Elite Parent Preserving Evolutionary Neural Architecture Search for Image Classification Mid-term Report

Bowen Zheng, Shijie Chen, Shuxin Wang
Department of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, Guangdong, China

Abstract—In this project, we propose an elite-parent preserving evolutionary framework for Neural Architecture Search on image classification problems. We have finished the evolutionary algorithm framework and the construction of neural architecture. Currently we focus on a cell-based search space where a neural architecture is composed of multiple interconnected cells. Experiments show the effectiveness of our algorithm in obtaining a neural architecture with good performance. In the future, we will try to add more evolutionary operations and try to enlarge the search space.

I. Introduction

The great leap of computing resources in the past few decades made it possible to fully utilize the potential of neural networks. In recent years neural networks outperformed traditional methods in many fields of research, especially image classification. However, the state-of-the-art architectures are carefully designed and tuned by researchers for a specific problem. Therefore, people start to think about automating the design of neural networks in the hope of finding the best-performing network architecture efficiently.

Neural architecture search is a research field focusing on automating the design of neural networks. Currently there are a few popular approaches, including reinforcement learning, Bayesian optimization, tree-based searching and genetic-based evolutionary algorithms.

This project focuses on NAS for image classification problems. The reason is that this area is well explored and there exist many high-performance hand-crafted neural architectures. They provide a good guidance and target to our project. In addition, neural networks for image classification are mostly built upon basic units including convolution, polling, normalization, and activation layers. This helps to shrink our search space.

II. Related Works

A lot of research works have been done on NAS. Some researchers have proposed algorithms that can design architectures with performance on par of or even better than state-of-the-art hand-crafted neural networks. Research topics in NAS are divided into three categories: search space, search strategy and performance estimation strategy.

A. Search Space

The search space of NAS determines the possible architectures a NAS algorithm can find.

The simplest search space is the simple multiple-layer structure, in which a neural network A is composed of multiple layers L_i connected to the neighboring layers. In this case, the search space can be described by (1) The maximum number of layers (2) The type and dimension of each layer and their hyper-parameters [1] [2].

In more recent studies, some researchers use a cell-based search space in which the possible architecture of cells is explored. A cell is nothing but a smaller neural network. The entire neural network is constructed by connecting several pre-defined cells. A cell has fewer layers but allows more complex architectures like skip connections between any layers [3] [4]. The cell could be some hand-crafted neural networks that have already been proofed effective. The search space is therefore decreased to the possible arrangements of cells.

In contrast to the above direction, some researchers also tried to search for effective cells and connect them at last in a predefined manner [5] [3]. The search space is greatly decreased in that each cell is comparably small. This method can also be easily transferred to other datasets [5] since the structure of cells is not fixed.

Recently, some researchers managed to optimize the overall architecture as well as the cells at the same time and obtained state-of-the-art result [6].

B. Search Strategy

Many different search strategies can be applied to explore the search space discussed above. These methods include Bayesian optimization, evolutionary methods, reinforcement learning and gradient-based methods.

Evolutionary algorithms have been used to evolve neural networks since 1989 [7]. Earlier works use genetic algorithms to both optimize the structure of neural networks and train the networks [8]. However, with the birth of back-propagation (BP), recent works of neural-evolution use genetic algorithms only for optimizing neural architectures and use BP to train the networks [9]. In the context of NAS, the individuals in the genetic algorithm are

neural network architectures and the genetic operations (crossover and mutation) are used to alter the architecture by adding/removing layers or change the connectivity of nodes.

Genetic algorithms show their diversity in how they sample parents, generate offspring and update population. Some work choose parents from a preto-optimal front [10] while others use tournament selection [11] [4] [9]. When generating offsprings, some algorithms randomly initialize the weight of child networks. In comparison, Lamarckian inheritance is used in [10] so that child networks could inherit the weight of its parent and the training cost is reduced. To update population, some algorithms abandon least capable individuals [9] while some delete the oldest individuals [4]. A more sophisticated policy is developed by [12] and [13] in which the age of the individuals are taken into account.

There are other methods that are used to implement NAS, including Bayesian optimization, reinforcement learning,tree-based search and gradient-based methods. We don't discuss them here since we use evolutionary algorithms in our project.

C. Performance Estimation Strategy

One important issue in neural architecture search is the estimation of neural network performance. This is critical in the population update policy of evolutionary algorithms.

The simplest way to estimate performance is to train every searched network from scratch and test the desired performance metric, e.g. accuracy on the validation set. However, the training of the neural network is very time and computation consuming.

An alternative is to estimate performance on lower fidelities. More specifically, to train the network for a shorter period of time [5] or on some subset of the dataset [14]. However, the estimate must ensure the result ranking of different networks must be the same as that of complete training. That is to say, there exists a trade-off between computational load and estimation fidelity.

Another approach to estimate performance is based on learning curve extrapolation [15]. This method accelerates estimation by stop poor performance networks in the early state of training based on statistical patterns of learning curves. Other researchers propose ways to predict neural network performance based on architectural and cell properties [11].

III. Methodology

We will develop an evolutionary neural architecture search algorithm based on the *age* of individuals. By incorporating *age* as a part of a gene, we can prolong the existence of good individuals [13] and kill average individuals when their age reaches a certain limit [12].

Combining the advantage of the above works, we propose a population update police based on *age* and that

preserves a parent whose child has good performance. An individual is dropped from the total population if its *age* exceeds a predefined *lifetime*. However, we prolong its life if its offspring performs well (e.g. within top $P\%$ in ranking). In this way, we hope to preserve good parent architectures in the population.

To test the effectiveness of our algorithm, we will experiment on image classification datasets including CIFAR-10, CIFAR-100 and will possibly extend to IMAGENET.

IV. System Design

A. Cell-based Search Space

We explore a cell-based search space with 5 hidden cells, as is illustrated in Fig.1. Each edge in Fig.1 represents a neural network unit. Each square represents a tensor in the neural network.

1) Neural Network Units: There are 7 possible neural network units that are suitable for image classification problems in our search space:

1) identity
2) 3×3 average polling
3) 3×3 max polling
4) 1×1 convolution
5) 3×3 depthwise-separable convolution
6) 3×3 dilated convolution
7) 3×3 convolution

The search space is the possible connections of the states using different neural network units.

2) Cell Architecture Representation: We represent the architecture of a cell by a vector R which is a vector of lists of tuple $R_i = \{(a_{i1}, b_{i1}), (a_{i2}, b_{i2}), ...\}$ where a_i marks the input state of state i and b_i shows the unit used in the state transition connection.
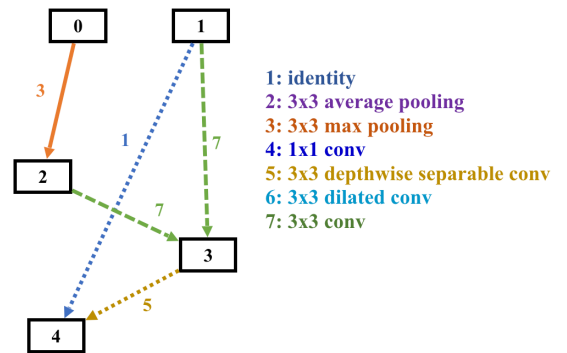

Fig. 1: An example artitecture of a cell with representation $R = \{\{\}, \{\}, \{(0, 3)\}, \{(2, 7), (1, 7)\}, \{(1, 1), (3, 5)\}\}$.

B. Overall Neural Network Architecture

As is illustrated in Fig.2 overall network consists of input layer, 6 ($N = 2$) identical cells, 3 polling layers, 1 global average polling(GAP) layer, 2 fully connected(FC) perceptron layers and 1 softmax layer.
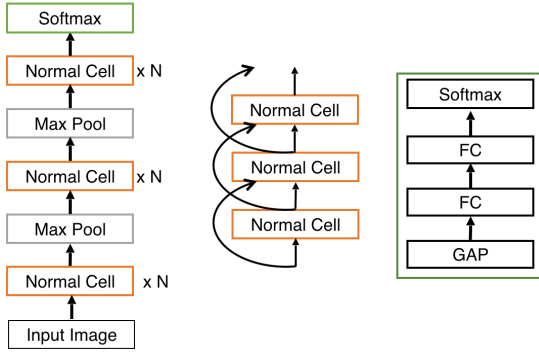
Fig. 2: The simplified overall artitecture of the total network. LEFT: outer structure. MIDDLE: detailed structure of normal cell stack. RIGHT: detailed structure of the Softmax unit.

For simplicity, all cells share the same structure.

C. Early Stop Performance Estimation

To accelerate the evolution process, we need to find a way to estimate the performance of a generated neural architecture. We use an early-stop strategy that trains the neural network with a small epoch value. This approach can noticeably cut the training time of a network and at the same time provides a relatively reliable performance comparison metric.

D. Evolutionary Algorithm Framework

Algorithm 1 Elite Parent Preserving Evolution

1: $population \leftarrow \phi$
2: $entireGen \leftarrow \phi$
3: $generation \leftarrow 0$
4: while $population$ size$< N$ do
5:     $newNetwork \leftarrow networkInit()$
6:     $trainNetwork(newNetwork)$
7:     add $newNetwork$ to $popution$ and $entireGen$
8: end while
9: while $genetation < G$ do
10:     $parent \leftarrow$ select a parent from population using tournament selection
11:     $child \leftarrow mutate(parent)$
12:     $trainNetwork(child)$
13:     add $child$ to $popution$ and $entireGen$
14:     if accuracy of $child$ is better than $P\%$ individuals in population then
15:         extend the $lifetime$ of $parent$ by t
16:     end if
17:     for all $individual$ in $population$ do
18:         update the age of $individual$
19:         remove current $individual$ if its age reaches its $lifetime$
20:     end for
21: end while
22: return the network model with highest accuracy in $entireGen$

For now, we follow the design of a cell based evolutionary NAS algorithm described in [13] except that we propose a population update policy to preserve good parents. Our evolutionary algorithm features an elite parent preserving strategy in which the lifetime of the parent whose child has a high accuracy rank within the total population is extended. In this way, we hope to preserve good genes in the population and produce more high-quality offsprings.

The *population* size is N and the algorithm evolves G generations in total. *entireGen* stores all network models that we generated. P and t are hyper-parameters controlling the actual lifespan of an individual.

In *networkInit()*, N initial network models are generated. New Networks are generated in *NetworkInitialization* with their *age* being set to 1 and *lifetime* being set to the default value. The architecture of the network is obtained by randomly adding legal connections. A legal architecture is one with all hidden states having a positive in-degree and without any go-back connections.

Algorithm 2 Network Initialization

Ensure: A randomly generated network.
1: $network \leftarrow$ new $Network$
2: $network.age \leftarrow 1$
3: $network.life \leftarrow$ DEFALT_LIFE
4: $arch \leftarrow \phi$
5: for all l in hidden layers do
6:     $edge_l \leftarrow \phi$
7:     for every input layers and hidden layers before l' do
8:         if $random < p$ then
9:             $e \leftarrow$ edge from l' to l with an randomly chosen neural network unit.
10:             $edge_l$.add(e)
11:         end if
12:     end for
13:     if in-degree of $l >0$ then
14:         $e \leftarrow$ legal edge to l with a random unit.
15:         $edge_l$.add(e)
16:     end if
17:     $arch$.add($edge_l$)
18: end for
19: $edge_{out} \leftarrow \phi$
20: for all i with 0 out-degree do
21:     $e \leftarrow$ edge from i to output layer with identity unit.
22:     $edge_{out}$.add(e)
23: end for
24: $arch$.add($edge_{out}$)
25: $network.arch = arch$
26: $network.accuracy \leftarrow getAcc()$
27: return The $network$ generated.

When sampling the parent individuals, we randomly

take X individuals from the population and pick the one with the highest estimated accuracy. The mutation operation includes adding, deleting and altering an edge(NN unit). Fig.3 gives an example. At the same time, the resulting model must be valid and conform to our representation definition. Currently, mutation operations are stochastic. A better heuristic approach can be used to improve performance.

Algorithm 3 Mutation Operation

Require: A network architecture $arch$
Ensure: The $arch$ with a mutation
 1: Randomly pick a mutate position between two hidden layer.
 2: **while** $arch$ is unchanged or $arch$ is illegal **do**
 3:     Apply mutation to the chosen position.
 4: **end while**
 5: $edge_{out} \leftarrow \phi$
 6: **for all** i with 0 out-degree **do**
 7:     $e \leftarrow$ edge from i to output layer with identity unit.
 8:     $edge_{out}$.add(e)
 9: **end for**
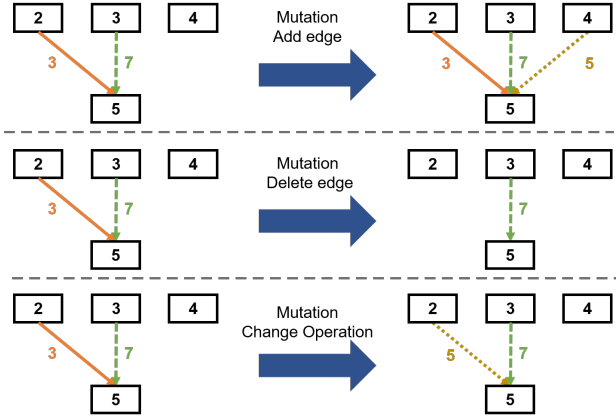10: $arch$.replace($edge_{out\_old}$,$edge_{out}$)
11: **return** arch

Fig. 3: Three mutation operations.

V. Experiments

A. specification

The followings are the parameters used in our experiment during search process.

 1) Population Size $population = 10$
 2) Sample Size $X = 5$
 3) Total Generations: 30
 4) Batch Size: 100
 5) Learning Rate: 0.001
 6) Epoch: 20
 7) Stack depth: $N = 2$
 8) Number of Channels: 32 for stack1, 64 for stack2 and 128 for stack3.

B. Result

Fig.4 shows the best model our algorithm found that reaches an accuracy of 70% on CIFAR-10 dataset.Currently, our model is still primitive. So this result is more of a validation of the correctness of our code.
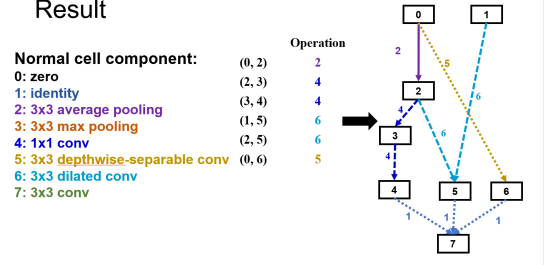

Fig. 4: Structure of the found network.

VI. Future Work

We have already defined the representation of neural architectures and have implemented the evolutionary search algorithm. The following are the improvements we are going to make for finishing this project.

1) Weight Sharing of Neural Networks: One major cost in NAS is the training of neural networks. Offspring networks are generated through mutation operation and share a lot of structures with their parents. Therefore, we can let the offspring inherit the weights of its parent. This will cut the training time cost significantly compared to training a neural network from scratch.

2) Heuristic Mutation Operation: Currently, the mutation operation is stochastic. Experiments show that such a random mutation operation has very limited searchability. We can try to design a heuristic mutation operation based on knowledge in neural network architectures and possibly graph theory.

3) More Efficient Performance Estimation: A better performance estimation approach is needed to accelerate the search process. We may try some approaches that require little training.

4) Find Proper Size of a Cell: The training cost is highly related to the amount of parameters in a neural network. We should find a way to set a proper size (number of hidden states) for the cell. This will help to find a good neural network architecture without enlarging the search space too much.

References

[1] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1251–1258, 2017.
[2] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," arXiv preprint arXiv:1611.02167, 2016.
[3] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," arXiv preprint arXiv:1806.02639, 2018.

[4] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," arXiv preprint arXiv:1802.01548, 2018.

[5] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8697–8710, 2018.

[6] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," arXiv preprint arXiv:1901.02985, 2019.

[7] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms.," in ICGA, vol. 89, pp. 379–384, 1989.

[8] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," Evolutionary computation, vol. 10, no. 2, pp. 99–127, 2002.

[9] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 2902–2911, JMLR. org, 2017.

[10] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," 2018.

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

[12] G. S. Hornby, "Alps: The age-layered population structure for reducing the problem of premature convergence," in Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06, (New York, NY, USA), pp. 815–822, ACM, 2006.

[13] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," CoRR, vol. abs/1802.01548, 2018.

[14] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," arXiv preprint arXiv:1605.07079, 2016.

[15] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.