Layered feature representation for differentiable architecture search

Jie Hao · William Zhu

Accepted: 15 February 2022 / Published online: 9 April 2022
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

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 solves 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

1 Introduction

Over the last few years, deep neural networks (DNN) have demonstrated powerful capabilities of feature extraction (Ravi and Zimmermann 2001; Guyon et al. 2008; Cai and Zhu 2018; Nixon and Aguado 2019) and data mining (Bramer 2007; Zhu 2009; Cai et al. 2020; Verma and Singh 2021.) Thus, DNN is applied to a large variety of challenging tasks, such as image recognition (Hu et al. 2018; Kaiming et al. 2016), speech recognition (Geoffrey et al. 2012; Alex et al. 2013), machine translation (Sutskever et al. 2014; Wu et al. 2016), and other complex tasks (Sunil et al. 2019; Lotfollahi et al. 2020; Heydarpour et al. 2016; Bobadilla et al. 2021). But designing an advanced neural network typically requires substantial efforts of human experts. To eliminate such a handcraft process, neural architecture search (NAS) (Zoph and Le 2016; Zoph et al. 2018; Real et al. 2019) 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) (Zoph and Le 2016; Zoph et al. 2018; Irwan et al. 2017) or evolutionary algorithms (EA) (Real et al. 2017; Liu et al. 2017; Real et al. 2019) to perform architecture search. Both of their searching procedure require 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 (Zoph et al. 2018)(RL-based) trains and evaluates more than 20,000 neural networks across 500 GPUs over 4 days. AmoebaNet (Real et al. 2019) (EA-based) even takes 3150 GPU-days to discover an optimal neural architecture.

To eliminate this high computational overhead, (Liu et al. 2018b) recently proposed a differentiable architecture search (aka DARTS), which relaxes the discrete 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 (Fig. 1).

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.

![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.](image-url)
Moreover, the optimization for cell architecture is simpler yet more efficient compared to DARTS, which is demonstrated by the diagnostic experiments in Sect. 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 (Thomas et al. 2018), i.e., reinforcement learning (RL)-based approaches, evolutionary algorithm (EA)-based approaches, and gradient-based approaches.

The early approaches (Zoph and Le 2016; Zoph et al. 2018; Bowen et al. 2016; Cai et al. 2018; Liu et al. 2018a) use RL to optimize the search policy for discovering optimal architectures. NASNet (Zoph et al. 2018) trains a recurrent neural network as a controller to decide the types and parameters of neural networks sequentially. ENAS (Hieu et al. 2018) 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 (Real et al. 2017, 2019; Wang et al. 2020; Ma et al. 2020). AmoebaNet (Real et al. 2019) 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 (Liu et al. 2018b, a, c; Xuanyi and Yang 2019) 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 (Liu et al. 2018c) proposes to constrain the architecture parameters to be one-hot to tackle the inconsistency in optimizing objectives between search and evaluation scenarios. GDAS (Xuanyi and Yang 2019) develops a differentiable sampler over the search space to avoid simultaneously training all the neural architectures in the space. DARTS+ (Liang et al. 2019) and PDARTS (Liu et al. 2018c) employ early stopping to restrict the excessive number of “skip” operations. FairDARTS (Chu et al. 2020) proposes the collaborative competition strategy to address the unfair advantage in exclusive competition. NASSA (Hao and Zhu 2021) 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 Sect. 3.1, which is a basis of our LFR-DARTS. Then, we describe the concrete search procedure of our algorithm in Sect. 3.2. Finally, in
A convolutional neural network (CNN) has a hierarchical structure so as to extract the layered visual features of images. As Simonyan et al. (2013); Zeiler and Fergus (2014) 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 (Szegedy et al. 2015; Kaiming et al. 2016; Hu et al. 2018) obey this rule consistently. But differentiable NAS algorithms (Zoph et al. 2018; Liu et al. 2018b) 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 $\alpha$ for architecture optimization in DARTS are shared between cells, leading to optimizing and producing only a common

### Table 1 The operation space for neural architecture search

| Operation       | O          |
|-----------------|------------|
| Zeroize         | Max_pool_3x3 |
| Avg_pool_3x3    | Skip_connect |
| Sep_conv_3x3    | Sep_conv_5x5 |
| Dil_conv_3x3    | Dil_conv_5x5 |

Sect. 3.3, we introduce a minimum entropy constraint and formulate the gradient optimization for the search network.

### 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 $O$ (Table 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:

$$f(i,j)(x_i) = \sum_{k=1}^{[O]} \left[ \frac{\exp(\alpha_{ok})}{\sum_{m=1}^{[O]} \exp(\alpha_{om})} \right] o_k(x_i)$$  \hspace{1cm} (1)$$

where $x_i$ is the feature maps of node $i$, and $\alpha$ 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:

$$x_j = \sum_{i<j} f(i,j)(x_i)$$  \hspace{1cm} (2)$$

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

$$x_{B−1} = \text{concat}(x_2, x_3, ..., x_{B−2})$$  \hspace{1cm} (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.

---

© Springer
cell. Second, the search procedure is complicated as DARTS
needs to alternatively optimize the architecture parameters
\( \alpha \) and network weights \( \omega \) by gradient descent. \( \alpha \) is trained
on the validation dataset and \( \omega \) 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:

\[
\text{NA}_k(x_{in}) = \varphi_k \times \text{norm}(x_{in})
\]  

(4)

\[
\text{norm}(x_{in}) = \frac{x_{in} - \mu}{\sqrt{\sigma^2 + \epsilon}}
\]  

(5)

where the trainable weight parameter \( \varphi_k \) is referred to as an
affine parameter which is used to weight each operation. \( \epsilon \) is
a very small value close to zero. \( x_{in} = \{x_{in}^1, x_{in}^2, \ldots, x_{in}^m\} \)
is the input tensor of the NA layer and the output tensor
of the \( k \)-th operation, and it contains \( m \) feature maps. \( \mu = \{\mu_1, \mu_2, \ldots, \mu_m\} \), \( \sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\} \) are mean vec-
tor and standard deviation vector of the mini-batch \( x_{in} \). \( \mu \)
and \( \sigma \) also contain \( m \) elements, and each element is corre-
sponding to a feature map of \( x_{in} \). The normalized function
\( \text{norm}(\cdot) \) partially comes from Batch Normalization (Ioffe
and Szegedy 2015), which is one of the most common and
useful normalization approaches in CNN models.

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

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

(6)

\[
\hat{x}_j^k = o_k(x_j)
\]  

(7)

where \( o_k(\cdot) \) is the \( k \)-th operation in a set of candidate operations and \( x_j \) is the input of the operation. \( \mu_k, \sigma_k \) 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 \( \varphi \), which is trained and updated
together with weight parameters \( \omega \) 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 param-
eters 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:

\[
S_k = \text{softmax}(\langle \varphi_k \rangle) = \frac{\exp(\langle \varphi_k \rangle)}{\sum_{j=1}^{[\mathcal{O}]} \exp(\langle \varphi_j \rangle)}
\]  

(8)

where \( \varphi_k \) 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 nec-
essary 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 \( \varphi \). For any operation, its weight parameters \( \omega \)
will be optimized and updated simultaneously with the corre-
sponding \( \varphi \). Optimizing the two kinds of parameters together
could make them vary synchronously, resulting in same result
by increasing one and deceasing another. Therefore, normal-
ization before reweighting is quite necessary since it makes
the results from different operations unified 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
\( \varphi \) together with weights \( \omega \) 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:

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

(9)

\[
H_p = -\sum_{j=1}^{[\mathcal{O}]} S_j \log S_j, \quad \text{s.t.} \sum_{j=1}^{[\mathcal{O}]} S_j = 1, 0 \leq S_j \leq 1
\]  

(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 can-
didate 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 $|O|$ is the size of the operation space. $\lambda$ 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 $\lambda$ 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 Sect. 4.3.2.

With the minimum entropy constrain, we formulate the optimization and gradient computing about affine parameters. According to Eqs. 6 and 9, the gradient w.r.t. an affine parameter $\phi_{k}$ can be computed as below:

$$\frac{\partial L}{\partial \phi_{k}} = \frac{\partial L_{CE}}{\partial \phi_{k}} + \lambda \left( \frac{\partial H_{k}}{\partial \phi_{k}} \right)$$

$$= \frac{\partial L_{CE}}{\partial F_{ij}(x_{j})} \text{norm} \left( \tilde{z}_{k}^{j} \right) - \lambda \sum_{j=1}^{|O|} (\log S_{j} + 1) \frac{\partial S_{j}}{\partial \phi_{k}}$$

(11)

where $\frac{\partial S_{j}}{\partial \phi_{k}}$ depends on the positive and negative of $\phi_{k}$, the result as follows:

$$\frac{\partial S_{j}}{\partial \phi_{k}} = \begin{cases} S_{j}(\delta_{jk} - S_{j}), & \phi_{k} \geq 0 \\ -S_{j}(\delta_{jk} - S_{j}), & \phi_{k} < 0 \end{cases}$$

(12)

where $\delta_{jk} = 1$ if $j = k$ or else $\delta_{jk} = 0$. From Eqs. 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 Algorithm 1.

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 (Jia et al. 2009). Following DARTS (Liu et al. 2018b), 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.
The cells discovered by LFR-DARTS algorithm; a Normal cell I; b Normal cell II; c Normal cell III; d Reduction cell IV; e Reduction cell V.

tings follow (Liu et al. 2018b). 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 Sect. 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 Hieu et al. (2018); Liu et al. (2018b). 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), (b) and (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
Table 2 Test classification error rates for LFR-DARTS, human-designed networks and other NAS architectures on CIFAR10

| Architecture                        | Test Error(%) | Params (M) | GPU days | Search Method |
|--------------------------------------|---------------|------------|----------|---------------|
| ResNet-18 Kaiming et al. (2016)      | 3.53          | 11.1       | –        | Manual        |
| DenseNet Huang et al. (2017)         | 3.46          | 25.6       | –        | Manual        |
| SENet Hu et al. (2018)               | 4.05          | 11.2       | –        | Manual        |
| NASNet-A Zoph et al. (2018)          | 3.41          | 3.3        | 1800     | RL            |
| NASNet-A + cutout Zoph et al. (2018) | 2.65          | 3.3        | 1800     | RL            |
| AmoebaNet-A + cutout Real et al. (2019) | 3.12          | 3.1        | 3150     | Evolution     |
| PNAS Liu et al. (2018a)              | 3.41          | 3.2        | 225      | SMBO          |
| ENAS Hieu et al. (2018)              | 3.54          | 4.6        | 0.5      | RL            |
| ENAS+cutout Hieu et al. (2018)       | 2.89          | 4.6        | 0.5      | RL            |
| DARTS(2nd order) + cutout Li et al.  | 2.82          | 3.4        | 1.6      | Gradient      |
| Random Sample Liu et al. (2018b)     | 3.49          | 3.1        | –        | –             |
| SNAS + cutout Liu et al. (2018c)     | 2.85          | 2.8        | 1.50     | Gradient      |
| GDAS + cutout Xuanyi and Yang (2019) | 3.87          | 3.4        | 0.21     | Gradient      |
| GDAS +cutout Xuanyi and Yang (2019)  | 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      |

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 $\chi$ 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 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 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

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 Zoph et al. (2018); Hieu et al. (2018), we construct our evaluation network by 15 cells and train it for 250 epochs with batch size 160, initial channels 48, weight decay $3 \times 10^{-5}$, momentum=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 Figs. 3 and 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.
Table 3  Test classification error rates for LFR-DARTS, human-designed networks and other NAS architectures on fashionMNIST.

| Architecture                      | Test Error(%) | Params (M) | GPU days | Search Method |
|-----------------------------------|---------------|------------|----------|---------------|
| VggNet Simonyan and Zisserman (2014) | 6.53          | 26.0       | –        | Manual        |
| ResNet-18 Kaiming et al. (2016)   | 5.10          | 11.1       | –        | Manual        |
| DenseNet Huang et al. (2017)      | 4.61          | 25.6       | –        | Manual        |
| NASNet-A + cutout Zoph et al. (2018) | 3.66          | 2.5        | 1800     | RL            |
| AmoebaNet-A + cutoutReal et al. (2019) | 3.67          | 2.3        | 3150     | Evolution     |
| PNAS Liu et al. (2018a)           | 3.89          | 2.5        | 225      | SMBO          |
| ENAS+cutout Hieu et al. (2018)    | 3.79          | 2.6        | 0.5      | RL            |
| DARTS(2nd order) + cutout (Liu et al. 2018b) | 3.77          | 2.2        | 1.6      | Gradient      |
| Random Sample Liu et al. (2018b)  | 3.95          | 2.5        | –        | –             |
| SNAS + cutout Liu et al. (2018c)  | 3.80          | 2.3        | 1.50     | Gradient      |
| GDAS + cutout Xuanyi and Yang (2019) | 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 4  Comparison with the state-of-the-art image classification methods on ImageNet

| Architecture                      | Accuracy (%) | Params (M) | GPU days | Search Method |
|-----------------------------------|--------------|------------|----------|---------------|
| Inception-V1 Szegedy et al. (2015) | 69.8         | 89.9       | 6.6      | Manual        |
| MobileNetV2 Howard et al. (2017)  | 72.0         | 91.0       | 3.4      | Manual        |
| ShuffleNetV2 Zhang et al. (2018) | 73.7         | –          | ~5       | Manual        |
| NASNet-A Zoph et al. (2018)       | 74.0         | 91.6       | 5.3      | 1800          | RL            |
| NASNet-B Zoph et al. (2018)       | 72.8         | 91.3       | 5.3      | 1800          | RL            |
| NASNet-C Zoph et al. (2018)       | 72.5         | 91.0       | 4.9      | 1800          | RL            |
| AmoebaNet-A Real et al. (2019)    | 74.5         | 92.0       | 5.1      | 3150          | Evolution     |
| AmoebaNet-B Real et al. (2019)    | 74.0         | 91.5       | 5.3      | 3150          | Evolution     |
| AmoebaNet Real et al. (2019)      | 75.7         | 92.4       | 6.4      | 3150          | Evolution     |
| PNAS Liu et al. (2018a)           | 74.2         | 91.9       | 5.1      | 225           | SMBO          |
| MnasNet Tan et al. (2019)         | 74.8         | 92.0       | 4.4      | –             | RL            |
| DARTS Liu et al. (2018b)          | 73.1         | 91.0       | 4.9      | 1.6           | Gradient      |
| SNAS Liu et al. (2018c)           | 72.7         | 90.8       | 4.3      | 1.5           | Gradient      |
| GDAS Xuanyi and Yang (2019)       | 74.0         | 91.5       | 5.3      | 0.21          | Gradient      |
| LFR-DARTS (Ours)                  | 74.5         | 91.7       | 4.9      | 0.45          | Gradient      |

Fig. 3  Loss learning curves of neural networks on three datasets (a: CIFAR10, b: FashionMNIST, c: ImageNet)
4.3.2 Effectiveness of entropy constraint

In this section, we experimentally verify the effectiveness of the entropy constraint mentioned in Sect. 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 $\lambda$ is closely related to the distribution of the importance score, so an appropriate scaling factor is a key. We conduct experiments to compare the effects of different values of $\lambda$ 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 $\lambda$ 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 $\lambda$ has different impacts on the results. In the initial phase of the training
The distribution of importance scores varies as training time under different $\lambda$ values. Figure 6 shows that the importance is a random distribution. After a period of training, 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 $\lambda$ could lead to unsatisfactory convergence results. As we can see, it gets worse while $\lambda \geq 0.05$. 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 net-
work 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 (grant number 61772120).

Author 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.

Funding This study was funded by National Natural Science Foundation of China (grant number 61772120).

Data availability The data that support the this study are available: CIFAR10 from “http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz”, fashionMNIST from “https://github.com/zalandoresearch/fashion-mnist”, and ImageNet from “https://image-net.org/challenges/LSVRC/2012/index.php”.

Declarations

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.

References

Baker B, Gupta O, Naik N, Raskar R (2016) Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167

Bello I, Zoph B, Vasudevan V, Le QV (2017) Neural optimizer search with reinforcement learning. arXiv preprint arXiv:1709.07417

Bobadilla J, Lara-Cabrera R, González-Prieto Á, Ortega F (2021) DeepFair: deep learning for improving fairness in recommender systems. Int J Interact Multimed Artif Intell 6(6):86–94

Bramer M (2007) Principles of data mining, vol 180. Springer, Berlin

Cai H, Chen T, Zhang W, Yu Y, Wang J (2018) Efficient architecture search by network transformation. In thirty-second AAAI conference on artificial intelligence

Cai Z, Yang X, Huang T, Zhu W (2020) A new similarity combining reconstruction coefficient with pairwise distance for agglomerative clustering. Inf Sci 508:173–182

Cai Z, Zhu W (2018) Multi-label feature selection via feature manifold learning and sparsity regularization. Int J Mach Learn Cybernet 9(8):1321–1334

Chu X, Zhou T, Zhang B, et al. (2020) Fair darts: eliminating unfair advantages in differentiable architecture search. In European conference on computer vision, pp 465–480

Deng J, Dong W, Socher R, Li L-I, Li K, Fei-Fei L (2009) Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, Ieee, pp 248–255

Dong X, Yi Y (2019) Searching for a robust neural architecture in four gpu hours. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1761–1770

Elksen T, Metzen JH, Hutter F (2018) Neural architecture search: a survey. arXiv preprint arXiv:1808.05377

Graves A, Mohamed A-R, Hinton G (2013) Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing, IEEE, pp 6645–6649

Guyon I, Gunn S, Nikravesh M, Zadeh LA (2008) Feature extraction: foundations and applications, vol 207. Springer, Berlin

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778

Heydarpour F, Abbasi E, Ebadi MJ, Karbassi Seyed-Mehdi (2016) Solving an Optimal Control Problem of Cancer Treatment by Artificial Neural Networks. In International Journal of Interactive Multimedia & Artificial Intelligence, 6(4)

Hinton G, Deng L, Yu D, Dahl G, Mohamed A-R, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Kingsbury B, et al. (2012) Deep neural networks for acoustic modeling in speech recognition. IEEE Signal processing magazine, 29

Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861

Hu J, Shen L, Sun G (2018) Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7132–7141

Huang G, Liu Z, Van Der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4700–4708
Ioffe S, Christian S (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167

Hao J, Zhu W (2021) Saliency: a new selection criterion of important architectures in neural architecture search. In Neural Computing and Applications, 1–15

Liang H, Zhang S, Sun J, et al. (2019) Darts+: improved differentiable architecture search with early stopping. In arXiv preprint arXiv:1909.06035

Liu C, Zoph B, Neumann M, Shlens J, Hua W, Li L-J, Li F-F, Alan Y, Jonathan H, Kevin M (2018a) Progressive neural architecture search. In Proceedings of the European Conference on Computer Vision (ECCV), pp 19–34

Liu H, Simonyan K, Vinyals O, Fernando C, Koray K (2017) Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436

Liu H, Simonyan K, Yiming Y (2018b) Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055

Lotfollahi M, Siavoshani MJ, Zade RSH, Saberian M (2020) Deep packet: a novel approach for encrypted traffic classification using deep learning. Soft Comput 24(3):1999–2012

Ma B, Li X, Xia Y, Zhang Y (2020) Autonomous deep learning: a genetic dcnn designer for image classification. Neurocomputing 379:152–161

Nixon M, Aguado A (2019) Feature extraction and image processing for computer vision. Academic press, Cambridge

Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z (2017) Automatic differentiation in pytorch, Alban Desmaison

Pham H, Guan MY, Zoph B, Le QV, Dean J (2018) Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268

Ravi V, Zimmermann H-J (2001) A neural network and fuzzy rule base hybrid for pattern classification. Soft Comput 5(2):152–159

Real E, Aggarwal A, Yanping H, Quoc VL (2019) Regularized evolution for image classifier architecture search. In proceedings of the AAAI conference on artificial intelligence, vol 33, pp 4780–4789

Real E, Moore S, Andrew S, Saurabh S, Yutaka Leon S, Jie T, Quoc V L, Alexey K (2017) Large-scale evolution of image classifiers. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp 2902–2911. JMLR. org

Saumya S, Singh JP, Dwivedi YK (2019) Predicting the helpfulness score of online reviews using convolutional neural network. Soft Computing, pp 1–17

Simonyan K, Vedaldi A, Zisserman A (2013) Deep inside convolutional networks: visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034

Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556

Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pp 3104–3112

Szegedy C, Wei L, Yangqing J, Pierre S, Scott R, Dragomir A, Dumitru E, Vincent V, Andrew R (2015) Going deeper with convolutions. pp 1–9

Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le QV. (2019) Mnasnet: Platform-aware neural architecture search for mobile. In proceedings of the IEEE conference on computer vision and pattern recognition, pp 2820–2828

Verma KK, Singh BM, (2021) Deep multi-model fusion for human activity recognition using evolutionary algorithms. In International Journal Of Interactive Multimedia And Artificial Intelligence, In press, 1–15

Wang H, Wang H, Xu K (2020) Evolutionary recurrent neural network for image captioning. Neurocomputing 401:249–256

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.