Neural Architecture Search Based on Evolutionary Algorithms with Fitness Approximation

Chao Pan and Xin Yao
Shenzhen Key Laboratory of Computational Intelligence,
University Key Laboratory of Evolving Intelligent Systems of Guangdong Province,
Department of Computer Science and Engineering,
Southern University of Science and Technology, Shenzhen 518055, China.
Email: 11930665@mail.sustech.edu.cn, xiny@sustech.edu.cn

***Abstract*—Designing advanced neural architectures to tackle specific tasks involves weeks or even months of intensive investigation by experts with extensive domain knowledge. In recent years, neural architecture search (NAS) has attracted the interest of many researchers due to its ability to automatically design efficient neural architectures. Among all the search strategies, evolutionary algorithms have achieved significant success as derivative-free optimization algorithms. However, the tremendous computational resource consumption of the evolutionary neural architecture search dramatically restricts its application. In this paper, we explore how to apply evolutionary algorithms based on fitness approximation to neural architecture search and propose NAS-EA-FA to accelerate the search. We exploit data augmentation and diversity in neural architectures to enhance the algorithm, and present NAS-EA-FA V2. Experiments show that NAS-EA-FA V2 is at least five times faster than other state-of-the-art neural architecture search algorithms like regularized evolution and iterative neural predictor on NASBench-101, and it is also the most effective and stable algorithm on NASBench-201 in our experiments. All the code used in this paper is available at https://github.com/fzjcdt/NAS-EA-FA.**

***Index Terms*—Neural Architecture Search, Evolutionary Algorithm, Fitness Approximation, Diversity**

I. Introduction

Deep learning [1] has achieved remarkable achievements in many fields, such as image classification, speech recognition, and machine translation. However, many of the advanced neural architectures are designed manually by researchers, which is time-consuming. Even experts with rich domain knowledge should take weeks or even months to design a state-of-the-art neural architecture capable of solving a specific task.

Therefore, neural architecture search (NAS) has gradually attracted researcher's attention. The search strategies in NAS mainly include reinforcement learning (RL), evolutionary algorithms (EAs), and gradient-based methods. The reinforcement learning-based NAS methods [2] [3] regard the generation of neural architectures as the action of the agent, and the accuracy on the validation dataset as the reward. The gradient-based methods [4] [5] turn the search space into a continuous space, and search the architectures based on the gradient information. However, reinforcement learning-based NAS methods typically consume large amounts of computational resources, while the hyper-network constructed by gradient-based methods require strong prior knowledge. For gradient-based methods, there are also some disputes about the weight sharing strategy [4].

As derivative-free optimization algorithms, evolutionary algorithms have shown powerful capabilities to tackle complex optimization problems [6], even the black-box optimization problems in large-scale space. Thus, the idea of evolving artificial neural networks has been widely explored for over 20 years [7]. With the development of deep learning, evolutionary algorithms are further employed for neural architecture search, and has achieved impressive results. The recent research shows that the neural architecture search based on evolutionary algorithm can converge faster in the early stage, and performs not worse than the other algorithms in the later stage [8].

As with RL-based NAS, the tremendous resource consumption of evolutionary neural architecture search greatly limits its application. Therefore, this paper explores how evolutionary algorithms based on fitness approximation can be applied to NAS to reduce the number of evaluations and speed up the search. A variety of techniques, including data augmentation and diversity of neural architectures, were verified to be useful in this paper and further applied to enhance the algorithm. Experimental results on the NASBench-101 and NASBench-201 datasets show that our algorithms (NAS-EA-FA and NAS-EA-FA V2) converge faster than other SOTA NAS algorithms, and are more efficient and stable.

The primary contributions of this paper are as follows:

- We propose NAS-EA-FA, an efficient framework for applying evolutionary algorithms with fitness approximation to NAS, and verify that its effectiveness is competitive with other state-of-the-art NAS algorithms.
- We further explain that data augmentation and enhanced diversity contribute to the efficiency and stability of NAS-EA-FA.

This paper is organized in the following order: the second section is related work, presenting the baseline and SOTA algorithms for comparison in our experiments. We propose the NAS-EA-FA and NAS-EA-FA V2 in section three and verify the benefits of data argumentation and high diversity of neural architectures. Section four introduces how to encode the neural architecture in the datasets NASBench-101 and NASBench-201. Section five is the experimental settings and results, and section six provides the conclusions.

II. Related Works

A. Random Search

The encoding method of the neural architecture directly determines the search space, whereas the neural architecture search algorithms are only to search for the optimal neural architecture in the search space. Many prior knowledge, such as skip-connection can be helpful in training deeper networks [9], etc., can be added to the neural architecture encoding methods to reduce the search space and improve the search speed. However, these human biases also limit the ability of neural architecture search algorithms to find novel and efficient structures.

Thus, random search [10] is commonly served as a lower bound baseline under a given encoding method or search space to verify whether other neural architecture search algorithms are significantly effective. It's worth pointing out that random search can be competitive when the search space is good enough [10].

B. Evolutionary Neural Architecture Search

Evolutionary algorithms are a series of algorithms inspired by natural evolution, such as evolutionary programming (EP), evolutionary strategy (ES) and genetic algorithms (GAs), which are characterized by population-based and stochasticity [11]. Through mutation, crossover and selection operators, evolutionary algorithms can tackle optimization problems effectively [6], even for the black-box optimization problems in large-scale space.

There have been many studies combining evolutionary algorithms and neural networks since 20 years ago [7]. With the development of deep learning, evolutionary algorithms have been further applied to neural architecture search [12]. From the perspective of individual representation, the evolutionary neural architecture search can be classified into two categories: overall structural encoding and cell-based encoding. The former encodes the complete neural architecture as an individual, while the latter encodes only the cell and the complete neural architecture consists of a stack of identical cells.

1) Overall Structural Encoding: [13] employs the directed acyclic graph to represent the whole neural architecture, which is the first time that the evolutionary algorithm was applied to the deep neural architecture search. 11 mutation operators were proposed, including modifying the connection and the type of node operations, while the crossover operators were verified to contribute little to the search. Due to the large search space, the algorithm was ran in parallel on 250 computers for 15 days. [14] divides the neural architecture into multiple stages to reduce the search space and performs the search by a genetic algorithm. The chain-structured neural architectures [15] can be very well suited to crossover operators, but cannot represent skip-connections. Therefore, many researchers commonly encapsulate multiple operations into blocks [16] [17] [18], such as ResNet block, DenseNet block, etc.

2) Cell-based Encoding: Inspired by the fact that many efficient artificially designed neural networks are composed of stacks of identical cells [9] [19] [20], many researchers have devised cell-only search algorithms to further reduce the search space [8] [21]. [22] presents the hierarchical cell search space where high-level motifs are made up of combinations of low-level motifs. To maintain the diversity of the neural architectures in population, [22] preserved all individuals and applied tournament selection with tournament size of 5% of the population size as the selection strategy. [23] divides cells into normal cells and reduce cells, with only the latter affecting the shape of the feature map. Aging strategy (regularized evolution) has been proposed as the selection operator by replacing the oldest individual in the population with a mutated one to explore the larger search space [23]. Regularized evolution found the first neural architecture (AmoebaNet-A) that surpassed the hand-designs and was the best performing algorithm in the experiments of [8]. Therefore, we also take regularized evolution as a state-of-the-art baseline for comparison in our experiments.

C. Neural Predictor and Fitness Approximation

Neural architecture search algorithms typically have to evaluate a significant number of neural architectures, and accurately evaluating a neural architecture consumes tremendous computational resources (several or even dozens of GPU hours). Therefore, many studies have proposed to predict the performance of neural architectures by the performance predictor (neural predictor).

[24] employs ν-support vector machine regression (ν-SVR) as the neural predictor and has achieved promising results in both natural language processing and computer vision domains. [25] showed that decision tree-based models can better process discrete data and gradient boosting decision tree (GBDT) was adopted as the neural predictor. Graph convolutional networks (GCN) are also applied to the representation of neural architectures, and the resulting lower dimensional vectors are then taken as input to the neural predictor [26].

An iterative neural predictor-based neural architecture search algorithm is provided in Algorithm 1: in each iteration, M neural architectures are randomly sampled and the top K neural architectures with the highest prediction accuracy are selected for real training and evaluation. Then, the obtained results are added to the training data to train a new neural predictor.

In evolutionary algorithms, it is generally necessary to evaluate the fitness of a huge number of individuals. The fitness of one single neural architecture is typically its accuracy on the validation set. Accessing this information requires a significant amount of computational resources. Similar to the neural predictor, the fitness approximation model is employed in evolutionary algorithms to approximate the real fitness of an individual to reduce the number of evaluations [27].

Algorithm 1: Iterative Neural Predictor

```
Input: Number of architectures $M$ to sample. Number of architectures $K$ to evaluate. Number of iterations $T$.
Output: A neural architecture.
1 $X = Y = \{\}$
2 for $t = 1, . . . , T$ do
3     Train neural predictor $f$ by $X$ and $Y$.
4     Randomly sample M architectures to form $X_s$.
5     Predict the accuracy $Y_s$ of the architectures $X_s$ by $f$.
6     Train and evaluate the architectures with top $K$ predicted accuracy in $X_s \setminus X$ to get $X_s^{'}$ and $Y_s^{'}$.
7     $X = X \bigcup X_s^{'}, Y = Y \bigcup Y_s^{'}$
8 end
Output: The best neural architecture in $X$.
```

III. NAS-EA-FA

In this section, we demonstrate how the evolutionary algorithm with fitness approximation can be applied to neural architecture search, and present the NAS-EA-FA algorithm (Algorithm 2). NAS-EA-FA consists of two core modules: the fitness approximation update module(line 4-6) and the evolutionary algorithm module (line 9-14). The representation of neural architecture in the two modules is different. In order to improve the performance of fitness approximation (FA) model, the genotype of the individuals in the evolutionary algorithm is refined before being used as training data for FA, which is described in detail in section 4. We also apply the same processes to the neural predictor-based algorithm to ensure a fair comparison.

In NAS-EA-FA, K randomly sampled neural architectures constitute the initial population, each with an initial fitness of 0 (line 2). We take the accuracy of the neural architecture on the validation set as the real fitness of the individual. In each iteration, the K individuals in the population with the highest fitness that have not been truly evaluated are evaluated and added to X and Y (line 4-5). We train a new fitness approximation $f(X) \to Y$ from scratch by X and Y (line 6), where the FA is typically a regression model. In the evolutionary algorithm module, the g-generation population P_g is derived by mutation from the previous population P_{g-1}, and the fitness of each individual is predicted by fitness approximation f (line 9-13). All the individuals generated by the evolutionary algorithm are stored in $P^{'}$, and $P^{'}$ is assigned to P_0 at the end of the EA module (line 12-14). In the next iteration, the K individuals with the highest predicted fitness in $P_0(P^{'})$ were evaluated.

From the description of the NAS-EA-FA algorithm, we can observe that its performance is influenced by two parts: the accuracy of the FA and the effectiveness of the EA. Therefore, we further enhance the performance of NAS-EA-FA in terms of data augmentation and the diversity of neural architectures, and propose NAS-EA-FA V2 (Algorithm 3).

A. Data Argumentation

In general, the more training data, the better the FA model will be. However, more training data also means more neural architectures need to be evaluated, which consumes a great deal of computational resources. We expect to be able to generate some accurate data based on existing training data without re-evaluation.

Each neural architecture may have a couple of isomorphisms that vary greatly in their representations, but have the equivalent functionality, which means that their performance is identical. Thus, when we evaluate a neural architecture, we also get the performance of its isomorphisms, and we can expand the training data of FA in this way. We generate all isomorphisms of the neural architectures by permuting the node order [8], and add all neural architectures to the FA training data X (line 7-8 in Algo 3).

B. Diversity of the Neural Architectures

The active selection of training data tends to achieve higher performance with the same amount of training data. In the domain of active learning, many strategies have been used to select training data, and increasing the diversity of training data has been shown to be an effective approach [28]. Therefore, we can increase the diversity of training data for FA to enhance its performance. Also, in evolutionary algorithms, higher population diversity contributes to locating global optimal solutions and avoiding early convergence [29].

We represent the neural architectures as 01 sequences of length n, and all evaluated neural architectures and their isomorphisms are stored in X. For a neural architecture $x^{'}$, we identify the neural architecture x in X that is most similar to $x^{'}$, and use the distance between $x^{'}$ and x as the distance between $x^{'}$ and X. The calculation formula is as follows:

$$D(X, x^{'}) = \min_{x \in X} \sum_{i=1}^{n} \mathbf{1}(x_i^{'} \neq x_i) \tag{1}$$

Where $\mathbf{1}(x)$ is 0-1 indicator function, when x is true, the function value is 1, otherwise it is 0.

In each iteration of NAS-EA-FA V2, we evaluate not only the K neural architectures with the highest fitness (line 4), but also the H neural architectures with the largest distance from the available training data X (line 5). The H neural architectures with the greatest distance from the previously

```
Algorithm 2: NASEA-FA
Input: Number of architectures $K$ to evaluate. Number of generations $G$. Number of iterations $T$.
Output: A neural architecture.
1  $X = Y = \{\}$
2  $P_0$ = Randomly sample $K$ architectures with an initial fitness of 0.
3  for $t = 1, \dots, T$ do
4      Train and evaluate the individuals with top $K$ predicted fitness in $P_0 \setminus X$ to get $X'$ and $Y'$.
5      $X = X \bigcup X', Y = Y \bigcup Y'$
6      Train fitness approximation $f$ by $X$ and $Y$.
7      $P_0 = X'$ with fitness $Y'$.
8      $P' = \{\}$
9      for $g = 1, \dots, G$ do
10         Generate $P_g$ from $P_{g-1}$ through evolutionary algorithm.
11         Predict the fitness of $P_g$ by $f$.
12         $P' = P' \bigcup P_g$
13     end
14     $P_0 = P'$
15 end
Output: The best neural architecture in $X$.
```

evaluated neural architectures will be part of the initial population in the next iteration (line 6), which will also be beneficial for increasing the diversity of the population and exploring a larger space.

C. Verification Experiments

We further demonstrate that data augmentation and higher data diversity contribute to improve the performance of fitness approximation model. We compare the performance of fitness approximation model trained on random sampling, random sampling with data argumentation and active selection based on Eq. 1 under different training sample sizes.

In NAS-EA-FA, we are not concerned with the numerical deviation of the FA predictions from the real results, but only with the accuracy of the relative ranking. In other words, we prefer the relative ranking of neural architecture performance to be accurate than the small mean square error. Therefore, in the experiment, we measure the performance of the fitness approximation model by the mean number of reverse pairs (MNRP). The range of MNRP is 0 to 1. The smaller the MNRP, the more accurate the fitness approximation model.

For a sequence y of length n, if $i < j$ and $y_i > y_j$, then (y_i, y_j) is an inverse pair. We rank the set of neural architectures X by their accuracy on the validation set from lowest to highest, and the FA prediction for X is y. The average number of reverse pairs in the sequence y is considered to evaluate the performance of the fitness approximation model. The MNRP is calculated as follows:

$$MNRP(y) = \frac{2}{n \times (n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{1}(y_i > y_j) \qquad (2)$$

Where $\mathbf{1}(x)$ is 0-1 indicator function and n is the number of neural architectures in X.

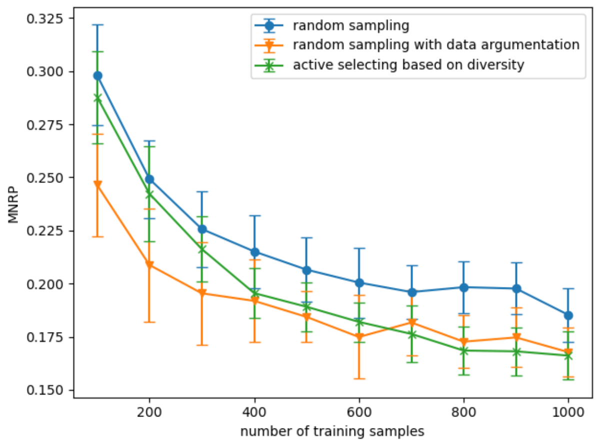

Fig. 1. The influence of data argumentation and diversity on fitness approximation model.

We adopte XGboost [30] with a learning rate of 0.1 as the fitness approximation model, NASbench-101 [8] as the dataset, and repeate the experiment 30 times. As shown in Figure 1, the error of the fitness approximation model becomes progressively lower as the training data increases. Compared with random sampling, data augmentation can significantly reduce MNRP when the amount of training data is relatively small. The performance of actively selecting widely differing neural architectures as training data is also not worse than random sampling at any point. This experiment reveals that data augmentation and higher diversity are indeed beneficial to enhance the performance of fitness approximation model.

Algorithm 3: NAS-EA-FA V2

```
Input: Number of top architectures $K$ to evaluate. Number of diversity architectures $H$ to evaluate. Number of
       generations $G$. Number of iterations $T$.
Output: A neural architecture.
1  $X = Y = \{\}$
2  $P_0$ = Randomly sample $K$+$H$ architectures with an initial fitness of 0.
3  for $t$ = 1, . . . , $T$ do
4      Train and evaluate the individuals with top $K$ predicted fitness in $P_0 \setminus X$ to get $X^{'}$ and $Y^{'}$.
5      Train and evaluate the individuals with top $H$ distance to $X$ in $P_0 \setminus X$ to get $X^{''}$ and $Y^{''}$.
6      $P_0 = X^{'} \bigcup X^{''}$ with fitness $Y^{'} \bigcup Y^{''}$.
7      $(X^{'}, Y^{'})$ = All isomorphisms of $(X^{'}, Y^{'})$.
8      $(X^{''}, Y^{''})$ = All isomorphisms of $(X^{''}, Y^{''})$.
9      $X = X \bigcup X^{'} \bigcup X^{''}, Y = Y \bigcup Y^{'} \bigcup Y^{''}$
10     Train fitness approximation $f$ by $X$ and $Y$.
11     $P^{'} = \{\}$
12     for $g$ = 1, . . . , $G$ do
13         Generate $P_g$ from $P_{g-1}$ through evolutionary algorithm.
14         Predict the fitness of $P_g$ by $f$.
15         $P^{'} = P^{'} \bigcup P_g$
16     end
17     $P_0 = P^{'}$
18 end
Output: The best neural architecture in $X$.
```

IV. Datasets

Massive computational resources make it impractical to compare different neural architecture search algorithms. Recently, with the public of some NAS datasets, we can verify the effectiveness of neural architecture search algorithms by directly querying the performance information of neural architectures in a smaller search space.

A. NASBench-101

NASBench-101 [8] is the first publicly available NAS dataset with 432K unique convolutional neural architectures. The neural architecture in NASBench-101 consists of stacks of identical cells, which is commonly used in many artificially designed neural architectures [9] [19]. The cell is represented by a directed acyclic graph (DAG) containing 7 nodes and up to 9 edges, where 5 internal nodes are one of 1×1 convolution, 3×3 convolution and 3×3 maximum pooling. Each neural architecture was trained three times on the CIFAR-10 dataset with the same training parameters, allowing direct querying of information such as validation accuracy, test accuracy and training time.

We take the type of nodes and the upper triangular adjacency matrix of the original cell as the genotype of the individual in the evolutionary algorithm. However, neural predictor and fitness approximation are regression models that are not suitable for taking discrete node-type data as input directly. Therefore, we represent the neural architecture in NASBench-101 by one-hot adjacency matrix encoding method [25] [26] [31], and the resulting 01 sequences are presented as inputs to the neural predictor and fitness approximation. As shown in Figure 2, we first delete the invalid nodes in the DAG and renumber the nodes. An invalid node implies that there is no path from input to output passing through this node. The one-hot encoding of the operations of the five internal nodes and the upper triangular adjacency matrix constitute the sequence of this cell. When the number of internal nodes is less than 5, empty nodes (none operation) are appended to ensure that the sequence length is the same for each cell.

B. NASBench-201

NASBench-201 [21], like NASBench-101, is a cell-based neural architecture dataset. However, in NASBench-201, the number of nodes and edges of a cell is fixed to 4 and 6, respectively, where nodes represent the sum of the feature maps and edges represent one of the following operations: 1×1 convolution, 3×3 convolution, 3×3 average pooling, skip connection, and zeroize. NASBench-201 contains 15,625 neural architectures, but only 8,764 unique neural architectures remain after removing isomorphisms and invalid neural architectures. Each neural architecture was trained three times on the CIFAR-10, CIFAR-100 and ImageNet-16-120 datasets, and NASBench-201 provides various data for each epoch of the model under each training, including the training time, accuracy, etc.

In NASBench-201, since all DAGs are densely connected, no additional encoding of the adjacency matrix of the neural architecture is expected. The genotype of the individual in the evolutionary algorithm is the type of 5 nodes in original cell. When generating the input (cell sequence) for neural predictor

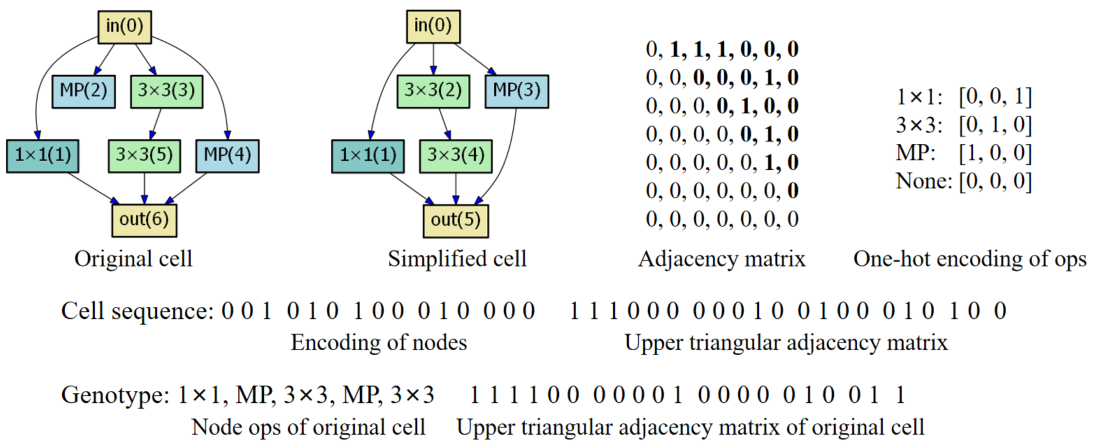

Fig. 2. An illustration of cell sequence generation and genotype representation of NASBench-101.

and fitness approximation, we will first simplified the cell. As shown in Figure 3, for the node whose input feature maps are all zero, we modify all the operations on its out-edges to zeroize. The simplified cell is functionally equivalent to the original cell. Then, the stack of one-hot encoding for each edge forms the sequence of this cell.

V. Experimants

A. Experimental Settings

In all the neural architecture search algorithms, if a neural architecture or its isomorphisms have been evaluated, there is no need to re-evaluate it, which can greatly reduce computational consumption. However, graph isomorphism problem is an NP problem, and the brute force solution is very expensive. Hence, we map the neural architecture to a hash value by the iterative graph hashing algorithm [32] [8] to determine whether two neural architectures are isomorphic under tolerable errors. When the hash value of a neural architecture to be queried has been recorded, its performance information can be obtained directly without any cost. In our experiments, we utilize this strategy for all neural architecture search algorithms to guarantee a fair comparison.

[25] proposed that gradient boosting decision tree (GBDT) is effective in NAS, so we use XGBoost [30], an elegant implementation of GBDT, as the regression model for neural predictor and fitness approximation. We follow the default parameters in XGBoost, except that the learning rate is set to 0.1. In regularized evolution (RE) and NAS-EA-FA, we follow the setting in [23] [8]. Tournament selection with the tournament size of 20% of the population size is employed as the selection strategy, while only bit wise mutation is applied to generate offsprings. Other parameters used in this paper are provided in table I.

When querying the performance of the neural architecture, the accuracy on the validation set is randomly selected from several runs. The performance information for the test set is not available during the run. We take the average accuracy of multiple runs on the test set as the final performance of the neural architecture [26] [25].

TABLE I
Experimental Parameter Settings

Dataset	Parameter	Value
NASBench-101	Time budget	6000000s
	Population size	100
	Tournament size	20%
	Ops mutation rate	$\frac{1}{5}$
	Connection mutation rate	$\frac{1}{21}$
	M in Algo 1	10000
	K in Algo 1 and 2	50
	K and H in Algo 3	30 and 20
	G in Algo 2 and 3	10
NASBench-201	Time budget (CIFAR10)	200000s
	Time budget (CIFAR100)	500000s
	Time budget (ImageNet16-120)	2000000s
	Population size	10
	Tournament size	20%
	Ops mutation rate	$\frac{1}{5}$
	M in Algo 1	1000
	K in Algo 1 and 2	5
	K and H in Algo 3	3 and 2
	G in Algo 2 and 3	10

All neural architecture search algorithms were run independently for 300 times, costing a total of 458 GPU years, and this experiment was only possible by relying on the NASBench101 and NASBench201 datasets.

B. Experimental Results

Figure 4 shows that, compared to random search, other neural architecture search algorithms are significantly more advanced in terms of convergence speed and final result. NAS-EA-FA approaches convergence at one million seconds, which is at least five times faster than other state-of-the-art algorithms, such as regularized evolution and iterative

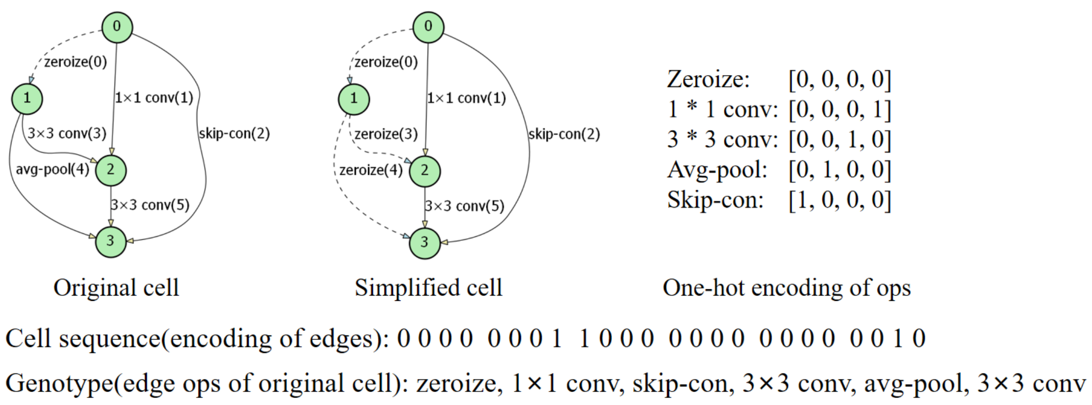

Fig. 3. An illustration of cell sequence generation and genotype representation of NASBench-201.

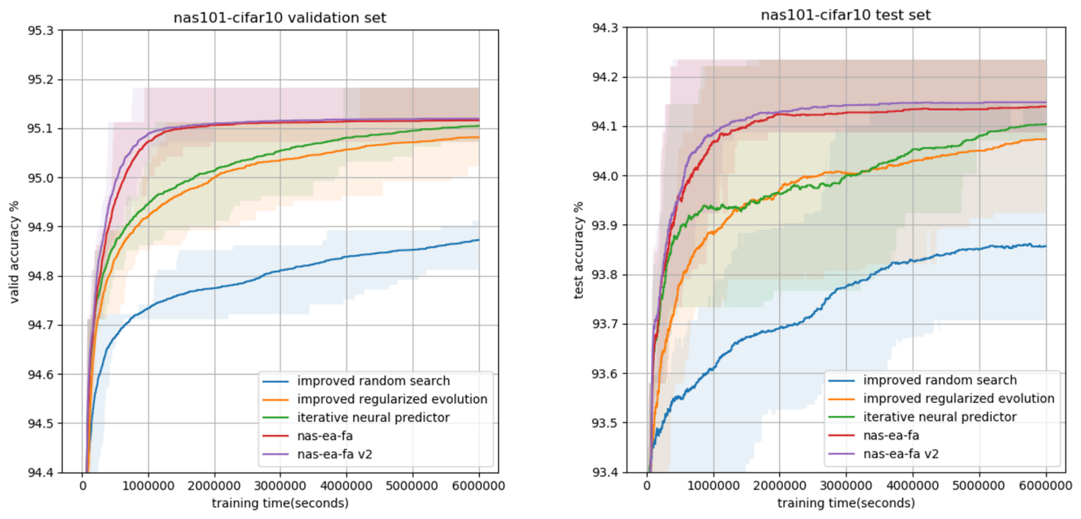

Fig. 4. Average accuracy of 300 runs on NASBench-101. The shaded area represents the results from the top 25% to the top 75% in 300 runs.

neural predictor. The terminal results of NAS-EA-FA and NAS-EA-FA V2 were similar, but NAS-EA-FA V2 converges faster because data augmentation and higher diversity making fitness approximation more accurate in the early stage. The final accuracy of the NAS-EA-FA V2 is 95.12%±0.06 on the CIFAR-10 validation set and 94.15%±0.15 on the test set.

In Table II, it can be observed that it is beneficial to avoid duplicate evaluation of neural architectures by querying their unique hash value. In particular, the improved regularized evolution is significantly better than the original regularized evolution in terms of average accuracy and standard deviation. NAS-EA-FA V2 not only achieved the best results on all data sets of NASBench-201, but also has the smallest standard deviation, indicating that NAS-EA-FA V2 is indeed effective and stable.

VI. Conclusion

In this paper, we explore how to apply evolutionary algorithms based on fitness approximation to neural architecture search and propose NAS-EA-FA and NAS-EA-FA V2. We further exploit the isomorphism of neural architecture to do data augmentation, and verify the importance of the diversity for fitness approximation model. Both techniques have been shown to be effective for fitness approximation model. In our experiments, NAS-EA-FA V2 performs at least five times faster than other state-of-the-art neural architecture search algorithms on NASBench-101, and it is also the most effective and stable algorithm on NASBench-201. Future work includes applying the approximation model to other components of evolutionary neural architecture search, such as mutation, crossover and initialization.

VII. Acknowledgement

This work is supported by the IEEE Computational Intelligence Society Graduate Student Research Grant 2020.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

TABLE II
EXPERIMENTAL RESULTS ON NASBENCH-201

NAS Methods	CIFAR10		CIFAR100		ImageNet16-120	
	Validation(%)	Test(%)	Validation(%)	Test(%)	Validation(%)	Test(%)
Random Search	91.07±0.28	90.68±0.34	71.60±0.80	71.41±0.94	45.81±0.56	45.48±0.68
Regularized Evolution	91.13±0.37	90.69±0.42	71.82±0.87	71.50±0.96	46.22±0.55	45.71±0.66
Improved Random Search	91.10±0.26	90.70±0.30	71.72±0.77	71.50±0.88	45.9±0.51	45.57±0.65
Improved Regularized Evolution	91.33±0.26	90.94±0.32	72.38±0.81	72.10±0.90	46.49±0.44	45.95±0.56
Iterative Neural Predictor	91.34±0.38	90.86±0.39	72.65±0.87	72.23±1.04	46.72±0.51	46.04±0.57
NAS-EA-FA	91.39±0.38	90.91±0.40	72.68±0.92	72.26±1.11	46.70±0.46	46.02±0.52
NAS-EA-FA V2	**91.55±0.21**	**91.08±0.24**	**73.09±0.64**	**72.76±0.76**	**46.90±0.28**	**46.28±0.39**

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

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

[4] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[5] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Advances in neural information processing systems*, pp. 7816–7827, 2018.

[6] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary computation*, vol. 3, no. 2, pp. 82–102, 1999.

[7] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[8] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," *arXiv preprint arXiv:1902.09635*, 2019.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[10] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in Artificial Intelligence*, pp. 367–377, PMLR, 2020.

[11] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[12] Y. Liu, Y. Sun, B. Xue, M. Zhang, and G. Yen, "A survey on evolutionary neural architecture search," *arXiv preprint arXiv:2008.10937*, 2020.

[13] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *arXiv preprint arXiv:1703.01041*, 2017.

[14] L. Xie and A. Yuille, "Genetic cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1379–1388, 2017.

[15] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.

[16] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing cnn architectures using the genetic algorithm for image classification," *IEEE Transactions on Cybernetics*, 2020.

[17] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Automatically evolving cnn architectures based on blocks," *arXiv preprint arXiv:1810.11875*, 2018.

[18] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the genetic and evolutionary computation conference*, pp. 497–504, 2017.

[19] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[21] X. Dong and Y. Yang, "Nas-bench-102: Extending the scope of reproducible neural architecture search," *arXiv preprint arXiv:2001.00326*, 2020.

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

[23] E. Real, A. Aggarwal, Y. Huang, and 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.

[24] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," *arXiv preprint arXiv:1705.10823*, 2017.

[25] R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture search with gbdt," *arXiv preprint arXiv:2007.04785*, 2020.

[26] W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, and P.-J. Kindermans, "Neural predictor for neural architecture search," in *European Conference on Computer Vision*, pp. 660–676, Springer, 2020.

[27] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft computing*, vol. 9, no. 1, pp. 3–12, 2005.

[28] K. Brinker, "Incorporating diversity in active learning with support vector machines," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 59–66, 2003.

[29] D. Sudholt, "The benefits of population diversity in evolutionary algorithms: a survey of rigorous runtime analyses," in *Theory of Evolutionary Computation*, pp. 359–404, Springer, 2020.

[30] T. Chen, T. He, M. Benesty, V. Khotilovich, and Y. Tang, "Xgboost: extreme gradient boosting," *R package version 0.4-2*, pp. 1–4, 2015.

[31] C. White, W. Neiswanger, S. Nolen, and Y. Savani, "A study on encodings for neural architecture search," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[32] C. Ying, "Enumerating unique computational graphs via an iterative graph invariant," *arXiv preprint arXiv:1902.06192*, 2019.