, the channel number of the deep cell’s output $c^{(i)}_{\text{deep}}$ can be obtained by

$$c^{(i)}_{\text{deep}} = N_{in} \times 2^{i-1} \times c, \quad i = 1, 2, \ldots, u$$

where $N_{in}$ represents the predefined intermediate node number of the cell and $c$ is the input channel number of mini BCNN$_1$. The channel numbers of broad and enhancement cells’ outputs in mini BCNN$_i$ are both $2 \times c^{(i)}_{\text{deep}}$. For those direct-connected cells and output nodes, corresponding knowledge embedding does not reduce the feature significance. In contrast, the significance of the indirect-connected features is reduced by a factor of a quarter, as shown in Fig. 5(a).

![Fig. 5. Embedding between indirectly connected cells and the output node.](image)

(a) Hand-crafted knowledge embedding. (b) Over-parameterized knowledge embedding module.

**B. Knowledge Embedding Search**

Different from vanilla BNAS, the KES algorithm aims to discover appropriate knowledge embedding module as a basic unit, is proposed to learn appropriate knowledge embedding in a differentiable way rather than designing by human.

1) **Over-Parameterized Knowledge Embedding Module:** For each knowledge embedding on an indirect edge, an over-parameterized knowledge embedding module, as shown in Fig. 5(b), is constructed to discover appropriate knowledge embeddings. Moreover, for the case of $2^u = c_e$, we employ two $1 \times 1$ convolutions with $c_e$ output channels.

There is an assumption that $c_e$ channels are fed into the indirect-connected knowledge embedding. The over-parameterized knowledge embedding consists of $n$ learnable knowledge embeddings with $2^i (i = 1, 2, \ldots, n)$ output channels and a single learnable knowledge embedding with $c_e$ output channels, where $n$ is the largest power of 2 less than $c_e$ determined by

$$\arg \max_n 2^n \quad \text{s.t.} \quad 2^n < c_e.$$  

Subsequently, the output of the over-parameterized knowledge embedding module $y_e$ is obtained by the channel-dimension concatenation of weighted $n + 1$ learnable knowledge embedding outputs as follows:

$$y_e = \phi \left( y_1^{(e)} 1, \ldots, y_{n+1}^{(e)} \right), \sum_{i=1}^{n+1} \gamma_i = 1, \gamma_i \in [0, 1]$$

where $y_i^{(e)}$ and $\gamma_i$ represent the weighted output and weight of $i$th learnable knowledge embedding, respectively.

2) **Learning Strategy:** After over-parameterized knowledge embedding module construction, Stacked BNAS aims to jointly optimize the knowledge embedding weights $\gamma$ and network weights $w$. The goal of KES is to discover $\gamma^*$ that minimizes the validation loss $L_{\text{val}}(w^*, \gamma^*)$, where $w^*$ is obtained by minimizing the training loss $L_{\text{train}}(w, \gamma^*)$. The bilevel optimization problem with lower-level variable $w$ and upper-level variable $\gamma$ can be represented as follows:

$$\min_{\gamma} L_{\text{val}}(w^*(\gamma), \gamma)$$

s.t. $w^*(\gamma) = \arg \min_w L_{\text{train}}(w, \gamma)$.
The optimization of (9) exactly is prohibitive because \( w^*(y) \) needs to be recomputed by the second term of (9) whenever \( y \) takes place any change [9]. Therefore, an approximate iterative optimization process is proposed as follows. For each gradient descent step, network weights \( w \) and knowledge embedding weights \( \gamma \) are optimized by alternating in the network and knowledge embedding spaces, respectively. At step \( t \), given the current knowledge embedding \( \gamma_{t-1} \), \( w_t \) is optimized by descending the training loss \( \mathcal{L}_{train}(w_t, \gamma_{t-1}) \). Subsequently, \( w_t \) is kept fixing and the over-parameterized knowledge embedding module is optimized with learning rate \( \xi \) by descending

\[
\nabla_y \mathcal{L}_{val}(w_t - \xi \nabla_w \mathcal{L}_{train}(w_t, \gamma_{t-1}), \gamma_{t-1}).
\]

Finally, every over-parameterized knowledge embedding module is replaced as the knowledge embedding with the largest weight by taking argmax.

C. Optimization Algorithm

RL [6], evolutionary algorithm [40], [41], and gradient-based algorithm [9] are three common optimization techniques for NAS. In this article, we employ the combination of the most efficient gradient-based optimization algorithm and the proposed Stacked BCNN to achieve extreme fast search efficiency.

To discover a high-performance Stacked BCNN, the gradient-based optimization pipeline is constructed as: 1) a continuous relaxation strategy for over-parameterized Stacked BCNN construction; 2) PCs for memory reduction; and 3) early stopping (ES) for efficiency improvement. In the over-parameterized Stacked BCNN, a mixture operation with respect to all candidate operators is inserted between two nodes of each cell to construct the NAS pipeline for differentiable architecture search. In particular, we call the Stacked BCNN which consists of several mixture operation-based mini BCNNs, as the over-parameterized Stacked BCNN. Moreover, the over-parameterized Stacked BCNN used in Stacked BNAs without KES (dubbed as case 1) and with KES (dubbed as case 2) are different. Case 1, the cell is over-parameterized, and the knowledge embedding is not over-parameterized as in Fig. 5(a). In case 2, for KES, both the cell and knowledge embedding are over-parameterized [see Fig. 5(b)]. The optimization algorithm of Stacked BNAs is shown in Algorithm 1.

1) Continuous Relaxation: In mini BCNN, each cell consists of 2 input nodes \( \{x(0), x(1)\}, N-3 \) intermediate nodes \( \{x(2), \ldots, x(N-2)\} \) and a single output node \( x(N-1) \). Each intermediate node \( x(i) \) can be computed by

\[
x(i) = \sum_{j<i} o(i,j)(x(j))
\]

where \( o(i,j) \) is the operator between \( x(i) \) and \( x(j) \) chosen from candidate operation set \( \mathcal{O} \). Moreover, the outputs of all intermediate nodes are combined to deliver \( x(N-1) \) by channel-dimension concatenation.

Subsequently, the over-parameterized Stacked BCNN is constructed by continuous relaxation [9]. Particularly, edge

\[
(i, j) \text{ of each cell is relaxed for mini BCNN by}
\]

\[
f(i,j)(x(j)) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha(o,i,j))}{\sum_{o' \in \mathcal{O}} \exp(\alpha(o',i,j))} o(x(j))
\]

where operation \( o(\cdot) \) is weighted by architecture weight \( \alpha(o) \).

2) Partial Channel Connections: The strategy of PC [10] is employed to improve the memory efficiency of Stacked BNAs and alleviate the performance collapse issue described in BNAs-v2 [13]. The continuous relaxation of Stacked BNAs with PC can be obtained by

\[
f_{i,j}^{\text{PC}}(x(j); M(i,j)) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha(o,i,j))}{\sum_{o' \in \mathcal{O}} \exp(\alpha(o',i,j))} o(M(i,j) * x(j))
\]

where \( M(i,j) \) represents the mask of the channel sample whose values are chosen from \( \{0, 1\} \).

Moreover, edge normalization is used to mitigate the undesired fluctuation in the search phase via \( \beta(i,j) \) as follows:

\[
x_{i,j}^{\text{PC}} = \sum_{j<i} \sum_{f<j} \frac{\exp(\beta(i,j))}{\sum_{f' < j} \exp(\beta(i,j))} \cdot f(i,j)(x(j))
\]

Finally, each operation of the best architecture is obtained by taking argmax as follows:

\[
o(i,j) = \arg\max_{o \in \mathcal{O}} \frac{\exp(\alpha(o,i,j))}{\sum_{o' \in \mathcal{O}} \exp(\alpha(o',i,j))} \cdot \frac{\exp(\beta(i,j))}{\sum_{f<i} \exp(\beta(i,f))}
\]

Algorithm 1: Stacked BNAs

1. Define \( p \) as early stopping epoch number, \( q \) as architecture repetitive time with zero initialization, \( kes \) as the flags represented using KES or not, \( archprev \) as previous architecture with None initialization;
2. For each edge \( (i, j) \), use (13) and (14) for continuous relaxation as \( f_{i,j}^{\text{PC}}(x(j); M(i,j)) \) parameterized by \( \alpha(i,j) \) and \( \beta(i,j) \);
3. Define architecture weight set \( \Theta = [\alpha, \beta, \gamma] \);
4. if \( kes \) then
5. Use (7) and (8) to construct over-parameterized knowledge embedding module parameterized by \( \gamma \);
6. \( \Theta = [\alpha, \beta] \);
7. end
8. while not converged do
9. Optimize \( w \) by descending \( \nabla_w \mathcal{L}_{train}(w, \Theta) \);
10. Optimize \( \Theta \) by descending \( \nabla_\Theta \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \Theta), \Theta) \);
11. Determine current architecture \( archcurr \) by taking argmax;
12. \( archprev = archcurr \);
13. if \( archcurr == archprev \) then
14. \( q = q + 1 \);
15. \( \text{// repetitive time} \);
16. \( \text{// early stopping} \);
17. \( q = 0 \);
18. \( \text{// repetitive time} \);
19. end
20. end
21. Output \( archcurr \) as the best architecture.
3) Early Stopping: As described in DARTS+ [42], there are two indices for ES: 1) the skip connection number in a single cell and 2) the number of stable epochs. On the one hand, the search procedure is stopped when there is more than one skip connection in one cell to avoid the performance collapse issue. PC contributes to reducing the predominance of skip connections, so this article does not choose the first index. On the other hand, the search procedure is stopped when the rank of architecture weights is no longer changed for a determined number of epochs. This index means that the search procedure stops when arriving at a saturated state. Above all, the second index is chosen for ES of Stacked BNAS using the following criterion:

Criterion 1: Stop the search procedure when the rank of architecture weights is no longer changed for three epochs.

Moreover, in the following sections, we employ repetitive time to determine when Stacked BNAS ES. Repetitive time means the number of consecutive occurrences of a specific architecture in the search procedure. The variable of repetitive time is set to 1 when an architecture appearing first. If the next architecture is identical to the previous one, the repetitive time is set to 2, else 1.

IV. Universal Approximation of Stacked BCNN

Given the initial input channel number $c$, the output of mini BCNN, with $C_i$ channels, i.e., (1), can be rewritten as follows:

$$y^{(i)} = \phi(x; \{\delta^{(i)}, \varphi^{(i)}, W_d^{(i)}, \theta_d^{(i)}, W_b^{(i)}, \theta_b^{(i)}, W_c^{(i)}, \theta_c^{(i)}\})$$

(16)

where $x$ represents input data. After GAP, each channel of $y^{(i)}$ is transformed into a single-pixel neuron-like feature map, so that it can be treated as $C_i$ neurons.

Given standard hypercube $I^d = [0; 1]^d \subseteq \mathbb{R}^d$ and any continuous function $f \in C(I^d)$, the proposed Stacked BCNN can be equivalently represented as follows:

$$f_{\bar{p}_{k,u}} = \sum_{z=1}^{Z} w_z \sigma \left(x; \{\phi, \delta, \varphi, W^{(1)}, \theta^{(1)}, \ldots, W^{(u)}, \theta^{(u)}\} \right)$$

(17)

where $Z = \sum_{i=1}^{u} C_i$ is the neuron number of the GAP output, $w$ represents the weight of the fully connected layer, $\bar{p}_{k,u} = (k, u, c, w_1, \ldots, w_Z, W, \theta)$ represents the set of overall parameters for the Stacked BCNN, and $\sigma$ is the activation function. Given the probability measure $\xi_{k,u}$, randomly generates variables on $\xi_{k,u} = (w_1, \ldots, w_Z, W, \theta)$. For compact set $\Omega$ of $I^d$, the distance between any continuous function and Stacked BCNN can be calculated as follows:

$$\chi_\Omega(f, f_{\bar{p}_{k,u}}) = \sqrt{\mathbb{E} \left[ \int_\Omega (f(x) - f_{\bar{p}_{k,u}}(x))^2 dx \right]}.$$  

(18)

Based on the above hypotheses, a theorem with proof of Stacked BCNN is given below.

Theorem 1: Given any continuous function $f \in C(I^d)$ and any compact set $\Omega \subseteq I^d$, Stacked BCNN with nonconstant bounded functions $\phi$, $\delta$, and $\varphi$, and absolutely integrable activation function $\sigma$ whose definition domain is $I^d$ so that

$$f_{\bar{p}_{k,u}} \sigma^2(x) dx < \infty,$$

has a sequence of $\{f_{\bar{p}_{k,u}}\}$ with probability measures $\xi_{k,u}$ satisfied that

$$\lim_{u \to \infty} \chi_\Omega(f, f_{\bar{p}_{k,u}}) = 0.$$  

(19)

Moreover, the trainable parameters $\xi_{k,u}$ are generated by $\xi_{k,u}^\prime$. Proof: Define input data $x$, nonconstant bounded functions $\phi$, $\delta$, $\varphi$, approximation function $f_{\bar{p}_{k,u}}$ of Stacked BCNN with $u$ mini BCNNs, probability distribution $\xi_{k,u}$ for trainable parameter generation, the weight matrix of fully connected layer $w^\prime = [w^\prime_1, \ldots, w^\prime_Z]^T$ where $Z = \sum_{i=1}^{u} C_i$ and supplement weight $w^\prime = [w^\prime_1, \ldots, w^\prime_{u+1}]^T$.

Stacked BCNN with $u^\prime$ (any integer) mini BCNNs can be computed by

$$f_{\bar{w}^\prime} = \sum_{z=1}^{Z} w_z^\prime \sigma \left(x; \{\phi, \delta, \varphi, W^{(1)}, \theta^{(1)}, \ldots, W^{(u^\prime)}, \theta^{(u^\prime)}\} \right).$$

(20)

Subsequently, Stacked BCNN with input data $x$ can approximate continuous function $f$ with bounded and integrable resident function $f_{\bar{w}^\prime} \in I^d$ as follows:

$$f_{\bar{w}^\prime}(x) = f(x) - f_{\bar{w}^\prime}(x).$$

(21)

As described in previous work [43], for $\forall \varepsilon > 0$, a function $f_{\bar{w}^\prime} \in C(I^d)$ can always be found to satisfy

$$\chi_\Omega(f_{\bar{w}^\prime}, f_{\bar{w}^\prime}) \leq \frac{\varepsilon}{2}.$$  

(22)

An extra mini BCNN (i.e., mini BCNN$u^\prime+1$) is defined to approximate $f_{\bar{w}^\prime}$ with $C_{u^\prime+1}$ channels. Mini BCNN$u^\prime+1$ can be equivalently expressed as follows:

$$f_{\bar{w}^{u^\prime+1}} = \sum_{i=1}^{C_{u^\prime+1}} w_z^{u^\prime} \sigma \left(x; \{\phi, \delta, \varphi, W^{(u^\prime+1)}, \theta^{(u^\prime+1)}\} \right)$$

(23)

Similarly, the composition function $\theta$ in (23) is absolutely integrable. According to [44, Th. 1], for $\forall \varepsilon > 0$, a sequence of $f_{\bar{w}^{u^\prime+1}}$ can be found to satisfy

$$\chi_\Omega(f_{\bar{w}^{u^\prime+1}}, f_{\bar{w}^{u^\prime+1}}) \leq \frac{\varepsilon}{2}.$$  

(24)

Moreover, the output of Stacked BCNN can be rewritten as follows:

$$f_{\bar{p}_{k,u}} = f_{\bar{w}^\prime} + f_{\bar{w}^{u^\prime+1}}.$$  

(25)

The distance between $f$ and $f_{\bar{p}_{k,u}}$ can be obtained by

$$\chi_\Omega(f, f_{\bar{p}_{k,u}}) = \sqrt{\mathbb{E} \left[ \int_\Omega (f(x) - f_{\bar{p}_{k,u}}(x))^2 dx \right]}.$$  

(26)
Therefore, a conclusion can be drawn as follows:

\[
\lim_{u, v \to \infty} a(f, f_{plu}) = 0.
\] (27)

In other words, the proposed Stacked BCNN can completely appropriate any continuous function.

V. Experiments and Results

In this section, the datasets used and implementation details are given first. Next, architecture search of Stacked BNAS without/with KES is introduced. Then, the best-performing architecture learned by Stacked BNAS on CIFAR-10 is transferred to solve large-scale image classification task on ImageNet, and the experimental results are analyzed. Furthermore, the learned architecture is also transferred to four image classification datasets to verify the generalization ability of the proposed Stacked BNAS. Finally, two groups of ablation experiments are performed to verify the effectiveness of Stacked BNAS in terms of efficiency and performance.

A. Datasets and Implementation Details

1) Datasets: CIFAR-10 and ImageNet are used to verify the effectiveness of the proposed Stacked BNAS. CIFAR-10 is a small-scale image classification dataset with 32 × 32 pixels that contains 50K training images and 10K test images. ImageNet contains approximately 1.3 M training data and 50K validation data with various pixels over 1000 object categories.

2) Implementation Details: The data preprocessing technique follows BNAS-v2 for CIFAR-10 and ImageNet. Architecture search without or with KES are performed. For the architecture search, the implementation is repeated with five times. For architecture evaluation, the mean value of three repetitive retraining experiments is treated as the index to determine the best architecture. Furthermore, the best architecture learned on CIFAR-10 is transferred to ImageNet, MNIST, FashionMNIST, NORB, and SVHN.

B. Experiments on CIFAR-10

1) Experimental Settings: As mentioned above, two experiments are implemented on CIFAR-10: 1) Stacked BNAS without KES and 2) Stacked BNAS with KES.

The above two experiments use many identical experimental settings for architecture search, as shown below. The initial input channel number is set to 16. The above over-parameterized Stacked BCNN is trained for 50 epochs. All training images are equally divided into two parts. On the one hand, 25K images are treated as the training data to update the network weights \( w \). On the other hand, another part is used as the validation data to optimize the architecture/embedding weights. To optimize the network weights \( w \), the SGD optimizer is used with a dynamic learning rate using an annealed decay manner, momentum of 0.9, and weight decay of \( 3 \times 10^{-4} \). Beyond that, the Adam optimizer is used to update the architecture/embedding weights, with weight decay of \( 1 \times 10^{-3} \). Architecture search is implemented on a single NVIDIA GTX 1080Ti GPU.

For architecture search of Stacked BNAS without KES, the over-parameterized Stacked BCNN consists of two mini BCNNs, where each one contains one broad cell and one enhancement cell. The batch size and network learning rate are set to 512 and 0.2, respectively. Moreover, the architecture learning rate is set to \( 6 \times 10^{-4} \). For architecture search of Stacked BNAS with KES, the over-parameterized Stacked BCNN consists of two mini BCNNs, where each one contains one deep cell, one broad cell, and one enhancement cell. Due to more memory requirements of KES, the batch size and network learning rate are set to 128 and 0.05, respectively. Beyond that, a larger learning rate of \( 2 \times 10^{-3} \) is set for architecture/embedding weights when knowledge embedding is learned.

For architecture evaluation, identical settings are employed except the number of deep cell which is 2 and 3 for Stacked BNAS without/with KES, respectively. The stacked BCNN consists of two mini BCNNs, where each one contains one broad cell and one enhancement cell. Following BNAS-v2, the learned Stacked BCNN with 44 input channels is trained for 2000 epochs using the SGD optimizer with a batch size of 128, learning rate of 0.05, momentum of 0.9, and weight decay of \( 3 \times 10^{-4} \). Moreover, architecture evaluation is implemented on a single NVIDIA Tesla P100 GPU.

2) Results and Analysis: For Stacked BNAS, the learned architecture is visualized in Fig. 6. For Stacked BNAS with KES, the best architecture and knowledge embedding are shown in Figs. 7 and 8, respectively. Furthermore, Table I summarizes the comparison of the proposed Stacked BNAS with other novel NAS approaches on CIFAR-10.

Fig. 6. Architecture learned by Stacked BNAS without KES. (a) Convolution cell. (b) Enhancement cell.

Fig. 7. Architecture learned by Stacked BNAS with KES. (a) Convolution cell. (b) Enhancement cell.
TABLE I
COMPARISON OF THE PROPOSED STACKED BNAS AND OTHER STATE-OF-THE-ART NAS APPROACHES ON CIFAR-10

| Architecture     | Error (%) | Params (M) | Search Cost (GPU days) | Number of Cells | Search Method | Topology |
|------------------|-----------|------------|------------------------|----------------|---------------|----------|
| LEMONADE [45]    | 3.05      | 4.7        | 80                     | -              | evolution     | deep     |
| DARTS (1st order) [9] | 3.00      | 3.3        | 0.45†                  | 20             | gradient-based | deep     |
| DARTS (2nd order) [9] | 2.76      | 3.3        | 1.50†                  | 20             | gradient-based | deep     |
| DARTS (random) [9] | 3.49      | 3.1        | -                      | 20             | -             | deep     |
| SNAS + mild constraint [46] | 2.98      | 2.9        | 1.50                   | 20             | gradient-based | deep     |
| SNAS + moderate constraint [46] | 2.85      | 2.8        | 1.50                   | 20             | gradient-based | deep     |
| SNAS + aggressive constraint [46] | 3.10      | 2.3        | 1.50                   | 20             | gradient-based | deep     |
| P-DARTS [47]     | 2.50      | 3.4        | 0.30                   | 20             | gradient-based | deep     |
| GDAS-NSAS [48]   | 2.73      | 3.5        | 0.40                   | 20             | gradient-based | deep     |
| PC-DARTS [10]    | 2.57      | 3.6        | 0.10                   | 20             | gradient-based | deep     |
| BNAS [6]         | 2.89      | 4.6        | 0.45                   | 17             | RL            | broad    |
| BNAS [11]        | 2.97      | 4.7        | 0.20                   | 5              | RL            | broad    |
| BNAS-CCLE [11]   | 2.95      | 4.1        | 0.20                   | 5              | RL            | broad    |
| BNAS-CCE [11]    | 2.88      | 4.8        | 0.19                   | 8              | RL            | broad    |
| BNAS-v2 [13]     | 2.79      | 3.7        | 0.05                   | 8              | gradient-based | broad    |
| Random           | 3.12      | 3.1        | -                      | 8              | -             | broad    |
| Stacked BNAS w/o KES (ours) | 2.71      | 3.7        | 0.02                   | 8              | gradient-based | broad    |
| Stacked BNAS w/ KES (ours) | 2.73      | 3.1        | 0.15                   | 10             | gradient-based | broad    |

† Obtained by DARTS using the code publicly released by the authors at https://github.com/quark8/darts on a single NVIDIA GTX 1080Ti GPU.

Fig. 8. Knowledge embedding learned by Stacked BNAS with KES, where \( c = 44 \). (a) First mini BCNN. (b) Second mini BCNN.

Table I shows the comparison of the proposed Stacked BNAS and other state-of-the-art NAS approaches on CIFAR-10. Stacked BNAS with KES discovers a Stacked BCNN with 2.73% test error and just 3.1 M parameters (approximately 16.2% less than Stacked BNAS). Moreover, the over-parameterized knowledge embedding module leads to more trainable parameters and memory requirements than vanilla Stacked BNAS, so larger costs are needed. As shown in Fig. 8, one indirect-connected knowledge embedding of each mini BCNN has more output channels than hand-crafted embedding. The above knowledge embedding changes lead to parameter reduction of the architecture learned by Stacked BNAS with KES.

Compared with NAS approaches with deep topology, Stacked BNAS obtains the best efficiency and competitive accuracy. In terms of random architecture, Stacked BCNN obtains 0.38% better accuracy than DARTS, which further examines the effectiveness of the proposed Stacked BCNN. Furthermore, Stacked BNAS is 5× faster than PC-DARTS whose efficiency ranks the best among all NAS approaches. Compared with BNASs, the proposed Stacked BNAS delivers better efficiency and accuracy. On the one hand, the efficiency of Stacked BNAS is approximately 10× and 2.5× faster than BNAS-v1 and BNAS-v2, respectively. On the other hand, the accuracy of Stacked BNAS is approximately 0.2% and 0.1% better than BNAS-v1 and BNAS-v2, respectively. Compared with previous BNASs, Stacked BNAS with KES can deliver better accuracy with approximately 16% and 33% parameter reduction, respectively.

3) Efficiency Difference Between Stacked BNAS Without or With KES: As shown in Table I, the efficiencies of Stacked BNAS without with KES are 0.02 and 0.15 GPU days, respectively. For that, there are two main reasons: 1) different structures of used Stacked BCNNs in the search phase and 2) various degrees of difficulty meeting Criterion 1.

On the one hand, in the architecture search phase of Stacked BNAS with KES, each mini BCNN contains a deep cell for learning appropriate knowledge embedding. Consequently, the first advantage of fast single-step training speed does not work. On the other hand, Stacked BNAS with KES has difficult satisfying Criterion 1 for ES.

1) The architecture repetitive times of Stacked BNAS are shown in Fig. 9. In the architecture search phase of Stacked BNAS, both the repetitive times of convolution and enhancement cells should be larger than 2. Each implementation can stop early before arriving at the maximum epoch so that Stacked BNAS delivers state-of-the-art efficiency.

2) For Stacked BNAS with KES, the architecture and embedding repetitive times are shown in Fig. 10. In this experiment, the ES strategy is that the convolution cell, enhancement cell, and knowledge embedding
Table II

| Architecture          | Test Err. (%) | Params (M) | Search Cost (GPU days) | FLOPs (M) | Search Method | Topology |
|-----------------------|---------------|------------|------------------------|-----------|---------------|----------|
| AmoebaNet-A [49]      | 25.5          | 8.0        | 5.1                    | 3150      | 555           | evolution | deep     |
| NASNet-A [8]          | 26.0          | 8.4        | 5.3                    | 1800      | 564           | RL       | deep     |
| PNAS [50]             | 25.8          | 8.1        | 5.1                    | 225       | 588           | SMBO     | deep     |
| GHN [51]              | 27.0          | 8.7        | 6.1                    | 0.84      | 569           | SMBO     | deep     |
| DARTS (2nd order) [9] | 26.7          | 8.7        | 4.7                    | 1.50      | 574           | gradient-based | deep |
| SNAS (mild) [46]      | 27.3          | 9.2        | 4.3                    | 1.50      | 522           | gradient-based | deep |
| BayesNAS [52]         | 26.5          | 8.9        | 3.9                    | 0.20      | -             | gradient-based | deep |
| PC-DARTS [10]         | 25.1          | 7.8        | 5.3                    | 0.10      | 586           | gradient-based | deep |
| PC-DARTS (ImageNet) [10] | **24.2**    | **7.3**   | **5.3**               | **3.80**  | **597**       | gradient-based | deep |
| BNAS-v2 (PC) (2nd order) [13] | 27.2       | 8.8        | 4.6                    | 0.09      | 475           | gradient-based | broad |

Stacked BNAS (ours)  | 26.4          | 8.9        | 4.7                    | **0.02**  | 485           | gradient-based | broad |

C. Experiments on ImageNet

1) Experimental Settings: To transfer the architecture learned by Stacked BNAS on ImageNet, three $3 \times 3$ convolutions are treated as the stem layers that reduce the input size from $224 \times 224$ to $28 \times 28$. Subsequently, a Stacked BCNN is constructed by 2 mini BCNNs, where each one contains two deep cells, one broad cell, and one enhancement cell.

For architecture evaluation, the SGD optimizer is chosen with a learning rate of 0.1 followed by a linear decay method, momentum of 0.9, and weight decay of $3 \times 10^{-5}$. Moreover, the Stacked BCNN is trained for 250 epochs with a batch size of 512 on four NVIDIA Tesla P100 GPUs. Other experimental settings follow PC-DARTS.

2) Results and Analysis: Table II summarizes the comparison of the proposed Stacked BNAS with other novel NAS approaches on ImageNet.

Compared with previous impactful NAS approaches [8], [49], [50], the proposed Stacked BNAS delivers competitive or better performance with a state-of-the-art efficiency of 0.02 GPU days which is 5 or 6 magnitudes faster. For efficient NAS approaches [9], [10], Stacked BNAS also obtains competitive or better performance with 1 or 2 magnitudes faster...
efficiency. Compared with BNAS-v2, Stacked BNAS obtains better performance in terms of top-1 and top-5 accuracy. As mentioned above, the main difference between BNAS-v2 and the proposed Stacked BNAS is the broad scalable architecture, so that the effectiveness of Stacked BCNN can be promised. Moreover, the search cost of BNAS-v2 is $4.5 \times$ higher than Stacked BNAS. Due to the broad topology of Stacked BCNN, Stacked BNAS delivers the best performance in terms of FLOPs, which is an important index to show the hardware friendliness of deep models.

D. Experiments on Other Datasets

To further verify the effectiveness of Stacked BNAS, similar with BNAS-v2, the architecture learned on CIFAR-10 is transferred to MNIST, FashionMNIST, NORB, and SVHN. Stacked BNAS employs identical experimental settings for the above four datasets. The Stacked BCNN consists of two mini BCNNs where each one has three deep cells, one broad cells, and one enhancement cells. SGD is chosen as the optimizer to train the Stacked BCNN from scratch for 300 epochs. Here, several important hyper-parameters are listed as follows: a batch size of 128 and an initial learning rate of 0.05. Other training hyper-parameters are identical to BNAS-v2. Similar to BNAS-v2, deep search space-based NAS approaches (e.g., NASNet [8], AmoebaNet [49], DARTS [9], and PC-DARTS [10]) consist of 20 cells and employ identical settings to Stacked BNAS except the learning rate of 0.025. Moreover, BNAS-v2 consists of eight cells and sets the learning rate to 0.025. Experimental results are shown in Table III.

Compared with the comparative approaches, Stacked BNAS delivers the best generalization ability. For MNIST, both Stacked BNAS and NASNet obtain the best performance, i.e., 99.64% accuracy. For FashionMNIST, NASNet also delivers the highest accuracy. The accuracy of Stacked BNAS is second-ranked. For NORB, the accuracy of Stacked BNAS is also second-ranked and AmoebaNet performs the best. For SVHN, Stacked BNAS is tied for the first place with 97.12% accuracy. Above all, Stacked BNAS shows the best generalization ability on all tasks.

E. Ablation Studies

Here, two groups of experiments are implemented: 1) one is to examine the effectiveness of PC and ES for efficiency improvement of Stacked BNAS and 2) the other is to verify the effectiveness of two scales of information, i.e., the output of the broad cell to the output node is denoted as b2o and the output of the deep cell to the enhancement cell is denoted as d2e.

1) PC and ES for Efficiency Improvement: In this group of experiments, there are four cases: 1) using neither PC nor ES; 2) using only PC; 3) using only ES; and 4) using both PC and ES. Each case is repeatedly performed five times following the experimental setting used for architecture search of Stacked BNAS without KES. Experimental results are shown in Table IV.

In case 1, the search cost of Stacked BNAS without PC and ES is 0.14 GPU days under a single NVIDIA GTX 1080Ti GPU. In case 2, Stacked BNAS employs PC to deliver an efficiency of 0.068 GPU days, which is $2 \times$ faster than case 1, because the strategy of PC contributes to improving a higher memory efficiency (using a larger batch size, i.e., 512) of Stacked BNAS than case 1. In the case of using ES, Stacked BNAS can stop the search phase at the 15th epoch and obtain an efficiency of 0.047 GPU days. When using both PC and ES, Stacked BNAS can not only search architecture with efficient memory of 512 batch size, but also stop early at the 12th epoch and delivers a state-of-the-art efficiency of 0.018 GPU days. Above all, both PC and ES play important roles in the efficiency improvement of Stacked BNAS.

2) Multiscale Feature Fusion for Performance Improvement: In this group of experiments, there are four cases: 1) using neither b2o nor d2e; 2) using only b2o; 3) using only d2e; and 4) using both b2o and d2e. Each case is repeatedly performed three times following the experimental setting used for architecture evaluation of Stacked BNAS without KES. Moreover, the mean accuracy of three repetitive experiments is treated as the final result. Experimental results are shown in Table V.

When using neither b2o nor d2e, Stacked BCNN degrades into vanilla BCNN, which lacks feature diversity for representation fusion and enhancement, so its test error is 3.14%. For case 2, each mini BCNN employs the scale information from the broad cell to the output node and delivers 96.95%
accuracy, which is approximately 0.1% higher than case 1. Compared with the scale information of b2o, d2e shows a greater contribution to performance improvement and obtains 97.12% accuracy, which is approximately 0.3% higher than case 1. In the last case, the combination of b2o and d2e further improves the performance of Stacked BCNN, i.e., 2.71% test error. Above all, multiscale feature fusion is effective for performance improvement of Stacked BCNN.

VI. CONCLUSION

BNASs deliver state-of-the-art efficiency via a novel broad scalable architecture named BCNN, which employs multiscale feature fusion and hand-crafted knowledge embedding to yield satisfactory performance with shallow topology. However, there are two issues in BCNN: 1) feature diversity loss for representation fusion and enhancement and 2) time consumption of knowledge embedding design. To solve the above issues, this article proposes Stacked BNAS. On the one hand, Stacked BNAS proposes a new broad scalable architecture named Stacked BCNN that can provide more feature diversities for representation fusion and enhancement than vanilla BCNN. On the other hand, a differentiable algorithm named KES is also proposed to learn appropriate knowledge embedding for Stacked BCNN in an automatic way instead of designing by machine learning experts. Benefiting from the combination of Stacked BCNN and an efficient optimization algorithm, the proposed Stacked BNAS delivers a state-of-the-art efficiency of 0.02 GPU days with competitive performance. Moreover, KES contributes to discovering a high-performance Stacked BCNN with fewer parameter counts. Moreover, the proposed Stacked BNAS shows powerful generalization ability on four image classification datasets.

Nevertheless, there has also a performance gap between deep and broad search spaces in terms of accuracy on ImageNet. In the future, a knowledge distillation technique will be introduced to improve the performance on ImageNet.

REFERENCES

[1] S. Wen et al., “Multilabel image classification via feature/label co-projection,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 51, no. 11, pp. 7250–7259, Nov. 2021.
[2] X. Tao, D. Zhang, Z. Wang, X. Liu, H. Zhang, and D. Xu, “Detection of power line insulator defects using aerial images analyzed with convolutional neural networks,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 50, no. 4, pp. 1486–1498, Apr. 2020.
[3] K. Zhang et al., “Multilevel image-enhanced sentence representation net for natural language inference,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 51, no. 6, pp. 3781–3795, Jun. 2021.
[4] J. Li et al., “ABCP: Automatic block-wise and channel-wise network pruning via joint search,” IEEE Trans. Cogn. Devel. Syst., early access, Dec. 20, 2022, doi: 10.1109/TCDS.2022.3230858.
[5] N. Li, Y. Pan, Y. Chen, Z. Ding, D. Zhao, and Z. Xu, “Hierarchic rank selection with progressively searching tensor ring network,” Complex Intell. Syst., vol. 8, pp. 771–785, Apr. 2022.
[6] H. Pham, M. Guo, B. Zoph, Q. Le, and J. Dean, “Efficient neural architecture search via parameters sharing,” in Proc. ICLR, 2018, pp. 4095–4104.
[7] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in Proc. ICLR, 2017, pp. 1–16.
[8] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in Proc. CVPR, 2018, pp. 8697–8710.
[9] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” in Proc. ICLR, 2018, pp. 1–13.
[10] Y. Xu et al., “PC-DARTS: Partial channel connections for memory-efficient architecture search,” in Proc. ICLR, 2020, pp. 1–13.
[11] 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., vol. 33, no. 9, pp. 5004–5018, Sep. 2022.
[12] X. Chu, B. Zhang, and R. Xu, “FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search,” 2019, arXiv:1907.01845.
[13] Z. Ding, Y. Chen, N. Li, and D. Zhao, “BNAS-v2: Memory-efficient and performance-collapse-prevented broad neural architecture search,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 52, no. 10, pp. 6259–6272, Oct. 2022.
[14] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” J. Mach. Learn. Res., vol. 20, no. 1, pp. 1997–2017, 2019.
[15] X. He, K. Zhao, and X. Chu, “AutoML: A survey of the state-of-the-art,” Knowl. Based Syst., vol. 212, Jan. 2021, Art. no. 106622.
[16] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, “Practical block-wise neural network architecture generation,” in Proc. CVPR, 2018, pp. 2423–2432.
[17] C. Liu et al., “Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation,” in Proc. CVPR, 2019, pp. 82–92.
[18] T. Wei, C. Wang, Y. Rui, and C. W. Chen, “Network morphism,” in Proc. IJCAI, 2016, pp. 564–572.
[19] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, “Hierarchical representations for efficient architecture search,” in Proc. ICLR, 2018, pp. 1–13.
[20] T. Chen, I. Goodfellow, and J. Shlens, “Net2Net: Accelerating learning via knowledge transfer,” 2015, arXiv:1511.05641.
[21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. CVPR, 2016, pp. 770–778.
[22] H. Jin, Q. Song, and X. Hu, “Auto-keras: An efficient neural architecture search system,” in Proc. SIGKDD, 2019, pp. 1946–1956.
[23] T. Wei, C. Wang, and C. W. Chen, “Modularized morphing of neural networks,” 2017, arXiv:1701.03281.
[24] Y. Chen, R. Gao, F. Liu, and D. Zhao, “ModuleNet: Knowledge-inherited neural architectural search,” 2020, arXiv:2004.05020.
[25] J. Fang, Y. Sun, Q. Zhang, Y. Li, W. Liu, and X. Wang, “Densely connected search space for more flexible neural architecture search,” in Proc. CVPR, 2020, pp. 10628–10637.
[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in Proc. CVPR, 2018, pp. 4510–4520.
[27] Y.-H. Pao and Y. Takefuji, “Functional-link net computing: Theory, system architecture, and functionalities,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 9, pp. 5004–5018, Sep. 2022.
[28] C. L. P. Chen, Z. Liu, and S. Feng, “Universal approximation capability of feed-forward neural network,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 29, no. 1, pp. 76–79, May 1992.
[29] C. L. P. Chen and J. Z. Wan, “A rapid learning and dynamic stepwise updating algorithm for flat neural networks and the application to time-series prediction,” IEEE Trans. Syst., Man, Cybern., B., Cybern., vol. 29, no. 1, pp. 62–72, Feb. 1999.
[30] C. L. P. Chen and Z. Liu, “Broad learning system: An effective and efficient incremental learning system without the need for deep architecture,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 1, pp. 10–24, Jan. 2018.
[31] C. L. P. Chen, Z. Liu, and S. Feng, “Universal approximation capability of broad learning system and its structural variations,” IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 4, pp. 1191–1204, Apr. 2019.
[32] H. Zhao, J. Zheng, W. Deng, and Y. Song, “Semi-supervised broad learning system based on manifold regularization and broad network,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 67, no. 3, pp. 983–994, Mar. 2020.
[32] L. Liu, L. Cai, T. Xie, and Y. Wang, “Self-paced broad learning system,” IEEE Trans. Cybern., early access, Jun. 29, 2022, doi: 10.1109/TCYB.2022.3181449.

[33] X. Xie, S. Hu, P. Shi, H. Shao, R. Li, and Z. Li, “Natural phase space reconstruction-based broad learning system for short-term wind speed prediction: Case studies of an offshore wind farm,” Energy, vol. 262, Jan. 2023, Art. no. 125342.

[34] X.-K. Cao, C.-D. Wang, J.-H. Lai, Q. Huang, and C. L. P. Chen, “Multiparty secure broad learning system for privacy preserving,” IEEE Trans. Cybern., early access, Jan. 24, 2023, doi: 10.1109/TCYB.2023.3235496.

[35] T. Zhang, X. Wang, X. Xu, and C. L. P. Chen, “GCB-net: Graph convolutional broad network and its application in emotion recognition,” IEEE Trans. Affect. Comput., vol. 13, no. 1, pp. 379–388, Jan.–Mar. 2022.

[36] N. Li, Y. Chen, and D. Zhao, “Dense attention: A densely connected attention-based vision transformer,” in Proc. Int. Joint Conf. Neural Netw. (IJCNN), 2023.

[37] T. Zhang, C. Lei, Z. Zhang, X.-B. Meng, and C. L. P. Chen, “AS-NAS: Adaptive scalable neural architecture search with reinforced evolutionary algorithm for deep learning,” IEEE Trans. Evol. Comput., vol. 25, no. 5, pp. 830–841, Oct. 2021.

[38] N. Li, Y. Chen, W. Li, Z. Ding, D. Zhao, and S. Nie, “BVIT: Broad attention-based vision transformer,” IEEE Trans. Neural Netw. Learn. Syst., early access, May 1, 2023, doi: 10.1109/TNNLS.2023.3264730.

[39] Z. Liu, C. L. P. Chen, S. Feng, Q. Feng, and T. Zhang, “Stacked broad learning system: From incremental flattened structure to deep model,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 51, no. 1, pp. 209–222, Jan. 2021.

[40] R. Shang, L. Jiao, F. Liu, and W. Ma, “A novel immune clonal algorithm for MO problems,” IEEE Trans. Evol. Comput., vol. 16, no. 1, pp. 35–50, Feb. 2012.

[41] R. Shang, K. Dai, L. Jiao, and R. Stolkin, “Improved memetic algorithm based on route distance grouping for multiobjective large scale capacitated arc routing problems,” IEEE Trans. Cybern., vol. 46, no. 4, pp. 1000–1013, Apr. 2016.

[42] H. Liang et al., “DARTS+: Improved differentiable architecture search with early stopping,” 2019, arXiv:1909.06335.

[43] W. Rudin, Real and Complex Analysis, New Delhi, India: Tata McGraw-Hill Educ., 2006.

[44] B. Igelnik and Y.-H. Pao, “Stochastic choice of basis functions in adaptive function approximation and the functional-link net,” IEEE Trans. Neural Netw., vol. 6, no. 6, pp. 1320–1329, Nov. 1995.

[45] T. Elsken, J. H. Metzen, and F. Hutter, “Efficient multi-objective neural architecture search via Lamarckian evolution,” in Proc. ICLR, 2018, pp. 1–23.

[46] S. Xie, H. Zheng, C. Liu, and L. Lin, “SNAS: Stochastic neural architecture search,” in Proc. ICLR, 2018, pp. 1–17.

[47] X. Chen, L. Xie, J. Wu, and Q. Tian, “Progressive differentiable architecture search: Bridging the depth gap between search and evaluation,” in Proc. ECCV, 2019, pp. 1294–1303.

[48] M. Zhang, H. Li, S. Pan, X. Chang, and S. Su, “Overcoming multi-model forgetting in one-shot NAS with diversity maximization,” in Proc. CVPR, 2020, pp. 7809–7818.

[49] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in Proc. AAAI, vol. 33, 2019, pp. 4780–4789.

[50] C. Liu et al., “Progressive neural architecture search,” in Proc. ECCV, 2018, pp. 19–34.

[51] C. Zhang, M. Ren, and R. Urtasun, “Graph hypernetworks for neural architecture search,” in Proc. ICLR, 2019, pp. 1–17.

[52] H. Zhou, M. Yang, J. Wang, and W. Pan, “BayesNAS: A Bayesian approach for neural architecture search,” in Proc. ICML, 2019, pp. 7603–7613.

Xiziang Ding received the Ph.D. degree in computer applied technology from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2022. He is currently working with Meitu, Beijing. His research interests include large language model, model compression, nature language processing, computer vision, neural architecture search, and deep reinforcement learning.

Yaran Chen (Member, IEEE) received the Ph.D. degree in control theory and control engineering 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 Multimodal Artificial Intelligence 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.

Nannan Li (Graduate Student Member, IEEE) received the B.S. degree in automation from the School of Automation Engineering, University of Electronic Science and Technology of China, Sichuan, China, in 2017. She is currently pursuing the Ph.D. degree in control theory and control engineering with The State Key Laboratory of Multimodal Artificial Intelligence Systems, Institute of Automation, Chinese Academy of Sciences, Beijing, China, and the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. Her research interests include computer vision and neural architecture search.

Dongbin Zhao (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees in material processing engineering from the Harbin Institute of Technology, Harbin, China, in 1994, 1996, and 2000, respectively. He is currently a Professor with the Institute of Automation, Chinese Academy of Sciences, Beijing, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. He has published six books, and over 100 international journal papers. His current research interests are in the area of deep reinforcement learning, computational intelligence, autonomous driving, game artificial intelligence, and robotics.

C. L. Philip Chen (Fellow, IEEE) received the master’s degree in electrical and computer engineering from the University of Michigan at Ann Arbor, Ann Arbor, MI, USA, in 1985, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 1988.

He is the Chair Professor and the Dean of the College of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His current research interests include cybernetics, systems, and computational intelligence.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.