Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Layered Feature Representation for Differentiable Architecture Search

Jie Hao
University of Electronic Science and Technology of China
William Zhu (wfzhu@uestc.edu.cn)
University of Electronic Science and Technology of China https://orcid.org/0000-0001-8898-9244

Research Article

Keywords: Neural architecture search, Di erentiable, Layered feature representation, Candidate operations

Posted Date: December 6th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-1086452/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Version of Record: A version of this preprint was published at Soft Computing on April 9th, 2022. See the published version at https://doi.org/10.1007/s00500-022-06907-1.

Noname manuscript No.
(will be inserted by the editor)

Layered feature representation for differentiable architecture search

Jie Hao · William Zhu*

Received: date / Accepted: date

Abstract Differentiable architecture search (DARTS) approach has made great progress in reducing the computational costs of designing automatically neural architectures. DARTS tries to discover an optimal architecture module, called as the cell, from a predefined super network containing all possible network architectures. Then a target network is constructed by repeatedly stacking this cell multiple times and connecting each one end to end. However, the repeated design pattern in depth-wise of networks fails to sufficiently extract layered features distributed in images or other media data, leading to poor network performance and generality. To address this problem, we propose an effective approach called *Layered Feature Representation for Differentiable Architecture Search* (LFR-DARTS). Specifically, we iteratively search for multiple cell architectures from shallow to deep layers of the super network. For each iteration, we optimize the architecture of a cell by gradient descent and prune out weak connections from this cell. Meanwhile, the super network is deepen by increasing the number of this cell to create an adaptive network context to search for a depth-adaptive cell in the next iteration. Thus, our LFR-DARTS can obtain the cell architecture at a specific network depth, which embeds the ability of layered feature representations into each cell to sufficiently extract layered features of data. Extensive experiments show that our algorithm solve the existing problem and achieves a more competitive performance on the datasets of CIFAR10 (2.45% error rate) , fashionMNIST (3.70%) and ImageNet (25.5%) while at low search costs.

Keywords Neural architecture search · Differentiable · Layered feature representation · Candidate operations

William Zhu*
Institute of Fundamental and Frontier Sciences at the University of Electronic Science and Technology of China, No.4, Sec. 2, Jianshe Beilu, Chengdu, China
Tel.: +086+15928727490
E-mail: wfzhu@uestc.edu.cn
ORCID: 0000-0001-8898-9244

1 Introduction

Over the last few years, deep neural networks (DNN) have demonstrated powerful capabilities of feature extraction [31,13,7,28] and data mining [4,48,6,40]. Thus, DNN is applied to a large variety of challenging tasks, such as image recognition [18,14], speech recognition [16,12], machine translation [37,42], and other complex tasks [34,26,15,3]. But designing an advanced neural network typically requires substantial efforts of human experts. To eliminate such a handcraft process, neural architecture search (NAS) [49,50,32] has been proposed to automatically search for a suitable neural network from a predefined search space. Its excellent performance has increasingly attracted researchers' attention.

Most NAS approaches apply reinforcement learning (RL) [49,50,2] or evolutionary algorithms (EA) [33, 24,32] to perform architecture search. Both of their searching procedure requires sampling and evaluating numerous architectures from a discrete search space to obtain the optimal one. The searching procedure is prohibitively computational overhead. For example, NASNet [50](RL-based) trains and evaluates more than 20,000 neural networks across 500 GPUs over 4 days. AmoebaNet [32] (EA-based) even takes 3150 GPU-days to discover an optimal neural architecture.

To eliminate this high computational overhead, Liu *et al.* [25] recently proposed a differentiable architecture search (aka DARTS), which relaxes the discrete

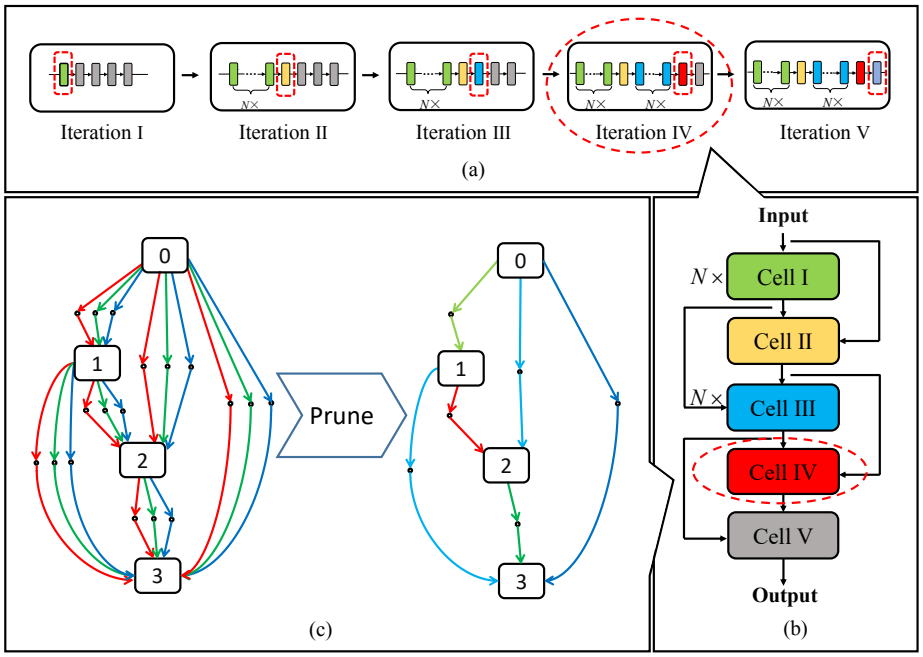


Fig. 1 Illustration of the proposed approach, *Layered Feature Representation for Differentiable Architecture Search* (LFR-DARTS). (a) The overall pipeline of LFR-DARTS. Suppose there are 5 target cells to be searched, and each cell is searched in an iteration. Note that the network depth gradually increases in each iteration, and this process is displayed in each box. (b) The fine structure of the search network containing multiple cells in a iteration. (c) Prune out weak operations for a cell currently being searched.

search space to be continuous and optimizes a common cell architecture in a super network (also called search network in the following) by gradient descent. Then, the identical cells are repeatedly stacked multiple times and connected end to end to construct a target network for a specific task. This kind of NAS approach indeed reduces the computational costs by the differentiable search strategy. However, this target network shows poor performance on testing datasets, especially when transferred to a large-scale dataset since this repeated and simple network structure in depth-wise is hard to sufficiently extract the layered features distributed in media data. In term of image data, the layered features express semantic information of different granularities. In general, the semantic information need to be handled by convolutional kernels with different configurations. But obviously the simple neural architecture from DARTS cannot fully extract and utilize these useful features. Therefore, how to search for cell architectures with layered feature representation for a target network becomes our research question.

To address the above problem, we propose an effective approach called *Layered Feature Representation for Differentiable Architecture Search* (LFR-DARTS). Specifically, we initialize a search network constituted by multiple cells with all candidate operations and then iteratively search for the architecture of each cell from shallow to deep layers of the search network. For each iteration, we first optimize the architecture of a specified cell by gradient descent and gradually prune out weak connections from this cell. To effectively learn the importance of candidate operations and highlight the optimal ones during this process, we design a new functional network layer called Normalization-Affine and introduce an entropy constraint for the operations being optimized. When obtaining the optimal architecture of a cell, we deepen the search network by increasing the number of this cell to N (a configurable hyperparameter) copies in the original location of network while keeping other cells unchanged, so as to create an adaptive network context to search for a depth-adaptive cell in the next iteration. Therefore, our LFR-DARTS makes each cell to be searched at a specific and adaptive network depth, which is conducive to embedding the ability of layered feature representations into each cell to sufficiently extract data features.

In terms of search efficiency, our approach takes shorter search time than DARTS since we constantly prune out weak operations from the search network to progressively accelerate the forward and backward propagation of the network. Moreover, the optimization for cell architecture is simpler yet more efficient compared to DARTS, which is demonstrated by the diagnostic experiments in Sec. 4.3.

We validate our LFR-DARTS on the image classification tasks of CIFAR10, fashionMNIST and ImageNet. We take only 0.45 GPU days (NVIDIA GTX1080Ti) to obtain an optimal neural architecture on the training dataset of CIFAR10. Our neural network achieves the state-of-the-art performance on validation dataset of CIFAR10 (*i.e.,* 2.65% test error rate with 2.7M parameters and 2.45% test error rate with 4.4M parameters). Then we transfer the neural architecture to other datasets of fashionMNIST and ImageNet. Under the same circumstances, our network achieves 3.70% test error rate on fashionMNIST (with 2.5M parameters) and 74.5% top1 accuracy on ImageNet (with only 4.9M parameters).

In summary, we make the following contributions in this work:

1. We propose a layered feature representation approach for differentiable architecture search to solve the problem of insufficient layered feature extraction in DARTS. Firstly, we design a hierarchical search scheme that is to search a depth-adaptive cell architecture in each search iteration. At the end of each iteration, we dynamically increase the number of the currently obtained cell to N copies in the original depth location so as to deepen the search network. Compared with other differentiable search approaches, our hierarchical and dynamic search scheme allows the discovered network to sufficiently extract feature information of different granularities and levels and integrate it to make decisions.

2. A new functional network layer (called as Normalization-Affine) and the entropy constraint are developed to highlight important operations among candidates, while suppressing other weak operations. That provides higher reliability for optimal architecture selection.

3. Extensive experiments show the advantages of our method in neural architecture search. Compared to other DARTS approaches, our discovered cells are able to represent different levels of feature information hidden in data. Therefore, our algorithm achieves competitive even better network performance and generalization on several datasets.

2 Related Work

In recent years, NAS is becoming a research hotspot in artificial intelligence. Many search algorithms have been proposed to explore neural networks. According to the strategies to explore the search space, the existing NAS approaches can be roughly divided into three categories [11], *i.e.*, reinforcement learning (RL)-based approaches, evolution algorithm (EA)-based approaches, and gradient-based approaches.

The early approaches [49, 50, 1, 5, 23] use RL to optimize the search policy for discovering optimal architectures. NASNet [50] trains a recurrent neural network as a controller to decide the types and parameters of neural networks sequentially. ENAS [30] reduces the computational burden of NASNet by sharing the weights of common operations among child networks. The EA-based methods apply evolutionary algorithms to evolve and optimize a population of network structures [33, 32, 41, 27]. AmoebaNet [32] encodes each neural architecture as a variable-length string. The string mutates and recombines to produce new population of networks. The high-performance networks will be remained and they generates the next promising generation.

But both RL-based and EA-based approaches require excessive computational overhead though achieving an advanced performance. To address this issue, the gradient-based approaches [25, 23, 44, 10] are proposed to accelerate the architecture search. Typically, DARTS relaxes the discrete search space to be continuous and utilizes gradient descent to jointly optimize neural architecture and network weights. SNAS [44] proposes to constrain the architecture parameters to be one-hot to tackle the inconsistency in optimizing objectives between search and evaluation scenarios. GDAS [10] develops a differentiable sampler over the search space to avoid simultaneously training all the neural architectures in the space. DARTS+ [22] RobustDARTS [46] and PDARTS [44] employ early stopping to restrict the excessive number of "skip" operations. FairDARTS [8] proposes the collaborative competition strategy to address the unfair advantage in exclusive competition. NASSA [21] designs a new importance metric of candidate operations for more reliable architecture selection. Although the gradient-based approaches show high search efficiency, their network structures lack the ability of layered feature representations.

3 Method

In this section, we present our proposed algorithm *Layered Feature Representation for Differentiable Architecture Search* (LFR-DARTS) in detail. We first introduce a classical differentiable NAS algorithm DARTS in Sec. 3.1, which is a basis of our LFR-DARTS. Then, we describe the concrete search procedure of our algorithm in Sec. 3.2. Finally, in Sec. 3.3, we introduce a minimum entropy constraint and formulate the gradient optimization for the search network.

Operation space	
• zeroize	• max_pool_3x3
• avg_pool_3x3	• skip_connect
• sep_conv_3x3	• sep_conv_5x5
• dil_conv_3x3	• dil_conv_5x5

Table 1 The operation space for neural architecture search.

3.1 Preliminary: DARTS

In DARTS, the goal of architecture search is to discover an optimal cell with the most important operations from a search network. The search network consists of L identical cells with the given candidate operations. These cells connect with each other in order, and each cell is considered as a directed acyclic graph (DAG) with B nodes, $\{x_0, x_1, ..., x_{B-1}\}$, where x_0, x_1 are two input nodes of this cell, x_{B-1} is the output node, and the others are intermediate nodes. The nodes are connected to predecessors by multiple kinds of operations (*e.g., convolution, pooling*). These operations share an operation space $\mathcal{O}$ 1, in which each operation is represented as $o(.)$. The feature transformation $f(.)$ from node i to the subsequent node j could be represented by the weighted sum of these operations:

$$f_{(i,j)}(x_i) = \sum_{k=1}^{|\mathcal{O}|} \frac{\exp(\alpha_{o_k}^{(i,j)})}{\sum_{m=1}^{|\mathcal{O}|} \exp(\alpha_{o_m}^{(i,j)})} o_k(x_i) \tag{1}$$

where x_i is the feature maps of node i, and α is the architecture parameter, which is used to weight its corresponding operation.

Each intermediate node ($\{x_2, x_3, ..., x_{B-2}\}$) is represented by all of its predecessors:

$$x_j = \sum_{i<j} f_{(i,j)}(x_i) \tag{2}$$

The output x_{B-1} of one cell is calculated by the concatenation of the intermediate nodes in the channel dimension:

$$x_{B-1} = \text{concat}(x_2, x_3, ..., x_{B-2}) \tag{3}$$

The output of this cell will be input of the next cell. The cell is a special information processing or feature extraction block. Thus, the internal architecture (including operation types and connection between nodes) of the cell is critical to the performance of a neural network.

3.2 The procedure of layered architecture search

A convolutional neural network (CNN) has a hierarchical structure so as to extract the layered visual features of images. As [35,45] describes, the discriminative information is hidden in feature maps of different layers, each layer has the characteristic of representing specific features. Many excellent network structures [38,14,18] obey this rule consistently. But differentiable NAS algorithms [50,25] just search single cell architecture (a normal cell and a reduction cell) in pre-defined search space, and then construct a target neural network by the repetitive cells. It contradicts the common sense and cannot be guaranteed that the neural network with repetitive and oversimplified structure is capable of sufficiently extracting layered features. It causes poor performance, especially when transferring the cell architecture to a large-scale dataset.

Following the characteristics of neural networks, we propose a new differentiable NAS algorithm called *Layered Feature Representation for Differentiable Architecture Search* (LFR-DARTS). Firstly, we specify the number of target cells to be searched and initialize a search network by a few identical cells that contain the same structure and candidate operations inside. These cells are connected in order, which makes each cell naturally placed in the different depths of the search network. Then, we iteratively search for multiple cells with different architectures from shallow to deep layers. For each iteration, we first optimize the architecture of a cell by gradient descent and gradually prune out weak connections from this cell. Once a cell discovers its optimal architecture, we will fix its architecture in the search network and then perform the search for a deeper-adaptive cell in the next iteration.

In order to embed the capability of layered feature representation into the cells, we dynamically increase the depth of the search network during the search process, rather than keeping the static state as in DARTS. Concretely, when we discover the optimal architecture of a cell (if it is a normal cell), we will increase its number to N copies in the original depth of the search network while simultaneously keeping other cells unchanged. In this way, our gradually growing search network creates an adaptive network context for searching optimal cells adaptive to different network depths.

But we find that there exist some problems when applying the architecture optimization strategy of DARTS to our search process. First, this optimization strategy in DARTS is just applied to searching a single cell, not multiple cells with different hierarchical features. Since the parameters α for architecture optimization in DARTS are shared between cells, leading to optimizing and producing only a common cell. Second, the search procedure is complicated as DARTS needs to alternatively optimize the architecture parameters α and network weights ω by gradient descent. α

is trained on the validation dataset and ω is trained on the training dataset respectively, which greatly consumes search time. To solve the problems of architecture optimization, we design a new functional layer called Normalization-Affine (NA), which follows intermediately after each candidate operation and provide us a selection indicator of optimal operations.

For any candidate operation, our NA functional layer first normalizes the output of this operation and then reweights the normalized result by a trainable parameter to learn its importance. We formulate the NA layer for any k-th operation in a set of candidate operations:

$$\mathrm{NA}_k(x_{in}) = \varphi_k \times norm(x_{in}) \tag{4}$$

$$norm(x_{in}) = \frac{x_{in} - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{5}$$

where the trainable weight parameter φ_k is referred to as an affine parameter which is used to weight each operation. ϵ is a very small value close to zero. $x_{in} = \{x_{in}^1, x_{in}^2, ..., x_{in}^m\}$ is the intput tensor of the NA layer and the output tensor of the k-th operation, and it contains m feature maps. $\mu = \{\mu_1, \mu_2, ..., \mu_m\}$, $\sigma = \{\sigma_1, \sigma_2, ..., \sigma_m\}$ are mean vector and standard deviation vector of the mini-batch x_{in}. μ and σ also contain m elements, and each element is corresponding to a feature map of x_{in}. The normalized function $norm(.)$ partially comes from Batch Normalization [20], which is one of the most common and useful normalization approaches in CNN models.

We combine Eq. 1, Eq. 4 and 5, get information conversion from node j to i, shown as Eq. 6:

$$\mathcal{F}_{ij}(x_j) = \sum_{k=1}^{|\mathcal{O}|} \varphi_k \times \frac{\hat{x}_j^k - \mu_k}{\sqrt{{\sigma_k}^2 + \epsilon}} \tag{6}$$

$$\hat{x}_j^k = o_k(x_j) \tag{7}$$

where $o_k(.)$ is the k-th operation in a set of candidate operations and x_j is the input of the operation. μ_k, σ_k denote the mini-batch mean and mini-batch standard deviation vector of the output of k-th operation.

Each NA layer, corresponding to any operation, contains a learnable affine parameter φ, which is trained and updated together with weight parameters ω by gradient descent. Since different cell is located in different depth of network, the affine parameters of the cell will be trained to learn the layered neural architectures. In addition, we optimize affine parameters and weight parameters in the same gradient descent step rather than alternate optimization, which saves half of the search time compared to DARTS.

Our dynamic search approach gradually prune out the weak operations from search network based on the affine parameters. The importance score S of an operation between any pair of nodes is defined as follows:

$$S_k = softmax(|\varphi_k|) = \frac{\exp(|\varphi_k|)}{\sum_{j=1}^{|\mathcal{O}|} \exp(|\varphi_j|)} \tag{8}$$

where φ_k denotes the affine parameter corresponding to the k-th operation in the operation space. The larger S_k is, the more likely the corresponding candidate operation is to be retained during the search process.

We might doubt whether the normalization is really necessary in the NA layer. We have found through experiments that directly using the affine parameter to weight an operation without beforehand normalization cannot achieve an ideal result. The reason is that the distribution of the outputs from different operations probably varies widely, which makes it pretty hard to identify importance of operations by the affine parameters φ. For any operation, its weight parameters ω will be optimized and updated simultaneously with the corresponding φ. Optimizing the two kinds of parameters together could make them vary synchronously, resulting in same result by increasing one and deceasing another. Therefore, normalization before reweighting is quite necessary since it makes the results from different operations uniformed so that the affine parameters can genuinely represent the importance of operations.

3.3 Network optimization with entropy constrain

During the search process, we optimize the affine parameters φ together with weights ω by the gradient descent. Then we try to pick out the operation with the highest importance score from the candidates. But the importance scores of operations between a pair of nodes could be very close to each other, which makes it challenging to select an optimal operation among them. Thus we consider adding an entropy constraint over these candidate operations to concentrate the high scores on one or few operations, then the operations with high scores can be identified and selected more easily. To this end, we redesign the loss function as follows:

$$\mathcal{L}(Y_{\text{train}}|\varphi, w, X_{\text{train}}) = \mathcal{L}_{\mathcal{CE}} + \lambda \sum_{p=1}^{B} H_p \tag{9}$$

$$H_p = -\sum_{j=1}^{|\mathcal{O}|} S_j \log S_j, \quad s.t. \sum_{j=1}^{|\mathcal{O}|} S_j = 1, 0 \le S_j \le 1 \tag{10}$$

where $\mathcal{L}_{\mathcal{CE}}$ is a general cross entropy loss function and $\sum_{p=1}^{B} H_p$ denotes the summation of entropies *w.r.t.* all candidate operations in the cell currently being

searched, H_p is the entropy of the p-th set of candidate operations. B is the number of nodes in a cell and $|\mathcal{O}|$ is the size of the operation space. λ is a scaling factor that controls the rate of convergence.

We try to minimize the loss function Eq. 9, and this optimization procedure forces the entropy to decrease so that the importance score distribution S tends to a single peak gradually. When the scaling factor λ is larger, the constraint is stronger, thus the difference of importance scores can get obvious through the training with the fixed number of steps. We verify the entropy constraint effectiveness, and details are presented in Sec. 4.3.2.

With the minimum entropy constrain, we formulate the optimization and gradient computing about affine parameters. According to Eq. 6 and Eq. 9, the gradient *w.r.t.* an affine parameter φ_k can be computed as below:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \varphi_k} &= \frac{\partial \mathcal{L}_{\mathcal{CE}}}{\partial \varphi_k} + \lambda(-\frac{\partial H_p^k}{\partial \varphi_k}) \\ &= \frac{\partial \mathcal{L}_{\mathcal{CE}}}{\partial \mathcal{F}_{ij}(x_j)} norm(\hat{x}_j^k) - \lambda \sum_{j=1}^{|\mathcal{O}|}(\log S_j + 1)\frac{\partial S_j}{\partial \varphi_k}\end{aligned} \tag{11}$$

where $\frac{\partial S_j}{\partial \varphi_k}$ depends on the positive and negative of φ_k, the result as follows:

$$\frac{\partial S_j}{\partial \varphi_k} = \begin{cases} S_j(\delta_{jk} - S_j), & \varphi_k \geq 0 \\ -S_j(\delta_{jk} - S_j), & \varphi_k < 0 \end{cases} \tag{12}$$

where $\delta_{jk} = 1$ if $j = k$ or else $\delta_{jk} = 0$. From Eq. 11 and 12, we can observe that the entropy constraint also delivers interactive information between different affine parameters, which pushes the competition of various operations. Moreover, there is no extra computational burden for training our search network just like a common convolutional neural network. The pseudocode of our proposed algorithm LFR-DARTS is presented in Alg. 1.

Algorithm 1 LFR-DARTS - Layered Feature Representation for Differentiable Architecture Search

Input: Training data: $(X_{\text{train}}, Y_{\text{train}})$; Total target cells: χ; Training epochs for each cell: T
Output: χ optimal cells

Initialize a search network $\mathcal{G}$ consisting of χ cells
while $(iter \leq \chi)$ **do**
  The target cell to be searched in $\mathcal{G}$: $\mathcal{C}_{\text{iter}}$
  Initialize the operation space $\mathcal{O}_{\text{iter}}$
  while $(epoch \leq \text{T})$ **do**
    Train the search network $\mathcal{G}$ and minimize $\mathcal{L}(Y_{\text{train}}|\varphi, w, X_{\text{train}}) = \mathcal{L}_{\mathcal{CE}} + \lambda(-\sum_{p=1}^{B} H_p)$
    if $epoch > 0$ and $epoch\%(T/3) == 0$ **then**
      Compute the importance score of each operation: $S_k = softmax(|\varphi_k|) = \frac{e^{|\varphi_k|}}{\sum_{j=1}^{|\mathcal{O}|} e^{|\varphi_j|}}$
      Prune out the operations with low importance scores from $\mathcal{O}_{\text{iter}}$
    end if
    $epoch = epoch + 1$
  end while
  if $\mathcal{C}_{\text{iter}} \subseteq$ normal cell **then**
    Fix the architecture of normal cell $\mathcal{C}_{\text{iter}}$ and expand it to N copies in the original position of $\mathcal{G}$
  end if
  $iter = iter + 1$
end while
return χ optimal cells

4 Experiments

In this section, we compare the performance of our algorithm LFR-DARTS with other NAS approaches and human-designed networks on the several popular image classification datasets, including CIFAR10, fashionMNIST and ImageNet [9]. Following DARTS [25], We conduct our experiment in two steps: (1) Architecture search: searching the optimal cell on the training dataset of CIFAR10; (2) Architecture evaluate: construct an evaluation network by the obtained cells and test its performance on the testing datasets of CIFAR10, fashionMNIST and ImageNet.

4.1 Architecture search and result

The initial search network $\mathcal{G}$ consists of $\chi = 5$ cells where two reduction cells (with stride = 2) are inserted between three normal cells (with stride = 1). The number of nodes $B = 7$ in a cell. The initial operation space $\mathcal{O}$ is same as [50, 25] and the size of space $|\mathcal{O}| = 8$ at the beginning of each iteration. The search network in the each iteration will be performed the search training of T epochs before obtaining the final architecture of the corresponding cell. The process of search training assures the the performance stability of the search network after each network prune. In our experiment, the value of T is set to 60 because we find through experiments that the accuracy of the search network keeps relatively steady after about 60 epoch training. Fewer training epochs usually lead to performance collapse as network parameters have not converged. More training epochs contribute little to the result. All of our experiments are run on one device with a CPU of Intel core i7-8700K and a GPU of NVIDIA GTX1080Ti.

The architecture search is implemented on the deep learning framework PyTorch [29] with initial channels

of 16 and a batch size of 96. The initial learning rate is 0.025 and then annealed down to zero following a cosine schedule. A standard SGD optimizer with momentum of 0.9 and weight decay of 3×10^{-4} is adopted. The hyperparameter λ is fixed at 5×10^{-3}. Other experiment settings follow [25]. The architecture search takes only 0.45 GPU days on a single GPU device.

We implement the experiment five times with different random seeds and pick out the best cells based on the validation performance. The cells discovered by LFR-DARTS algorithm are presented in Fig. 2.

4.2 Architecture evaluate

4.2.1 Evaluation on CIFAR10

An evaluation network consisting of 16 cells discovered in 4.1 is trained from scratch for 600 epochs on the CIFAR10 with mini-batch=50, initial learning_rate=0.025, init_channels=36, drop_path_prob=0.15, momentum=0.9, weight_decay=3×10^{-4} and auxiliary towers of weight=0.4. We use standard image preprocessing and data augmentations, *i.e.*, randomly cropping, horizontally flipping and batch normalization. Other settings remain the same as [30,25]. Two reduction cells as Fig. 2(d) and Fig. 2(e) are located at 1/3 and 2/3 of the total depth of the evaluation network, respectively. The other positions of the network are filled with three other kinds of the normal cells, *i.e.*, Fig. 2(a), 2(b) and 2(c).

To explore the performance limitation of our discovered neural architecture, we further increase the initial channels to 50 for the evaluation network which contains 15 cells and more parameters (denoted as large settings in Table 2). We compare our neural network with other networks designed by experts and other NAS methods under fair conditions where the parameters are less than 5M for all the NAS networks. Every evaluation network is trained 5 times using different random seeds and the results prove that our discovered network has excellent performance and strong stability.

The test results and the comparison with other approaches are summarized in Table 2. As shown in Table 2, our LFR-DARTS achieves a test error rate of 2.65% with only 2.7M parameters on the validation dataset of CIFAR10. With more parameters (4.4M), LFR-DARTS further reduces the error rate to 2.45%, which almost outperforms the existing state-of-the-art works with less computational cost than DARTS.

4.2.2 Evaluation on fashionMNIST

The discovered cell architectures are first transferred to another dataset called fashionMNIST, consisting of a training set of 60,000 images and a testing set of 10,000 images. Each image is a 28x28 grayscale image associated with a label from 10 classes. The evaluation network is constructed by 15 cells, 36 initial channels. We training this evaluation network by a SGD optimizer with learning rate=0.025, momentum=0.9, weight_decay=3×10^{-4} and auxiliary towers of weight=0.4, mini-batch size=96. Other configurations are same as that in Sec. 4.2.1. This network is training for 300 epochs and reaches a test error rate of 3.70% with 2.5M parameters and a test error rate of 3.63% with 4.1M parameters. The comparison results of other algorithms are displayed in Tab. 3.

4.2.3 Evaluation on ImageNet

We transfer our architecture discovered on CIFAR10 to a large-scale dataset named ImageNet, and the result also demonstrates excellent generalization performance. Following the mobile setting in [50,30], we construct our evaluation network by 15 cells and train it for 250 epochs with batch size 160, initial channels 48, weight decay 3×10^{-5}, monmentum=0.9 and initial SGD learning_rate=0.1 (decayed linearly to 0.0). We keep other hyper-parameters and settings as that on CIFAR10. We compare our algorithm with other approaches and the results are presented in Table 4.

We compare the complete architecture search and evaluation process of ours, DARTS and random search on three datasets, respectively. Please refer to Fig. 3 and Fig. 4. Fig. 3 illustrates the loss learning curves on CIFAR10 (a), fashionMNIST (b) and ImageNet (c). Fig. 4 shows the training/testing process at the evaluation stage on the same three datasets. The results show that our method achieves better performance and generalization than the baseline methods.

4.3 Diagnostic experiments

4.3.1 Efficiency of architecture search

Our LFR-DARTS algorithm has shown high search efficiency and space utilization. In the experiment, the architecture search contains χ iterations. In each iteration, we make statistics on the time and space cost of one epoch training, and the results are presented in Table 5. For each iteration, we gradually remove the weak operations from the search network at 3 steps, respectively. At the beginning, it spends 196 seconds in

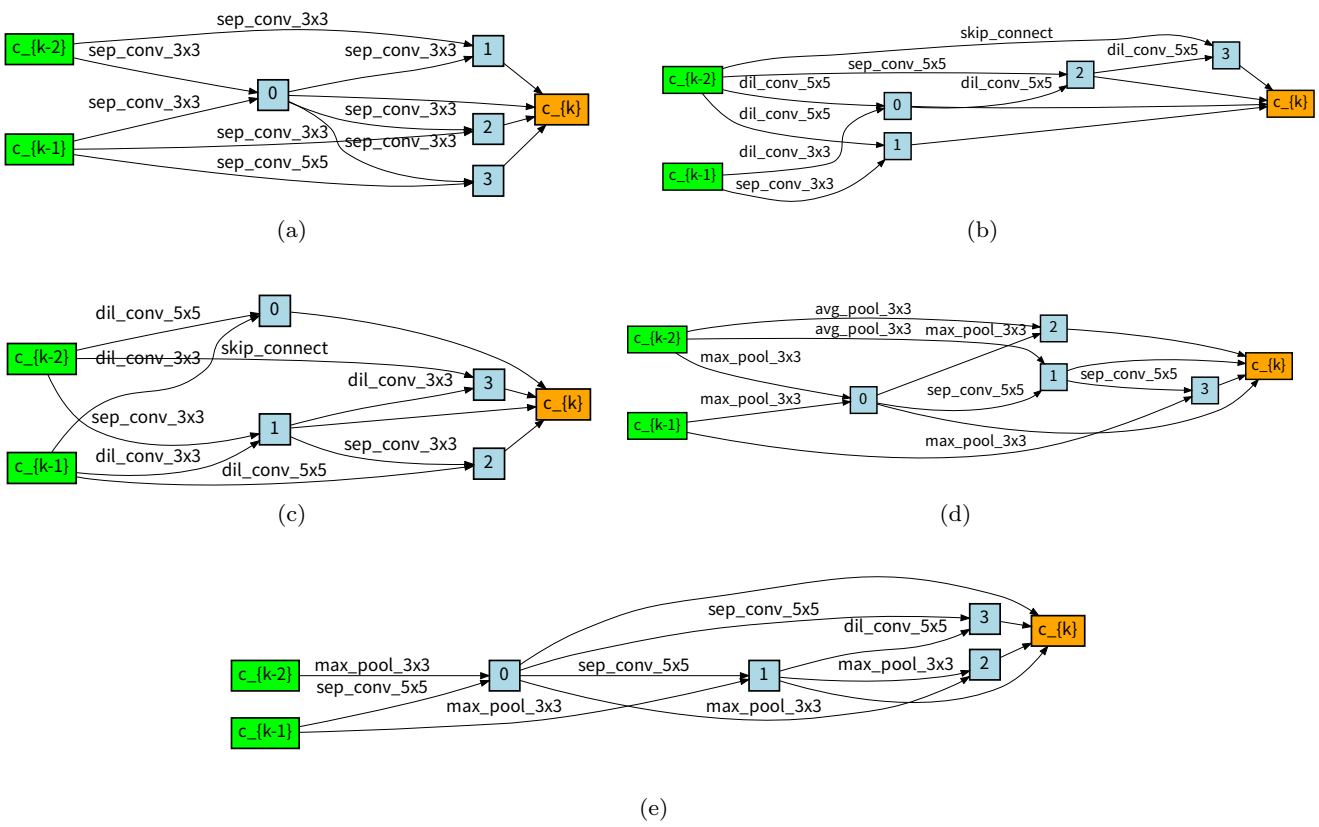


Fig. 2 The cells discovered by LFR-DARTS algorithm; (a) Normal cell I; (b) Normal cell II; (c) Normal cell III; (d) Reduction cell I; (e)Reduction cell II.

Architecture	Test Error(%)	Params (M)	GPU days	Search Method
ResNet-18 [14]	3.53	11.1	-	Manual
DenseNet [19]	3.46	25.6	-	Manual
SENet [18]	4.05	11.2	-	Manual
NASNet-A [50]	3.41	3.3	1800	RL
NASNet-A + cutout [50]	2.65	3.3	1800	RL
AmoebaNet-A + cutout[32]	3.12	3.1	3150	Evolution
PNAS [23]	3.41	3.2	225	SMBO
ENAS [30]	3.54	4.6	0.5	RL
ENAS+cutout [30]	2.89	4.6	0.5	RL
DARTS(2nd order) + cutout[25]	2.82	3.4	1.6	Gradient
Random Sample [25]	3.49	3.1	-	-
SNAS + cutout [44]	2.85	2.8	1.50	Gradient
GDAS + cutout [10]	3.87	3.4	0.21	Gradient
GDAS +cutout [10]	2.93	3.4	0.21	Gradient
LFR-DARTS + cutout	2.65	2.7	0.45	Gradient
LFR-DARTS + cutout (large settings)	2.45	4.4	0.45	Gradient

Table 2 Test classification error rates for LFR-DARTS, human-designed networks and other NAS architectures on CIFAR10.

one epoch training with 10094M GPU memory usage and reduces to 87 seconds with 8676M GPU memory usage at the last. It clearly shows that the training time and space costs constantly get decreasing, which proves that our method speeds up the search process.

To further show the difference in the search efficiency between our algorithm and DARTS, we investigate the forward-propagation and backward-propagation time of our method and DARTS during the search process. For the search network of ours and DARTS, we set the same batch size=32, training

Architecture	Test Error(%)	Params (M)	GPU days	Search Method
VggNet [36]	6.53	26.0	-	Manual
ResNet-18 [14]	5.10	11.1	-	Manual
DenseNet [19]	4.61	25.6	-	Manual
NASNet-A + cutout [50]	3.66	2.5	1800	RL
AmoebaNet-A + cutout[32]	3.67	2.3	3150	Evolution
PNAS [23]	3.89	2.5	225	SMBO
ENAS+cutout [30]	3.79	2.6	0.5	RL
DARTS(2nd order) + cutout [25]	3.77	2.2	1.6	Gradient
Random Sample [25]	3.95	2.5	-	-
SNAS + cutout [44]	3.80	2.3	1.50	Gradient
GDAS + cutout [10]	3.76	2.4	0.21	Gradient
LFR-DARTS + cutout	3.70	2.5	0.45	Gradient
LFR-DARTS + cutout (large settings)	3.63	4.1	0.45	Gradient

Table 3 Test classification error rates for LFR-DARTS, human-designed networks and other NAS architectures on fashion-MNIST.

Architecture	Accuracy (%)		Params (M)	GPU days	Search Method
	Top1	Top5			
Inception-V1 [38]	69.8	89.9	6.6	-	Manual
MobileNetV2 [17]	72.0	91.0	3.4	-	Manual
ShuffleNetV2 [47]	73.7	-	∼5	-	Manual
NASNet-A [50]	74.0	91.6	5.3	1800	RL
NASNet-B [50]	72.8	91.3	5.3	1800	RL
NASNet-C [50]	72.5	91.0	4.9	1800	RL
AmoebaNet-A [32]	74.5	92.0	5.1	3150	Evolution
AmoebaNet-B [32]	74.0	91.5	5.3	3150	Evolution
AmoebaNet [32]	75.7	92.4	6.4	3150	Evolution
PNAS [23]	74.2	91.9	5.1	225	SMBO
MnasNet [39]	74.8	92.0	4.4	-	RL
DARTS [25]	73.1	91.0	4.9	1.6	Gradient
SNAS [44]	72.7	90.8	4.3	1.5	Gradient
GDAS [10]	74.0	91.5	5.3	0.21	Gradient
LFR-DARTS (Ours)	74.5	91.7	4.9	0.45	Gradient

Table 4 Comparison with the state-of-the-art image classification methods on ImageNet.

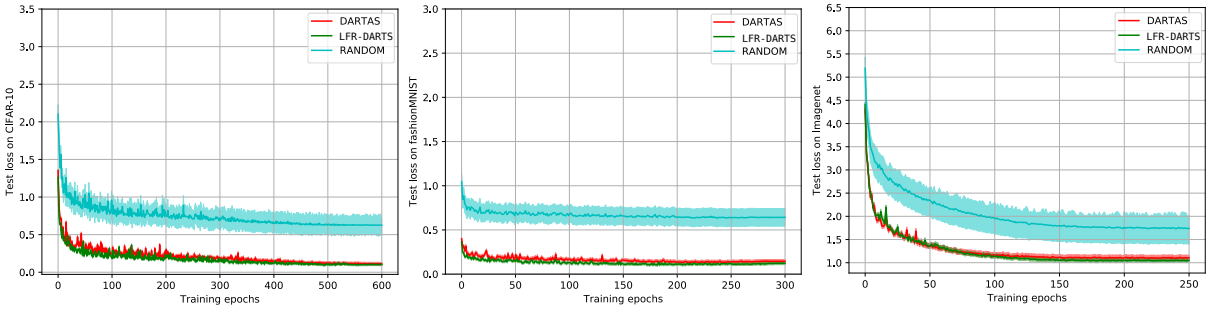
Pruning steps	Cell I (s/M)	Cell II (s/M)	Cell III (s/M)	Cell IV (s/M)	Cell V (s/M)
1	196/10094	181/8676	139/8676	123/8676	95/8676
2	179/9512	163/8310	131/8676	112/8676	92/8676
3	169/9512	156/8310	123/8676	104/8676	87/8676

Table 5 The search time/space costs for the cell in different iterations.

epochs=300, and then we monitor the time changes of once propagation. The results are displayed in Fig. 5. The propagation time in our method descends step by step in Fig. 5(a) as the search network constantly drops weak operations. But DARTS needs to conduct a bilevel optimization of architecture parameters and network weights simultaneously. We measure the propagation time and illustrate it in Fig. 5(b). As can be seen, our method still keeps a faster search process than DARTS even in the initial phase. The search process gets accelerated gradually in the later phases, which is also suitable for searching deeper networks.

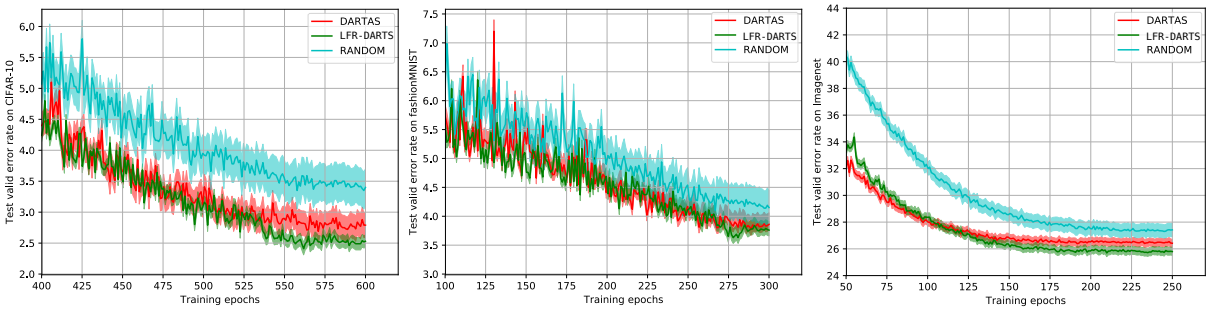
4.3.2 Effectiveness of entropy constraint

In this section, we experimentally verify the effectiveness of the entropy constraint mentioned in Sec. 3.3. The entropy constraint is a part of the loss function Eq. 9. The minimum entropy over candidate operations makes the distribution of operations' importance score tend to a single or few peaks. This distribution makes it easier to filter out optimal operations since the constraint highlights the most important operation. In fact, the parameter λ is closely related to the distribution of the importance score, so an appropriate scaling factor



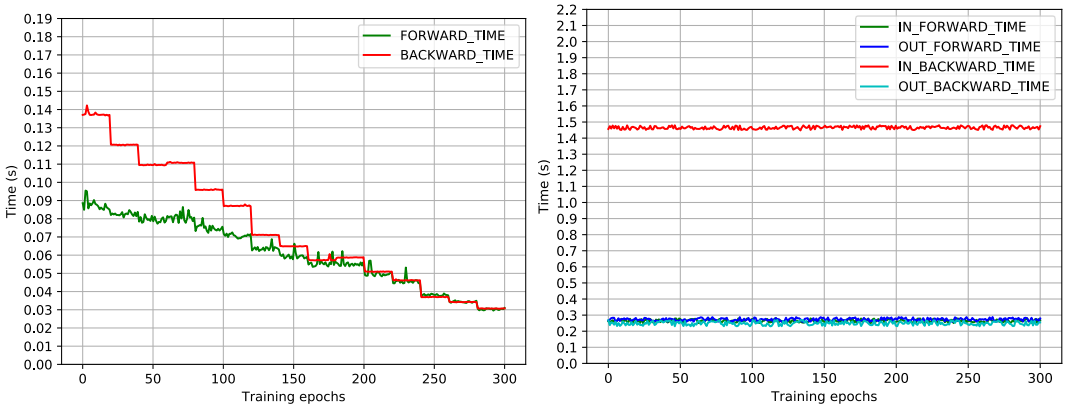
(a) (b) (c)

Fig. 3 Loss learning curves of neural networks on three datasets ((a): CIFAR10, (b): FahsionMNIST, (c): ImageNet).



(a) (b) (c)

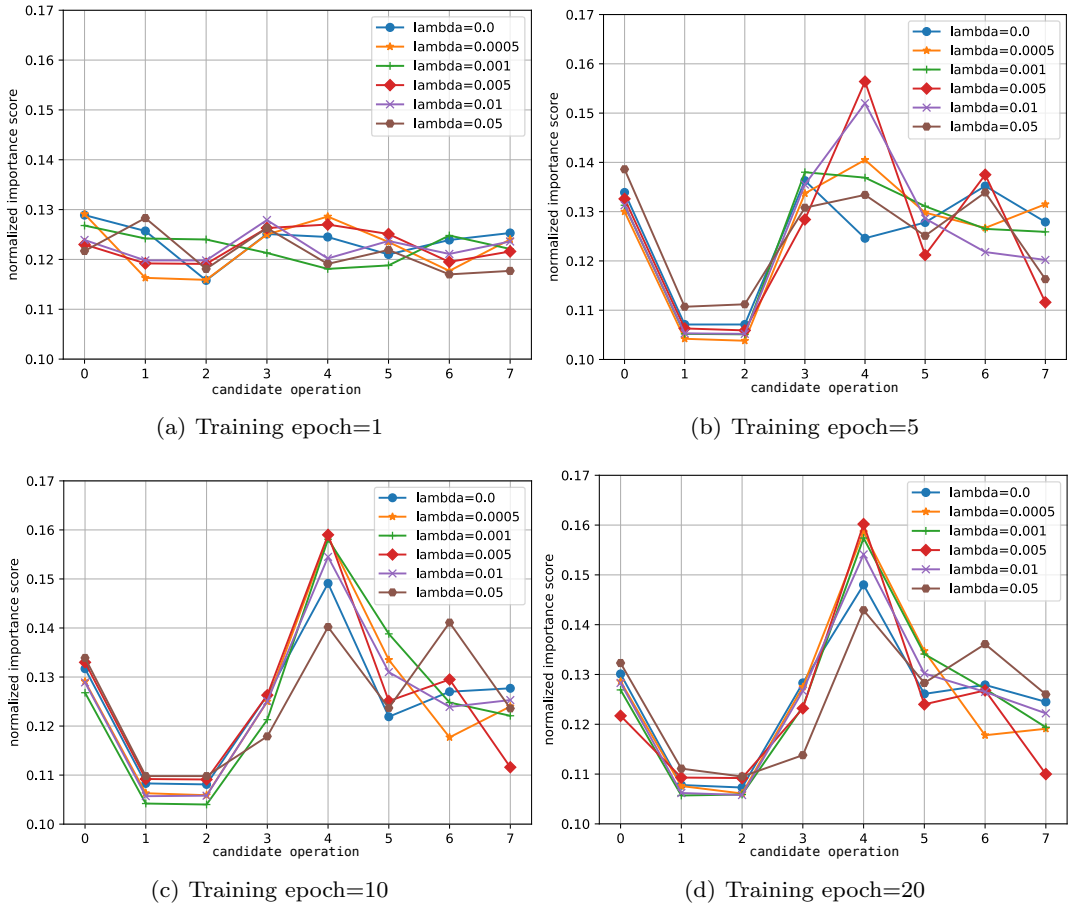
Fig. 4 Training and testing error rate of neural networks on three datasets ((a): CIFAR10, (b): FahsionMNIST, (c): ImageNet).



(a) LFR-DARTS (b) DARTS

Fig. 5 The efficiency comparison of once forward and backward propagation during the search process.

is a key. We conduct experiments to compare the effects of different values of λ on the results. During a search stage, we randomly chose one set of operations (containing 8 operations) in a searching cell and observe its differences of importance scores under different λ settings. The result is displayed as Fig. 6, where $\lambda = 0.0$ means no entropy constraint in the experiment. The four sub-figures show the distributes of importance score at four different training epochs. The distributes vary as the training, and various λ has different impacts on the results. In the initial phase of the training 6(a), the importance is a random distribution. After a period of training 6(d), the high scores are concentrated on few operations. From the extensive experiments, we find that the importance score converges better under the condition of $\lambda = 0.005$ than other settings. Too large or small values of λ could lead to unsatisfactory convergence results. As we can see, it gets worse while



(a) Training epoch=1

(b) Training epoch=5

(c) Training epoch=10

(d) Training epoch=20

Fig. 6 The distribution of importance scores varies as training time under different λ.

$\lambda \geq 0.05$. So an appropriate entropy constraint *w.r.t.* candidate operations can make a positive contribution to the process of architecture search.

5 Discussion

From the visualized results of the discovered cells in Fig 2, we get some interesting observations consistent with common sense are that shallow network layers prefer to select small separable convolutional kernel (as Fig. 2(a)) and deeper layers prefer large dilated separable convolutional kernel (as Fig. 2(c)). Therefore, these discovered cells show striking depth-adaptive characteristic. That is small-size convolutional kernels in the shallow layers do well in extracting the fine-grained feature information of data. The large-size convolutional kernels in the deeper layers are conducive to processing the fused features. Sufficient layered information can provide more reliable basis for making decisions. However, the network architectures stacked repeatedly in DARTS cannot meet the requirement of feature extraction. LFR-DARTS takes this problem into consideration, thus improves the performance of differentiable approaches. Our method also provides a valuable reference for developing more elaborate and useful architecture cells.

In addition, our search process is divided into multiple stages and performed iteratively. Each cell architecture is searched based on the obtained cells and the current network depth. Although, these cells are not the global optima, their combination provides an approximately optimal solution for the architecture search. The greedy search scheme is currently adopted by most differentiable search methods. Thus, there are lots of promising improvements in this greedy search scheme.

Our work has shown many advantages in designing network architectures for image tasks. It is also worth applying our method to other fields, such as object detection, natural language processing *etc.* We will further explore the differentiable approaches of architecture search to solve the problems of model automated design in other fields.

6 Conclusion

In this paper, we propose a novel differentiable NAS algorithm called *Layered Feature Representation for Differentiable Architecture Search* (LFR-DARTS) to solve the existing problem of insufficient layered feature representation. In this way, LFR-DARTS improves the performance and generalization of discovered network architectures compared to other differentiable NAS algorithms. Specifically, we develop a layered and dynamic architecture search scheme to discovered multiple optimal cells from shallow to deep layer and gradually prunes out the weak operations from the search network. Besides, to effectively learn the importance of candidate operations and highlight the optimal ones during search process, we design a new functional layer Normalization-Affine and introduce an entropy constraint for the operations. The extensive experiments on the image classification tasks demonstrate our algorithm can achieve better performance while requiring low computational costs.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (61772120).

Compliance with ethical standards

Funding: This study was funded by National Natural Science Foundation of China (grant number 61772120).
Conflict of Interest: The authors declare that they have no conflict of interest.
Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

Authorship contributions

All authors contributed to the study conception and design. The experiment design and implementation are done by Jie Hao. The first draft of the manuscript was written by Jie Hao and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

References

1. Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.
2. Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. *arXiv preprint arXiv:1709.07417*, 2017.
3. Bobadilla, Jesús and Lara-Cabrera, Raúl and González-Prieto, Ángel and Ortega, Fernando. DeepFair: Deep Learning for Improving Fairness in Recommender Systems. *International Journal of Interactive Multimedia & Artificial Intelligence*, 6(6), 86–94, 2021.
4. Max Bramer. *Principles of data mining*, volume 180. Springer, 2007.
5. Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
6. Zhiling Cai, Xiaofei Yang, Tianyi Huang, and William Zhu. A new similarity combining reconstruction coefficient with pairwise distance for agglomerative clustering. *Information Sciences*, 508:173–182, 2020.
7. Zhiling Cai and William Zhu. Multi-label feature selection via feature manifold learning and sparsity regularization. *International Journal of Machine Learning and Cybernetics*, 9(8):1321–1334, 2018.
8. Chu X, Zhou T, Zhang B, et al. Fair darts: Eliminating unfair advantages in differentiable architecture search. In *European conference on computer vision*, pages 465–480, 2020.
9. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
10. Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.
11. Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.
12. Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.
13. Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.
14. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
15. Heydarpour, F and Abbasi, E and Ebadi, MJ and Karbassi, Seyed-Mehdi. Solving an Optimal Control Problem of Cancer Treatment by Artificial Neural Networks. In *International Journal of Interactive Multimedia & Artificial Intelligence*, 6(4), 2016.
16. Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.
17. Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco

Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
18. Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.
19. Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
20. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
21. Jie Hao, William Zhu. Saliency: a new selection criterion of important architectures in neural architecture search. In *Neural Computing and Applications*, 1–15, 2021.
22. Liang H, Zhang S, Sun J, et al. Darts+: Improved differentiable architecture search with early stopping. In *arXiv preprint arXiv:1909.06035*, 2019.
23. Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.
24. Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.
25. Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.
26. Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020.
27. Benteng Ma, Xiang Li, Yong Xia, and Yanning Zhang. Autonomous deep learning: A genetic dcnn designer for image classification. *Neurocomputing*, 379:152–161, 2020.
28. Mark Nixon and Alberto Aguado. *Feature extraction and image processing for computer vision*. Academic press, 2019.
29. Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
30. Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.
31. Vadlamani Ravi and H-J Zimmermann. A neural network and fuzzy rule base hybrid for pattern classification. *Soft Computing*, 5(2):152–159, 2001.
32. Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.
33. Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017.
34. Sunil Saumya, Jyoti Prakash Singh, and Yogesh K Dwivedi. Predicting the helpfulness score of online reviews using convolutional neural network. *Soft Computing*, pages 1–17, 2019.
35. Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.
36. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
37. Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
38. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. pages 1–9, 2015.
39. Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.
40. Verma, K. K., and Singh, B. M. Deep Multi-Model Fusion for Human Activity Recognition Using Evolutionary Algorithms. In *International Journal Of Interactive Multimedia And Artificial Intelligence*, In press, 1–15, 2021.
41. Hanzhang Wang, Hanli Wang, and Kaisheng Xu. Evolutionary recurrent neural network for image captioning. *Neurocomputing*, 2020.
42. Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
43. Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.
44. Liu C, Zoph B, Neumann M, et al. Progressive neural architecture search. *Proceedings of the European conference on computer vision (ECCV)*, 19–34, 2018.
45. Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
46. Zela, Arber and Elsken, Thomas and Saikia, Tonmoy and Marrakchi, Yassine and Brox, Thomas and Hutter, Frank. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020.
47. Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.
48. William Zhu. Relationship between generalized rough sets based on binary relation and covering. *Information Sciences*, 179(3):210–225, 2009.
49. Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

50. Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Front Page

Jie Hao and William Zhu*

University of Electronic Science and Technology of China

No.4, Sec. 2, Jianshe Beilu, Chengdu, China

* The corresponding author: wfzhu@uestc.edu.cn, +086+15928727490

William Zhu's ORCID: 0000-0001-8898-9244