Electronic Research Archive

http://www.aimspress.com/journal/era

ERA, 32(1): 263–292.
DOI: 10.3934/era.2024013
Received: 18 October 2023
Revised: 26 November 2023
Accepted: 12 December 2023
Published: 22 December 2023

Research article

Bridge the gap between fixed-length and variable-length evolutionary neural architecture search algorithms

Yunhong Gong[1], Yanan Sun[1], Dezhong Peng[1,2,*] and Xiangru Chen[1]

[1] College of Computer Science, Sichuan University, Chengdu 610065, China

[2] National Innovation Center for UHD Video Technology, Chengdu 610095, China

* **Correspondence:** Email: pengdz@scu.edu.cn.

Abstract: Evolutionary neural architecture search (ENAS) aims to automate the architecture design of deep neural networks (DNNs). In recent years, various ENAS algorithms have been proposed, and their effectiveness has been demonstrated. In practice, most ENAS methods based on genetic algorithms (GAs) use fixed-length encoding strategies because the generated chromosomes can be directly processed by the standard genetic operators (especially the crossover operator). However, the performance of existing ENAS methods with fixed-length encoding strategies can also be improved because the optimal depth is regarded as a known priori. Although variable-length encoding strategies may alleviate this issue, the standard genetic operators are replaced by the developed operators. In this paper, we proposed a framework to bridge this gap and to improve the performance of existing ENAS methods based on GAs. First, the fixed-length chromosomes were transformed into variable-length chromosomes with the encoding rules of the original ENAS methods. Second, an encoder was proposed to encode variable-length chromosomes into fixed-length representations that can be efficiently processed by standard genetic operators. Third, a decoder cotrained with the encoder was adopted to decode those processed high-dimensional representations which cannot directly describe architectures into original chromosomal forms. Overall, the performances of existing ENAS methods with fixed-length encoding strategies and variable-length encoding strategies have both improved by the proposed framework, and the effectiveness of the framework was justified through experimental results. Moreover, ablation experiments were performed and the results showed that the proposed framework does not negatively affect the original ENAS methods.

Keywords: neural architecture search; evolutionary algorithm; variable-length encoding; fixed-length encoding; autoencoder

1. Introduction

Deep neural networks (DNNs) are the state of the art methods for computer vision [1–3], natural language processing [4–6] and so on. Generally, the performance of neural architectures heavily depends on their skeletons and the associated hyperparameters. In terms of skeletons, outstanding architectures such as Visual Geometry Group (VGG) [7], Residual Network (ResNet) [8] and Densely Network (DenseNet) [9] are all carefully designed by researchers with rich expertise. In terms of hyperparameters, it is a challenging and time-consuming process for experts to manually tune those hyperparameters according to the empirical values gained from extensive experimentations [10].

Evolutionary Algorithm (EA) based neural architecture search (ENAS) methods have been frequently used, searching for not only the optimal neural architectures but also the weights of neural networks simultaneously. However, most ENAS methods utilize fixed-length encoding strategies to limit the encoding space so that the search complexity can be heavily reduced. Moreover, standard genetic operators (especially the crossover operator) have achieved huge benefits in fixed-length strategies because the fixed-length chromosomes can be easily processed by standard genetic operators. However, the depth of searching structures remains fixed because of the fixed-length encoding strategies. For example, Gradient-based Evolution Algorithm (GEA) [11] focuses on 10-layer convolutional neural networks (CNNs) and uses a fixed-length strategy to encode the parameters of each layer into fixed-length chromosomes with 9 convolutional units and a max-pooling unit. Genetic network (GeNet) [12] proposes a 3-stage network with 3, 4, and 5 nodes in each stage, and uses fixed-length encoding strategies to encode gene information into a fixed-length chromosome. As a result, the number of search blocks, which are parts of the whole skeleton, is also fixed. In a multiobjective EA method called NASTSGA-NET [13], the target architecture is constructed by stacking 5 nodes containing 2 predefined operators, and the depth of searching structures stays unchanged during evolution due to the number of nodes being predefined. However, the optimal depths of neural networks are not known at the beginning, and the depth heavily affects the performance of different tasks on different datasets. As the depths of networks increase, the ability to extract and process data would be enhanced, which helps models achieve better performance [14]. However, this benefit cannot last and reaches the upper limit quickly. ResNet [8] concluded that the deeper network is, the higher the training and test errors. A similar observation is also found in graphic neural networks (GNNs). For GNNs, greater depth enables the capability to make use of longer-range interactions, which can be beneficial in many scenarios [15]. However, GNNs tend to suffer from performance degradation as they become deeper [16–18]. As a result, the optimal depth of neural networks should be taken as an unknown priori, and the length of chromosomes should be altered by generation.

To solve this issue, the variable-length encoding strategies are studied in optimization problems. Ma et al. [19] design a variable-length coding structure to flexibly represent the solutions of variable multi-objective optimization problems. Muwafaq et al. [20] propose a novel variable-length multi-objective whale optimization to cover the variable number of cloudlets for deployment. Variable-length chromosomes (VLCs) are obtained by combining simple well-tested short chromosomes to form more complex chromosomes that cover all the features of the problem under consideration [21]. Relying on the VLC-based genetic algorithm (GA) solutions to applications [22–25], the ENAS community has designed variable-length encoding strategies to

determine the optimal depth of neural networks during the search. The EvoCNN [26] encodes the model architectures into variable-length chromosomes by a flexible gene encoding strategy. In recurrent neural network (RNN) searching tasks, variable-length encoding strategies are often adopted. VLC_GA [27] dynamically determined the number of layers and their connections during evolution to find the optimal structure. In heterogeneous evolution [28], the variable node topologies are searched, and the nodes of the topology for searching are varied between 6 and 15. AutoGraph [29], a GNN searching task, automatically generates variable-length and deeper architectures during evolution. Compared with those of the fixed-length encoding strategies, the variable-length strategies achieve better results during search. However, the standard genetic operators that are important to GAs are hardly used to generate balanced child chromosomes. The shallow neural networks represented by short chromosomes lose the ability to extract meaningful features, and the very deep neural networks represented by long chromosomes may achieve higher training and test errors [8]. As a result, those ENAS methods with variable-length encoding strategies all developed the standard genetic operator with complex designs. An evolutionary CNN search method called EvoCNN proposed the unit alignment to collect three blocks into three lists based on orders, then the crossover operator is performed to exchange the units at the same positions. VLC_GA and heterogeneous evolution solve the imbalanced child chromosomes by inserting a new fragment of genes at a random position or shrinking the long chromosomes by choosing a fragment randomly. An ENAS method used in the covid-19 classification task called EAVL-COVID [30] also changed the length of chromosomes to alter CNN hyperparameters encoding space based on the proposed growth operator and shrink operator. AutoGraph [29] removed the traditional crossover operator and developed the mutation operator, which adds a new layer to the GNN model during search to generate the variable-length and deeper architectures automatically. Even though those developed operators alleviate the imbalance caused by standard genetic operators, some situations are not entirely resolved, or the developed operators may introduce new problems. Although EvoCNN achieves notable advancements with variable-length encoding strategies, the length of chromosomes still remains fixed during evolution. In addition, the convolutional, pooling, and fully connected blocks must be handled due to the unit alignment. For balance operations proposed by VLC_GA, heterogeneous evolution, and EAVL-COVID, randomness is introduced and increases the difficulty of fitting [30].

In this paper, we aim to simultaneously improve the performance of existing ENAS methods based on GAs by exploiting both fixed-length encoding strategies and variable-length encoding strategies. To achieve this goal, the objectives are specified below.

1) A framework is proposed to transform the fixed-length encoding strategies into variable-length encoding strategies with the encoding rules of the original ENAS methods. Usually, the length of chromosomes determines the depth of the search structures, and the framework improved those existing ENAS methods with fixed-length encoding strategies by taking advantage of the variable-length strategies.

2) To achieve the benefit of standard genetic operators, an encoder is proposed to encode generated variable-length chromosomes into a fixed-length representation. Those high-dimensional representations can be efficiently processed by the standard operators, especially the crossover operator.

3) As the high-dimensional representations cannot be directly adopted by ENAS methods, a decoder is designed to recover those processed representations to their original chromosomal forms. Notably, the encoder and the decoder are cotrained in an unsupervised way with the sequence-to-sequence (seq2seq) autoencoder.

4) From the experimental results on three common NAS searching tasks, CNN, RNN, and GNN, as well as the comparison with state of the art ENAS methods with fixed-length and variable-length encoding strategies, we demonstrate that the proposed framework indeed improves the performance of the original ENAS methods by simultaneously adopting variable-length encoding strategies and standard genetic operators.

2. Related works

The related works are presented in three subsections to help the readers understand the workflow of the ENAS (Subsection 2.1), the various encoding spaces that are transformed into variable-length encoding strategies (Subsection 2.2) and the seq2seq autoencoder, which is used to obtain the encoder and the decoder by unsupervised cotraining (Subsection 2.3).

2.1. The workflow of ENAS

As a significant branch of artificial intelligence, neural architecture search (NAS) technologies are studied to automatically achieve the best candidate architecture on a specific dataset. The general NAS procedure can be categorized into the encoding space, the searching strategy, and the evaluation [31]. The encoding space defines which architectures can be represented in principle. The evaluation measures the performance of the candidate architecture. The searching strategy, which is the key to NAS, details how to efficiently explore the encoding space. There are many different searching strategies that effectively and comprehensively explore the encoding space with an exploration-exploitation trade-off, such as reinforcement learning [32–34], Bayesian optimization [35–37] and EAs [20, 38, 39]. EAs are natural heuristic algorithms that help to find the exact or approximate global optimal solution efficiently with a stochastic approach [40], and they have unique advantages over other searching strategies. Compared with that of reinforcement learning, EAs have a faster search speed, especially in the early stages [41]. Because the Bayesian optimization strategy relies cubically on the number of observations [42], that strategy explores the large-scale searching space with more expensive evaluations than those of the EAs.

The ENAS methods find the optimal combination of hyperparameters through an evolutionary simulation process. To explore the searching space and optimize the combination of hyperparameters, the crossover and mutation are important operations in ENAS methods. Crossover generates new individuals by randomly selecting and exchanging partial genes from two chromosomes to simulate the hybridization process during biological reproduction. The crossover allows the superior genes to be passed on to the next generation, and helps searching strategies explore diverse searching space. The mutation randomly alters specific genes to generate new chromosomes during biological reproduction. The significance of mutation lies in preventing being trapped in local optimal solutions. In summary, crossover increases search diversity, and mutation helps escape local optima.

2.2. Encoding space

The design of the encoding space forms a key component of neural architecture search because the parameters to build the target architecture are searched in the encoding space. To encode the predefined encoding space into genetic chromosomes, the encoding space can be divided into three categories according to the basic units they adopt. These categories are the chain-based, block-based, and cell-based encoding spaces [43]. Figure 1(a) shows a chain-based encoding space, and the "gray" parts in the figure participate in evolution. The optimizing parameters in each layer are encoded into a gene, and the genes are stacked following the order of layers to form chromosomes. As a result, the length of chromosomes represents the depth of the architecture. The block-based encoding space is divided into macro space and micro space. In most ENAS methods, the macro space is defined by experts, and the micro space is the target to be optimized. Figure 1(b) demonstrates a skeleton of the block-based encoding spaces. The block-based neural network is stacked with 3 convolutional layers, a pooling layer and 2 blocks, which are the search targets. Additionally, only the "gray" parts in Figure 1(b) participate in evolution, which means that the parameters that form the search structures in every block are encoded into a gene and the length of chromosomes is equal to the number of blocks. The cell-based encoding space is a particular case of the block-based space where there is only one topological relationship in a cell, and the searched topology of the neural network is stacked m times. Unlike the block-based encoding space, the topological nodes are encoded into genes to form chromosomes. Figure 1(c) shows a cell-based encoding space, and there are 5 nodes in a cell. The length of the chromosomes is 5, which is equal to the number of searching nodes.

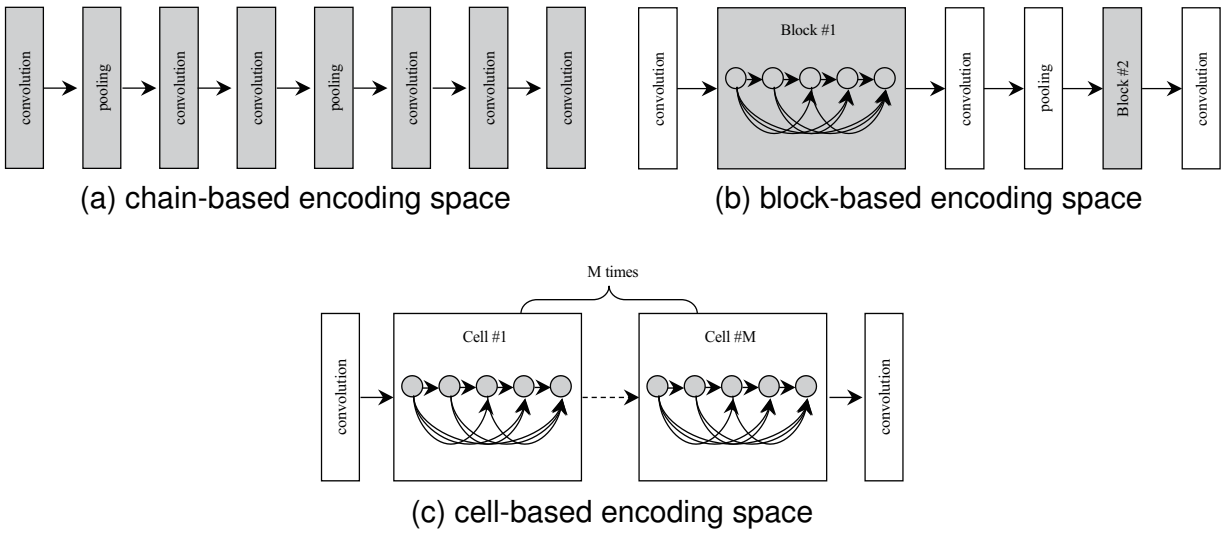

Figure 1. Examples of different encoding spaces in ENAS.

2.3. Representation learning with autoencoder

The autoencoder is an unsupervised neural network that contains an encoder and a decoder, which can learn useful representations with little or no supervision. Autoencoders have been successfully applied to dimensionality reduction [44–46], noise reduction [47–49], data generation [50–52], and

data representation [53–55]. The encoder can learn a mapping function that encodes observations to representations, and the decoder reconstructs the original observations through the representations.

The principle of the autoencoder is that it is given the input space $\boldsymbol{X} \in \mathcal{X}$ and the feature space $\boldsymbol{H} \in \mathcal{F}$. The encoder f encodes $\boldsymbol{X}$ in input space $\mathcal{X}$ to $\boldsymbol{H}$ in feature space $\mathcal{F}$, and decoder g recovers $\boldsymbol{H}$ in feature space $\mathcal{F}$ to $\boldsymbol{X}$ in input space $\mathcal{X}$. The autoencoder finds the mapping between f and decoder g to minimize the reconstruction error of the inputs:

$$\begin{cases} f : \mathcal{X} \to \mathcal{F} \\ g : \mathcal{F} \to \mathcal{X} \\ f, g = \arg\min\limits_{f,g} \|\boldsymbol{X} - g[f(\boldsymbol{X})]\|^2. \end{cases} \tag{2.1}$$

With the ability to effectively model variable-length sequence data and to produce fixed-length embeddings, the seq2seq autoencoder has achieved state of the art results in numerous applications. In [56], a general end-to-end multilayered long short-term memory (LSTM) based autoencoder is proposed to map variable-length sequences into fixed-length representations and translate those representations into the target sequences. Compared with the standard system with unlimited vocabulary, the deep LSTM-based autoencoder performs better with a limited vocabulary. In [57], a method is proposed to learn fixed-length vector representations of variable-length phonetic audio segments. In [58], a seq2seq autoencoder is developed to reduce the variable-length patient phenotypes into low-dimensional embeddings. The reconstructed features from the richer representations are less noisy and more robust to raw data, which successfully helps predict actionable interventions.

3. Proposed algorithm

The goal of this work is to improve the performance of the existing ENAS methods by taking advantage of the fixed-length encoding strategies and the variable-length encoding strategies. The proposed framework with the autoencoder-based evolutionary algorithm (AEEA) scheme is described in Algorithm 1, where our contributions are highlighted in bold and italics.

Before search, the encoder and decoder should be cotrained. To achieve this goal, various variable-length chromosomes are initialized from the encoding space based on the initialization rules of each ENAS method (line 1). Different genes among all chromosomes are assigned a unique numerical ID, and a dictionary is built to link genes and IDs (line 2). The next step is to describe the chromosomes by sequential forms and a special flag called <EOS> whose ID is 0, is added behind every sequence. With this flag, the encoder and decoder can process the variable-length sequences (line 3). Based on the prepared sequences, the encoder and decoder are cotrained (line 4), and the trained encoder and decoder are the keys of the proposed AEEA scheme.

During the search, the variable-length individuals are initialized by the variable-length encoding strategy to construct the population (line 6). The proposed AEEA scheme mainly contains 5 steps (lines 7–12). In the beginning, the generated chromosomes are translated into sequences followed by the flag "0" based on the built dictionary (line 7). The trained encoder is applied to indirectly encode sequences into fixed-length representations (line 8). Because the high-dimensional representations are easily performed by a one-point crossover operator, the redesigned crossover and

mutation operators in the original methods are replaced by the standard genetic operators (lines 9 and 10). All processed representations are decoded into sequences by the decoder (line 11). To recover variable-length chromosomes from sequences, the decoded sequences are translated based on the built dictionary (line 12). Finally, the candidate architectures are selected to build the next generation after proper evaluation (lines 13 and 14).

Algorithm 1: The proposed framework with AEEA scheme

1 ***Initialize various variable-length chromosomes based on the initialization rules***;
2 Build a dictionary for different genes with unique numbers (IDs);
3 Represent variable-length chromosomes by IDs in sequence forms and add a special "<EOS>" token whose ID is 0, behind every sequence;
4 ***Cotrain the encoder and the decoder based on the variable-length sequences***;
5 **while** *terminated criterion is not satisfied* **do**
6 ***Initialize variable-length chromosomes based on the initialization rules of each ENAS method***;
7 Represent variable-length chromosomes by IDs in sequence forms based on the built dictionary;
8 ***Indirectly encode sequences to the fixed-length representations with the trained encoder***;
9 Perform a one-point crossover operator on fixed-length representations;
10 Perform swap mutation operator on fixed-length representations;
11 ***Decode processed representations to sequences by the trained decoder***;
12 Translate those sequences into chromosomes based on the built dictionary;
13 Evaluate the fitness of each of the candidates on chosen datasets.;
14 Select candidate architectures;
15 **end**

3.1. Transform fixed-length encoding strategies into variable-length encoding strategies

The fixed-length encoding strategies encode optimized parameters into a specific length of chromosomes, and the values of each gene in those chromosomes are generated by the encoding strategies. In our work, the length of chromosomes is no longer a constant but a range, and all genes are filled with values following the rules of fixed-length encoding strategies.

For example, a fixed-length ENAS method called EANN [59] finds the optimal combination of eight blocks, and each block is uniquely represented by binary strings with 3 bits. The length of the chromosomes is 19, and each gene in the chromosomes is filled with a 3-bit string. Since the length of chromosomes affects the depth of architectures, the fixed-length encoding strategy can be transformed into the variable-length encoding strategy by altering the length of chromosomes. For example, the length of chromosomes is set to $[14, 23]$, and the value of each gene is generated following the rule of the original EANN. As a result, the variable-length encoding strategy can search $[14, 23]$ blocks that form a DNN.

3.2. Cotrain the encoder and the decoder in an unsupervised way

There are two main purposes of the autoencoder in this work. First, the encoder can indirectly encode the variable-length sequences into fixed-length representations. As a result, the encoder can be regarded as the indirect encoding function; second, the decoder can correctly decode the processed representations to input sequences. Figure 2 describes the workflow of unsupervised cotraining.

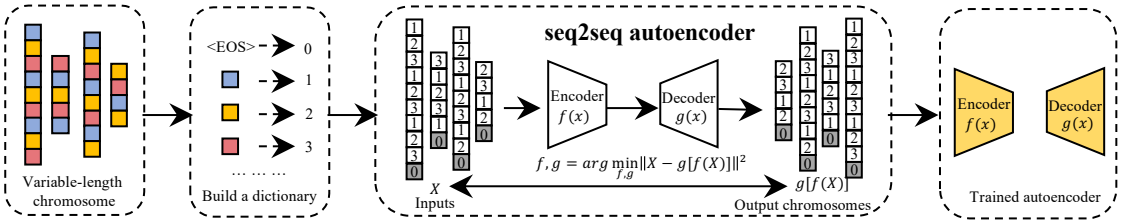

Figure 2. Unsupervised cotrain the encoder and the decoder following the workflow of seq2seq autoencoder.

With the ability to manage the length independency sequences, the Gated Recurrent Unit (GRU) [60], which is a variant of RNNs and has fewer parameters than LSTMs, is introduced to the autoencoder. In this paper, all chromosomes are translated into sequences, which are column vectors denoted by lowercase letters. Matrices are represented by uppercase letters. In GRU, the decision at time $t-1$ affects the decision at time t. The GRU output is founded by iteratively calculating the following two equations:

$$c^{<t>} = \mathcal{H}(W_{c^{<t>}} x^{<t>} + U c^{<t-1>} + b_c) \tag{3.1}$$

$$\bar{x}^{<t>} = W_{\bar{x}^{<t>}} h^{<t>} + b_{\bar{x}^{<t>}}, \tag{3.2}$$

where x is the input, $\bar{x}$ is the output, and c is the hidden vector at time slices $t = 1$ to T. Furthermore, W represents the weights, and b represents the biases. $\mathcal{H}$ is the activation function used in the hidden space.

Unlike LSTMs, which have four gate functions, there are only two gate functions, an updated gate Γ_u and a reset gate Γ_r, in the GRU.

$$\Gamma_u = \sigma(W_u x^{<t>} + U_u h^{<t-1>} + b_u) \tag{3.3}$$

$$\Gamma_r = \sigma(W_r x^{<t>} + U_r h^{<t-1>} + b_r) \tag{3.4}$$

where W, U, b are coefficients specific to the gate and σ is the sigmoid function. The new memory $\widetilde{c}^{<t>}$ uses the reset gate to store past relative information and the final memory $c^{<t>}$ uses the update gate to decide the information that needs to be retained:

$$\widetilde{c}^{<t>} = tanh(W x^{<t>} + U(\Gamma_r \odot c^{<t-1>}) + b_c) \tag{3.5}$$

$$c^{<t>} = \Gamma_u \odot c^{<t-1>} + (1 - \Gamma_u) \odot \widetilde{c}^{<t>}, \tag{3.6}$$

where *tanh* is the hyperbolic tangent activation function.

To help the decoder recover the transformed hidden vectors to the output vectors correctly, the proposed system is regarded as the translation model [61]. In this work, the hidden vector h is generated by the last hidden state of the encoder based on an input vector $x = (x^{<1>}, ..., x^{<T>})$, then estimating the probability of $\widetilde{x} = (\widetilde{x}^{<1>}, ..., \widetilde{x}^{<T>})$ with a formulation whose initial hidden state is set to the indirect representation h of x:

$$p(\widetilde{x}|x) = \prod_{t=1}^{T} p(\widetilde{x}^{<t>}|h, \widetilde{x}^{<1>}, ..., \widetilde{x}^{<t-1>}). \tag{3.7}$$

The core to achieving the mentioned goals of this subsection involved training the GRU-based autoencoder with the generated sequences by maximizing the log probability of input vector S to minimize the difference between input and correct output vector T, so the training objective is:

$$1/|\mathcal{S}| = \sum_{(S,T)\in\mathcal{S}} \log p(T|S), \tag{3.8}$$

where $T = S$. Once training is complete, the recovery outputs, according to the GRU-based autoencoder:

$$\hat{T} = \arg\max_{T} p(T|S), \tag{3.9}$$

To train the autoencoder, which keeps the input sequences and target sequences the same, the training samples for each ENAS method should be generated.

3.3. The AEEA scheme

As shown in Figure 3, the AEEA scheme contains a dictionary that links the genes and numbers, the trained encoder, the one-point crossover operator, the swap mutation operator and the trained decoder. In the proposed AEEA scheme, the variable-length individuals followed by a special flag "<EOS>" are translated into sequences, which are encoded into fixed-length representations. As those high-dimensional representations are easily utilized by the standard genetic operators, the one-point crossover strategy divides the fixed-length representations into head and tail parts. Before recombination, only the tail parts are swapped. Figure 4(a) illustrates the crossover operator on fixed-length representations. After crossover, the mutation operator is performed to expand the diversity of the population and to accelerate convergence so that the population can obtain the ability to explore better solutions and not fall into the local optimum. As the indirect encoding function, a popular mutation operator called swap mutation [62] is applied to the AEEA scheme. Figure 4(b) pictures the swap mutation operator in the AEEA scheme. The mutation operator probabilistically mutates the recombined representations, and the mutation probability is ϵ. If a representation is satisfied with mutation criteria ϵ, two units in this representation are randomly selected and their values are swapped. Finally, the processed representations are decoded by the trained decoder, and the decoded sequences are translated into chromosomes which demonstrate the candidate architectures based the built dictionary.

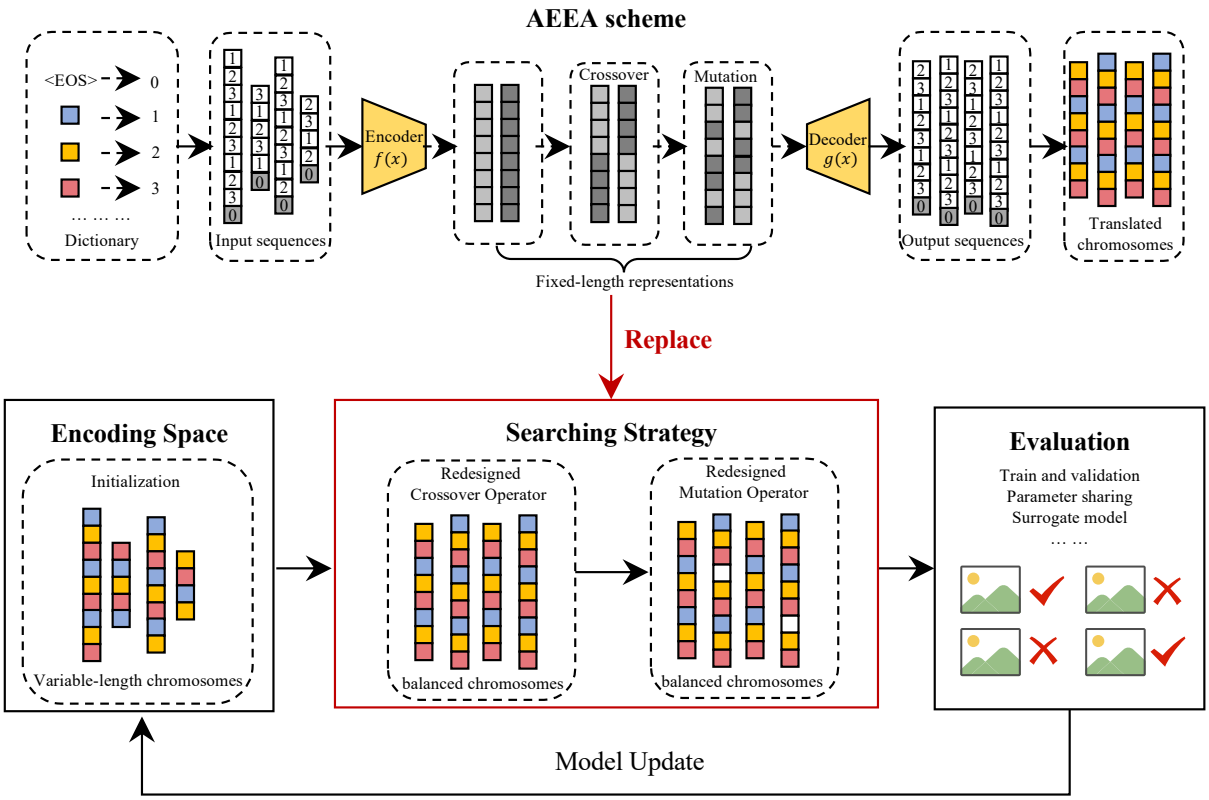

Figure 3. Replace crossover and mutation operators with the proposed AEEA scheme.

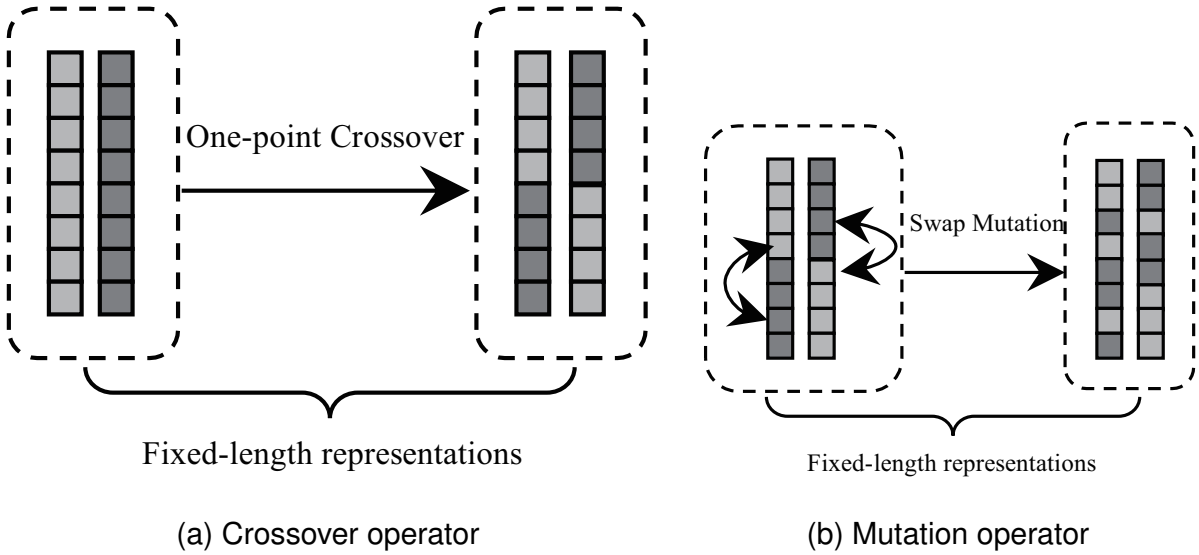

Figure 4. An example of crossover operator and mutation operator in the AEEA.

4. Experimental design

Since the proposed framework with the AEEA scheme aims at the shortcomings of most ENAS methods based on GAs, three main kinds of architectures in the deep learning community are chosen: CNN, RNN, and GNN. To study the effectiveness of the proposed framework, a significant quantity of quantified experiments were performed on state of the art ENAS methods. The following subsections introduce the chosen ENAS methods, the benchmark datasets, and the parameter settings.

4.1. The chosen ENAS methods

To verify the effectiveness of the proposed framework, various ENAS methods with chain-based, block-based, and cell-based encoding spaces are all included, and the selected ENAS methods should be processed accordingly based on their encoding spaces. In terms of chain-based encoding space, the parameters that form a layer are filled in a gene so that the length of chromosomes is equal to the corresponding structures. For block-based encoding space, the parameters that form a block are filled in a gene so that the length of chromosomes equals the number of searching blocks. For cell-based encoding space, each node and its relationship with other nodes are encoded into a gene so that the length of chromosomes equals the number of search nodes. To confirm the conjecture of the superiority of the standard genetic operators, variable-length ENAS methods with the developed operators are also selected. As the framework is designed to process the variable-length chromosomes, the encoding rules of the ENAS methods with variable-length encoding strategies remain unchanged. Some critical information is presented in Table 1.

Table 1. Important information about the chosen ENAS methods.

Fixed-length ENAS methods				
Method	Length	Operator	Target	Database
GEA [11]	10	standard	CNN	CIFAR10, CIFAR100
EANN [59]	21	standard	CNN	CIFAR10, CIFAR100
GeNet [12]	3	standard	CNN	CIFAR10, CIFAR100
IMMU-NET [63]	6	standard	CNN	CIFAR10, CIFAR100
NSGA-NET [13]	5	standard	CNN	CIFAR10, CIFAR100
GGNN [64]	2	standard	GNN	CORA, PUBMED
Variable-length ENAS methods				
Method	Length	Operator	Target	Database
EvoCNN [26]	[2, 10]	developed	CNN	CIFAR10, CIFAR100
VLC_GA [27]	[1, 8]	developed	RNN	Penn TreeBank, WikiText-2
HRN #1 [28]	[6, 15]	developed	RNN	Penn TreeBank, WikiText-2
HRN #2 [28]	[6, 15]	developed	RNN	Penn TreeBank, WikiText-2
AutoGraph [29]	undefined	developed	GNN	CORA, PUBMED

4.1.1. CNN architecture

CNNs are the most popular architectures to achieve fantastic results on image tasks. Generally, the CNN architecture is composed of convolutional layers, pooling layers, and fully connected layers.

ENAS methods build outstanding CNN architectures on image tasks by optimizing parameters, depth, and connections between units. There are some excellent ENAS methods for image classification tasks based on different encoding spaces:

GEA is a chain-based evolution to optimize the CNN parameters proposed in [11]. GEA encodes the CNN in a fixed-length chromosome with 10 numbered genes.

EvoCNN [26] is a famous chain-based ENAS method focused on the variable-length chromosomes by using two different types of genes, namely, the convolutional layer and pooling layer. In the experiment, the maximum lengths of each kind of layers are set to be the same as 5, which means the maximal depths of candidates are 10. To address the variable-length chromosomes during crossover, a method called unit alignment for chromosome recombination is designed to align those genes at the top, and crossover operators are performed at the same position.

EANN [59] is a block-based ENAS method with 8 proposed basic building blocks. They are ResNET-2015 [8], fb.no_relu [65], fb.bn_after_add [65], Inception [66], no_act [65], RestNet-2016 [67], ResNet-2016-1 $\times$ 1-cov [67] and Wide-Inception [68]. EANN searched the optimal combination of 8 selected blocks in a chromosome with 21 genes.

GeNet [12] is a particular block-based ENAS method. In each of the 3 blocks, the topology is not generated by preparation such as EANN, but by searching. In the experiments, a 3-stage network is adopted with 3, 4, and 5 nodes in each stage. To represent the internode connections, GeNet uses $\frac{1}{2}K_s(K_s - 1)$ bits in each stage, where K_s is the number of searching nodes in the s stage. For GetNet, the binary representation to describe a stage is encoded into a gene, and the length of chromosomes during evolution is always 3.

IMMU-NET [63] is an important ENAS method that has 7 kinds of layers and optimizes the combination of those layers to build a block. In IMMU-NET, a block contains 7 layers, and 6 blocks are searched to build the whole architecture. Similarly, all information to describe a block is encoded into a gene, and the length of chromosomes during evolution is always 6.

NSGA-NET [13] uses directed acyclic graphs (DAGs) consisting of 5 nodes to construct both types of blocks (a block uses a stride of two). Each node is a two-branched structure, mapping two inputs to one output. For pairs of inputs chosen, 12 options are provided to form a computation operation.

4.1.2. RNN architecture

As the architectures of processing the temporal data are repeatedly updated or rebuilt, the depth and density of the connectivity patterns are explored by the ENAS methods to yield a more rigorous automated examination [69]. In other words, the representations of RNN architecture are usually variable. Due to the variable-length representations, the child chromosomes generated by the one-point crossover strategy are generally imbalanced. As a result, the issue is resolved by the insert and shrink mutation [27, 28].

VLC_GA [27] is a chain-based ENAS for searching the RNN architectures. The proposed VLC_GA dynamic determined the number of layers and their connections during evolution to find the optimal structure. The number of layers is flexible, and the length of chromosomes is set from 1 to 8.

Heterogeneous recurrent networks (HRN #1) [28] is a block-based ENAS method. In HRN #1, the search structures in 3 blocks are repeated 30, 40 and 30 times. The HRN #1 is a variable-length representation as the search topology nodes vary between 6 and 15. In HRN #1, 3 developed mutation operators are proposed to address the limitations of the variable-length strategy.

HRN #2 is also proposed in [28]. Unlike HRN #1, HRN #2 is a cell-based ENAS method because the topology in two RNN cells is the same. The searched topology with [6, 15] nodes is repeated 100 times to create RNN CELL 1 and RNN CELL 2.

4.1.3. GNN architecture

As the basic operations of GNNs, the ENAS methods on graph tasks are complex and diverse, ranging from node-level to graph-level problems, with different settings, objectives, and constraints.

GeneticGNN (GGNN) [64] is a chain-based ENAS method optimizing the GNN architectures and evolving GNN structures with 5 searching components. In GGNN, the number of layers is set to 2, so the length of chromosomes is always 2.

AutoGraph [29] is an ENAS method based on chain representations. AutoGraph removed the traditional crossover operator in EA to generate the variable-length and deeper architectures automatically. It uses the mutation operator, which adds a new layer to the GNN model. AutoGraph applies the same setting of attention function, attention head, hidden dimension, aggregation function, activation function, skip connection and layer add information into a gene to describe a layer of the GNN. Due to the depth of architectures being changed during evolution, the length of chromosomes is undefined.

4.2. Benchmark datasets

To accurately evaluate the effectiveness of the framework, the datasets are selected according to various searching tasks.

4.2.1. CNN searching task

To justify the effectiveness of the proposed framework, the community of the CNN searching task often evaluates the performance of image classification tasks.

CIFAR10 [70] is a famous public dataset with 6000 examples of each of the 10 classes in the computer vision community, and many classic and state of the art networks have achieved great results on CIFAR10.

CIFAR100 [70] is a famous dataset that has 600 examples of each of the 100 nonoverlapping classes in the dataset. Compared with the classification on CIFAR10, the CNN searching tasks on CIFAR100 are more challenging and complex.

4.2.2. RNN searching task

Experiments described here have focused on predicting the next word in various datasets to determine the final evaluation of the RNN searching tasks.

Penn TreeBank (PTB) [71] is one of the most well-known character-level and word-level corpora used for the evaluation of models for sequence labeling.

WikiText-2 (WT2) [72] is a language modeling corpus that features a far more extensive vocabulary and retains the original case, punctuation and numbers. WT2 is over two times larger than PTB and is well-suited for models that can exploit long-term dependencies.

4.2.3. GNN searching task

The performance of the GNN searching task is assessed in a node classification scenario on graph-like datasets, where the nodes with known labels assign a class to nodes with unknown labels.

CORA [73] is a dataset that consists of 2708 scientific publications (nodes) with 5429 edges which represent one publication cited after the other. All those publications are classified into one of 7 classes.

PUBMED is a dataset that contained 19,717 publications (nodes) and 44,338 edges. All those publications are classified into one of 3 classes.

4.3. Parameter settings

The experiments in this work are divided into two parts. First, the ENAS methods with fixed-length encoding strategies are developed with the proposed framework. Second, the variable-length ENAS methods with the developed operators are also selected to confirm the benefit of standard genetic operators.

The first experiments follow two steps. First, we verify whether the proposed AEEA scheme affects the traditional fixed-length strategies. The symbol "+" after each selected ENAS method means the AEEA scheme is implemented in those fixed-length ENAS methods without changing their encoding space. Next, the second step implements the proposed framework in the chosen methods. The fixed-length strategies are transformed into variable-length strategies, and the symbol "*" is added behind the end of the names. The length settings in the first experiments are shown in Table 2. In detail, because the encoding spaces are unchanged in the first step, the length settings of the chosen methods in the first step follow the rule of the original ENAS methods, which are concluded in Table 1. For the second step, the fixed-length strategies are transformed into variable-length strategies, and the encoding space is altered. For GEA*, the length of chromosomes is set to [5, 15]. The number of units in EANN* is set to [15, 25]. For GeNet*, the number of stages is set to [2, 5]. Like GeNet*, IMMU-NET* and NSGA-NET* change the searching blocks, which are set to [4, 8] and [4, 6]. The number of layers in GGNN* is set to [2, 4].

Table 2. Length settings in first experiments.

Method	Length	Type
GEA+	10	fixed, chain-based
EANN+	21	fixed, block-based
GeNet+	3	fixed, block-based
IMMU-NET+	6	fixed, block-based
NSGA-NET+	5	fixed, block-based
GGNN+	2	fixed, chain-based
GEA*	[5, 15]	variable, chain-based
EANN*	[15, 25]	variable, block-based
GeNet*	[2, 5]	variable, block-based
IMMU-NET*	[4, 8]	variable, block-based
NSGA-NET*	[4, 6]	variable, block-based
GGNN*	[2, 4]	variable, chain-based

The second experiment confirms the importance of standard operators in the ENAS field. The

proposed framework can also process the variable-length chromosomes by the AEEA scheme with standard genetic operators. As a result, variable-length ENAS methods with developed genetic operators are selected. For example, EvoCNN proposed a unit alignment operator. AutoGraph removes the crossover operation. The VLC_GA designs insert and shrink mutation to address imbalanced child chromosomes. In this experiment, the encoding spaces are not altered as the proposed framework is designed to process variable-length chromosomes with standard genetic operators. We note that the search length is not defined in AutoGraph due to the mutation operator. To apply the proposed AEEA method in AutoGraph, the depth during evolution is set as $[2, 10]$ based on the comparison of the GNN models with different blocks on datasets in [29]. The symbol "+" after the name of the variable ENAS methods means they are implemented by the proposed AEEA scheme without changing their encoding space.

To unsupervised cotrain the encoder and the decoder with two GRU layers, the autoencoder should be trained in an unsupervised manner with the generated 100,000 training samples and 20,000 testing samples. For training the autoencoder until convergence, the batch size, epochs, learning rate (lr) and weight decay are set to 512, 50, 1e-4, and 1e-5, respectively. Adam is used as the optimizer, and mean-square error loss (MSELoss) is used as the criterion. For a fair comparison, the numbers of generations and individuals are set as 30 and 50, respectively, which means that the max evolution architecture is 1500. As the crossover and mutation operators of traditional ENAS methods are replaced by the AEEA scheme during evolution, the mutation rate ϵ is set as 0.2. For a specific generation, the number of reserved elites and abandoned offspring with the worst fitness are both 10. We note that hyperparameter settings not specifically stated are the same as those in the original methods. Table 3 concludes the settings in the AEEA scheme and search strategy.

Table 3. Parameter setting of contraining autoencoder and search.

	Type	Value
AEEA scheme	# samples to train autoencoder	100,000
	# samples to test autoencoder	20,000
	# layers of encoder	2
	# layers of decoder	2
	# input neurons	50
	# output neurons	50
	# hidden neurons	512
	# epochs	50
	# batch size	512
	lr	1e-4
	weight decay	1e-5
	optimizer	Adam
	criterion	MSELoss
Search strategy	# generations	30
	# population	50
	# reserved elites	10
	# abandoned offsprings	10
	probability ϵ of mutation operator	0.2

There are three search tasks: CNN ENAS for image classification, RNN ENAS for word prediction and GNN ENAS for node classification. To fairly evaluate the performance of candidate architectures during evolution, the settings of each task are the same. For the CNN searching task, batch size, epochs, lr, weight decay, optimizer and criterion are 256, 100, 1e-4, 1e-5, Adam and CrossEntropyLoss (CEL), respectively. The evaluated metric is classified error. For the RNN searching task, the settings of batch size, epochs, lr, weight decay, optimizer and criterion are 128, 100, 1e-4, Adam, 1e-3, and CEL, respectively. The evaluated metric is Perplexity. For the GNN searching task, hyperparameters of batch size, epochs, lr, weight decay, optimizer, and criterion are 32, 100, 1e-2, Adam, 5e-4 and NegativeLogLikelihoodLoss (NLL), respecetively. The evaluated metric is classified error. The parameter settings of each task are concluded in Table 4.

Table 4. Parameter setting of evaluation.

task	# epoch	batch size	lr	weight decay	optimizer	criterion	evaluated metric
CNN	100	256	1e-4	1e-5	Adam	CEL	Classified error
RNN	100	128	1e-4	1e-3	Adam	CEL	Perplexity
GNN	100	32	1e-2	5e-4	Adam	NLL	Classified error

In addition, the proposed AEEA method is implemented by PyTorch, and each copy of the code runs on a computer with 8 GPU cards with the identical model number GTX2080Ti.

5. Experimental results and analysis

In this section, the experimental results and analysis of the proposed framework are recorded in detail. Due to the CNN search tasks, RNN search tasks, and GNN search tasks all included, the chosen ENAS methods and corresponding variants are performed experiments under the same experimental situation based on the kinds of tasks for a fair comparison. As a result, the evolving curves of all methods with the fixed-length encoding strategies are packaged in Figure 5, and the methods with the variable-length encoding strategies are packaged in Figure 6. For each picture in Figures 5 and 6, the experimental dataset is shown on the top of the evolving curves, which are generated by the chosen methods and corresponding variants by generations.

To demonstrate the effectiveness of the proposed framework, the results and analysis are recorded in Subsection 5.1. To confirm the importance of standard genetic operators in the ENAS field, the comparison between the variable-length ENAS methods with developed operators and those methods that are developed with the proposed AEEA scheme is presented in Subsection 5.2.

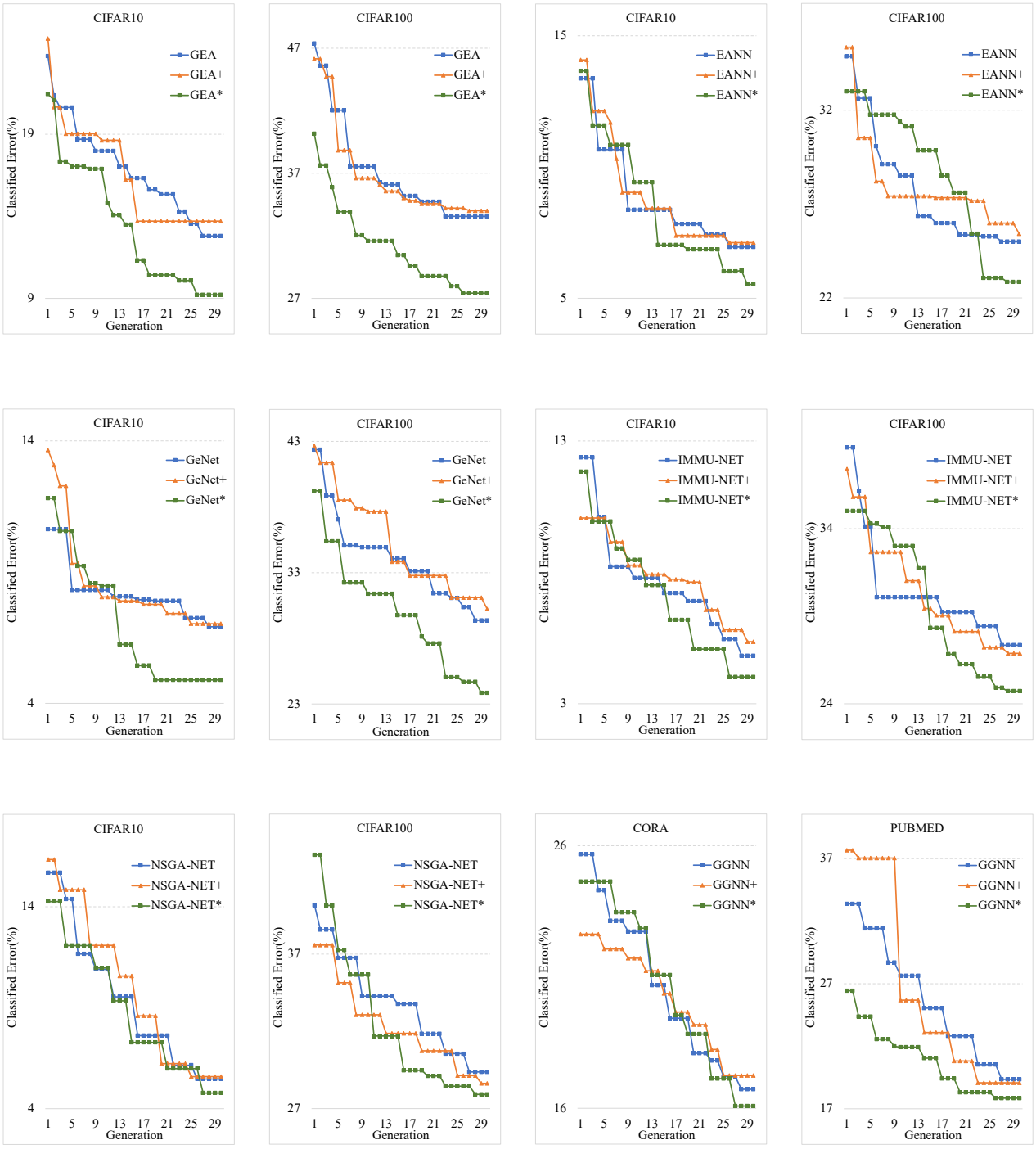

Figure 5. Evolving curves of fixed-length ENAS methods.

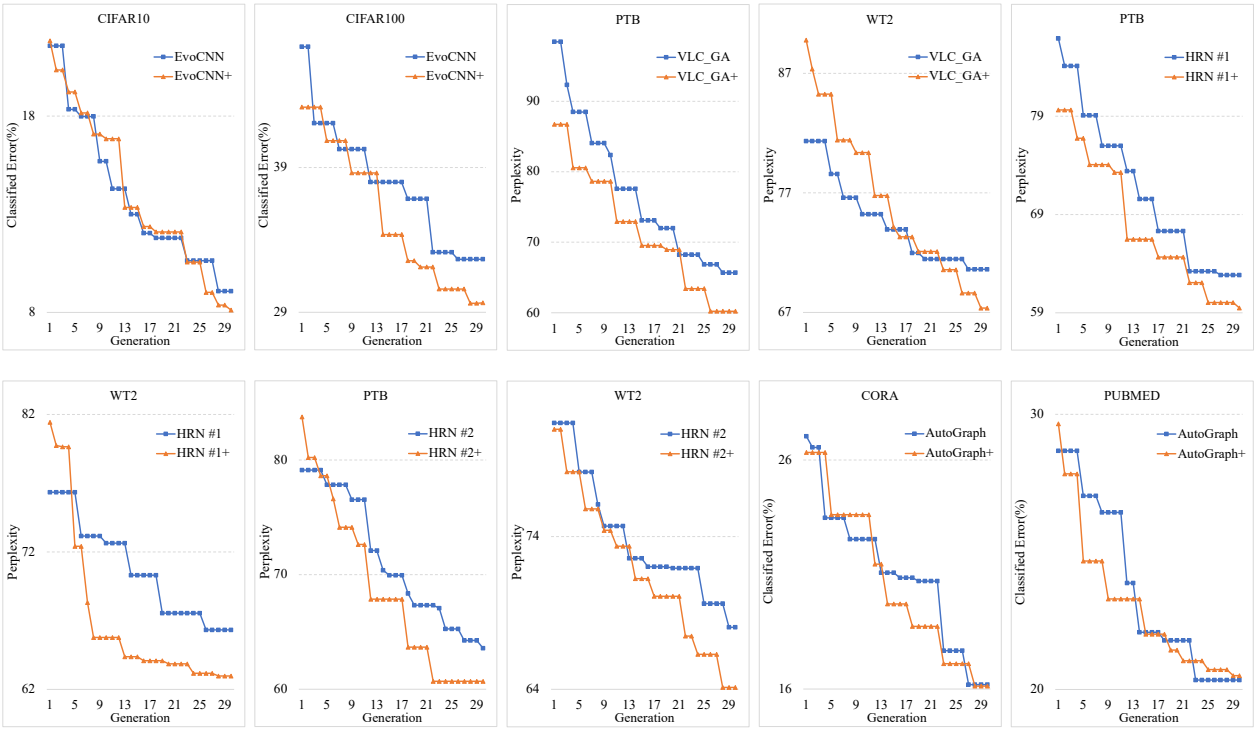

Figure 6. Evolving curves of variable-length ENAS methods.

5.1. Experimental results of fixed-length ENAS methods

The experiments in this subsection are divided into two steps. First, whether the proposed AEEA method has negative effects on those fixed-length ENAS methods is investigated. In this step, the fixed-length ENAS methods have performed the proposed AEEA scheme without changing their encoding spaces. In Table 5, the results are expressed in the form of *mean* ± *std*, where *mean* and *std* denote the mean and standard deviation of classified errors of 1500 architectures generated by each ENAS method, respectively. The *mean* values describe the overall ability to find suitable architectures, and the *std* values describe the stability of the ENAS methods. The experimental results show that the proposed AEEA scheme does not significantly affect the performance of the fixed-length ENAS methods. Compared with the original methods, the performances of those methods with the proposed AEEA scheme have not significantly decreased the performance.

Apart from the classified error metric, another important aspect of demonstrating the efficiency of NAS is the computational complexity of the methods. Since the theoretical analysis of the computational complexity of different NAS methods is challenging, we compare the computation time spent on GPUs and GPU-Days by each approach to arrive at the reported architectures. Following [13], the number of GPU-Days is calculated by multiplying the number of employed GPU cards by the execution time (in units of days). For a fair comparison, all chosen methods and their variants are performed experiments in the same experimental situation. Compared to the original ENAS methods, the AEEA scheme does not significantly affect the computational complexity. However, the complexity of developed methods with the proposed framework is heavily increased.

The reason is that the fixed-length ENAS methods are transformed into the variable-length ENAS methods. In return, the search space and search parameters are increased.

Table 5. The evaluated performance of each fixed-length ENAS method with the proposed AEEA method.

	CIFAR10		CIFAR100	
Name	Error (%)	GPU-Days	Error (%)	GPU-Days
GEA	22.52 ± 9.72	32	53.63 ± 20.08	61
GEA+	21.53 ± 7.83	32	52.79 ± 18.78	60
GEA*	19.61 ± 8.41	34	51.11 ± 21.72	65
EANN	12.06 ± 5.11	41	40.37 ± 15.36	78
EANN+	11.75 ± 4.63	40	41.99 ± 16.54	78
EANN*	10.47 ± 4.93	43	38.10 ± 14.46	81
GeNet	12.19 ± 5.27	19	42.95 ± 13.58	38
GeNet+	12.35 ± 5.32	19	43.53 ± 13.30	38
GeNet*	12.11 ± 5.49	21	40.83 ± 14.07	40
IMMU-NET	11.14 ± 6.31	20	44.27 ± 16.92	41
IMMU-NET+	10.64 ± 5.28	20	44.14 ± 17.26	41
IMMU-NET*	10.04 ± 5.72	22	42.90 ± 17.03	42
NSGA-NET	12.50 ± 7.03	29	43.39 ± 14.02	59
NSGA-NET+	12.71 ± 6.75	29	43.10 ± 14.47	58
NSGA-NET*	11.09 ± 6.32	30	42.51 ± 14.60	62
	CORA		PUBMED	
Name	Error (%)	GPU-Days	Error (%)	GPU-Days
GGNN	20.54 ± 3.81	0.4	25.02 ± 5.67	1.1
GGNN+	20.48 ± 3.23	0.4	24.14 ± 6.30	1.0
GGNN*	19.82 ± 3.11	0.6	24.71 ± 6.07	1.4

In the second step, the fixed-length strategies are transformed into variable-length strategies with the proposed framework. The experimental results of the optimal structures searched by each method are shown in Table 6. The proposed framework transforms the fixed-length GEA, EANN, GeNet, IMMU-NET, and NSGA-NET into dynamic search spaces for CNN search tasks. On the CIFAR10 and CIFAR100 datasets, the optimal architectures searched by ENAS methods with the proposed framework achieve better performance. In detail, the test error of fixed-length GeNet is 6.92% on the CIFAR10 dataset and 29.37% on the CIFAR100 dataset. However, the variable-length GeNet* won 2.03% on the CIFAR10 dataset and 5.52% on the CIFAR100 dataset. The biggest improvement appears in GEA. On the CIFAR100 dataset, the classification error of variable-length GEA* is only 27.42%, which is 6.13% less than that of the original GEA with a fixed-length encoding strategy. Similar results are obtained on the GNN searching tasks. For the GGNN*, the optimal depths of layers are 3 on the CORA and PUBMED datasets. The classified error of GGNN* is 16.09% on the CORA dataset, which is 0.64% better than the performance of GGNN. A similar situation appears in the PUBMED dataset, GGNN* achieves a 1.51% improvement with the proposed framework.

Table 6. The optimal performance of each fixed-length ENAS method with the proposed framework.

Name	CIFAR10		CIFAR100	
	Error (%)	Length	Error (%)	Length
GEA	12.80	10	33.55	10
GEA*	9.22	11	27.42	13
EANN	6.95	21	25.01	21
EANN*	5.53	19	22.85	23
GeNet*	4.89	4	23.85	5
IMMU-NET	4.83	6	27.35	6
IMMU-NET*	4.02	6	24.72	8
NSGA-NET	5.13	5	28.91	5
NSGA-NET*	4.31	5	26.21	6
Name	CORA		PUBMED	
	Error (%)	Length	Error (%)	Length
GGNN	16.73	2	19.35	2
GGNN*	16.09	3	17.84	3

5.2. Experimental results of variable-length ENAS methods

Although the fixed-length encoding strategies dominate the ENAS methods, some variable-length encoding methods have been proposed to explore the search space dynamically. However, the standard genetic operators cause an imbalance between the generated chromosomes. Variable-length ENAS methods solve this problem by developing standard genetic operators. Like VLC_GA, HRN #1, and HRN #2, they develop the mutation operator into insert and shrink operators. The insert operator is performed on the too-short child chromosomes, and the shrink operator is performed on the too-long chromosomes. The EvoCNN proposed a unit alignment to relieve this problem and achieved a good result in the CNN searching task. AutoGraph removes the mutation operator and the length of chromosomes is randomly added or cut during evolution.

In the AEEA scheme, the trained encoder is an indirect encoding function that encodes the variable-length sequences into fixed-length representations. The genetic operators are the only difference between variable-length strategies and those with the proposed AEEA scheme. As a result, the importance of standard genetic operators in the ENAS field can be confirmed by implementing the AEEA scheme on the variable-length ENAS methods. Table 7 shows the evaluated performance of the variable methods with developed operators and those ENAS methods with the proposed AEEA scheme. Similarly, the results are expressed in the form of $mean \pm std$, where *mean* and *std* denote the mean and standard deviation of the classified errors or perplexities of 1500 architectures generated by each ENAS method. The *mean* values describe the overall ability to find exemplary architectures, and the *std* values describe the stability of the ENAS methods. Moreover, Table 8 shows the optimal performance searched by each ENAS method with and without the proposed AEEA scheme.

Table 7. The evaluated performance of each variable ENAS methods with the proposed AEEA method.

	CIFAR10		CIFAR100	
Name	Error (%)	GPU-Days	Error (%)	GPU-Days
EvoCNN	22.50 ± 13.42	34	58.15 ± 25.49	69
EvoCNN+	19.86 ± 11.74	32	50.83 ± 21.18	65
	PTB		WT2	
Name	Perplexity	GPU-Days	Perplexity	GPU-Days
VLC_GA	86.52 ± 20.84	1.6	91.63 ± 21.03	2.9
VLC_GA+	75.45 ± 15.23	1.2	84.06 ± 16.70	2.2
HRN #1	81.22 ± 18.39	0.8	84.36 ± 18.03	1.5
HRN #1+	74.04 ± 14.56	0.7	79.08 ± 16.11	1.2
HRN #2	81.51 ± 17.92	0.8	85.63 ± 17.57	1.9
HRN #2+	75.57 ± 14.88	0.8	79.75 ± 15.62	1.6
	CORA		PUBMED	
Name	Error (%)	GPU-Days	Error (%)	GPU-Days
AutoGraph	23.24 ± 7.04	0.3	31.18 ± 10.84	0.9
AutoGraph+	22.64 ± 6.51	0.4	32.03 ± 11.53	0.9

Overall, variable ENAS methods developed with the proposed AEEA scheme achieve competitive performance on ENAS tasks. As shown in Table 7, the average classified error of EvoCNN+ on the CIFAR10 dataset is 19.86%, which is 2.64% less than the average error of EvoCNN. On the CIFAR100 dataset, the EvoCNN with the developed unit alignment operator produces the variable-length chromosomes and achieves an average classified error of 58.15% in 1500 searched architectures, which is 7.32% worse than EvoCNN+. Regarding the standard deviation, EvoCNN+ with the proposed AEEA scheme also outperforms EvoCNN with the developed operators. Compared with that of EvoCNN+, the standard deviation of EvoCNN has a 1.68% and a 4.31% gap on the CIFAR10 dataset and CIFAR100 dataset, respectively. All variable methods in RNN searching tasks developed the operators by randomly shrinking too-long chromosomes and inserting genes in too-short chromosomes. A similar situation appears in the RNN searching tasks, which means that the developed ENAS methods with the proposed AEEA scheme win on the PTB dataset and WT2 dataset. In detail, VLC_GA+ performs better on both datasets, with 11.07 average perplexities on PTB and 7.57 average perplexities on WT2. Because of the one-point crossover strategy, the VLC_GA obtains many too-short or too-long child chromosomes during evolution. After decoding target models, the shallow models with short chromosomes lose the ability to represent data, and complex models with long chromosomes are hard to train in minor epochs. As a result, the stability of VLC_GA has worse performance on both the PTB dataset and WT2 dataset. The standard deviations of VLC_GA have 5.61 and 4.33 larger than VLC_GA+ on both datasets. A similar situation appears in HRN #1 and HRN #2. On the mentioned two datasets, the HRN #1+ and HRN #2+ achieve a substantial lead on average perplexities and standard deviations. In experiments of the variable-length ENAS methods, the computational complexity is also investigated. We note that all ENAS methods and corresponding variants are performed in the same experimental situation. There is a huge difference between the fixed-length ENAS methods and variable-length ENAS methods. Compared

with the original ENAS methods, the computational complexity of the developed methods by the proposed frameworks is significantly decreased. Although the resigned crossover operators and mutation operators can temporarily solve the issues caused by the variable-length encoding strategies, the standard genetic operators are more suitable for GAs.

Table 8. The optimal performance of each variable ENAS method with the proposed AEEA method.

Name	CIFAR10 Error (%)	Length	CIFAR100 Error (%)	Length
EvoCNN	9.08	8	32.66	9
EvoCNN+	8.12	9	29.65	9
Name	PTB Perplexity	Length	WT2 Perplexity	Length
VLC_GA	65.68	6	70.60	7
VLC_GA+	60.22	7	67.36	8
HRN #1	62.83	10	66.33	13
HRN #1+	59.48	12	62.97	12
HRN #2	63.59	14	68.06	15
HRN #2+	60.69	13	64.13	13
Name	CORA Error (%)	Length	PUBMED Error (%)	Length
AutoGraph	16.20	4	20.34	3
AutoGraph+	16.13	3	20.50	3

The optimal structures searched by each method are also concluded in Table 8. Overall, the variable methods with the proposed AEEA scheme that uses the standard genetic operator improves the performance on the CNN and RNN searching tasks compared to that of the developed operators used by the original ENAS methods. For the CNN searching task, EvoCNN+ achieves 8.12 and 29.65 classified errors by using a 9-layer CNN on CIFAR10 and CIFAR100, respectively. For RNN searching tasks, HRN #1+ occupies the first place with 59.48 perplexities on the PTB dataset, which is 3.35 less than HRN #1, and 62.97 perplexities on the WT2 dataset, which is 3.36 less than HRN #1.

More experiments are performed on AutoGraph, as the experimental results show that the proposed framework with the AEEA scheme does not significantly improve the performance of AutoGraph. As seen from Table 8, AutoGraph+ leads the performance on the CORA dataset with only 0.07% classified error and falls behind with 0.16% classified error on the PUBMED dataset. To further study this issue, the oversmoothing problem is first investigated, following research [29]. Moreover, ablation experiments are designed to verify the impact of different genetic operators on the AutoGraph.

In terms of the oversmoothing problem, the performance of the GNN generated by AutoGraph with different numbers of graph layers is concluded in a table chart. Similarly, the comparison of the optimal models with different numbers of layers is illustrated in Figure 7. Due to AutoGraph removing traditional crossover operations and adding layers during evolution, deeper architectures may be generated during searching. We focus this research mainly on the GNNs with 2 to 10 layers, so the number of layers is not in [2, 10] and is packaged into others. Figure 7 pictures the performance

evaluation on the PUBMED and CORA datasets. The best architectures searched by AutoGraph and AutoGraph+ have 4 and 3 layers, respectively. The structures with 3 graph layers achieved first-rank results on the PUBMED dataset. In this experiment, an important conclusion is confirmed that more complex or DNNs may not consistently achieve excellent performance [74].

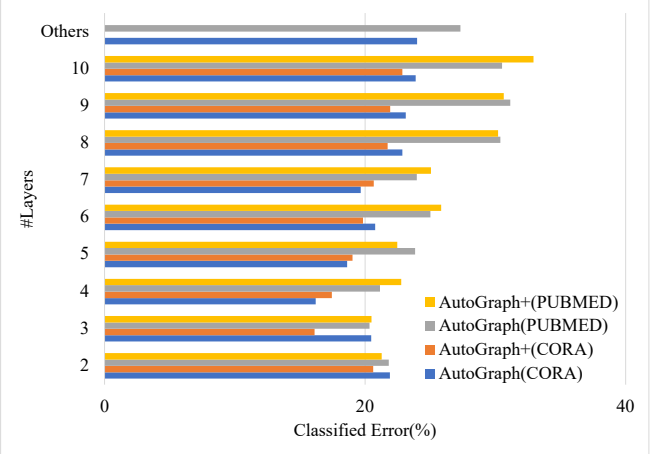

Figure 7. Comparison of the optimal models with different numbers of blocks.

To further study the effectiveness of the various genetic operators on AutoGraph, ablation experiments are designed based on the results of the oversmoothing problem. The optimal depth searched by AutoGraph and AutoGraph+ on the CORA dataset and the PUBMED dataset concentrated on 3 and 4 layers. Two variants of AutoGraph with the one-point crossover operator are designed in the ablation experiments, and the depth of the searched GNN is 3 and 4. The two variants are AutoGraph-3l, whose length is 3, and AutoGraph-4l, whose length is 4. For comparison, the experimental conditions, which are concluded in Tables 3 and 4, remain unchanged. Based on the experimental settings, the only difference between the designed methods is the types of genetic operators. The ablative results are shown in Table 9. The performance of AutoGraph is not significantly impacted by the crossover operator. Unlike other ENAS methods with variable-length encoding strategies that introduce many complexities and randomness, AutoGraph removed the crossover operator and the mutation operator added a layer one by one. This difference helps AutoGraph search the neural networks at the optimal depth.

Table 9. The comparison of different types of crossover operator.

Name	Type of operators	Results	
		CORA	PUBMED
AutoGraph	remove crossover	23.24 ± 7.04	31.18 ± 10.84
AutoGraph+	the AEEA scheme	22.64 ± 6.51	32.03 ± 11.53
AutoGraph-3l	one-point crossover	22.93 ± 5.34	31.42 ± 10.17
AutoGraph-4l	one-point crossover	23.14 ± 6.31	30.84 ± 11.09

6. Conclusions

To improve the performance of the existing ENAS methods based on GAs, a framework with the AEEA scheme was proposed to simultaneously exploit fixed-length encoding strategies and variable-length encoding strategies. Rather than predefining the depth of neural networks before the search, the proposed approach helps the existing ENAS methods with fixed-length encoding strategies dynamically determine the optimal depth of neural networks during the search. Moreover, the ENAS methods with variable-length encoding strategies also benefited from the proposed framework. With the AEEA scheme, the variable-length chromosomes can be processed by the standard crossover operator and mutation operator.

The effectiveness of the proposed framework was justified by experimental results. First, the performance of the ENAS methods with fixed-length encoding strategies was improved by the proposed framework, which uses those existing ENAS methods and transforms them into variable-length ENAS methods. The developed ENAS methods performed better than the original ENAS methods. Second, the performance of the ENAS methods with variable-length encoding strategies was also boosted because the encoder encodes the variable-length chromosomes into fixed-length representations that are easily processed by standard genetic operators. With the experimental results on benchmark datasets, the proposed framework with the AEEA scheme outperforms most state of the art methods from the automatic category.

Based on the present work, deeper research or further developments can be conducted in three areas. First, the proposed framework with the AEEA scheme focuses only on the GA-based ENAS methods. It is important that the proposed framework could be generalized to other branches of EA, such as genetic programming and evolutionary strategy. Second, efficient ENAS methods are all excluded in this work for a fair comparison. The reason is that most of the efficient ENAS methods design the approximate components or predictive components that may generate additional errors. In the future, the effectiveness of the proposed framework on the efficient ENAS methods needs to be verified. Third, the unique encoding strategy of each ENAS method is always the main contribution that distinguishes it from other ENAS methods. In return, the forms of chromosomes generated by each ENAS method have a large difference. Those chromosomes are recorded as sequences to train the seq2seq autoencoder, and the various forms of chromosomes can be regarded as different datasets. As a result, the proposed AEEA scheme must be retrained for each ENAS method, which requires considerable computational resources. A unified method should be researched to represent all those various encoding strategies so that the proposed AEEA scheme needs to be trained only once.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (U19A2078 and 62372315), Sichuan Science and Technology Planning Project (2022YFSY0047, 2023YFG0033, 2023ZHCG0016, 2023YFQ0020 and 2023ZYZYTS0077) and the Chengdu Science and Technology Project (2023-XT00-00004-GX and 2021-JB00-00025-GX).

Conflict of interest

The authors declare there are no conflicts of interest.

References

1. A. Esteva, K. Chou, S. Yeung, N. Naik, A. Madani, A. Mottaghi, et al., Deep learning-enabled medical computer vision, *NPJ Digit. Med.*, **4** (2021), 1–9. https://doi.org/10.1038/s41746-020-00376-2
2. A. Bhargava, A. Bansal, Fruits and vegetables quality evaluation using computer vision: A review, *J. King Saud Univ. Comput. Inf. Sci.*, **33** (2021), 243–257. https://doi.org/10.1016/j.jksuci.2018.06.002
3. Z. Wang, Q. She, T. E. Ward, Generative adversarial networks in computer vision: A survey and taxonomy, *ACM Comput. Surv.*, **54** (2021), 1–38. https://doi.org/10.1145/3439723
4. Y. Gu, R. Tinn, H. Cheng, M. Lucas, N. Usuyama, X. Liu, et al., Domain-specific language model pretraining for biomedical natural language processing, *ACM Trans. Comput. Healthcare*, **3** (2021), 1–23. https://doi.org/10.1145/3458754
5. D. H. Maulud, S. R. Zeebaree, K. Jacksi, M. A. M. Sadeeq, K. H. Sharif, State of art for semantic analysis of natural language processing, *Qubahan Acad. J.*, **1** (2021), 21–28. https://doi.org/10.48161/qaj.v1n2a40
6. I. Guellil, H. Saâdane, F. Azouaou, B. Gueni, D. Nouvel, Arabic natural language processing: An overview, *J. King Saud Univ. Comput. Inf. Sci.*, **3** (2021), 497–507. https://doi.org/10.1016/j.jksuci.2019.02.006
7. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, preprint, arXiv:1409.1556.
8. 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*, IEEE, (2016), 770–778. https://doi.org/10.1109/cvpr.2016.90
9. G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, (2017), 4700–4708. https://doi.org/10.1109/cvpr.2017.243
10. J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., **24** (2011), 1–9.
11. N. Mitschke, M. Heizmann, K. H. Noffz, R. Wittmann, Gradient based evolution to optimize the structure of convolutional neural networks, in *2018 25th IEEE International Conference on Image Processing*, IEEE, (2018), 3438–3442. https://doi.org/10.1109/icip.2018.8451394
12. L. Xie, A. Yuille, Genetic cnn, in *Proceedings of the IEEE International Conference on Computer Vision*, IEEE, (2017), 1379–1388. https://doi.org/10.1109/iccv.2017.154
13. Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. D. Goodman, W. Banzhaf, et al., Multiobjective evolutionary design of deep convolutional neural networks for image classification, *IEEE Trans. Evol. Comput.*, **25** (2020), 277–291. https://doi.org/10.1109/tevc.2020.3024708

14. X. Xiao, M. Yan, S. Basodi, C. Ji, Y. Pan, Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm, preprint, arXiv:2006.12703.
15. K. Zhou, Y. Dong, K. Wang, W. S. Lee, B. Hooi, H. Xu, et al., Understanding and resolving performance degradation in deep graph convolutional networks, in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, ACM, (2021), 2728–2737. https://doi.org/10.1145/3459637.3482488
16. Q. Li, Z. Han, X. M. Wu, Deeper insights into graph convolutional networks for semi-supervised learning, in *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI Press, **32** (2018), 1–8. https://doi.org/10.1609/aaai.v32i1.11604
17. K. Oono, T. Suzuki, On asymptotic behaviors of graph cnns from dynamical systems perspective, preprint, arXiv:1905.10947.
18. D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, X. Sun, Measuring and relieving the over-smoothing problem for graph neural networks from the topological view, in *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI Press, **34** (2020), 3438–3445. https://doi.org/10.1609/aaai.v34i04.5747
19. L. Ma, Y. Liu, G. Yu, X. Wang, H. Mo, G. G. Wang, et al., Decomposition-based multiobjective optimization for variable-length mixed-variable pareto optimization and its application in cloud service allocation, *IEEE Trans. Syst. Man Cybern.: Syst.*, **53** (2023), 7138–7151. https://doi.org/10.1109/tsmc.2023.3295371
20. L. Muwafaq, N. K. Noordin, M. Othman, A. Ismail, F. Hashim, Cloudlet based computing optimization using variable-length whale optimization and differential evolution, *IEEE Access*, **11** (2023), 45098–45112. https://doi.org/10.1109/access.2023.3272901
21. R. Domala, U. Singh, A survey on state-of-the-art applications of variable length chromosome (vlc) based ga, in *Advances in Artificial Intelligence and Data Engineering*, Springer, **1133** (2021), 615–630. https://doi.org/10.1007/978-981-15-3514-7_47
22. A. Maruyama, N. Shibata, Y. Murata, K. Yasumoto, M. Ito, P-tour: A personal navigation system with travel schedule planning and route guidance based on schedule, *IPSJ J.*, **45** (2004), 2678–2687.
23. M. Alajlan, A. Koubaa, I. Chaari, H. Bennaceur, A. Ammar, Global path planning for mobile robots in large-scale grid environments using genetic algorithms, in *2013 International Conference on Individual and Collective Behaviors in Robotics*, IEEE, (2013), 1–8. https://doi.org/10.1109/icbr.2013.6729271
24. J. J. Lee, D. W. Kim, An effective initialization method for genetic algorithm-based robot path planning using a directed acyclic graph, *Inf. Sci.*, **332** (2016), 1–18. https://doi.org/10.1016/j.ins.2015.11.004
25. Z. Qiongbing, D. Lixin, A new crossover mechanism for genetic algorithms with variable-length chromosomes for path optimization problems, *Expert Syst. Appl.*, **60** (2016), 183–189. https://doi.org/10.1016/j.eswa.2016.04.005
26. Y. Sun, B. Xue, M. Zhang, G. G. Yen, Evolving deep convolutional neural networks for image classification, *IEEE Trans. Evol. Comput.*, **24** (2019), 394–407. https://doi.org/10.1109/TEVC.2019.2916183

27. M. H. Aliefa, S. Suyanto, Variable-length chromosome for optimizing the structure of recurrent neural network, in *2020 International Conference on Data Science and its Applications*, IEEE, (2020), 1–5. https://doi.org/10.1109/icodsa50139.2020.9213012

28. A. Rawal, J. Liang, R. Miikkulainen, Discovering gated recurrent neural network architectures, in *Deep Neural Evolution*, Springer, (2020), 233–251. https://doi.org/10.1007/978-981-15-3685-4_9

29. Y. Li, I. King, Autograph: Automated graph neural network, in *International Conference on Neural Information Processing*, Springer, **12533** (2020), 189–201. https://doi.org/10.1007/978-3-030-63833-7_16

30. Y. Gong, Y. Sun, D. Peng, P. Chen, Z. Yan, K. Yang, Analyze covid-19 ct images based on evolutionary algorithm with dynamic searching space, *Complex Intell. Syst.*, **7** (2021), 3195–3209. https://doi.org/10.1007/s40747-021-00513-8

31. T. Elsken, J. H. Metzen, F. Hutter, Neural architecture search: A survey, preprint, arXiv:1808.05377.

32. B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, preprint, arXiv:1611.01578.

33. 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*, IEEE, (2018), 8697–8710. https://doi.org/10.1109/cvpr.2018.00907

34. Z. Zhong, J. Yan, W. Wu, J. Shao, C. L. Liu, Practical block-wise neural network architecture generation, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, (2018), 2423–2432. https://doi.org/10.1109/cvpr.2018.00257

35. H. Jin, Q. Song, X. Hu, Auto-keras: Efficient neural architecture search with network morphism, preprint, arXiv:1806.10282.

36. K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, E. P. Xing, Neural architecture search with bayesian optimisation and optimal transport, preprint, arXiv:1802.07191.

37. F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in *Learning and Intelligent Optimization*, Springer, **6683** (2021), 507–523. https://doi.org/10.1007/978-3-642-25566-3_40

38. Y. Sun, B. Xue, M. Zhang, G. G. Yen, An experimental study on hyper-parameter optimization for stacked auto-encoders, in *2018 IEEE Congress on Evolutionary Computation*, IEEE, (2018), 1–8. https://doi.org/10.1109/cec.2018.8477921

39. Y. Du, Y. Fan, X. Liu, Y. Luo, J. Tang, P. Liu, et al., Multiscale cooperative differential evolution algorithm, *Comput. Intell. Neurosci.*, **2019** (2019), 1–18. https://doi.org/10.1155/2019/5259129

40. V. P. Ha, T. K. Dao, N. Y. Pham, M. H. Le, A variable-length chromosome genetic algorithm for time-based sensor network schedule optimization, *Sensors*, **21** (2021), 3990. https://doi.org/10.3390/s21123990

41. 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*, AAAI Press, **33** (2019), 4780–4789. https://doi.org/10.1609/aaai.v33i01.33014780

42. J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, et al., Scalable bayesian optimization using deep neural networks, preprint, arXiv:1502.05700.

43. Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, K. C. Tan, A survey on evolutionary neural architecture search, *IEEE Trans. Neural Networks Learn. Syst.*, **34** (2021), 550–570. https://doi.org/10.1109/TNNLS.2021.3100554

44. Y. Wang, H. Yao, S. Zhao, Auto-encoder based dimensionality reduction, *Neurocomputing*, **184** (2016), 232–242. https://doi.org/10.1016/j.neucom.2015.08.104

45. M. Sakurada, T. Yairi, Anomaly detection using autoencoders with nonlinear dimensionality reduction, in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, ACM, (2014), 4–11. https://doi.org/10.1145/2689746.2689747

46. W. Wang, Y. Huang, Y. Wang, L. Wang, Generalized autoencoder: A neural network framework for dimensionality reduction, in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, IEEE, (2014), 496–503. https://doi.org/10.1109/cvprw.2014.79

47. X. Lu, Y. Tsao, S. Matsuda, C. Hori, Speech enhancement based on deep denoising autoencoder, in *Interspeech*, ISCA, (2013), 436–440. https://doi.org/10.21437/interspeech.2013-130

48. H. T. Chiang, Y. Y. Hsieh, S. W. Fu, K. H. Hung, Y. Tsao, S. Y. Chien, Noise reduction in ecg signals using fully convolutional denoising autoencoders, *IEEE Access*, **7** (2019), 60806–60813. https://doi.org/10.1109/access.2019.2912036

49. L. Yasenko, Y. Klyatchenko, O. Tarasenko-Klyatchenko, Image noise reduction by denoising autoencoder, in *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies*, IEEE, (2020), 351–355. https://doi.org/10.1109/dessert50317.2020.9125027

50. Z. Wan, Y. Zhang, H. He, Variational autoencoder based synthetic data generation for imbalanced learning, in *2017 IEEE Symposium Series on Computational Intelligence*, IEEE, (2017), 1–7. https://doi.org/10.1109/ssci.2017.8285168

51. S. Semeniuta, A. Severyn, E. Barth, A hybrid convolutional variational autoencoder for text generation, preprint, arXiv:1702.02390.

52. W. Xu, S. Keshmiri, G. Wang, Adversarially approximated autoencoder for image generation and manipulation, *IEEE Trans. Multimedia*, **21** (2019), 2387–2396. https://doi.org/10.1109/tmm.2019.2898777

53. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P. A. Manzagol, L. Bottou, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, *J. Mach. Learn. Res.*, **11** (2010), 3371–3408.

54. M. Tschannen, O. Bachem, M. Lucic, Recent advances in autoencoder-based representation learning, preprint, arXiv:1812.05069.

55. S. Lauly, H. Larochelle, M. M. Khapra, B. Ravindran, V. Raykar, A. Saha, et al., An autoencoder approach to learning bilingual word representations, preprint, arXiv:1402.1454.

56. I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in *Advances in Neural Information Processing Systems*, Curran Associates, Inc. **27** (2014), 1–9.

57. Y. A. Chung, C. C. Wu, C. H. Shen, H. Y. Lee, L. S. Lee, Audio word2vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder, preprint, arXiv:1603.00982.
58. H. Suresh, P. Szolovits, M. Ghassemi, The use of autoencoders for discovering patient phenotypes, preprint, arXiv:1703.07004.
59. Z. Chen, Y. Zhou, Z. Huang, Auto-creation of effective neural network architecture by evolutionary algorithm and resnet for image classification, in *2019 IEEE International Conference on Systems, Man and Cybernetics*, IEEE, (2019), 3895–3900. https://doi.org/10.1109/smc.2019.8914267
60. K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, et al., Learning phrase representations using rnn encoder-decoder for statistical machine translation, preprint, arXiv:1406.1078.
61. P. Koehn, Europarl: A parallel corpus for statistical machine translation, in *Proceedings of Machine Translation Summit X: Papers*, (2005), 79–86.
62. C. Moon, J. Kim, G. Choi, Y. Seo, An efficient genetic algorithm for the traveling salesman problem with precedence constraints, *Eur. J. Oper. Res.*, **140** (2002), 606–617. https://doi.org/10.1016/s0377-2217(01)00227-2
63. K. Chen, W. Pang, Immunetnas: An immune-network approach for searching convolutional neural network architectures, preprint, arXiv:2002.12704.
64. M. Shi, D. A. Wilson, X. Zhu, Y. Huang, Y. Zhuang, J. Liu, et al., Evolutionary architecture search for graph neural networks, preprint, arXiv:2009.10199.
65. A. Karpathy, Lessons learned from manually classifying cifar-10, Andrej Karpathy blog, 2011. Available from: http://karpathy.github.io/2011/04/27/manually-classifying-cifar10.
66. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, et al., Going deeper with convolutions, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, (2015), 1–9. https://doi.org/10.1109/cvpr.2015.7298594
67. K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in *European Conference on Computer Vision*, Springer, **9908** (2016), 630–645. https://doi.org/10.1007/978-3-319-46493-0_38
68. S. Zagoruyko, N. Komodakis, Wide residual networks, preprint, arXiv:1605.07146.
69. T. Desell, A. ElSaid, A. G. Ororbia, An empirical exploration of deep recurrent connections using neuro-evolution, in *International Conference on the Applications of Evolutionary Computation*, Springer, **12104** (2020), 546–561. https://doi.org/10.1007/978-3-030-43722-0_35
70. A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, 2009.
71. M. P. Marcus, B. Santorini, M. A. Marcinkiewic, Building a large annotated corpus of english: The penn treebank, *Tech. Rep. (CIS)*, **1993** (1993), 237.
72. S. Merity, C. Xiong, J. Bradbury, R. Socher, Pointer sentinel mixture models, preprint, arXiv:1609.07843.

73. A. K. McCallum, K. Nigam, J. Rennie, K. Seymore, Automating the construction of internet portals with machine learning, *Inf. Retr.*, **3** (2000), 127–163. https://doi.org/10.1023/A:1009953814988

74. J. Wei, Y. Tay, Q. V. Le, Inverse scaling can become u-shaped, preprint, arXiv:2211.02011.