J. Nonlinear Var. Anal. 5 (2021), No. 1, pp. 119-140
Available online at http://jnva.biemdas.com
https://doi.org/10.23952/jnva.5.2021.1.08

ARCHITECTURE SELF-ATTENTION MECHANISM: NONLINEAR OPTIMIZATION FOR NEURAL ARCHITECTURE SEARCH

JIE HAO, WILLIAM ZHU*

Institute of Fundamental and Frontier Sciences,
University of Electronic Science and Technology of China, Chengdu, China

Abstract. Neural Architecture Search (NAS) is a very prevalent method of automatically designing neural network architectures. It has recently drawn considerable attention since it relieves the manual design labour of neural networks. However, existing NAS methods ignore the interrelationships among candidate architectures in the search space. As a consequence, the objective neural architecture extracted from the search space suffers from performance unstable due to the interrelationship collapse. In this paper, we propose architecture self-attention mechanism for neural architecture search (ASM-NAS) to address the above problem. Specifically, the proposed architecture self-attention mechanism constructs the interrelationships among architectures by interacting information between any two candidate architectures. Through learning the interrelationships, it selectively emphasizes some architectures important to the network while suppressing unimportant ones, which provides significant references for the architecture selection. Therefore, we improves the performance stability of the architecture search by the above startegy. Besides, our proposed method is high-efficiency and executes architecture search with low time and space costs. Compared to other advanced NAS approaches, our ASM-NAS is able to achieve better architecture search performance on the image classification datasets of CIFAR10, CIFAR100, fashionMNIST and ImageNet.
Keywords. Neural architecture search; Nonlinear optimization; Self-attention; Candidate architectures.

1. Introduction

In recent years, deep neural networks [1, 2, 3, 4, 5, 6] have achieved great success in many computing tasks due to their powerful capabilities of feature extraction [7, 8, 9] and data mining [10, 11, 12]. But designing high-quality neural networks requires a large amount of expert knowledge and sufficient experiments. To eliminate such a tough handcraft process, neural architecture search (NAS) [13, 14, 15] has been proposed to automatically discover advanced neural networks from a predefined search space. Impressively, NAS has recently drawn considerable attention since it outperforms hand-crafted architectures in many important tasks.

Early NAS approaches, *i.e.*, based on reinforcement learning (RL) [13, 14] and evolutionary algorithms (EA) [15, 16], require sampling and evaluating the candidate architectures from a

*Corresponding author.
E-mail address: wfzhu@uestc.edu.cn (W. Zhu).
Received December 29, 2020; Accepted January 18, 2021.

search space repeatedly. During this process, each candidate architecture is learned independently and is discarded if its performance is not competitive. As a result, these methods need excessive computational resources to search for optimal neural architecture.

To overcome this problem, many recent result focus on improving the computational efficiency of NAS by the weight sharing strategy [17, 18, 19]. They encode the search space into a single over-parameterized super network which contains all possible architectures, *i.e.*, network operation types (*e.g.*, convolution, pooling) and connection styles. Each network operation or connection is assigned a weight parameter to learn its importance. All the candidate architectures share the weight parameters which are optimized together in the super network. The super network is trained once and then the optimal neural architecture will be identified based on the value of shared weights. In this way, the calculation cost is significantly reduced. However, the weight sharing based NAS methods, such as DARTS [18], ignore the interrelationships among candidate architectures (*i.e.*, operations or connections) during the search. As a consequence, the objective neural architecture extracted from the super network suffers from performance unstable due to the interrelationship collapse.

In this paper, we propose architecture self-attention mechanism which is a nonlinear optimization for neural architecture search. The architecture self-attention mechanism constructs the interrelationships among architectures by interacting information between any two candidate architectures. Through learning the interrelationships, it is able to allocate the network attention to the architectures more reasonably to selectively emphasizes some architectures important to the network while suppressing less useful ones. Then we execute architecture selection according to the importance of candidate architectures. Besides, our architecture search is high-efficiency since the architecture self-attention mechanism changes optimization process and reduces the calculational complexity.

We evaluate the effectiveness of our method on the image classification tasks of CIFAR10, CIFAR100, fashionMNIST and ImageNet. We could discover an optimal neural network with only 0.55 GPU-days on a single NVIDIA GTX2080Ti GPU. The optimal neural network achieves state-of-the-art performance on CIFAR10 (2.49% test error rate), CIFAR100 (15.60% test error rate) and fashionMNIST (3.70% test error rate), respectively. Furthermore, we transfer the neural architecture searched on CIFAR10 to the complicated dataset ImageNet to verify the stability and generalization performance of our method. Under the MobileNet settings as [20], our neural network achieves high performance of 74.6% accuracy with 5.5M parameters.

2. Related Work

Neural architecture search. Neural architecture search has received significant attention in the last few years. The goal is to find automatic methods of designing neural architectures to replace conventional handcrafted ones. To this end, many search algorithms have been proposed to discover optimal architectures by specific search strategies. Pioneer works on NAS employ reinforcement leanring (RL) or evolutionary algorithms (EA) to search for the best architecture. These methods need to repeat this process: sampling a neural architecture from a predefined search space and then evaluating it. RL based methods [13, 14, 17, 21] will optimize its policy by the evaluation result and EA based methods [15, 22, 23] update its population or gene according to the result. However, these methods require excessive computational overhead though achieving an advanced performance. For example, NASNet [14] needs 500 GPUs over four days to finish the architecture search.

To solve the problem of computational overhead, many works [18, 24, 25, 26] adopt the weight sharing strategy where all the candidate architectures in the search space share the weights (or architecture parameters) that are trained together by the gradient descent. In this way, this kind of methods eliminate the repeated process of sampling and evaluating candidate architectures. DARTS [18] converts the discrete search space into a continuous one by building a super network that subsumes all the possible candidate architectures. It trains the shared weights of architecture candidates to represent the importance of these architectures. PDARTS [26], based on DARTS, addresses the performance difference of the search network and evaluation network by progressively increasing the depth of searched architectures. GDAS [25] develops a differentiable sampler over the search space to avoid simultaneously training all the candidate architectures in the search space. PC-DARTS [27] addresses the memory efficiency problem of DARTS by sampling a small part of the super network to reduce the redundancy in network space. Based on the typical weight sharing method DARTS, we propose architecture self-attention mechanism to learn the interrelationship among the candidate architectures. Our architecture self-attention mechanism also could be seamlessly embedded into other weight sharing methods.

Self-attention mechanism. It is well known that attention plays an important role in human perception [28, 29]. The human visual system selectively focuses on salient parts of images in order to capture visual structure better. Our architecture self-attention mechanism is related to self-attention [30] and SENet [6] for computer vision. A self-attention module computes the response at a position in an image or a sequence by attending to all positions and taking their weighted average in an embedding space. It can allocate the available processing resources towards the most informative components of input. For example, SENet focuses on the channel relationship of convolution neural networks and proposes a novel architecture unit, "Squeeze-and-Excitation", to capture this relationship. CBAM [31], compared to SENet, adds the spatial dimensions into the attention to boost the model representation.

3. Preliminaries: A weight sharing based NAS method: DARTS

For the case of convolutional neural networks, DARTS [18] searches for a normal cell and a reduction cell to build up a final target network. A cell is considered as a directed acyclic graph (DAG) with N sequential nodes, $\{x_0, x_1, ..., x_{N-1}\}$, where x_0, x_1 are input nodes of this cell, x_{N-1} is the output node, and the others are intermediate nodes. Each node is connected to all its predecessors by edges, and there is a set of candidate operations on each edge. The set of candidate operations includes: $\{$*zeroize, max_pool_3×3, avg_pool_3×3, skip_connect, sep_conv_3×3, sep_conv_5×5, dil_conv_3×3, dil_conv_5×5*$\}$, which is denoted as the operation space $\mathscr{O}$. A candidate operation $o(.)$ performs a specific transformation over the input and then delivers the result to its next node. This transformation from node i to the next node j could be represented by mixed operations $\overline{o}_{i,j}(x_i)$:

$$\overline{o}_{i,j}(x_i) = \sum_{k=1}^{|\mathscr{O}|} \frac{\exp(\alpha^k_{o_{i,j}})}{\sum_{\ell=1}^{|\mathscr{O}|} \exp(\alpha_{o^\ell_{i,j}})} o^k_{i,j}(x_i), \tag{3.1}$$

where each operation $o^k_{i,j}(.)$ is associated with an architecture parameter $\alpha^k_{o_{i,j}}$, which is trained to represent the importance of its corresponding operation. x_i is the feature map of node i. The intermediate node j aggregates all inputs from its predecessors:

$$x_j = \sum_{i<j} \bar{o}_{i,j}(x_i). \tag{3.2}$$

The output node concats the results of the intermediate nodes in the channel dimension:

$$x_{N-1} = \text{concat}(x_2, x_3, ..., x_{N-2}). \tag{3.3}$$

The architecture search problem is essentially to learn optimal architecture parameters α and network weights w to minimize validation loss $\mathscr{L}_{val}(w^*(\alpha), \alpha)$. DARTS resolves this problem with a bi-level optimization:

$$\begin{aligned} &\min_{\alpha} \mathscr{L}_{val}(w^*(\alpha), \alpha) \\ \text{s.t.} \quad & w^*(\alpha) = \operatorname*{argmin}_{w} \mathscr{L}_{train}(w, \alpha). \end{aligned} \tag{3.4}$$

This optimization process is performed alternately for a fixed number of steps. Then, DARTS selects the operations with maximal *softmax* value of $\alpha_{o_{i,j}}$ from the candidates to connect each pair of nodes in the cells. Finally, DARTS generates two cells, a normal cell (the size of output tensor is equal to input tensor) and a reduction cell (the size of output tensor is half of the input tensor).

4. Method

To solve performance unstable in architecture search, we propose architecture self-attention mechanism which optimizes for neural architecture search in Section 4.1. Based our architecture self-attention mechanism, we develop a lightweight network module named architecture self-attention module which is conveniently integrated into a super network for architecture search. The detailed content is described in Section 4.2. Then, we show the concrete architecture search process with the architecture self-attention module in Section 4.3. Finally, the theoretical convergence analysis of our algorithm is presented in Section 4.4. Our method can capture the interrelationships among different operations to provide a more valuable reference for architecture selection.

4.1. **Architectural self-attention mechanism.** Using self-attention as a primary mechanism for representation learning has seen widespread adoption in natural language processing and computer vision. The ability of self-attention to learn to focus on important regions within a context or an image has made it a critical component in neural transduction models for several modalities. Self-attention mechanism computes the response at a position in an image or a sequence by attending to all positions. Then it produces an attention coefficient and uses the coefficient to multiply each position or region of inputs to weight this part. Inspired by the self-attention mechanism, we propose a generic architecture self-attention mechanism, a nonlinear optimization for neural architecture search. This mechanism focus on important operations in the super network and allocates more attention to these operations by learning. Specifically, we compute the attention coefficient $a^k_{i,j}$ of an operation $o^k_{i,j}$ by interacting it with other candidate

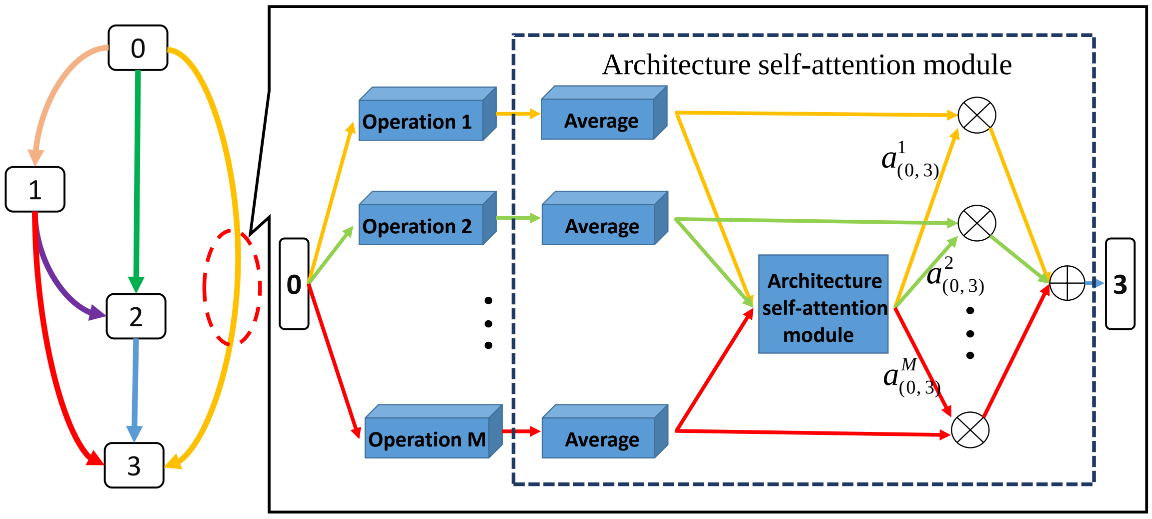


FIGURE 1. Illustration of the proposed approach, *Architecture Self-attention Mechanism: Nonlinear Optimization for Neural Architecture Search* (ASM-NAS). As an example, we show the process how information is propagated from node 0 to node 3 in a cell (suppose a cell contains 4 nodes). There are M candidate operations on an edge connecting each pair of nodes. Different color lines represent different operation paths. This architecture self-attention module aggregates the information from each operation and allocating attention coefficient $a^k_{i,j}$ on the different operation.

operations $o^{\rho}_{i,j}|_{\rho=1,2,\ldots|\mathscr{O}|}$ in the search space:

$$a^k_{i,j} = \frac{1}{\gamma}\sum_{\rho=1}^{\mathscr{O}} f(g(o^k_{i,j}(x_i)), g(o^{\rho}_{i,j}(x_i))), \tag{4.1}$$

where the function $f(.,.)$ computes a response between the k-th operation and other candidate operations. The function $g(.)$ computes a representation of an operation. For simplicity, we refer to $f(.,.)$ as a relational function and $g(.)$ as a representation function. At last, the result is normalized by a factor γ.

Then each candidate operation $o^k_{i,j}$ is multiplied by its corresponding attention coefficient $a^k_{i,j}$. According Eq. 4.1, the mixed operations in Eq. 3.1 from node i to the next node j is formulated as:

$$\bar{o}_{i,j}(x_i) = \sum_{k=1}^{|\mathscr{O}|} a^k_{i,j} o^k_{i,j}(x_i). \tag{4.2}$$

Therefore, the transformation result is the sum of all candidate operations weighted by their respective attention coefficients. Each coefficient is embedded with the interrelationships from other candidate operations. By the network optimization, *i.e., stochastic gradient descent (SGD)*, the attention coefficients are progressively trained to an optimal distribution *w.r.t.*, operations. Then, we perform architecture selection according to the attention coefficients the attention coefficients.

4.2. **Architectural self-attention module in NAS.** To make architecture self-attention mechanism work well in our NAS, we develop architectural self-attention module which is a nonlinear module and could be conveniently integrated into the network layer. The architectural self-attention module achieves the function of Eq. 4.1 to produce the attention coefficients and weights each operation by its corresponding coefficient. Thus, our module contains three parts: relational function $f(.,.)$, representation function $g(.)$ and weighting operations by attention coefficients. To this end, we first need to find an appropriate relational function $f(.,.)$ and a representation function $g(.)$ since they play a critical role in capturing the interrelationships. We first consider the representation function $g(.)$, which obtains the information embedding of each operation. Formally, the outputs of a set of candidate operations can be denoted as $\mathbf{Y}_{i,j} = [\mathrm{y}_{i,j}^1, \mathrm{y}_{i,j}^2, \ldots, \mathrm{y}_{i,j}^{|\mathscr{O}|}]$, where each element $\mathrm{y}_{i,j}^k = o_{i,j}^k(x_i)$ and $\mathrm{y}_{i,j}^k \in \mathbb{R}^{H\times W\times C}$.

To effectively represent the output, we use average feature of $\mathrm{y}_{i,j}^k$ as its information embedding, which is denoted as:

$$\begin{aligned} e_{i,j}^k &= g(\mathrm{y}_{i,j}^k) \\ &= \frac{1}{H\times W\times C}\sum_{l=1}^{H}\sum_{m=1}^{W}\sum_{n=1}^{C}\mathrm{y}_{i,j}^k[l,m,n], \end{aligned} \tag{4.3}$$

where $[l,m,n]$ is the position index of an output tensor. The average scalar $e_{i,j}^k$ can be interpreted as an embedding whose statistics are expressive for the whole output of an operation.

The relational function $f(.,.)$ aims to compute interrelationships by interacting between any two operations. To achieve this goal, the function must meet two criteria: (1) it can effectively learn the interrelationships at a low cost of time and space; (2) it must be structure-friendly so that it can be easily applied to the neural network layer. But it is actually quite difficult to extract this kind of abstract relationship by an explicit mathematical function. We employ a simple module with 4 fully-connected and activation layers to simulate the action mechanism of $\sum_{\rho=1}^{\mathscr{O}} f(g(o_{i,j}^k(x_i)), g(o_{i,j}^\rho(x_i)))$ in Eq. 4.1. Each module, following the function $g(.)$, acts on $\mathbf{E}_{i,j} = [e_{i,j}^1, e_{i,j}^2, \ldots, e_{i,j}^{|\mathscr{O}|}]$ and then outputs $\mathbf{z}_{i,j}$,

$$\mathbf{z}_{i,j} = \sigma(\Theta_4\delta(\Theta_3\delta(\Theta_2\delta(\Theta_1\mathbf{E}_{i,j})))), \tag{4.4}$$

where $\delta(.)$ refers to the nonlinear activation function ReLU [32] function and $\sigma(.)$ refers to the nonlinear activation function sigmoid function, $\Theta = \{\Theta_1, \Theta_2, \Theta_3, \Theta_4\}$ are the learnable weights of 4 fully-connected layers, respectively. The weights Θ are part of the network parameters and are optimized together with other network parameters w. Then the output vector $\mathbf{z}_{i,j}$ is normalized by the *softmax* function. According to Eq. 4.1 and Eq. 4.4, we obtain attention coefficient $a_{i,j}^k$ as follows:

$$a_{i,j}^k = softmax(\mathbf{z}_{i,j}, k) = \frac{\exp(z_{i,j}^k)}{\sum_{\ell=1}^{|\mathscr{O}|}\exp(z_{i,j}^\ell)}. \tag{4.5}$$

The attention coefficients can be calculated efficiently by Eq. 4.5. Then the attention coefficients are brought into Eq. 4.2, and the weighted operation outputs could be obtained. Through the optimization of the super network, the coefficients will focus on the important operations of the super network. Meanwhile, the impacts of unimportant operations on the results (*i.e.,* network loss) will be suppressed. Refer to Figure. 1 for more details about our method and architecture self-attention module.

4.3. **Architecture search.** The architecture search process consists of two stages, architecture training and architecture selection. We execute architecture training on the training dataset and architecture selection on the validation dataset, respectively. The network weights (w,Θ) are optimized during architecture training, and then those operations corresponding to maximal attention coefficients $a=\{a_{0,1}^{1},...,a_{i,j}^{k}...\}$ are recognized as optimal ones among candidates. Our objective function is designed as follows:

$$\begin{aligned} &\min_{a} \mathscr{L}_{val}(w,\Theta) \\ \text{s.t.} \quad & w,\Theta = \operatorname{argmin} \mathscr{L}_{train}(w,\Theta), \end{aligned} \tag{4.6}$$

where the learnable parameters w and Θ are both network weights that can be optimized simultaneously by stochastic gradient descent.

Architecture training. We first construct an over-parameterized super network that contains L cells with architecture self-attention modules. Then we train the super network on the training dataset to minimize the network loss $\mathscr{L}_{train}(w,\Theta)$.

Architecture selection. When this super network is trained for a large number of iterations, we execute architecture selection on the validation dataset. At this stage, we will obtain L optimal cells in order by architecture selection. Specifically, the structure and network weights of the super network remain fixed throughout this stage. For each cell in the super network, we perform network forward propagation on the whole validation dataset and simultaneously count the attention coefficient $a_{i,j}^{k}$ of each operation. The operation with maximal $a_{i,j}^{k}$ among a set of candidate operations is kept, and other operations are removed from the cell. Notably, there is no network backward propagation at this stage, so architecture selection is carried out efficiently.

Discussion. DARTS resolves the architecture search problem as a bi-level optimization. But differently, our method only needs to optimize the network weights (Θ are also part of the network weight parameters) as does that in a conventional neural network. Therefore, we change the optimization process of super network and reduce the calculational complexity. That saves plenty of time for our architecture search compered to other weight sharing based methods, such as DARTS. Besides, our architecture self-attention modules produce adaptive architectures for different network layers. Our method also improves the flexibility and diversity of the neural architecture, which leads to better network generalization.

4.4. **Convergence analysis of Algorithm.** During architecture search, the parameters (w,Θ) in super network $\mathscr{G}$ are optimized by *stochastic gradient descent* (SGD) method, as described in Step 4 of Algorithm 1. For simplicity, we represent the parameters (w,Θ) by $\{\omega\}$ as both w and Θ are free network parameters. The training data $\mathscr{D}_{train}=\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$ is uniformly distributed, where x_i is a input data and y_i is the corresponding label (or target) of x_i. In particular, let us represent a sample (x_i,y_i) by a random variable ξ_i. Feed a input data x_i into the super network $\mathscr{G}$, and get the prediction output $\hat{y}_i$. we represent this process as:

$$\hat{y}_i = f(x_i,\omega), \tag{4.7}$$

Algorithm 1 Architecture Self-attention Mechanism: Nonlinear Optimization for Neural Architecture Search (ASM-NAS)

```
Input: Training data $\mathscr{D}_{train}$; Validation data $\mathscr{D}_{val}$; Operation space $\mathscr{O}$; The number of cells $L$
Output: The optimal cells
 1: Initialize a super network $\mathscr{G}$ consisting of $L$ cells with architecture self-attention modules.
 2: Initialize network weights $(w, \Theta)$.
 3: while not converged do
 4:     Update the weights $(w, \Theta)$ of $\mathscr{G}$ by descending $\nabla_{w,\Theta}\mathscr{L}_{train}(w, \Theta)$.
 5: end while
 6: $\ell = 0$.
 7: while $\ell < L$ do
 8:     Forward propagate the network $\mathscr{G}$ on $\mathscr{D}_{val}$.
 9:     Count all the attention coefficients $a$ in the cell$_\ell$.
10:     Replace $\bar{o}_{i,j}$ with $o^k_{i,j} = \text{argmax}_{k \in |\mathscr{O}|} a^k_{i,j}$ for each edge $(i, j)$ in the cell$_\ell$.
11:     $\ell = \ell + 1$.
12: end while
13: return The optimal cell architectures [cell$_1$, cell$_2$, ..., cell$_L$].
```

where $f(.) : X \to Y$ donotes the function transformation of the network $\mathscr{G}$, from an input space X to an output space Y. If loss function is r, then the expctection loss on n data samples is:

$$F(\omega) = \mathbb{E}[\mathscr{L}_{train}(\omega)] = \frac{1}{n}\sum_{i=1}^{n} r(f(x_i, \omega), y_i). \tag{4.8}$$

The objective function F taken respect to parameters ω is optimized by gradient descent to reach to the minimal expected loss. We use $g(\omega_i, \xi_i)$ to represent a stochastic gradient, *i.e.*,

$$g(\omega_i, \xi_i) = \nabla r(f(x_i, \omega), y_i). \tag{4.9}$$

The update of the parameters ω is defined by the iteration:

$$\omega_{k+1} \leftarrow \omega_k - \eta \frac{1}{n}\sum_{i=1}^{n} g(\omega_i, \xi_i) = \omega_k - \eta \nabla F(\omega_k), \tag{4.10}$$

where k indicates the k-th iteration, and η is the stepsize of the update in each iteration.

To establish the convergence guarantees for our algorithm, we present two fundamental lemmas based on the previous work [33]. The two lemmas are built upon an assumption of smoothness of the objective function and an assumption about first and second moments of stochastic vectors $\{g(\omega_i, \xi_i)\}$. In particular, the optimization of a deep neural network is essentially a non-convex optimization problem. Thus, the following convergence analysis of our algorithm is based on non-convex optimization.

Assumption 4.1 (Lipschitz-continuous objective gradients). The objective function $F : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable and the gradient function of F, namely, $\nabla F : \mathbb{R}^d \to \mathbb{R}^d$, is Lipschitz continuous with Lipschitz constant $L > 0$, *i.e.*,

$$\|\nabla F(\omega) - \nabla F(\overline{\omega})\|_2 \le L\|\omega - \overline{\omega}\|_2 \text{ for all } \{\omega, \overline{\omega}\} \subset \mathbb{R}^d. \tag{4.11}$$

where ω and $\overline{\omega}$ are two arbitrary variables of F.

Assumption 4.1 ensures the gradient of F does not change arbitrarily quickly with respect to the parameters. Based on this assumption, an important consequence is that

$$F(\omega) \leq F(\bar{\omega}) + \nabla F(\overline{\omega})^T(\omega - \overline{\omega}) + \frac{1}{2}L\|\omega - \overline{\omega}\|_2^2 \text{ for all } \{\omega, \overline{\omega}\} \subset \mathbb{R}^d. \tag{4.12}$$

The proof of inequality 4.12 is proved in Appendix B.1.

Under Assumption 4.1, we can obtain Lemma 4.1.

Lemma 4.1. *Under Assumption 4.1, the iterations of SGD satisfy the following inequality for all $k \in \mathbb{N}$, where k is the index of the number of algorithm iterations:*

$$\mathbb{E}_{\xi_k}[F(\omega_{k+1})] - F(\omega_k) \leq -\eta \nabla F(\omega_k)^T \mathbb{E}_{\xi_k}[g(\omega_k, \xi_k)] + \frac{1}{2}\eta^2 L \mathbb{E}_{\xi_k}[\|g(\omega_k, \xi_k)\|_2^2]. \tag{4.13}$$

where $\mathbb{E}_{\xi_k}[.]$ is an expected value taken respect to the distribution of the random sample ξ_k given ω_k. For simplicity, ξ_k represents random samples in the k-th iteration, not a single sample, as mentioned above. Please refer to Appendix B.2 for detailed proof of inequality (4.13).

In order to limit the harmful effect of $\mathbb{E}_{\xi_k}[\|g(\omega_k, \xi_k)\|_2^2]$ in ineuqlity (4.13), we restrict the variance of $g(\omega_k, \xi_k)$:

$$\mathbb{V}_{\xi_k}[g(\omega_k, \xi_k)] := \mathbb{E}_{\xi_k}[\|g(\omega_k, \xi_k)\|_2^2] - \|\mathbb{E}_{\xi_k}[g(\omega_k, \xi_k)]\|_2^2. \tag{4.14}$$

Next, we further give the assumption of first and second moments about $g(\omega_k, \xi_k)$.

Assumption 4.2 (First and second moment limits). The objective function F and SGD algrithm satisfy the following:

(a) The sequence of iteration $\{\omega_k\}$ is contained in an open set over which F is bounded below by a scalar F_{inf}.

(b) There exist scalars $\mu_G \geqslant \mu > 0$ such that, for all $k \in \mathbb{N}$,

$$\nabla F(\omega_k)^T \mathbb{E}_{\xi_k}[g(\omega_k, \xi_k)] \geq \mu \|\nabla F(\omega_k)\|_2^2 \text{ and} \tag{4.15a}$$

$$\|\mathbb{E}_{\xi_k}[g(\omega_k, \xi_k)]\|_2 \leq \mu_G \|\nabla F(\omega_k)\|_2. \tag{4.15b}$$

(c) There exist scalars $M \geqslant 0$ and $M_V \geqslant 0$ such that, for all $k \in \mathbb{N}$,

$$\mathbb{V}_{\xi_k}[g(\omega_k, \xi_k)] \leq M + M_V \|\nabla F(\omega_k)\|_2^2. \tag{4.16}$$

The Assumption 4.2(a) requires the objective function F to be bounded bellow a scalar over the explored region. Assumption 4.2(b) states that the expected value of $-g(\omega_k, \xi_k)$ is a direction of sufficent descent for F from ω_k. Assumption 4.2(c) states that the variance of $g(\omega_k, \xi_k)$ is restricted and proportional to $\|\nabla F(\omega_k)\|_2^2$ if noise $M = 0$.

By Assumption 4.2 and Definition 4.14, the second moment of $g(\omega_k, \xi_k)$ satisfies

$$\mathbb{E}_{\xi_k}[\|g(\omega_k, \xi_k)\|_2^2] \leq M + M_G \|\nabla F(\omega_k)\|_2^2 \text{ with } M_G := M_V + \mu_G^2 \geq \mu^2 > 0. \tag{4.17}$$

Lemma 4.2. *Under Assumptions 4.1 and 4.2, the iterations of SGD satisfy the following inequalities for all $k \in \mathbb{N}$:*

$$\mathbb{E}_{\xi_k}[F(\omega_{k+1})] - F(\omega_k) \leq -\mu\eta \|\nabla F(\omega_k)\|_2^2 + \frac{1}{2}\eta^2 L \mathbb{E}_{\xi_k}[\|g(\omega_k, \xi_k)\|_2^2] \tag{4.18a}$$

$$\leq -(\mu - \frac{1}{2}\eta L M_G)\eta \|\nabla F(\omega_k)\|_2^2 + \frac{1}{2}\eta^2 L M. \tag{4.18b}$$

The detailed proof for ineuqality (4.18b) is presented in Appendix B.3.

According to the above Lemmas 4.1 and 4.2, if the stepsize η satisfies

$$0 < \eta \leq \frac{\mu}{LM_G}. \tag{4.19}$$

we obtain from the total expectation of (4.18b) that

$$\begin{aligned}\mathbb{E}[F(\omega_{k+1})] - \mathbb{E}[F(\omega_k)] &\leq -(\mu - \frac{1}{2}\eta LM_G)\eta \mathbb{E}[\|\nabla F(\omega_k)\|_2^2] + \frac{1}{2}\eta^2 LM \\ &\leq -\frac{1}{2}\mu\eta\mathbb{E}[\|\nabla F(\omega_k)\|_2^2] + \frac{1}{2}\eta^2 LM.\end{aligned} \tag{4.20}$$

Summing $k \in \{1, ..., K\}$ on both sides of the above inequality, we arrive at

$$F_{\inf} - F(\omega_1) \leq \mathbb{E}[F(\omega_{K+1})] - F(\omega_1) \leq -\frac{1}{2}\mu\eta \sum_{k=1}^{K} \mathbb{E}[\|\nabla F(\omega_k)\|_2^2] + \frac{1}{2}K\eta^2 LM. \tag{4.21}$$

Rearranging inequality (4.21), one finds that the expected sum or average of squares of ∇F corresponding to SGD iterations satisfy the following inequalities for all $K \in \mathbb{N}$ (K is the total number of algorithm iterations):

$$\mathbb{E}[\sum_{k=1}^{K} \|\nabla F(\omega_k)\|_2^2] \leq \frac{K\eta LM}{\mu} + \frac{2(F(\omega_1) - F_{\inf})}{\mu\eta}, \tag{4.22a}$$

$$\mathbb{E}[\frac{1}{K}\sum_{k=1}^{K} \|\nabla F(\omega_k)\|_2^2] \leq \frac{\eta LM}{\mu} + \frac{2(F(\omega_1) - F_{\inf})}{K\mu\eta} \xrightarrow{K\to\infty} \frac{\eta LM}{\mu}. \tag{4.22b}$$

If $M = 0$, *i.e.*, there is no noise, then the norm of the gradient in inequality (4.22b) remains finite. If the difference $\mathbb{E}[F(\omega_{k+1})] - \mathbb{E}[F(\omega_k)]$ in inequality (4.20) is bounded below a deterministic quanity, then our algorithm converges to an optimal point. If $M > 0$, *i.e.*, there exists noise, then the norm of the gradient of the objective function F in the inequality (4.22b) is related to the update stepsize η, the total number of iterations K. If we adjust the stepsize η small enough and increase the iterations K, our algorithm converges to the neighbourhood of the optimal point. The above convergence analysis suggests that we need to choose an appropriate update stepsize (*i.e.*, learning rate) and the total number of iterations during architect search to make our super network $\mathscr{G}$ converge to the optimality.

5. Experiments

We demonstrate the effectiveness of our proposed method on four popular image datasets (*i.e.,* CIFAR10, CIFAR100 [34], fashionMNIST, and ImageNet [35]). CIFAR10 consists of 60K images, all of which are of a spatial resolution of 32 × 32. These images are equally distributed over 10 classes, with 50K training and 10K testing images. CIFAR100 has the same images' configuration as CIFAR10 except for containing more object categories (*i.e.,* 100 classes). FashionMNIST consists of a training set of 60K images and a test set of 10K images. Each image is a 28×28 grayscale image, associated with a label from 10 classes. ImageNet contains 1,000 object categories, and 1.3M training images and 50K validation images, all of which are high-resolution and roughly equally distributed over all classes.

Following DARTS [18] as well as conventional architecture search approaches, we use a stage for architecture search and another stage for architecture evaluation. In the architecture search stage, the goal is to determine the best sets of attention coefficients $\{a_{i,j}^k\}$ for each edge. To this end, the original training set is partitioned into two parts, with the first part for optimizing network weights, *i.e.*, (w,Θ), and the second part for architecture selection. In the architecture evaluation stage, we test the performance of obtained neural architectures on the validation dataset by assembling cells to form a complete network.

5.1. **Architecture search.** We construct an over-parameterized super network $\mathscr{G}$ consisting of $L = 15$ cells connected in order, where the cell located at 1/3 and 2/3 of the total network depth is set as reduction cell (with stride = 2), and other cells are normal cells (with stride = 1). These cells are modified by plugging our architecture self-attention modules into each edge. We initialize the number of network channels as 16 and increase the channel number to 32, 64 at specific network depths. The number of nodes N in each cell is 7. The search space $\mathscr{O}$ is same as [14, 18], and the size of space $|\mathscr{O}| = 8$.

We optimize the super network for 30 epochs through the training dataset during the architecture search. The initial learning rate is 0.025 and then annealed down to zero following a cosine schedule. A standard momentum SGD is used to optimize the super network with momentum = 0.9 and weight decay=3×10^{-4}. Then we execute architecture selection based on the trained super network with the network weights (w,Θ) frozen. For each cell in the super network, we execute forward propagation of the network through the validation dataset and simultaneously count the attention coefficients $a_{i,j}$ for each edge (i,j). After traversing the validation dataset, we select the optimal operation and remove others for each edge in the cell. We search for neural architectures on CIFAR10, CIFAR100 and fashionMNIST, respectively. The searched normal and reduction cells on the three datasets are visualized in Appendix A.

5.2. **Architecture evaluation.** To demonstrate the performance of our discovered neural architectures, we construct an evaluation network by these cells searched on a training dataset. The evaluation network consists of 15 cells, where the 5th, 10th are reduction cells. We train the evaluation network with batch size 64, initial learning rate 0.025, momentum 0.9, drop probability 0.2, weight decay 3×10^{-4} and an auxiliary tower of weight 0.4. The standard image preprocessing techniques (including randomly cropping, horizontally flipping, and normalization) are applied to the input images. We train the evaluation network from scratch for 600 epochs and test the accuracy of this network. The same evaluation process is executed separately on the different neural architectures discovered on the three datasets.

Under the fair conditions, we compare the performance and search costs of our method with other NAS methods. The comparison results on the three small scale datasets are displayed in the Table. 1, 2, 3, respectively. In merely 0.55 GPU-days, our method achieves an error rate of 2.49% on CIFAR10, 15.60% on CIFAR100 and 3.70% on fashionMNIST, with both search time and accuracy surpassing the baseline, DARTS.

To verify the generalization of the architectures, we transfer the cells discovered on CIFAR10 to a large scale dataset ImageNet. The evaluation network consisting of 15 cells is trained for 250 epochs with initial channels=54, initial SGD learning rate=0.15 (annealed down to zero following a linear schedule), momentum=0.9, weight decay=3×10^{5}. The testing and comparison results are summarized in the Table. 4. The result proved that our searched architectures excellent performance and generalization.

5.3. **Diagnostic experiments.**

Architecture	**Test Error(%)**	**Params (M)**	**GPU days**	**Search Method**
ResNet [4]	4.61	1.7	-	Manual
DenseNet [5]	3.46	25.6	-	Manual
SENet [6]	4.05	11.2	-	Manual
NASNet-A [14]	3.41	3.3	1800	RL
NASNet-A + cutout [14]	2.65	3.3	1800	RL
AmoebaNet-A + cutout [15]	3.12	3.1	3150	Evolution
PNAS [24]	3.41	3.2	225	SMBO
ENAS [17]	3.54	4.6	0.5	RL
ENAS+cutout [17]	2.89	4.6	0.5	RL
DARTS(2nd order) + cutout[18]	2.82	3.4	1.6	Gradient
PDARTS + cutout [26]	2.50	3.4	0.3	Gradient
PC-DARTS + cutout [27]	2.57	3.6	0.1	Gradient
Random Sample [18]	3.49	3.1	-	-
GDAS + cutout [25]	3.87	3.4	0.21	Gradient
GDAS +cutout [25]	2.93	3.4	0.21	Gradient
ASM-NAS + cutout	2.59	3.1	0.55	Gradient
ASM-NAS + cutout (large settings)	2.49	4.0	0.55	Gradient

TABLE 1. Test classification error rates for ASM-NAS, human-designed networks and other NAS architectures on CIFAR10.

Architecture	**Test Error(%)**	**Params (M)**	**GPU days**	**Search Method**
ResNet-101 [4]	22.22	25.3	-	Manual
DenseNet-161 [5]	21.56	26.0	-	Manual
SENet-50 [6]	21.42	26.5	-	Manual
NASNet-A + cutout [14]	18.34	3.3	1800	RL
AmoebaNet-A + cutout[15]	18.38	3.1	3150	Evolution
PNAS [24]	19.53	3.2	225	SMBO
ENAS+cutout [17]	17.92	3.4	0.5	RL
DARTS(2nd order) + cutout[18]	17.54	3.4	1.6	Gradient
PDARTS + cutout[26]	16.55	3.4	0.3	Gradient
PC-DARTS + cutout [27]	17.62	3.4	0.1	Gradient
Random Sample [18]	20.70	3.1	-	-
GDAS + cutout [25]	18.38	3.4	0.21	Gradient
ASM-NAS + cutout	15.60	3.1	0.55	Gradient

TABLE 2. Test classification error rates for ASM-NAS, human-designed networks and other NAS architectures on CIFAR100.

5.3.1. *Performance stability research.* In this subsection, we investigate the performance stability of our method and DARTS by experiments. Specially, we estimate the impact of a single operation selected by our method or DARTS on the network. The experiment is performed on a super network $\mathcal{G}$ that has been trained ever during architecture search. As mentioned before, there are 14 edges in a cell and 8 operations on each edge. Then, we replace a mixed operations $\overline{o}_{i,j}$ of edge (i, j) with an optimal operation $o^k_{i,j}$ recognized by our method or DARTS, and then test the accuracy of the modified network. We operate on these 14 edges in this cell

Architecture	**Test Error(%)**	**Params (M)**	**GPU days**	**Search Method**
ResNet-18 [4]	5.10	11.1	-	Manual
DenseNet [5]	4.61	25.6	-	Manual
NASNet-A + cutout [14]	3.66	2.5	1800	RL
AmoebaNet-A + cutout [15]	3.67	2.3	3150	Evolution
PNAS [24]	3.89	2.5	225	SMBO
ENAS+cutout [17]	3.79	2.6	0.5	RL
DARTS(2nd order) + cutout [18]	3.77	2.2	1.6	Gradient
Random Sample [18]	3.95	2.5	-	-
GDAS + cutout [25]	3.76	2.4	0.21	Gradient
ASM-NAS + cutout	3.70	2.6	0.45	Gradient

TABLE 3. Test classification error rates for ASM-NAS, human-designed networks and other NAS architectures on fashionMNIST.

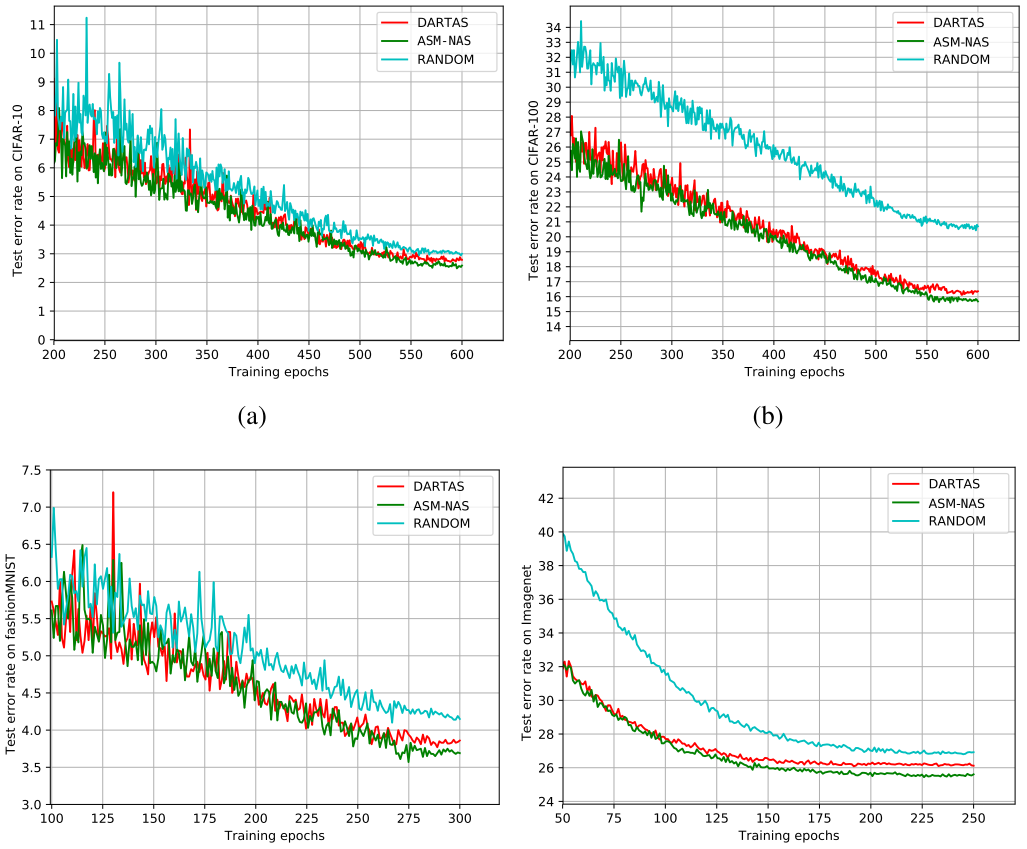
Architecture	**Accuracy (%)**		**Params**	**GPU**	**Search**
	Top1	**Top5**	**(M)**	**days**	**Method**
Inception-V1 [3]	69.8	89.9	6.6	-	Manual
MobileNetV2 [20]	72.0	91.0	3.4	-	Manual
ShuffleNetV2 [36]	73.7	91.5	5.0	-	Manual
NASNet-A [14]	74.0	91.6	5.3	1800	RL
NASNet-B [14]	72.8	91.3	5.3	1800	RL
NASNet-C [14]	72.5	91.0	4.9	1800	RL
AmoebaNet-A [15]	74.5	92.0	5.1	3150	Evolution
AmoebaNet-B [15]	74.0	91.5	5.3	3150	Evolution
PNAS [24]	74.2	91.9	5.1	225	SMBO
DARTS [18]	73.1	91.0	4.9	1.6	Gradient
PDARTS [26]	75.3	92.3	5.1	0.3	Gradient
PC-DARTS [27]	74.9	92.2	5.3	0.1	Gradient
SNAS [19]	72.7	90.8	4.3	1.5	Gradient
GDAS [25]	74.0	91.5	5.3	0.21	Gradient
ASM-NAS (Ours)	74.6	91.9	5.5	0.55	Gradient

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

separately and keep the modified network the same with $\mathscr{G}$ except for one edge each time. The experimental results are displayed in Figures 3(a) and 3(b).

The red line in the figure means the accuracy baseline of the original super network and the green one means the accuracy of the modified super network. As you can see, The change in accuracy induced by an operation is smaller by our method, which means our method is more stable and effective. That also implies our method can focus on the important operations by considering the interrelationships of operations. Thereby we can enhance the impact of these important operations on network performance and weaken other operations. In contrast, DARTS does not consider the interrelationship of operations, resulting in selected architecture unstable.

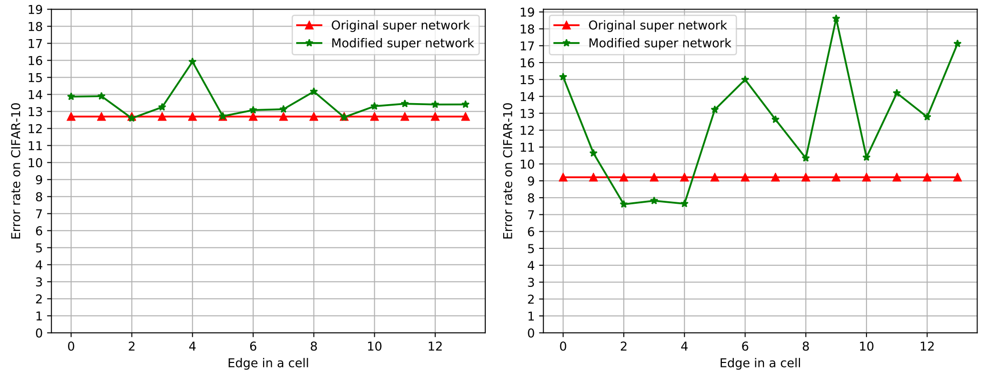
5.3.2. *Efficiency of architecture search.* Our algorithm shows high efficiency in architecture search. We experimentally explored and compared the difference between our method and



(a) (b)

(c) (d)

FIGURE 2. Accuracy comparison of 3 search methods on four datasets ((a)CIFAR10, (b)CIFAR100, (C)fashionMNIST, (d)ImageNet).



(a) ASM-NAS (b) DARTS

FIGURE 3. The performance stability research. Modify the cell in super network by replacing an edge with the optimal operation and test the accuracy of the modified and original super network. Note, the x-axis indicates the edge index in a cell.

DARTS during the architecture search. As mentioned before, DARTS resolves the architecture search with a bi-level optimization, *i.e.,* one level for weight parameters and the other for architecture parameters. In contrast, our method performs only single-level optimization. We construct the super network of DARTS and our ASM-NAS with 15 cells, respectively. An image is fed into the super network, and then we count the time of once optimization of two methods. The results are summarized in Table. 5.

Method	**Weight param optimization**(s)	**Architecture param optimization**(s)	**total** (s)
DARTS	1.41	9.96	11.37
ASM-NAS	1.43	-	1.43

TABLE 5. The cost of once optimization for DARTS and ASM-NAS.

From the result, we can find that our method takes about 1.43s to perform once optimization, which is much faster than DARTS (taking about 11.37s). Especially, DARTS takes the most time to optimize the architecture parameters. But our method avoids this inefficient optimization. Moreover, the bi-level optimization of DARTS needs more training iterations to approach the optimal point of the network.

5.3.3. *Learning adaptive neural architectures.* Our ASM-NAS could learn adaptive neural architectures for different network layers by the architecture self-attention module. The self-attention module located in the different layers is capable of mining the characteristics of this network layer. To verify the above view, we visualize the attention coefficients $a(i,j)$ in each cell that has been trained during architecture search. The details are illustrated with heatmaps in Figure. 4. The subfigures (a)∼(o) are corresponding to the cells 1∼15, where (e), (j) are reduction cells and others are normal cells. We can get some interesting observations from the heatmaps. The cells in the shallow layers prefer the separable convolution operations with size 3×3, such as subfigure 4(a), 4(b), 4(c). These operations could extract coarse-grained information or features of inputs. The cells in the middle and deeper layers prefer the convolution operations with lager size 5×5, such as subfigure 4(f)∼4(l). The convolution operations with a large receptive field could represent complex hierarchical information in the network. The cells in the deep layers prefer the pooling operations (such as, 4(n), 4(o)), which fusions of information from the upper layers and reduces the size of feature maps. Therefore, the neural architectures discovered by our method conform to the general network characteristics.

6. CONCLUSION

In this paper, we propose architecture self-attention mechanism, a nonlinear optimization for neural architecture search. Our method ASM-NAS captures the interrelationships among candidate architectures and embeds the information into the attention coefficients, which provides an important reference for architecture selection. Therefore, we address network performance unstable due to lacking the interrelationship of candidate architectures. Besides, ASM-NAS

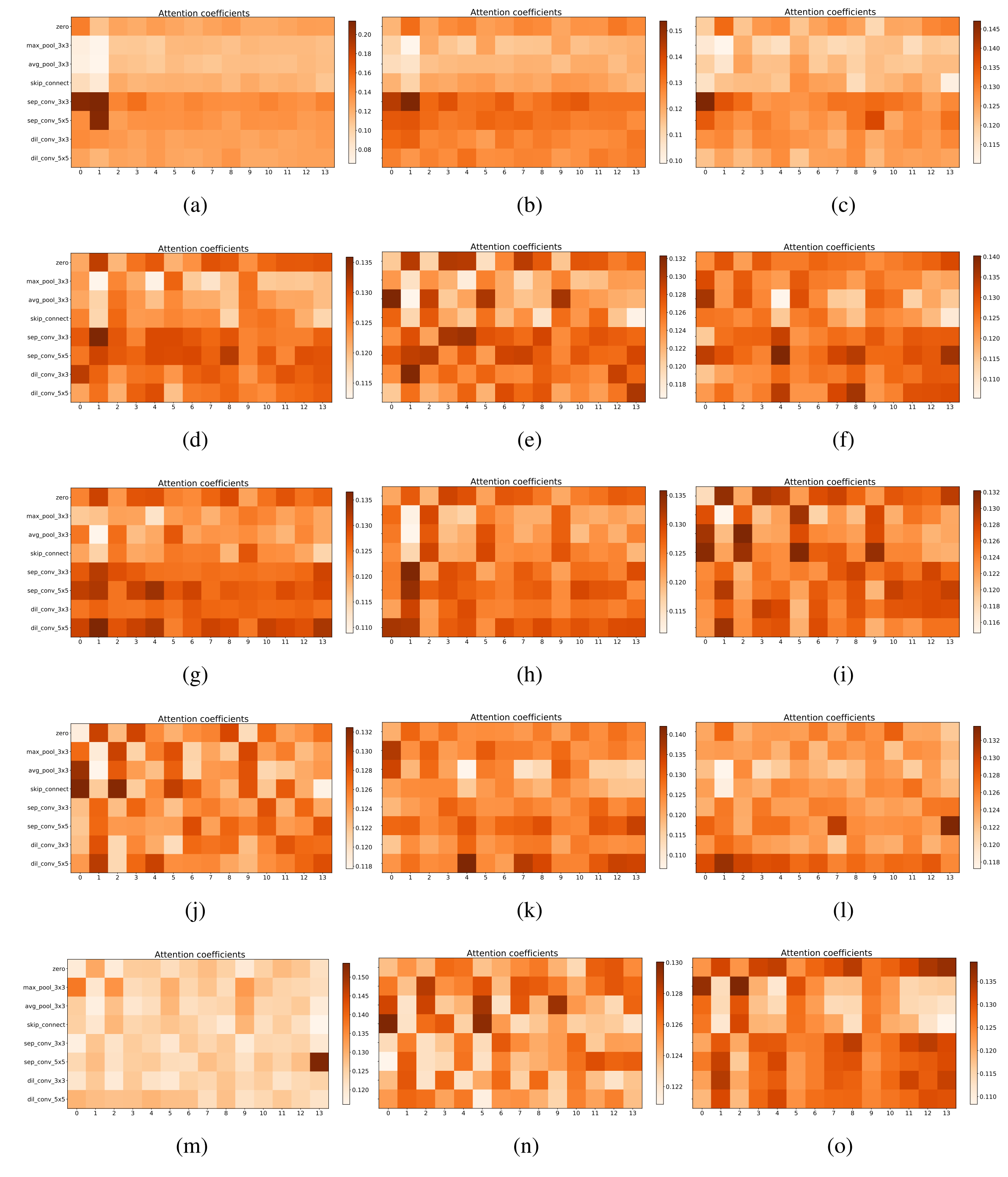


FIGURE 4. The attention coefficients of our searched neural architecture.

could efficiently execute architecture since reducing the optimization complexity. The extensive experiments on the image classification tasks demonstrate our algorithm can achieve better performance compared to other NAS methods while at low computational costs.

Acknowledgements

The authors are grateful to the reviewers for useful suggestions which improved the contents of this paper. The work was supported by the National Natural Science Foundation of China (61772120).

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, In: Advances in Neural Information Processing Systems, pp. 1097-1105, 2012.

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

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1-9, 2015.

[4] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778, 2016.

[5] G. Huang, Z. Liu, L.V. Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, In; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4700-4708, 2017.

[6] J. Hu, L. Shen, G. Sun, Squeeze-and-excitation networks, In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7132-7141, 2018.

[7] I. Guyon, S. Gunn, M. Nikravesh, L.A. Zadeh, Feature Extraction: Foundations and Applications, vol. 207, Springer, 2008.

[8] Z. Cai, W. Zhu, Multi-label feature selection via feature manifold learning and sparsity regularization, Int. J. Machine Learn. Cybernetics, 9 (2018), 1321-1334.

[9] M. Nixon, A. Aguado, Feature Extraction and Image Processing for Computer Vision, Academic press, 2019.

[10] M. Bramer, Principles of Data Mining, vol. 180, Springer, 2007.

[11] W. Zhu, Relationship between generalized rough sets based on binary relation and covering, Info. Sci. 179 (2009), 210-225.

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

[13] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, arXiv preprint arXiv:1611.01578, 2016.

[14] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8697-8710, 2018.

[15] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 4780-4789, 2019.

[16] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu, Hierarchical representations for efficient architecture search, arXiv preprint arXiv:1711.00436, 2017.

[17] H. Pham, M.Y. Guan, B. Zoph, Q.V. Le, J. Dean, Efficient neural architecture search via parameter sharing, arXiv preprint arXiv:1802.03268, 2018.

[18] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, arXiv preprint arXiv:1806.09055, 2018.

[19] S. Xie, H. Zheng, C. Liu, L. Lin, Snas: stochastic neural architecture search, arXiv preprint arXiv:1812.09926, 2018.

[20] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861, 2017.

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

[22] E. Real, S. Moore, A. Selle, S. Saxena, Y.L. Suematsu, J. Tan, Q.V. Le, A. Kurakin, Large-scale evolution of image classifiers, In: Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 2902-2911, 2017.

[23] T. Elsken, J.H. Metzen, F. Hutter, Efficient multi-objective neural architecture search via lamarckian evolution, arXiv preprint arXiv:1804.09081, 2018.

[24] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.J. Li, L.L. Fei, A. Yuille, J. Huang, K. Murphy, Progressive neural architecture search, In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 19-34, 2018.

[25] X. Dong, Y. Yang, 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, 2019.

[26] X. Chen, L. Xie, J. Wu, Q. Tian, Progressive differentiable architecture search: Bridging the depth gap between search and evaluation, In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1294-1303, 2019.

[27] Y. Xu, L. Xie, X. Zhang, X. Chen, G.J. Qi, Q. Tian, H. Xiong, Pc-darts: Partial channel connections for memory-efficient differentiable architecture search, arXiv preprint arXiv:1907.05737, 2019.

[28] L. Itti, C. Koch, E. Niebur, A model of saliency-based visual attention for rapid scene analysis, IEEE Trans. Pattern Anal. Machine Intell. 20 (1998), 1254-12598.

[29] M. Corbetta, G.L. Shulman, Control of goal-directed and stimulus-driven attention in the brain, Nature Rev. Neurosci. 3 (2002), 201-215.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Info. Processing Sys. 30 (2017), 5998-6008.

[31] S. Woo, J. Park, J.Y. Lee, I.S. Kweon, Cbam: Convolutional block attention module, In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 3-19, 2018.

[32] V. Nair, G.E. Hinton, Rectified linear units improve restricted boltzmann machines, In: Proceedings of the 27th International Conference on Machine Learning, pp. 807-814, Haifa, 2010.

[33] L. Bttou, F.E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, SIAM Rev. 60 (2018), 223-311.

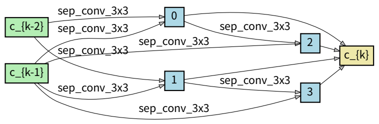
[34] A. Krizhevsky, Learning multiple layers of features from tiny images, Technical Report TR-2009, University of Toronto, Toronto, 2009.

[35] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, F. Li, Imagenet: A large-scale hierarchical image database, In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248-255, 2009.

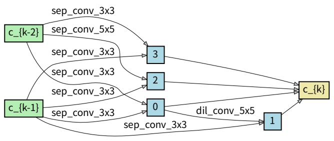
[36] X. Zhang, X. Zhou, M. Lin, J. Sun, Shufflenet: An extremely efficient convolutional neural network for mobile devices, In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6848-6856, 2018.

APPENDIX A.

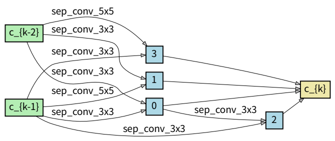
We display the cell architectures that discovered on CIFAR10 5, CIFAR100 6, and fashion-MNIST 5, respectively. 15 optimal cell architectures are searched on three datasets, and are denoted in the subfigure (a)∼(o). We connect the 15 cells from end to end to construct an evaluation network to test its performance as described in Section 5.2.



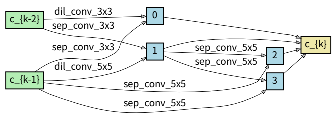
(a)



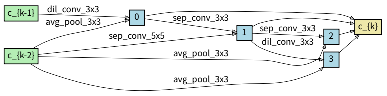
(b)



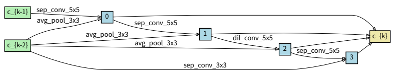
(c)



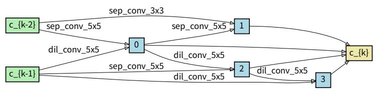
(d)



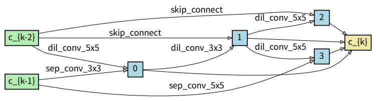
(e)



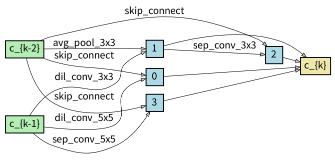
(f)



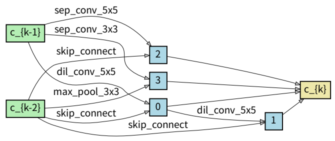
(g)



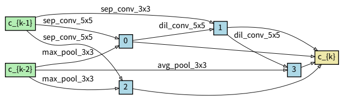
(h)



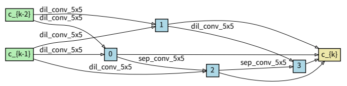
(i)



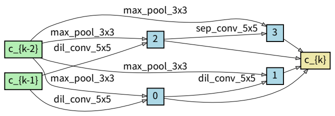
(j)



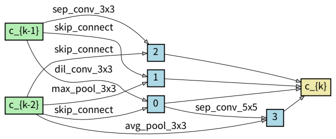
(k)



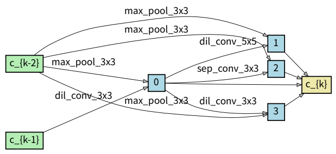
(l)



(m)



(n)



(o)

FIGURE 5. The cell architectures discovered on CIFAR10.

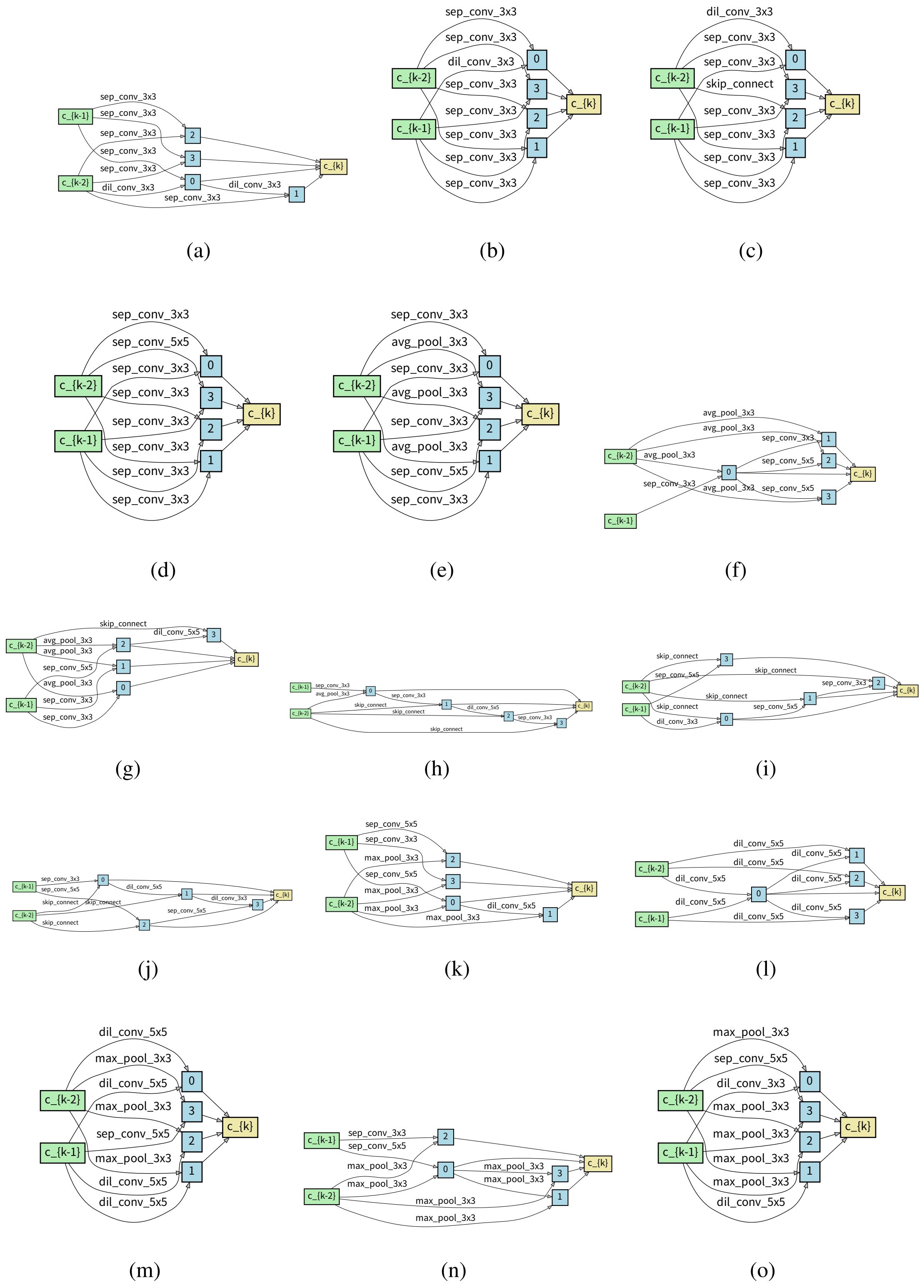


FIGURE 6. The cell architectures discovered on CIFAR100.

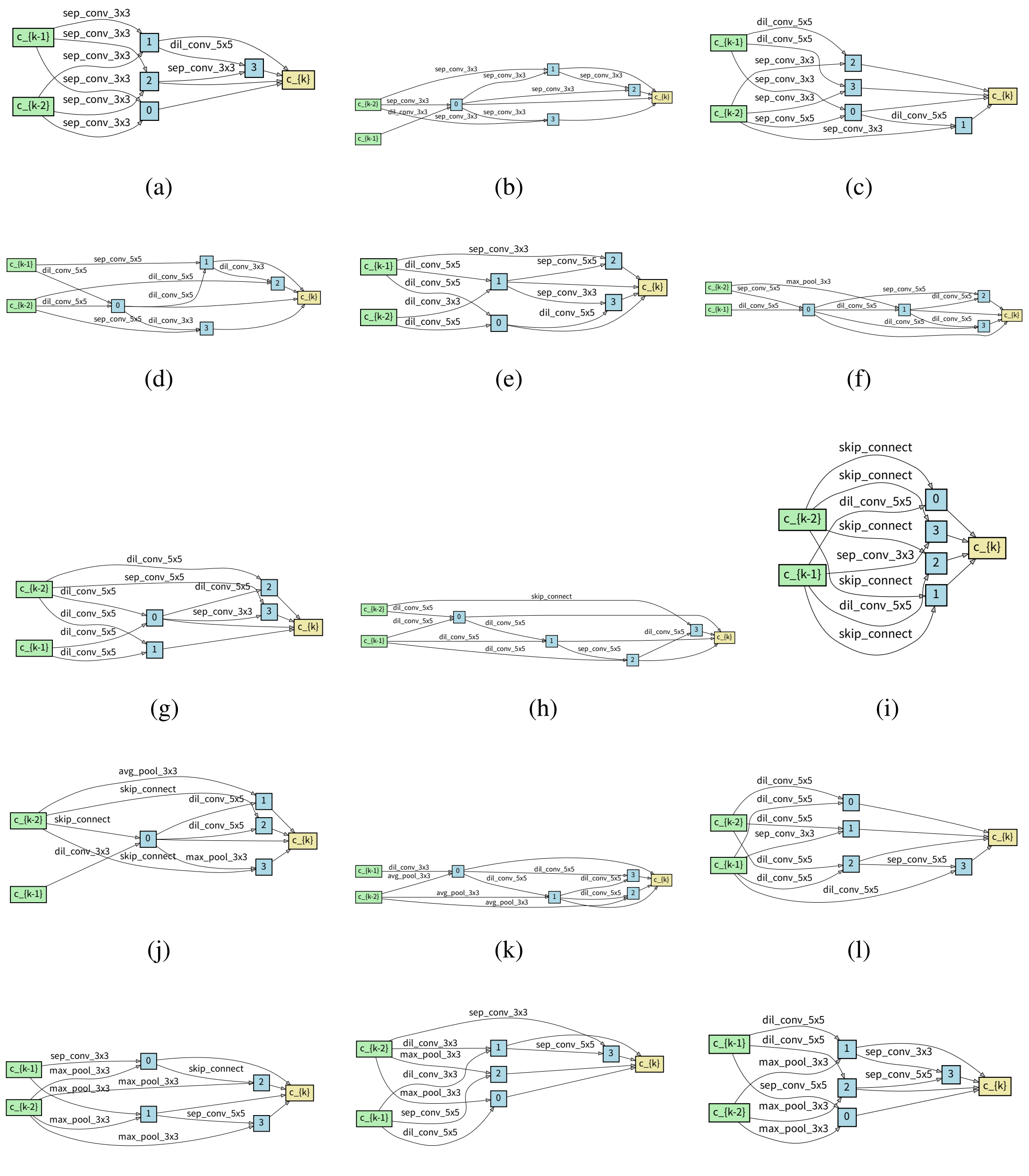


FIGURE 7. The cell architectures discovered on fashionMNIST.

APPENDIX B. PROOFS

B.1. **Inequality 4.12.**

$$\begin{aligned} F(\omega) &= F(\overline{\omega}) + \int_0^1 \frac{\partial F(\overline{\omega}+t(\omega-\overline{\omega}))}{\partial t} dt \\ &= F(\overline{\omega}) + \int_0^1 \nabla F(\overline{\omega}+t(\omega-\overline{\omega}))^T(\omega-\overline{\omega})dt \\ &= F(\overline{\omega}) + \nabla F(\overline{\omega})^T(\omega-\overline{\omega}) + \int_0^1 [\nabla F(\overline{\omega}+t(\omega-\overline{\omega})) - \nabla F(\overline{\omega})]^T(\omega-\overline{\omega})dt \\ &\leq F(\overline{\omega}) + \nabla F(\overline{\omega})^T(\omega-\overline{\omega}) + \int_0^1 L\|t(\omega-\overline{\omega})\|_2\|\omega-\overline{\omega}\|_2 dt \\ &= F(\overline{\omega}) + \nabla F(\overline{\omega})^T(\omega-\overline{\omega}) + \frac{1}{2}L\|\omega-\overline{\omega}\|_2^2. \end{aligned} \tag{B.1}$$

B.2. **Inequality 4.13.**

From Assumption 4.1, we can get that,

$$\begin{aligned} F(\omega_{k+1}) - F(\omega_k) &\leq \nabla F(\omega_k)^T(\omega_{k+1}-\omega_k) + \frac{1}{2}L\|\omega_{k+1}-\omega_k\|_2^2 \\ &\leq -\eta \nabla F(\omega_k)^T g(\omega_k,\xi_k) + \frac{1}{2}\eta^2 L\|g(\omega_k,\xi_k)\|_2^2. \end{aligned} \tag{B.2}$$

Taking expecitations in these inequalities *w.r.t,* the distribution of ξ_k, and noting that ω_{k+1}, but not ω_k, depends on ξ_k, we obtain the desired result.

B.3. **Inequality 4.18b.**

From Lemma 4.1 and inequality (4.15a), we have

$$\begin{aligned} \mathbb{E}_{\xi_k}[F(\omega_{k+1})] - F(\omega_k) &\leq -\eta \nabla F(\omega_k)^T \mathbb{E}_{\xi_k}[g(\omega_k,\xi_k)] + \frac{1}{2}\eta^2 L \mathbb{E}_{\xi_k}[\|g(\omega_k,\xi_k)\|_2^2] \\ &\leq -\mu\eta\|\nabla F(\omega_k)\|_2^2 + \frac{1}{2}\eta^2 L \mathbb{E}_{\xi_k}[\|g(\omega_k,\xi_k)\|_2^2]. \end{aligned} \tag{B.3}$$

Considering Assumption 4.2 and (4.17), we obtain (4.18b).