A Distributed Framework for EA-Based NAS

Qing Ye, Yanan Sun, *Member, IEEE*, Jixin Zhang, and Jiancheng Lv, *Member, IEEE*

Abstract—Evolutionary Algorithms (EA) are widely applied in Neural Architecture Search (NAS) and have achieved appealing results. Different EA-based NAS algorithms may utilize different encoding schemes for network representation, while they have the same workflow. Specifically, the first step is the initialization of the population with different encoding schemes, and the second step is the evaluation of the individuals by the fitness function. Then, the EA-based NAS algorithm executes evolution operations, e.g., selection, mutation, and crossover, to eliminate weak individuals and generate more competitive ones. Lastly, evolution continues until the max generation and the best neural architectures will be chosen. Because each individual needs complete training and validation on the target dataset, the EA-based NAS always consumes significant computation and time inevitably, which results in the bottleneck of this approach. To ameliorate this issue, this article proposes a distributed framework to boost the computation of the EA-based NAS algorithm. This framework is a server/worker model where the server distributes individuals requested by the computing nodes and collects the validated individuals and hosts the evolution operations. Meanwhile, the most time-consuming phase (i.e., individual evaluation) of the EA-based NAS is allocated to the computing nodes, which send requests asynchronously to the server and evaluate the fitness values of the individuals. Additionally, a new packet structure of the message delivered in the cluster is designed to encapsulate various network representations and support different EA-based NAS algorithms. We design an EA-based NAS algorithm as a case to investigate the efficiency of the proposed framework. Extensive experiments are performed on an illustrative cluster with different scales, and the results reveal that the framework can achieve a nearly linear reduction of the search time with the increase of the computational nodes. Furthermore, the length of the exchanged messages among the cluster is tiny, which benefits the framework expansion.

Index Terms—Distributed framework, evolutionary algorithm (EA), neural architecture search (NAS), evolutionary neural networks

1 Introduction

Nowadays, Deep Neural Networks (DNNs) have been widely applied in various fields and have made remarkable achievements, such as image classification [1], [2], speech recognition [3], [4] and segmentation [5]. Most of the DNNs are designed manually, while designing a novel DNN for a specific task heavily relies on human experts' knowledge and experience, and may take many trials before achieving meaningful results. Due to the limitation of human knowledge about neural architecture, the best architecture of a neural network to solve a specific problem can be extremely complicated and hard to design. Therefore, Neural Architecture Search (NAS)[6], [7], [8], a technique to automatically discover efficient architectures of DNNs, is gaining ground. NAS aims to generate a robust and well-performing neural architecture by selecting and combining different basic components from a predefined search space adopting the well-designed searching strategies, including Bayesian optimization [9], [10], evolutionary methods [11], [12], reinforcement learning (RL) methods [13], and gradient-based methods [14], [15].

In recent years, the NAS approaches based on the evolutionary algorithm (i.e., EA-based NAS) are in the spotlight because of their high robustness and promising performance [16], [17], [18]. For example, Elsken *et al.* [19] discovered a Convolutional Neural Network (CNN) by using the genetic algorithm [20] to solve large-scale visual recognition, where a binary encoding method was developed to represent the architectures. Liu *et al.* [21] proposed a novel hierarchical genetic representation to fulfill efficient architecture search, where an adjacency matrix G was employed to specify the neural network graph of operations. Wang *et al.* [22] proposed a hybrid differential evolution method to search for the best architectures of CNNs for image classification. Suganuma*et al.* [23] used the Cartesian genetic programming [24] to realize the NAS. Esteban *et al.* [25] proposed the regularized evolution method for CNN architecture, where an age property to favor the younger genotypes had been introduced to the modification of the tournament selection.

Although the experiments of the previous works [25] have shown that the EA-based NAS performs better than some other approaches (e.g., RL-based methods, Bayesian optimization) for searching DNNs with higher accuracy, the searching cost of the EA-based NAS is much expensive. For instance, the Hierarchical NAS method [21] employed 300 GPU Days to search the architecture on CIFAR10. The main reasons resulted in the time-consuming issue of the EA-based NAS are as follows. First, the accuracy of the DNN is always used to model the fitness value (i.e., the accuracy of a DNN in EA-based NAS algorithm) of the individual, which will be accumulated to adjust the search policies, thus a DNN is necessary to evaluate on the target dataset independently. However, the training and validation of a

- *Qing Ye, Yanan Sun, and Jixin Zhang are with the School of Computer Science, Sichuan University, Chengdu 610017, China. E-mail: {fuyeking, jixin.zhang.scu}@gmail.com, ysun@scu.edu.cn.*
- *Jiancheng Lv is with the School of Computer Science, State Key Laboratory of Hydraulics and Mountain River Engineering, Sichuan University, Chengdu 610017, China. E-mail: lvjiancheng@scu.edu.cn.*

Manuscript received 29 June 2020; revised 15 Oct. 2020; accepted 30 Nov. 2020. Date of publication 23 Dec. 2020; date of current version 11 Feb. 2021.
(Corresponding authors: Yanan Sun and Jiancheng Lv.)
Recommended for acceptance by P. Balaji, J. Zhai, and M. Si.
Digital Object Identifier no. 10.1109/TPDS.2020.3046774

DNN will consume a lot of time. In one generation, dozens of individuals need to be validated. Second, several individuals with superior performance are chosen as parents after evaluation, and many new off-springs requiring evaluation will be generated by executing mutation and crossover operations in each new generation. Third, in order to search for a neural architecture with superior performance as much as possible, a large population size and evolution times are always set, which means hundreds of individuals need to be decoded, trained and validated on the target dataset completely. Generally, the EA-based NAS approaches can not avoid the time-consuming issue.

To ameliorate the time-consuming issue of the EA-based NAS methods, the available acceleration methods fall into two main categories in this paper. First, several acceleration approaches for the evaluation of the DNNs are proposed, such as Low fidelity [26], weights sharing [27], and prediction model [28]. However, the bias in the estimate of the DNN performance is introduced because of the reduction of training in these methods. Second, some famous distributed training methods have been developed to boost the training of a large-scale DNN, such as data parallelism and model parallelism [29]. Particularly, the data parallelism refers that the large dataset is divided into multiple different parts with small size, and the DNN is simultaneously trained on multiple different computing nodes with different dataset parts, while the model parallelism means that the DNN model is divided into different small models (e.g., each layer in the DNN may be assigned to a different node) and each computing node performs a part of the model on the entire dataset. However, two distributed methods mainly aim at the training of a single large-scale DNN and the parameter synchronization across multiple nodes also requires a lot of communication overhead. Besides, the distributed training of a DNN may also result in the problem of model inconsistency, which is not suitable for the training of EA-based NAS. In addition, some enterprises and research institutions have also developed distributed training platforms. For example, Uber designed and developed a distributed training platform called Horovod [30], and Vishnu *et al.* proposed the distributed TensorFlow with MPI [31]. However, it is difficult to reproduce these approaches due to the budget-limitation of the high-performance computers and Infiniband networks [32] used in these platforms.

In summary, the time-consuming problem has always been an obstacle to the popularization of the EA-based NAS, while the existed acceleration methods aforementioned are not suitable or can not be employed directly to the EA-based NAS. To alleviate the time-consuming issue plainly, this paper proposes a distributed framework to reduce the training time of the EA-based NAS, which consists of server and computing nodes. To the best of our knowledge, it is the first work focusing on the specified distributed framework for the EA-based NAS. Particularly, the framework consigns the individual evaluation (i.e., the most time-consuming phase of the EA-based NAS) to computing nodes. Meanwhile, the initialization, selection, crossover, and mutation operations are consigned to the server node. All nodes are connected through socket communication and asynchronous working to achieve maximum utilization of the computational resources. The specific contributions are as follows:

1) Employ popular master-slave architecture to build a distributed framework and develop an asynchronous communication scheme from scratch. The server maintains two processes for each computing node to ensure an asynchronous data exchange. The proposed framework can be easily extended according to the computation requirements of a specific algorithm.
2) Create a new packet structure to carry the communication data among the cluster in an efficient way, which contains the representation of DNNs in the EA-based NAS algorithm. Different EA-based NAS algorithms can utilize this framework by encapsulating various network representations into the proposed packet structure.
3) Design an EA-based NAS algorithm named EA-Pelee for image classification as a case to investigate the performance of the proposed framework. Multiple perspective analysis of the experimental results illustrates the efficiency and scalability of the proposed framework.

The following structure of this paper is organized as follows. The related work of the EA-based NAS and acceleration methods are reviewed in Section 2. Section 3 illustrates the details of the proposed distributed framework. The experiment design and the result analysis are documented in Section 4. Finally, the conclusions are drawn in Section 5.

2 Related Work

In this section, the related literature about the EA-based NAS, and other state-of-the-art works focusing on the acceleration related to EA-based NAS are presented briefly.

2.1 EA-Based NAS

The evolutionary algorithm (EA) is a type of population-based algorithm inspired by the biological evolution with a metaheuristic or stochastic optimization character. The EA can produce highly optimized solutions in a wide range of problem settings, which makes it a mature global optimization approach in Neural Architecture Search (NAS). In the EA-based NAS methods, competitive deep neural networks (DNNs) can be designed automatically by adopting genetic operations such as selection, crossover, and mutation. Typically, the EA-based NAS methods comprehensively require three stages: initialization, evaluation, and evolution. The overview of an EA-based NAS is shown in Fig. 1.

The initialization phase consists of two steps: network encoding and generation of the initial population. Encoding schema is used to represent each neural network as an individual which is also called chromosome in the evolutionary algorithms. After the encoding representation, a population of n individuals will be initialized on behalf of the first generation. The population's initialization varies with different algorithms, while each individual can be decoded into a unique neural network.

At the stage of the evaluation, each individual is first decoded into the corresponding DNN and then the DNN is trained and validated on the target datasets

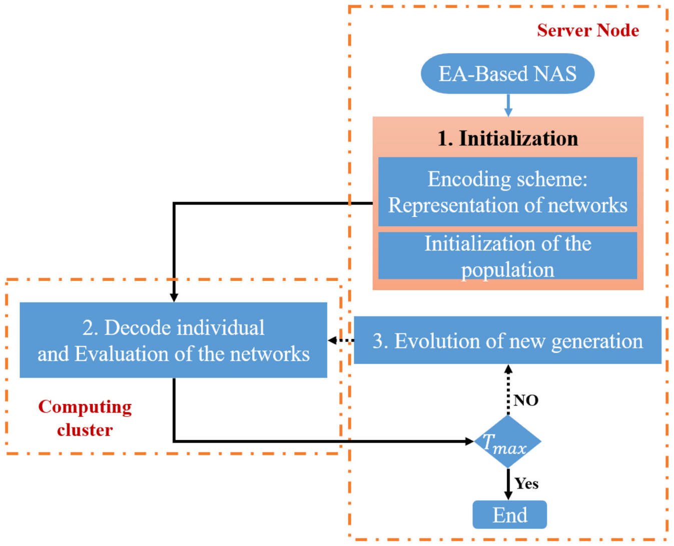
Server Node
EA-Based NAS
1. Initialization
Encoding scheme: Representation of networks
Initialization of the population
2. Decode individual and Evaluation of the networks
3. Evolution of new generation
NO
Computing cluster
T_{max}
Yes
End

Fig. 1. The overall view of the EA-based NAS algorithms. The proposed distributed framework consigns the time-consuming evaluation stage to a computing cluster. Meanwhile, the initialization and evolution of the EA-based NAS will be conducted on the server node. The details of the proposed framework will be specified in Section 3.

completely. Note that it may take hours to evaluate a DNN, and the accuracy of this DNN is usually treated as an individual's fitness value. Particularly, the whole population with fitness values will be exploited to adjust the evolution. It means that more than dozens of individuals in a population need to be evaluated. Obviously, the evaluation stage is a time-consuming period of the EA-based NAS.

In the evolution phase, three main genetic operations will be executed to guide the birth of the next generation: *a) Selection* acts as a force increasing the quality of individuals. The individuals with superior performance are chosen as parents to generate more competitive ones, while weak individuals are eliminated. There are a lot of strategies in how to select an individual, such as random selection, tournament selection [21], and so on. *b) Crossover* ensures a new offspring has inherited the genetic information of both its parents, while recombination mutation creates necessary diversity as well as novelty. *c) Mutation* means the chromosomes of the individual change randomly with a certain probability. In the context of the EA-based NAS, mutations are local operations, such as adding or removing a layer or a block (i.e., a network cell), altering the hyper-parameters of a layer, or training hyper-parameters. A number of new individuals will be generated by adopting these operations in each evolution.

The evaluation and evolution stages loop and iterate until the number of the generations reaches the cap (T_{max}) or the performance of the generated DNNs meets the demand. It is worth mentioning that the simplest way of evaluation is to train a DNN on the training data and evaluate its performance on the validation data. However, training each raw DNN from scratch frequently yields computational demands in the order of hundreds of GPU Days [33]. Furthermore, in order to pursue a DNN model with superior performance, the population size and evolution generations are always set large. Therefore, the time-consuming problem of the EA-based NAS is aggravated inevitably, which is urgent to be settled.

2.2 Acceleration Approaches

The time-consuming drawback of the EA-based NAS algorithm leads to the emergence of several approaches for speeding up the performance evaluation, which is summarized into two main strategies in this paper.

2.2.1 *Acceleration of the Evaluation*

The evaluation of a DNN is an essential phase of the NAS algorithm. Since the training time is highly related to the size of the dataset, the low fidelity methods were proposed to reduce the training time, which trained the DNN on a subset of the original data [34] or on lower-resolution images [26]. However, when the difference between the complete evaluation and the abridged one becomes larger, the bias in the estimate of performance increases too. Another possible way to boost the individual evaluation of the EA-based NAS builds upon early stopping, including employing learning curve extrapolation [35] or supporting performance prediction to control the way of searching DNNs based on cell properties [28]. Particularly, if the model is predicted to perform poorly on the validation dataset, the evaluation operation should be terminated or the individuals with bad performance would be eliminated. The main challenge of the early stopping methods is that the predictions in a relatively large search space need to be made based on relatively scanty evaluations. In addition, different neural architectures can share weights because of the same search space, so that there is no need to train every DNN to converge for a long time. Many techniques of weight sharing were proposed, for instance, Transfer Neural AutoML [36] gains knowledge from prior tasks to accelerate NAS. ENAS [27] shares parameters among child networks and new networks succeed in the weights of previous architectures. However, the drawback of the weight sharing methods is that not all architectures are provided with shareable weights. To shorten the training time of a large-scale DNN, the distributed methods have been proposed to accelerate the training of the DNN, which can be generally divided into two different categories: data parallelism and model parallelism.

2.2.2 *Distributed Deep Learning*

In the data parallelism, every worker hosts a copy of the DNN and trains the DNN on a subset of the training dataset. Then the parameters calculated by each worker need to be synchronized, aggregated, and redistributed by well-designed methods. As shown in Fig. 2, all sub gradients($\bigtriangledown w$) of each worker are collected by the parameter server, and new parameters (w_{i+1}) would be calculated, and then distributed to the workers. This classical realization of the data parallelism is named Synchronous Stochastic Gradient Descent (SSGD) [37], [38], which is simple but inefficient because the workers are forced to wait for the slowest one at every iteration. Particularly, additional network communication and synchronization costs may overwhelm the benefit obtained from the extra workers. Another improved method called asynchronous SGD (ASGD) [39], [40] overcomes this drawback by removing any explicit synchronization amongst the workers. However, this method introduced another

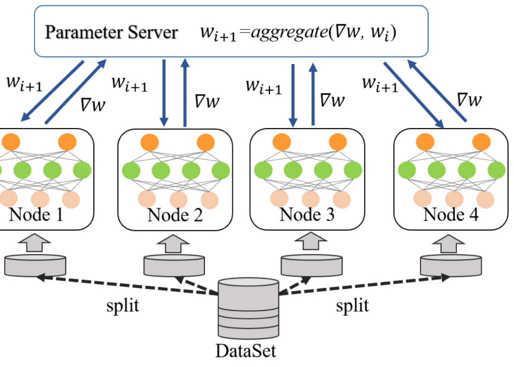

Fig. 2. The illustrative architecture of the data parallelism.

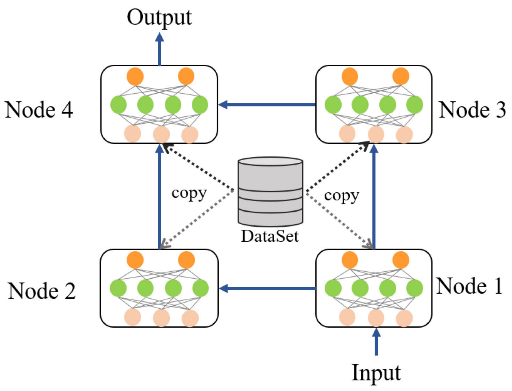

Fig. 3. The illustrative architecture of the model parallelism.

"gradient staleness" problem, which may result in slow convergence speed or even non-convergence of the model. In the model parallelism, each worker hosts a partition of the DNN (e.g., several layers) which can conserve memory (i.e., the full network is not stored in one place), and the dataset needs to be copied to all workers as shown in Fig. 3. Note that parameters need to be transmitted between the adjacent workers. Indeed, the model parallelism is not commonly used because of the difficulty of implementation and the additional communication cost.

Particularly, the popular deep learning platforms, such as Pytorch[1] and Tensorflow,[2] also provide the distributed module [41] including data parallelism and model parallelism. In addition, Horovod [30], a distributed library proposed by Uber Technologies, employs efficient inter-GPU communication via ring reduction and requires only a few lines of modification to user code, enabling faster, easier distributed training in TensorFlow. In these methods, the dataset or DNN model should be split and distributed to different computational nodes. The acceleration efficacy is not guaranteed due to the communication and synchronization overhead among multiple nodes. In addition, several AutoML toolkits such as NNI[3] and Paddleslim,[4] also integrate a few classic NAS algorithms. However, their goal is to provide interfaces for users to implement their NAS algorithms in different environments easily instead of the acceleration of EA-based NAS methods.

1. https://pytorch.org/docs/stable/_modules/torch/distributed.html
2. https://tensorflow.google.cn/
3. https://github.com/microsoft/nni
4. https://github.com/PaddlePaddle/PaddleSlim

TABLE 1
The Summary of Existed Acceleration Methods for the EA-Based NAS

Category	Method	Focus
Model evaluation	Low fidelity [26] Wights sharing [42] Early stopping [43] Surrogate [28]	a single DNN
Distributed training	Data Parallelism [29] Model Parallelism	a large-scale DNN
Distributed platform	Horovod [30], Pytorch ..	a single DNN

To summarize, the acceleration methods aforementioned are summarized in Table 1. Most of the accelerative strategies concentrate on the acceleration of the evaluation of a DNN or the improvement of the algorithm itself. Especially, no distributed framework is dedicated to the EA-based NAS. Fortunately, these ideas motivate us to design a distributed framework to take on the time-consuming bottleneck of the EA-based NAS in a plain way. Specifically, the proposed framework is built based on the master-slave model (i.e., inspired by Parameter Server (PS) [44] model in distributed DNN training), which focuses on boosting the evolution phase of the EA-based NAS algorithms by consigning the evaluation of individuals to an extensible computing cluster as shown in Fig. 1. Simultaneously, the server node is responsible for the initialization and evolution of the EA-based NAS. Note that all computing nodes work asynchronously to maximize the utilization of the computing cluster, which is inspired by the ASGD with data parallelism. Furthermore, this proposed framework is decoupled to specific Evolutionary Algorithms, it means that any EA-based NAS method can consider employing this framework to realize distributed training by adjusting the algorithm flow through interfaces. In addition, the accelerative strategies aforementioned can also be built on this framework to further reduce the training time of the individual evaluation.

3 The Proposed Distributed Framework

In this section, the motivation and implementation of the proposed distributed framework are presented detailly.

3.1 Motivation

The evolutionary algorithms (EA) have been widely employed to implement the Neural Architecture Search (NAS) and achieved appealing results. However, the issue of the great computational resources and time consumed is inevitable, because the evaluation of each individual needs completely validation on the target dataset and the total number of the individual is relatively large. A variety of acceleration methods have emerged, while they only focus on the individual evaluation or the distributed training of a single large-scale DNN model. The fact that the EA-based

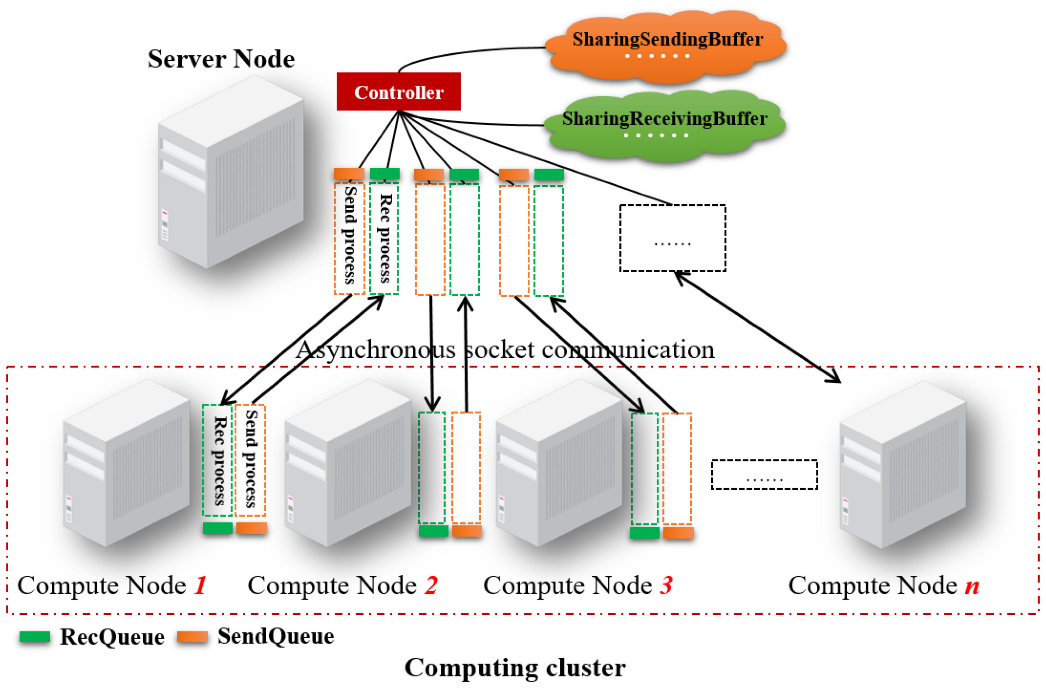

Fig. 4. The overall view of the distributed training architecture.

NAS algorithms are inherently suited to parallelism means that a population with several individuals can be simultaneously evaluated within acceptable time. Steady increase in computing power, including high performance computer with thousands of graphics processing units cores or adding computing nodes, will contribute to speeding up the computing of the EA-based NAS.

Particularly, the time-consuming evaluation of raw individuals in the EA-based NAS can be formalized as a producer-consumer problem [45]. The so-called producer-consumer problem actually comprises two types of workers. One is the producer for tasks, and the other is the consumer for the consumption of tasks. To decouple the relationship between the producers and the consumers, the method of the shared memory (i.e., store the task) is commonly used. The producers only need to put the task into the shared memory, while they do not need to care about the behavior of the consumers. Instead, the consumers only need to get the task from the shared memory and do not need to care about the behavior of the producers. In principle, the more consumers the framework has, the faster the task will be completed. Similarly, the evolution of the EA-based NAS is modeled as the producers, while the evaluation of the individuals is modeled as the consumers. Hence, improving the capacity of the consumers can effectively reduce the time consumed.

In this work, a distributed framework with two types of nodes (i.e., server node, computing node), inspired by the producer-consumer model has been developed to significantly reduce the training time of the EA-based NAS. The evolution of the EA-based NAS algorithm is assigned to the server node (i.e., producers), and the individual evaluation phase that requires the most computational resources is consigned to distributed computing nodes (i.e., consumers). The detailed implementation of the server and computing nodes are presented in Sections 3.4 and 3.5, respectively.

3.2 Framework Overview

The overall architecture of the proposed framework is shown in Fig. 4, which consists of the *server node* and the computing nodes. Specifically, the *server node* is responsible for performing the evolution on the population and the management of the cluster, while a set of computing nodes are responsible for the evaluation of the individuals. Generally, each *computing node* has multiple GPU cards that are suitable for training DNNs. The framework has no special requirements on the performance of the computing nodes, and all computing nodes work asynchronously. Thus it is easy to enhance the cluster by adding computers with different performances. Furthermore, the *computing nodes* and the *server node* communicate through the socket communication interface based on the TCP/IP protocol and the exchanged data are packed by a well-designed data structure, which will be introduced in Section 3.3.

As shown in Fig. 4, the *server node* has a couple of sharing buffers to store the data to be exchanged. The sharing $SendingBuffer$ contains the evolutionary individuals with unknown fitness values of each generation which will be sent to the computing nodes, while the sharing $ReceivingBuffer$ stores the individuals whose fitness values have already been calculated by the *computing nodes* respectively. Each *computing node* establishes a connection with the server node through socket communication. Once the connection is established, the server node and the computing node will create a couple of processes corresponding to each other for communication and transmitting messages. Specifically, a $RecQueue$ will be created to provide storage for the $RecProcess$, while a $SendQueue$ is provided for the $SendProcess$ as shown in the Fig. 4. Particularly, the $RecProcess$ is responsible for receiving all data packets sent by the other side, and pushing the received packets into $RecQueue$. In contrast, the $sendProcess$ takes a data packet from the $SendQueue$ corresponding to this process, then

TABLE 2
The Definition of the New Data Structure

Members	Type	Annotation
operation_type	Integer	**0** represents applying for new data, **1** represents replied processed data
ind_len	Integer	the size of the individuals carried by a message
ind_set	List	store individuals (i.e., an individual represents a network)
termination_flag	Boolean	**True** means the end of evolution

sends it to the other side. Especially, they are only responsible for sending and receiving the packets and all processes are running independently and monitoring whether there are packets in the queues to be processed. In addition, a process created as a $controller$ in the server node completes several tasks such as the post-processing of the received packets, creating reply packets carrying the unevaluated individuals, the evolution of the population, and so on.

3.3 Definition of the Data Packet

The encoding scheme is the key to the EA-based NAS. The EA-based NAS algorithms differ in how they adopt encoding schemes for the network representation. Recently, various types of encoding schemes have been proposed to support the EA-based NAS. For instance, Genetic CNN [46] utilized a binary encoding schema to represent the network architecture, i.e., 1 means that two nodes are connected while 0 means they are separated. Cartesian genetic programming (CGP) [24] employed the directed acyclic graphs to represent the network. In addition, Cellular Encoding (CE) [47] encrypted a family of neural networks into a set of labeled trees, which is based on a simple graph grammar. Recently, some works also have employed an indirect encoding scheme to represent the DNNS [48], [49]. For example, Elske *et al.* [50] introduced the network morphism operators acting on the space of neural networks that preserve the function a network represents, obviating training from scratch and thereby substantially reducing the required training time per network.

To make the proposed distributed framework applicable to various evolutionary algorithms, we develop a new packet structure to encapsulate the different chromosomes (i.e., the network representations based on the various encoding schemes). The different packets will be communicated as messages among the cluster and the details of the packet structure are shown in the Table 2. Specifically, the packets sent by the computing node in this proposed framework are divided into two types. The first type of packet is utilized for requesting data (i.e., *operation_type* = 0), which indicates that the computing node requests data for the first time. The second type of packet is utilized for transmitting the processed data (i.e., *operation_type* = 1), which indicates that all individuals in this type packet have already been evaluated by the computing node. Particularly, the second type packet contains the feedback of the previous individuals as well as the request for new data. The packet as a reply sent by the server node is simple, which only contains several raw individuals to be evaluated. Particularly, if the framework needs to be terminated, the server node will broadcast a packet with $termination\_flag$ set $True$. This new packet structure could in principle be utilized to encapsulate the chromosomes of any evolutionary algorithm for Neural Architecture Search.

3.4 Server Node

The server node is the controller of the distributed framework, which creates a main process to handle the packets from the computing nodes and the evolution of the population. Furthermore, another task of the main process is to manage the entire cluster, including initializing the network connections and broadcasting the termination message. Particularly, all computing nodes initiate requests to the server node asynchronously, i.e., the server node will receive a packet from the computing node at any time and needs to respond immediately. Therefore, the main process in the server node needs to keep running during the whole process of the EA-based NAS algorithms. The workflow of the server node is detailed in Algorithm 3.4.

Algorithm 1. Algorithm Flow of the Server Node

```
1  Wait for all compute nodes to apply for socket connection : *conn_list*;
2  Create a couple of sharing buffers: $SendingBuffer$, $ReceivingBuffer$, both are set to empty;
3  for each conn $i \in conn\_list$ do
4      Create a couple of queues: $SendQueue_i$ and $RecQueue_i$. $//i$ indicates the index of a connection;
5      Create a couple of processes: $SendProces_i$, $RecProcess_i$;
6  end
7  Generation number: $t = 0$. Initial population $P^t : \{p_1^t,\ p_2^t,\ \ldots\ldots,\ p_n^t\}$;
8  Put the $P^t$ into the sharing $SendingBuffer$;
9  for $t = 0, Rec\_size = 0, T_{max}$ do
10     Process the packet from the $RecQueue$ sent by computing nodes;
11     if $operation\_type = 1$ then
12         Take out the evaluated individuals and store in the $RecevingBuffer$;
13         $Rec\_size = Rec\_size + ind\_num$;
14         if $Rec\_size == n$ then
15             Execute evolution operations to generate new population: $P^{t+1}$: selection, mutation, crossover, then repeat step 8;
16             $t = t + 1$;
17             $ReceivingBuffer.clear()$;
18             $Rec\_size = 0$;
19         end
20     end
21     Create a packet with new individuals and push into $SendQueue$;
22 end
23 Notify all nodes to stop evaluation;
24 return the best individuals(i.e., the best performing DNNs);
25 End;
```

At Step 1, the server node listens for all connection requests from the computing nodes, and all established connections for the different computing nodes are stored in $conn\_list$. Then two sharing buffers (i.e., $SendingBuffer$

TABLE 3
Hardware Configuration

Item	Value
OS	Centos7
GPU	GTX1080
Memory	16G
Hard disk	1T

and $ReceivingBuffer$) will be created. For each connection, the server node creates a pair of processes and queues to ensure the reliability of the communication channel for each computing node and runs all the processes at Step 5. Note that a connection corresponds to a computing node. For example, if there are three computing nodes, the server node will create three couples of $SendProcess$ and $RecProcess$. In the server node, all $SendProcess$ share the $SendBuffer$ and all $RecProcess$ share the $ReceivingBuffer$. Especially, the difference between buffer and queue is that the buffer stores individuals while the queue stores packets. Subsequently, the initialization of the population is completed at Step 8. Note that the specific generation of the population varies with the different encoding strategies and the random seeds of different EA-based NAS algorithms. Then, the population $P^t = \{p_1^t, p_2^t, \ldots\ldots, p_i^t\}$ waiting to be evaluated is stored in $SendingBuffer$. After the initialization of the framework, the main process starts to process the received packets at Step 10. Specifically, when the server node takes out a packet from the $RecQueue$, various operations will be performed according to the type of the packet. If the $operation\_type$ of the packet is 0, it is the first time that a computing node applies for a task. The server node will directly respond a packet to the computing node, which contains several individuals (i.e., the number is set in gen_num in the Table 2) to be evaluated. Otherwise, if the $operation\_type$ of the packet is 1, it indicates that the computing node has finished the evaluation of the previous individuals and is waiting for a new task. The server node will collect the evaluated individuals in sharing $ReceivingBuffer$, then reply to the computing node with a packet containing raw individuals to be evaluated. Particularly, the server node will generate a new population P^{t+1} by performing evolutionary operations after the whole population P^t is evaluated (i.e., the size of the evaluated individuals equals the size of the population). Steps from 9 to 22 will be repeated until t reaches the max generation T_{max}. Lastly, a termination message will be broadcasted to stop the evaluation and the best DNN is chosen from the results searched by the EA-based NAS algorithm.

3.5 Computing Node

The main task of the computing node is to complete the evaluation of the individuals and calculate the fitness values, which is the most time-consuming phase of the EA-based NAS. The raw individual will be fetched from the packet sent by the server node and then decoded into a DNN. Then, the DNN will be trained and validated on the target dataset. Subsequently, the fitness value of the individual will be specified as the accuracy of the DNN. The workflow of the computing node is detailed in Algorithm 2. Briefly, Steps from 1 to 3 refer to the initialization of the computing node. Note that $RecPrcess$ and $SendProcess$ are running, which indicates that if a packet arrives or needs to be sent, the processes can take action immediately. Particularly, the packets received by $RecPrcess$ are pushed into the $RecQueue$, while the $SendProcess$ picks a packet from the $SendQueue$ and sends this packet to the server node. Steps from 5 to 10 complete the evaluation of a neural network and a packet carried the evaluated individuals is pushed into $SendQueue$. Then, this packet will be sent to the server node by another running process (i.e., $SendProcess$), which indicates that the previous task has been completed and the computing node is applying for a new task. Specifically, if the $RecQueue$ is $NULL$, it means all tasks have been completed and the computing node is idle, a new request packet will be created and sent at Step 12. The computing node keeps working until it receives a termination notification from the server node (i.e., the $termination\_flag$ of the packet is *true*). Note that during the whole execution of the EA-based NAS method, there is no synchronization required thus all computing nodes are fully occupied.

Algorithm 2. Algorithm Flow of the Computing Node

```
1  Apply to $server\ node$ for a socket connection by $ip$ and $port$;
2  Create $RecQueue$ and $SendQueue$ to provide temp storage
     for $RecProcess$ and $SendProcess$;
3  Create a pair of processes: $SendProces$ and $RecProcess$, both
     run independently;
4  while $termination == False$ do
5      if Take a packet from $RecQueue$ then
6          Fetch the $ind\_set$ from the packet;
7          Decode individuals $p_i^t$ and constructing DNNs;
8          Train and validate the models on the target dataset;
9          Assign the accuracy to the fitness value;
10         Create a packet containing the processed individuals
             (i.e., $ind\_set$) with fitness values, then push it into
             $SendQueue$.;
11     end
12         Initiate task request to the server node //idle;
13     end
14 end
15 End;
```

4 Experiment and Analysis

To verify the efficiency of the distributed framework proposed in this paper, an illustrative cluster is built in our lab environment, which comprises one server node and four computing nodes with the same hardware configurations. Then, several experiments are conducted on this framework. The details of the computers are shown in Table 3. Particularly, the proposed framework is implemented from scratch with Python 3.6,[5] which can be employed to deploy the EA-based NAS building on other deep learning platforms, such as Pytorch,[6] Tensorflow [2]. We also apply the genetic algorithm to design a NAS method named evolutionary Pelee (i.e., EA-Pelee) based on modules of the

5. https://www.python.org/
6. https://pytorch.org/

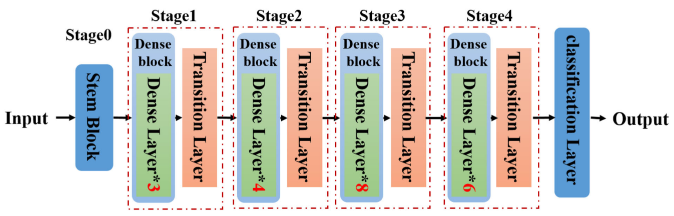

Fig. 5. The overall view of the Pelee architecture.

TABLE 4
The Description of Search Space

Parameter	Value range
Dense way	1, 2
Growth rate	8, 16, 32
The repetition of the dense layer	[1,10]

PeleeNet [51] , which is utilized to search for the optimal neural architecture of CNNs (i.e., an efficient architecture for image recognition on mobile devices with higher accuracy or lightweight). Particularly, the EA-Pelee is treated as an example of the EA-based NAS to investigate the performance of the proposed distributed framework. Several experiments are conducted on this framework with different scales. Specifically, the training time consumed by the EA-based NAS on a single node is treated as the baseline. Lastly, the comparison and multiple perspectives analysis of the training efficiency under different cluster scales are conducted to justify the effectiveness and scalability of the proposed framework. Note that the proposed framework is designed to improve the training efficiency not the accuracy of the EA-based NAS.

Particularly, PeleeNet is an image classification neural network on mobile devices with promising performance, which is manually designed and can be further optimized. An EA-based NAS method called EA-Pelee based on PeleeNet is proposed to search for more efficient Deep Neural Networks(DNN), such as a DNN having fewer parameters while without performance compromised. The EA-Pelee also focuses on the task of Image Classification, which will be detailed in Section 4.1. The CIFAR10 [52] is chosen as the benchmark dataset in the following experiments. The dataset consists of 60,000 color images with the size of 32x32 for 10 different classes. Particularly, each class has 6,000 images including 5,000 training images and 1,000 test images. Most popular EA-based NAS are conducted on this dataset, such as CARS [18]. Then the searched superior architectures are evaluated on larger datasets such as ImageNet [53]. In principle, the performance of the proposed framework is independent of the NAS algorithms and datasets. We only verify the efficiency of this proposed framework on CIFAR10 in an acceptable time with limited computational resources. The EA-Pelee will be trained on the proposed framework with different scales. The experimental results (e.g., consuming time, and DNNs searched by the EA-Pelee) will be accumulated and analyzed. Specifically, in Section 4.2, the consuming time of one computing node is treated as a baseline to compute the speedup of the cluster with the proposed framework, and the consuming time of the state-of-the-art methods is treated as benchmarks to demonstrate the efficiency of the proposed framework in Section 4.5.

4.1 Evolutionary Pelee

The PeleeNet is a variant of the DenseNet [54] architecture running on mobile devices for image classification, which has strict constraints on memory and computational budget. PeleeNet follows the connectivity patterns and the key design principals of the DenseNet and achieves a compelling result. For instance, the accuracy of PeleeNet is higher than the one built with the original DenseNet architecture and MobileNet [55]. Particularly, the PeleeNet is stacked of three important modules:

1) *Dense Layer*: PeleeNet exploited a 2-way dense layer to get different scales of receptive fields. One way uses a 3x3 kernel size. The other way uses two stacked 3x3 convolution. The structure is shown in Fig. 5.
2) *Stem Block*: The stem block can effectively improve the feature expression ability without adding computational cost too much better than other more expensive methods, e.g., increasing channels of the first convolution layer or increasing growth rate.
3) *Transition Layer without Compression*: This module keeps the number of output channels the same as the number of input channels in transition layers.

The entire architecture of the PeleeNet is shown as Fig. 5. The entire network consists of a stem block and four stages of feature extractor, where several dense layers and a transition layer form a stage. The transition layer can adjust the number of output channels. To avoid compression, the number of output channels is the same as the number of input channels in transition layers in PeleeNet. The multi-stage structure is commonly used in the large model design. For example, ShuffleNet [56] exploited a three-stage structure and shrunk the feature map size at the beginning of each stage. Note that the repetitions of the dense layer in four stages are different, the number is manually set as 3, 4, 8, and 6 without explanation, this efficient mobile device network can be further optimized by searching smaller repetitions of the dense layer.

To achieve this, we apply the Genetic Algorithm [20] to search for a more efficient network (i.e., smaller model size or higher classification accuracy) based on the modules of PeleeNet. Specifically, an integer coding scheme is proposed to represent the hyper parameters of the PeleeNet. For each stage, we use three integers to represent the characteristics of the block: one number for the dense way of the dense layer, one number for the repetition of the dense layer, and another number for the increasing growth rate between two adjoin dense layers. Hence the gene of the dense block can be represented as *[dense way, number of the dense layer, growth rate]*. The specific description of the search space is shown in the Table 4. For example, a gene of [2, 6, 8] means that one stage of the network has 6 dense layers, every dense layer has two dense ways and the growth rate between the two dense layers is 8, respectively. To reduce the complexity of network search, the main architecture of the PeleeNet with four stages is retained. Hence the individual can be represented by a chromosome, which consists of 12 integers with four genes. Particularly, different genes have various

TABLE 5
The Consuming Time and Speedup of the Distributed Framework With Different Number of Computational Nodes

EA-based NAS on Proposed Framework	1 node	2 nodes	3 nodes	4 nodes
Total individuals	233	219	236	226
Time(GPU hours)	83.43	44.85	32.7	26.36
Experimental speedup	**1**	**1.86**	**2.55**	**3.25**

parameters and the classification accuracy of a DNN is set as the fitness value of the individual.

4.2 Speedup Analysis

In this section, the parameter setting of the following experiments is presented briefly. We create an initial population with N = 20 individuals, and t_{max} is set as 20 rounds. The probabilities of mutation and crossover for each individual (or pair) are specified as P_M = 0.58 (i.e., mutation probability), P_C = 0.5 (i.e., crossover probability), respectively. The relatively high mutation and crossover probabilities are set to facilitate new structures to be generated. The Russian roulette selection is performed to determine which individuals survive to generate offsprings as the next generation. The experiments with the same setting are conducted on different scale clusters. Each DNN model is trained by Adam optimizer [57], and the loss function is specified to the cross-entropy. The $batchsize$ is set as 256 (i.e., the number of samples selected at one time), and $epochsize$ (i.e., the number of times to train the DNN on the target dataset) is set as 50. The range of the ind_len in a packet defined in Table 2 is $[0, \frac{pop\_size}{n}]$, which will be detailed in Section 4.4. In the experimental configuration, pop_size is 20 and the cluster scale n is 4. Thus, the available value of the ind_len is in [0, 5], which is manually initialized to 3 and then decreases gradually to 1 as a computing node sends a new task request. Particularly, the ind_len could be adjusted according to the population size and cluster scale. The time consumed on different cluster scales is summarized in Table 5, where the first row indicates the EA-based NAS framework with different distributed cluster scales n. Meanwhile, the second row presents the total individuals of the EA-Pelee in different experiments. Moreover, the consumed time and the experimental speedup on various clusters are detailed in the third and fourth rows, respectively. The formula of the speedup is defined as Equation (1):

$$S_p = \frac{T_s}{T_n}. \tag{1}$$

Particularly, S_p represents the speedup. T_s and T_n represent the training time on single node and multi-nodes, and n denotes the cluster scale respectively. In addition, the comparisons between the experimental speedup and theoretical speedup with different cluster scales are shown in Fig. 6.

As can be seen from Table 5, the experimental results reveal that the total evaluated individuals are different because of the randomness of the algorithm, but the training time of EA-Pelee can be effectively reduced based on the proposed distributed framework, and linear efficiency

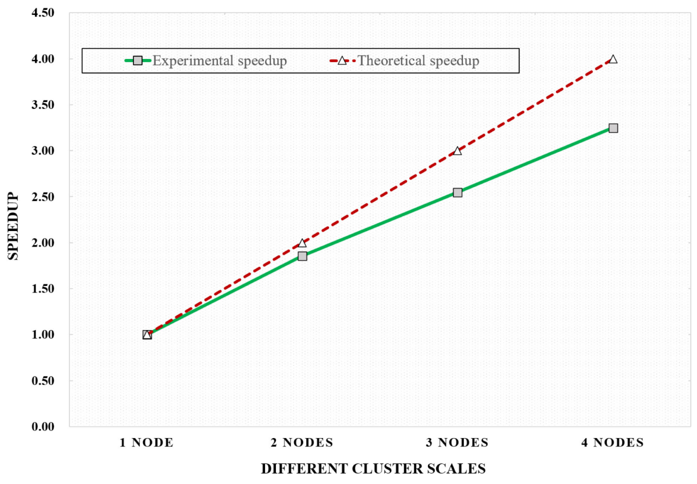

Fig. 6. The comparison of the experimental speedup and the theoretical speedup.

can be obtained (i.e., the green line in the Fig. 6). Moreover, the speedup keeps rising with the increase of the computing nodes employed. Although the theoretical speedup (i.e., the red line in Fig. 6) cannot be achieved and the disparity between the theoretical and the experimental speedup widens with the extension of the cluster, the speedup increases more obvious when the number of computing nodes is relatively small and linear acceleration can be achieved. The main reason is that the cluster communication and evolution cost increase as the cluster expands, and the acceleration will continue to decline. In addition, the usage of mutually exclusive resources (i.e., the sharing queues), and the evolution of each generation of individuals also consume time. Roughly speaking, the additional cost is acceptable compared to the significant acceleration effect of the proposed framework according to the experimental results.

4.3 Analysis of the Distributed Framework With Inconsistent Performance Nodes

As mentioned in Section 3.2, the proposed framework has no performance requirements on the computational nodes. Namely, nodes with different performances can still be employed to support this proposed framework for EA-based NAS. We replace one GPU card of the cluster composed of four same performance GPUs (i.e., GTX1080, 12G of memory) with a low-performance GPU card (i.e., GTX1060, 6G of memory). The parameter setting is the same as Section 4.2. The EA-Pelee is conducted on two different simulative clusters, respectively. The number of evaluated individuals on different nodes are summarized in Fig. 7. Particularly, the number of individuals calculated by the nodes with equal performance is approximately the same, while the number of individuals calculated by weak nodes is relatively small, which is consistent with the performance difference. Refer to the Table 5, the total consumed time of the cluster_2 is *26.36 hours*, while it of the cluster_1 is *29.53 hours* which is also below the consumed time of two (i.e., *44.85 hours*) or three nodes (i.e., *32.7 hours*). It demonstrates that the straggler (e.g., Node1 in cluster_1) does not degrade the whole performance of the cluster, and makes its

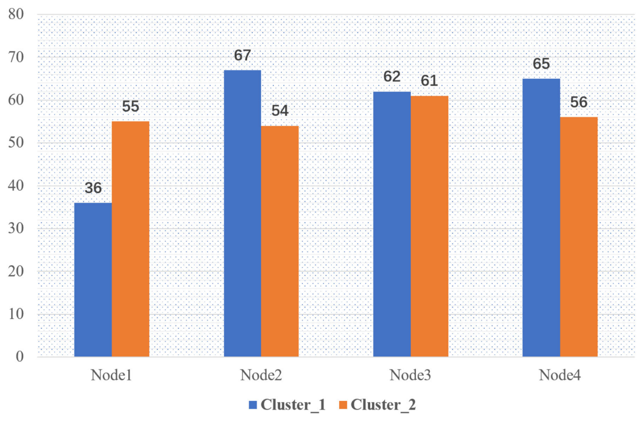

Fig. 7. The comparison of the number of evaluated individuals on different nodes. The *cluster_1* is equipped with three GTX1080 and one GTX1060 (i.e., Node1), while the *cluster_2* is equipped with four GTX1080.

contribution to the distributed EA-based NAS. The proposed framework provides an alternative choice of the distributed EA-based NAS for the research group with limitation of computing resources.

4.4 Communication Analysis

Generally, the network representations of a DNN in the EA-based NAS methods are always integers or some other sample data structure (e.g., binary, tree), so the storage required for the encoding of the DNN is relatively small, which means that the communication cost required for the exchanged packets is small. Specifically, the length of a message distributed in our cluster can be defined as Equation (2):

$$L = l_p + l_{ind} * ind\_len, ind\_len \in \left[0, \frac{pop\_size}{n}\right]. \quad (2)$$

In Equation (2), n is the number of the computing nodes. l_p denotes the length of the proposed data structure, l_{ind} means the length of an individual respectively, and ind_len represents the number of individuals carried by a message at a time, whose minimum value is 0 and the maximum value is $\lceil \frac{pop\_size}{n} \rceil$. Obviously, the whole length L of a message increases with ind_len, because the l_p and l_{ind} are fixed. Particularly, even if the maximum value of the ind_len is obtained, the message size is KB level. In order to decrease the communication rounds and improve the utilization of the cluster, we suggest that the ind_len should not be set as a fixed value, and it can be set as a relatively large value at the beginning. As the number of individuals to be evaluated decreases, the ind_len gradually decreases to 1. Moreover, only two processes need to be created when the cluster expands a computing node, so the total processes in the server are $2n$ and the maintenance cost is affordable. In summary, the overall communication cost is low when the cluster expands, which makes the expansion of the whole cluster flexible.

4.5 Efficiency Analysis

In this section, the searching results of the EA-Pelee are compared with some state-of-the-art methods and the classification accuracy rates on CIFAR10 are summarized in

TABLE 6
The Comparison of the Performances of Different State-of-the-Art Algorithms and Our EA-Pelee on CIFAR 10

Model	Accuracy (%)	GPU Days	Resources GPUs	Parameters (million)
VGG [58]	92.6	-	-	15.2
ResNet [59]	93.39	-	-	1.7
Pelee [51]	94.2	-	-	2.04
MetaQNN [60]	93.08	100	10	-
BlockQNN [61]	96.46	96	-	39.8
Hierarchical [21]	96.37	300	200	15.7
AmoebaNet [25]	96.7	3150	-	3.2
CGP [23]	94.02	30.4	-	2.64
EA-Pelee-A	**94.01**	**5.3**	**4**	**0.94**
EA-Pelee-B	**93.6**	**5.3**	**4**	**0.85**

Note that the comparison of the accuracy and mode size (i.e, parameters) is to demonstrate that the EA-Pelee is a valid case, while the comparison of the GPUDays is to indicate the efficiency of the proposed framework.

Table 6. Then, the efficiency analysis of the proposed framework is stated. Note that the proposed framework focuses on the acceleration of the EA-based NAS not the improving accuracy of an EA-based NAS algorithm. In fact, the proposed framework can not improve the accuracy of the algorithm. The analysis of different EA-based NAS algorithms' performances in terms of the model size and accuracy is only to justify that the special case of the EA-Based NAS (i.e, EA-Pelee) works effectively. Specifically, the first column describes the models created by different methods. The VGG and ResNet are hand-crafted CNN architectures that are designed by human experts, whereas the MetaQNN, and BlockQNN are the models constructed by the RL (reinforcement learning)-based method. Additionally, Hierarchical [21], CGP, and AmoebaNet are EA-based methods. The EA-Pelee-A and EA-Pelee-B are chosen from the results refer to the proposed EA-Pelee. Besides, the details of the training time and resources are set in the third and fourth columns respectively. In particular, GPU Days [16] are defined by $GPUDays = N * D$, where N represents the number of the GPUs, D represents the practical number of the days consumed for the searching. Lastly, the size of the weight parameters of a DNN is provided in the fifth column.

As can be seen in Table 6, the accuracy of the EA-Pelee is competitive with the state-of-the-art methods. In particular, the size of the parameters of the EA-Pelee-A and EA-Pelee-B outperforms all hand-crafted models (e.g., ResNet, VGG) and the DNNs generated by some NAS methods (e.g., MetaQNN, BlockQNN). In addition, the proposed framework takes far less time than other methods at the same level of accuracy, as shown in Table 5. Specifically, the MetaQNN takes 100 GPU Days to achieve 93 percent accuracy, while the EA-Pelee-A with 94.01 percent accuracy just needs 5.3 GPU Days. Although Hierarchical method achieved better accuracy, it has consumed 300 GPU Days. Furthermore, compared to the native Pelee, EA-Pelee-A achieves 50 percent compression of the model with the sacrifice of a little network performance, which is more suitable for running on mobile devices with storage limitation. The EA-Pelee is a valid example of the EA-based NAS. In summary, the EA-Pelee as a case built on the proposed framework could find a competitive DNN with reasonable time and affordable resources, which also states the efficiency of the proposed framework.

According to the experiments and analysis above, the proposed distributed framework can effectively reduce the training time of EA-based NAS compared to the single node and is easy to expend because of the low additional cost.

5 Conclusion

Evolutionary Algorithms (EA) are widely utilized as a kind of important optimization algorithm in Neural Architecture Search (NAS), but the EA-based NAS algorithms consume significant computation and time because of the expensive evaluation of each neural network on the target dataset, which may result in a bottleneck to the wide application of the EA-based NAS. To ameliorate this issue, this paper proposes a special distributed framework for the EA-based NAS. The main contributions are as follows. First, the typical master-slave model is employed in the proposed framework, which consists of the server and computing nodes. Particularly, all computing nodes request tasks asynchronously from the server node without interfering with each other, which can make maximum utilization of the computing resources. Second, a new data structure is designed to encapsulate the individual coding information, which has low communication overhead and can contribute to the flexible expansion of the distributed cluster. Third, an EA-based NAS algorithm is designed as a case to investigate the efficiency of the proposed framework. The experiments show that the proposed framework can effectively reduce the training time of the EA-based NAS and simultaneously possess good expansibility. Since the proposed framework has no excessive requirements on the performance of computational nodes, ordinary computers with different capacities can be unitized to build the distributed framework for EA-based NAS. The proposed framework provides a plain way for the research groups that can not afford the high-performance computing, to pursue the distributed training of the EA-based NAS.

In the future, more efforts will be made to improve the efficiency and scalability of the proposed framework, such as further improvement of speedup on the proposed framework, paralleling the evolution process to multiple nodes instead of one server node.

Acknowledgments

This work was supported by the National Key R&D Program of China under Contract 2017, YFB1002201, in part by the National Natural Science Fund for Distinguished Young Scholar under Grant 61625204, and in part by the Key Program of the National Science Foundation of China under Grant 61836006.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[3] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[4] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *Proc. IEEE Workshop Autom. Speech Recognit. Understanding*, 2015, pp. 167–174.

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, 2020..

[6] H. Chen *et al.*, "Anti-bandit neural architecture search for model defense," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 70–85.

[7] Y. Fu, W. Chen, H. Wang, H. Li, Y. Lin, and Z. Wang, "AutoGAN-Distiller: Searching to compress generative adversarial networks," 2020, *arXiv:2006.08198*.

[8] Y. Li *et al.*, "Neural architecture search for lightweight non-local networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10294–10303. [Online]. Available: http://dx.doi.org/10.1109/cvpr42600.2020.01031

[9] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 115–123. [Online]. Available: http://proceedings.mlr.press/v28/bergstra13.html

[10] H. Zhou, M. Yang, J. Wang, and W. Pan, "BayesNAS: A Bayesian approach for neural architecture search," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 7603–7613.

[11] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Apr. 2020.

[12] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.

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

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

[15] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 7816–7827. [Online]. Available: http://papers.nips.cc/paper/8007-neural-architecture-optimization.pdf

[16] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," 2019, *arXiv: 1908.00709*.

[17] H. Al-Sahaf *et al.*, "A survey on evolutionary machine learning," *J. Roy. Soc. New Zealand*, vol. 49, pp. 205–228, May 2019.

[18] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Aging evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019.

[19] T. Elsken *et al.*, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[20] D. M. Mukhopadhyay, M. O. Balitanas, F. A. Alisherov, S. Jeon, and D. Bhattacharyya, "Genetic algorithm: A tutorial review," *Int. J. Grid Distrib. Comput.*, vol. 2, pp. 25–32, 2013.

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

[22] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *Proc. Australas. Joint Conf. Artif. Intell.*, 2018, pp. 237–250.

[23] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 5369–5373. [Online]. Available: https://doi.org/10.24963/ijcai.2018/755

[24] J. Miller, "Gecco 2013 tutorial: Cartesian genetic programming," pp. 715–740, 2013.

[25] E. Real, A. Aggarwal, Y. Huang, and Q. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4780–4789.

[26] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of ImageNet as an alternative to the CIFAR datasets," 2017, *arXiv: 1707.08819*.

[27] T. Elsken, J. H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," in *Proc. 6th Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1hymrkDf

[28] C. Liu *et al.*, "Progressive neural architecture search," 2017, *arXiv: 1712.00559*.

[29] J. Dean, G. S. Corrado, R. Monga, C. Kai, and A. Y. Ng, "Large scale distributed deep networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.

[30] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," 2018, *arXiv: 1802.05799*.

[31] A. Vishnu, C. Siegel, and J. Daily, "Distributed TensorFlow with MPI," 2016, *arXiv:1603.02339*.

[32] N. S. Islam *et al.*, "High performance RDMA-based design of HDFS over InfiniBand," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–12.

[33] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, vol. 70, pp. 2902–2911.

[34] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian optimization of machine learning hyperparameters on large datasets," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 528–536. [Online]. Available: http://proceedings.mlr.press/v54/klein17a.html

[35] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proc. 24th Int. Conf. Artif. Intell.*, 2015, pp. 3460–3468.

[36] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural AutoML," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 8366–8375.

[37] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of DNNs with natural gradient and parameter averaging," 2014, *arXiv:1410.7455*.

[38] S. Shi *et al.*, "A distributed synchronous SGD algorithm with global top-k sparsification for low bandwidth networks," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 2238–2247.

[39] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware Async-SGD for distributed deep learning," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 2350–2356.

[40] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 3043–3052. [Online]. Available: http://proceedings.mlr.press/v80/lian18a.html

[41] M. Abadi *et al.* , "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 265–283.

[42] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104. [Online]. Available: http://proceedings.mlr.press/v80/pham18a.html

[43] M. Mahsereci, L. Balles, C. Lassner, and P. Hennig, "Early stopping without a validation set," 2017, *arXiv:1703.09580*.

[44] M. Li, D. G. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 19–27.

[45] K. Jeffay, "The real-time producer/consumer paradigm: A paradigm for the construction of efficient, predictable real-time systems," in *Proc. ACM/SIGAPP Symp. Appl. Comput.*, 1993, pp. 796–804.

[46] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1388–1397.

[47] F. Gruau, "Cellular encoding as a graph grammar," in *Proc. IEE Colloq. Grammatical Inference: Theory Appl. Alternatives*, 1993, pp. 17/1–1710.

[48] M. Kim and L. Rigazio, "Deep clustered convolutional kernels," 2015, *arXiv:1503.01824*.

[49] C. Fernando *et al.*, "Convolution by evolution: Differentiable pattern producing networks," in *Proc. Genetic Evol. Comput. Conf.*, 2016, pp. 109–116.

[50] T. Elsken, J. Hendrik Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," 2018, *arXiv: 1804.09081*.

[51] R. J. Wang, X. Li, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 1963–1972. [Online]. Available: http://papers.nips.cc/paper/7466-pelee-a-real-time-object-detection-system-on-mobile-devices.pdf

[52] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Comput. Sci. Dept., Univ. of Toronto, Jan. 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[53] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[54] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.

[55] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv: 1704.04861*.

[56] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recogn.*, 2018, pp. 6848–6856.

[57] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[58] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

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

[61] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2423–2432.

Qing Ye received the BS and MS degrees from the School of Computing, Sichuan University, Chengdu, China. He is currently working toward the PhD degree with the School of Computing, Sichuan University, Chengdu, China. His main research interests include distributed machine learning, deep learning, and neural architecture search.

Yanan Sun (Member, IEEE) received the PhD degree in engineering from the Sichuan University, Chengdu, China, in 2017. He is currently a professor (research) with the College of Computer Science, Sichuan University, China. Prior to that, he was a research fellow with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research topics are evolutionary algorithms, deep learning, and evolutionary deep learning. He is the leading organizer of the First Workshop on Evolutionary Deep Learning, the leading organizer of the Special Session on Evolutionary Deep Learning and Applications in CEC19, and the founding chair of the IEEE CIS Task Force on Evolutionary Deep Learning and Applications.

Jixin Zhang is currently working toward the undergraduate degree in computer science and financial engineering with the School of Computing, Sichuan University, Chengdu, China. Her main research interests include machine learning and data analytic.

Jiancheng Lv (Member, IEEE) received the PhD degree in computer science and engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2006. He was a research fellow with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. He is currently a professor with the Data Intelligence and Computing Art Laboratory, College of Computer Science, Sichuan University, Chengdu, China. His research interests include neural networks, machine learning, and big data.