This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems

EfficientAutoGAN: Predicting the rewards in reinforcement-based neural architecture search for Generative Adversarial Networks

Yi Fan, Xiulian Tang, Guoqiang Zhou, and Jun Shen, *Senior Member, IEEE*

***Abstract*—This paper is inspired by human's memory and recognition process to improve Neural Architecture Search (NAS), which has shown novelty and significance in the design of Generative Adversarial Networks (GAN), but the extremely enormous time consumption for searching GAN architectures based on reinforcement learning (RL) limits its applicability to a great extent. The main reason behind the challenge is that, the performance evaluation of sub-networks during the search process takes too much time. To solve this problem, we propose a new algorithm, EfficientAutoGAN, in which a Graph Convolution Network (GCN) predictor is introduced to predict the performance of sub-networks instead of formally assessing or evaluating them. Experiments show that EfficientAutoGAN saves nearly half of the search time and at the same time, demonstrates comparable overall network performance to the state-of-the-art algorithm, AutoGAN, in the field of RL-based NAS for GAN.**

***Index Terms*—RL-based neural architecture search, generative adversarial networks, cognition embodied, time complexity, performance predictor, graph convolution network**

I. Introduction

IN human's cognitive process, pattern recognition or image processing is different from and often faster than neural network based machines, because intuition plays very important roles. Brains' reflections are often based on prediction or similarity in rewards rather than exactly optimal architecture, counter comparison or finely recalculated hyperparameters. This paper considers on how to mimic this inspiration from human and makes a step-ahead contribution to reduce the huge time consumption problem in the search of network architectures in deep learning networks. Our focus is using predicted network architectures based on rewards concept in reinforced learning to save time on the search of optimal architectures in Generative Adversarial Networks (GAN) [1].

GAN is a type of deep learning neural networks emerging since 2014. Different from the traditional single networks, GAN is composed of two parts, a generator network and a discriminator network. The two parts have opposite objective functions and finally achieve a relative balance through alternating optimization, which can be applied to the production of high-quality images and other generative occasions.

The rationality of network architecture is one of the premises to ensure the acceptable performance of GAN. For this reason, people spend great effort and time in designing and adjusting network architectures. In recent years, with the promising research on Neural Architecture Search (NAS), it has become possible to automatically design a GAN, which can free GAN designers from the heavy labour.

Among various NAS methods, reinforcement learning (RL)-based method is a relatively mature solution [2]–[7], the application of which in GAN has received preliminary results [8], [9]. Within this method, a controller will be built to learn how to select the appropriate architecture in the search space. During the search process, several sub-networks[1] will be sampled and their performance will be validated. The validation results will be used as rewards in reinforcement learning for optimizing the controller.

However, a serious drawback has limited more advanced application of this method. In general, the performance of a GAN is mainly evaluated by the Inception Score (IS) and Frchet Inception Distance (FID) score of the images generated from it. Unfortunately, the time complexity for calculating IS and FID is high owing to additional neural networks [10], [11] involved in this process, which far exceeds the time of evaluating the network with the same size for image classification. Furthermore, in the methods proposed by [8] and [9], the performance testing of a network occupies most of the time in the whole search process, which makes the RL-based NAS for GAN less efficient. For example, figure 1 displays different operations performed in an iteration of AutoGAN [9] (search phase). It can be seen that in the iteration, the time consuming of training GAN only occupies a small part at the beginning. Besides, although the total time of training controller has a large span, the time consuming of each training is very short, corresponding to the thin vertical line in the item 'training controller'. The rest of the time is all spent calculating IS.

To tackle this problem, this article proposes the EfficientAutoGAN algorithm to reduce the search time of GAN while maintaining the quality of search results. In our algorithm,

Y. Fan and X. Tang are with College of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, 210046 China, e-mail: 1018041129, 1018041106@njupt.edu.cn.

G. Zhou is with College of Computer Science, Nanjing University of Posts and Telecommunications and State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210046 China, e-mail: zhougq@njupt.edu.cn.

J. Shen is with the School of Computing and Information Technology, University of Wollongong, NSW 2522, Australia, e-mail: jshen@uow.edu.au.

(Corresponding author: Guoqiang Zhou and Jun Shen).

[1]Because different sub-networks differ mainly in their architecture, we sometimes use the word, architecture, to emphasize sub-network with special architecture. We do not explicitly distinguish between "sub-network" and "architecture" in this paper.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems

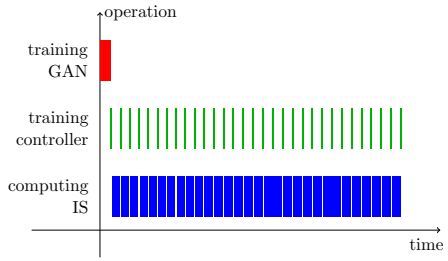


Fig. 1. Different operations performed in an iteration of the AutoGAN algorithm (search phase). Imagine a brush that moves chronologically from left to right, paying attention to what the algorithm is doing constantly, and moving to the corresponding position to paint with a specific color. It can be seen that computing IS takes up most of the time. Note that this figure is just a schematic. In fact, the ratio of the time for training the controller to the time for calculating IS is less than that drawn in the figure. If the figure were drawn with the real ratio, the processes of training controller and the break points between the processes of calculating IS would be too small to be visible.

a predictor is introduced to learn, and predict in essence, the relationship between architecture and performance. At the initial moment of the search, numerous sub-networks are still being sampled and evaluated, but the IS and FID scores obtained by evaluation will be used to train the predictor in addition to playing the role of rewards. In the later period of the search, IS and FID scores of some sub-networks can be predicted by the predictor since it has been trained well to a certain degree.

The benefits of this approach are intuitive and proved by our experiments. On the one hand, since quite a lot of computation tasks for IS and FID scores are replaced by a predictor, which only concerns the relationship between performance and architecture without the weight of networks, the speed of both training and prediction is high. On the other hand, as in [9], all the candidate architectures shares their weights [4], which is believed to affect the selection of the optimal structure in some studies [12]–[14]. Generally speaking, when using the weight-sharing mechanism, the sampled architecture will get more training than other architectures, resulting in the artificially high performance and, furthermore, the artificially high frequency of being sampled, finally falling into the local optimal solution. However, this problem can be significantly moderated by using the predictor to determine the rewards only according to the network architecture itself, instead of the parameters in the network.

The contribution of this paper mainly lies in the following aspects:

- A performance prediction mechanism is introduced in the process of RL-based NAS for GAN, which greatly reduces the time consumption of search.
- A performance predictor appropriate for GAN is designed.
- The comparative experiment proves that the proposed EfficientAutoGAN algorithm can maintain comparable search results and respective performance while reducing the search time.

The organisation of this paper is as follows: Section II summarizes the related work related; Section III reviews the basic knowledge involved in this paper; Section IV introduces the proposed algorithm, EfficientAutoGAN, in detail; Section V verifies the effectiveness and efficiency of this algorithm through experiments and comparative analysis; Section VI concludes the full paper.

II. Related work

A. Generative Adversarial Networks

Generative Adversarial Networks (GAN) was first proposed by Goodfellow in 2014 [1], since then it was widely used to generate realistic images [15]–[20]. In order to improve the quality of images generated by GAN, many researchers adopted various practical methods such as modifying the loss function of discriminator network [21], [22], adding regularization term [16], [19], [23], [24], introducing attention mechanism [17], [25], and conducting progressive training [18]. In order to further improve the performance of GAN, researchers further designed the deep residual generator network to generate high-resolution images [16], [19], [24], [25].

However, the architecture of GAN is artificially designed or adjusted in all the above methods. Among them, some researchers modify the macro framework of the model and directly employ the existing excellent architecture on the micro architecture, such as [26]. The others optimize a specific unit in the network on a micro level [24]. However, it is known that GAN is a system organized by many micro components according to the macro framework. These methods can bring limited improvement to the performance of GAN, but cannot achieve automatic and continuous optimization. In this sense, it is necessary to search GAN architecture on the basis of existing model.

B. Neural Architecture Search

The original idea of Neural Architecture Search (NAS) had been proposed in the 1990s [27], but it was not widely used until 2016 with the significant improvement of computing power. According to the search strategy, the existing NAS methods can be roughly divided into three categories: Reinforcement Learning-based (RL) methods [2]–[7], Evolutionary Algorithm (EA) methods [28]–[30] and gradient-based methods [31]–[35].

In general, there have been abundant research results about NAS [36], [37], but most of them are limited to the task of image classification. In the literature of GAN, the works are relatively few. Among them, [8] and [9] first applied RL-based methods for searching GAN. Then different methods were employed to complete the task of searching GAN, such as gradient-based methods [38] and EA methods [39], [40]. In addition, [41] also searches the architecture of conditional GAN (cGAN) in the process of model compression. From a comprehensive view of these methods, the gradient-based method takes GAN's loss function as the optimization direction, which has a small time consuming, while the complex relationship between GAN's loss function and network performance is easy to lead to unstable search results. The RL-based methods and evolutionary methods decouple the GAN itself and the architecture selection model to achieve

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems

strong expansibility, but the time consumed for evaluating the performance is enormous, which this paper is devoted to solving.

C. Network Performance Prediction

The main task of network performance prediction is to obtain network performance from networks having been trained and then predict the performance of untrained or insufficiently trained networks. [42], [43] and [44] satisfied the requirements of predicting the final performance on the condition that the network is initially trained, while [6] and [45] used the traditional supervised learning method to predict the network performance solely according to the network architecture with a Recurrent Neural Network (RNN).

Different from [6] and [45], [46] and [47] used Graph Convolutional Network (GCN) as performance predictors, and the accuracy was further improved. GCN can effectively model the adjacency relationship between all nodes in a graph, and neural network, as a member of directed acyclic graph (DAG), is also within the scope of application of GCN. Therefore, this paper decided to use GCN as the performance predictor after sufficient discussion.

III. Preliminaries

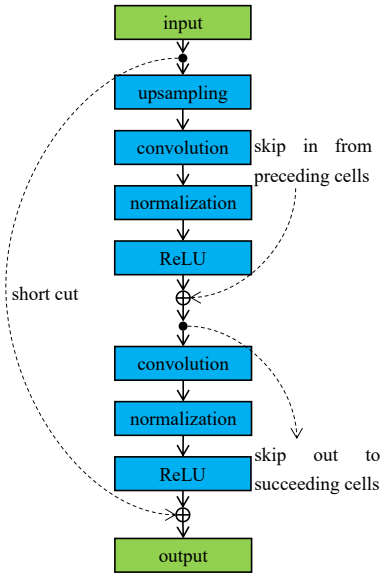
Since AutoGAN [9] is a state-of-the-art RL-based NAS method for GAN, the improvement proposed in this paper will be taking AutoGAN as a baseline benchmark work. This section is a brief review on search space and search strategy of AutoGAN. In reality, our method can be employed in every RL-based NAS method for GAN. However, starting with AutoGAN helps us to explain our method with a solid and richer foundation.

A. Search Space

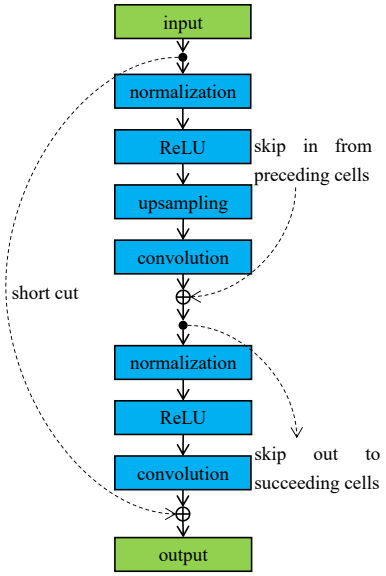
Like most search methods with scalable architectures [3], [4], [6], both the generator network and the discriminator network are composed of several linearly connected cells and the number in both networks are equal. The architecture of the cell is shown in figure 2. It contains five hyper-parameters, which are explained in table I.

Under this search space, the number of optional network architecture in the i-th cell is $2 \times 3 \times 3 \times 2 \times 2^{i-1} = 36 \times 2^{i-1}$. In order to ensure the stability of the search process, the architecture of the discriminator network is fixed in advance and only the architecture of the generator network is searched. So when the target network consists of three cells, the search space is 373,248.

Note that AutoGAN employs the weight-sharing mechanism, in which all the optional networks are integrated into a large network (known as shared GAN), and the common parts of each optional network share the same copy of the parameters during the search. Each time a network is sampled, the corresponding branch is activated and the parameters are optimized, which affects other branches that include this parameter.



(a) Post-activation cell



(b) Pre-activation cell

Fig. 2. The architecture of the cell. The generator is a linear combination of several post-activation cells or pre-activation cells. The path 'short cut' borrows the idea of residual networks [48]. The item 'skip in' starts from the second cell, and its data comes from the 'skip out' in the preceding cell(s).

B. Search Strategy

Similar to the search strategy of [4], AutoGAN uses a controller to learn the policy of sampling sub-networks. In the search process, the shared GAN and the controller are trained alternately.

During the training of shared GAN, the controller outputs the score of each architecture, and the sub-networks will be sampled according to the score, in which the higher the score, the higher the probability of being sampled. The sampled sub-networks are then activated from the shared GAN and trained.

During the training of the controller, there is still a sampling operation similar to that of training shared GAN. But instead of training the sampled sub-networks, we directly calculate the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems

TABLE I
HYPER-PARAMETERS IN THE CELL

name	meaning	possible value
C	the order of convolution and normalization	pre-activation convolution, post-activation convolution
N	the type of normalization	batch normalization, instance normalization, no normalization
U	the type of upsampling	bilinear upsampling, nearest upsampling, deconvolution
SC	whether the short cut between the input and the output exists	yes, no
skip in	the output of which preceding cells adds to the current cell	a list of yes/no, which length equals to the number of preceding cells

IS of the images generated from it. Applying the framework of policy gradient method, IS will be used as rewards for the training of controller. Specifically, suppose the parameters in the controller are θ_c, and the expected performance of the sampled sub-networks are $J(\theta_c)$, then the gradient of $J(\theta_c)$ is

$$\nabla_{\theta_c} J(\theta_c) = \frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_c} \log P\left(a_t | a_{(t-1):1}; \theta_c\right) (R_k - b), \tag{1}$$

where m is the number of sub-networks in an iteration, T is the number of parameters contained in a sub-network, a_i $(i = 1, 2, 3, \cdots, T)$ is the value of the i-th parameter, R_k is the evaluation result of the sub-networks (IS in this scenario), and b is the historical average evaluation result.

In the initial phase of the search, both the generator network and the discriminator network have only one cell. Every time after completing a certain iterations, the shared GAN will be expanded. In the expansion process, several optimal sub-networks, called candidate networks, will be discovered based on the output of the controller and further validation of their performance. Candidate networks will serve as search space for preceding cells in the following search process.

IV. METHOD

In this section the proposed method, EfficientAutoGAN, will be introduced. First we describe the component of the predictor, and then we present the search framework of EfficientAutoGAN.

A. The design of the predictor

The role of the predictor is to learn the relationship between architecture and performance of sub-networks. On the one hand, since there are two nonlinear connections in the generator network, short cut and skip in (see figure 2), its architecture is similar to a directed acyclic graph, in which various operations can be regarded as nodes and the data flow between operations can be regarded as edges. Combined with the existing research results [46], [47], we believe that GCN can better learn the characteristics of sub-network architectures and is more suitable for the predictor in this scenario. On the other hand, as discussed in [9], with the method of progressive search proposed in [6] being used, it is necessary to construct the whole network architecture not from the assignment of the controller only, but also from the candidate networks. For this reason, the information about candidate networks should also be incorporated into the predictor.

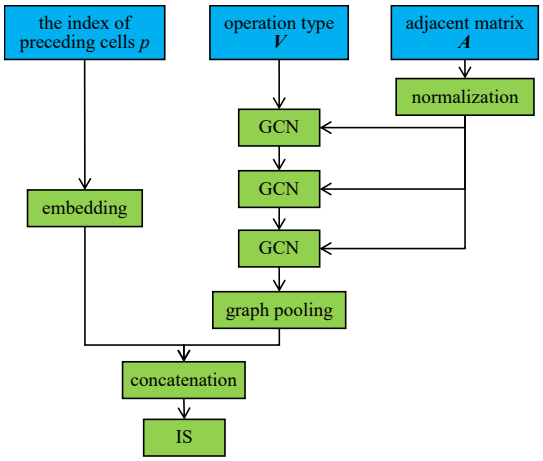


Fig. 3. The predictor used in EfficientAutoGAN. It takes the index of preceding cells $\boldsymbol{p}$, operation type $\boldsymbol{V}$ and adjecent matrix $\boldsymbol{A}$ as input, and the performance of GAN, IS, as output. The purpose of GCN is to combine the information of each operation with its neighbour, while graph pooling is used to achieve the global character of the network. Besides, embedding and normalization can map the label to latent variables. Finally, concatenation combine the components of two vectors by producing $[x_1, x_2, \cdots, x_m, y_1, y_2, \cdots, y_n]$ on condition that $[x_1, x_2, \cdots, x_m]$ and $[y_1, y_2, \cdots, y_n]$ is given.

TABLE II
INDEX OF ALL THE OPERATIONS

index	operation name
0	input
1	batch normalization
2	instance normalization
3	ReLU
4	bilinear upsampling
5	nearest upsampling
6	deconvolution
7	convolution
8	output

The predictor used in this paper is shown in figure 3. The input includes the operation type of each node, $\boldsymbol{V}$, the adjacency matrix of the graph, $\boldsymbol{A}$, and the index of the candidate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems

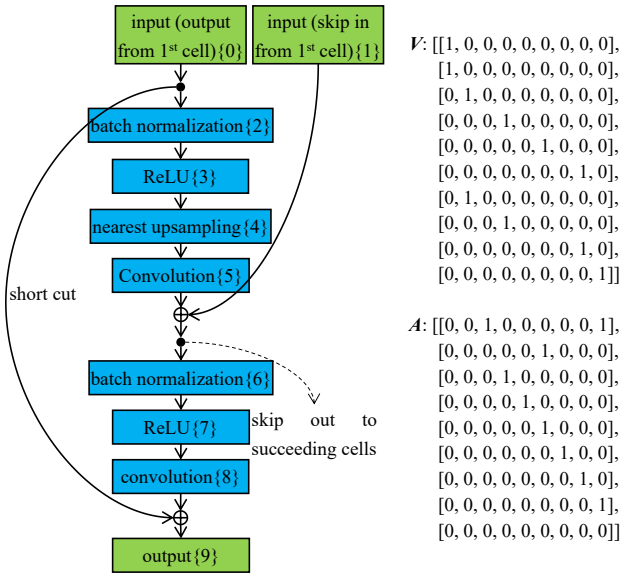


Fig. 4. A cell and its corresponding $\boldsymbol{V}$ and $\boldsymbol{A}$. It can play the role of the second cell in a generator. First, number each node in any order just like the number in braces. Then investigate the operation type of each node, and get $\boldsymbol{V}$. For example, the operation type of node 2 is batch normalization, the index corresponding to which is 1 according to table II. Therefore, for row 2 of $\boldsymbol{V}$ (the number of rows starts from 0), only the element in column 1 is 1, while the rest elements are 0. Finally, the adjacency of nodes in the graph is investigated, and the adjacency matrix is generated and denoted as $\boldsymbol{A}$.

network, p. Here $\boldsymbol{V}$ and $\boldsymbol{A}$ are encoded by the operation and their connection mode in sub-network, which completely describe the sub-network architecture. In the coding process, all nodes and all possible operations are firstly numbered as 1, 2, ..., n and 1, 2, ..., m accordingly to serve as the index. The corresponding relationship between operation index and operation name is shown in table II. The i-th line of $\boldsymbol{V}$ is the one-hot encoding of the i-th node's operation type, and the i-th line, the j-th column of $\boldsymbol{A}$, indicates whether the edge from the i-th node to the j-th node exists. Note that each 'skip in' element of the cell requires a separate input operation. For example, when the third cell of the generator receives skip in from the first and second cells at the same time, the corresponding DAG has three input nodes, corresponding to skip in of the first node, output and skip in of the second node respectively. For the sake of understanding, we demonstrate a cell and its corresponding $\boldsymbol{V}$ and $\boldsymbol{A}$, as shown in figure 4.

As shown in figure 3, our method absorbs the idea of the performance predictor in [45] for reference. The predictor processes the $\boldsymbol{V}$ and $\boldsymbol{A}$, which contain the current cell information, plus p, which contains the information about candidate networks, separately, and then merges them. Because GCN operation does not have the ability to process the embedding variables, we can not merge the information about p at first. Besides, the reason why we do not feed the entire network architecture composed of the current cell and preceding cell(s) into GCN is that with the expansion of shared GAN, the scale of the architecture will grow exponentially. If a shared GAN contains too many cells in the real application scenario, the efficiency of the predictor will be severely restricted.

The architecture of GCN is described below. Since each row of $\boldsymbol{V}$ implies the operation information of a node, if $\boldsymbol{A}$ is left multiplied by it, each row of $\boldsymbol{V}$ will be replaced by the operation information of subsequent nodes. Essentially, with abstraction, $\boldsymbol{AV}$ reflects the forward flow of information in the cell. However, to fully model the architecture of a cell, the reverse information flow, which can be implied by $\boldsymbol{A}^T\boldsymbol{V}$, is also important. Therefore, we use a both-way information flow similar to that in [46] by simultaneously using the term $\boldsymbol{AV}$ and $\boldsymbol{A}^T\boldsymbol{V}$ in GCN. For latent variable $\boldsymbol{V}$ and adjacency matrix $\boldsymbol{A}$, the forward calculation process is

$$\boldsymbol{V}' = \frac{1}{2}\mathrm{ReLU}\left(\boldsymbol{AVW}_1\right) + \frac{1}{2}\mathrm{ReLU}\left(\boldsymbol{A}^T\boldsymbol{VW}_2\right), \quad (2)$$

where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are two independent parameters to be optimized during the training of the predictor, $\boldsymbol{A}^T$ represents the transpose of $\boldsymbol{A}$, and $\boldsymbol{V}'$ is the output latent variable. The two terms in equation (2) are used to simulate the forward and backward information flow of sub-networks respectively.

B. The search framework

For concreteness, we build our algorithm based on AutoGAN in this paper. In order to train the controller, the IS of the sampled sub-networks need to be calculated as rewards. In EfficientAutoGAN, part of the IS are still obtained through validation, but they will also be used to train the predictor so the remaining IS can be given by the predictor.

Suppose that c sub-networks will be sampled in each iteration, and in the i-th iteration, the proportion of the IS obtained through validating sub-networks is η. Since the predictor is continuously trained during the search process and the accuracy of prediction is constantly improved, η should decrease with the increase of i. Here, we set η as exponential decay in each iteration. Specifically, the value η in the i-th iteration is

$$\eta\left(i\right) = \exp\left\{\frac{i}{e_m}\ln\eta_T\right\}, \quad (3)$$

where e_m represents the total epoch in the entire search process, and η_T represents η in the final epoch.

In each epoch, the predictor will conduct both the training and prediction. As sub-networks are trained throughout the search, the IS will increase over time. However, the training samples of the predictor are all from the history, so the predictor has the tendency of underestimating the current IS. Hence, we have to regularize the IS.

The flow of training the controller in EfficientAutoGAN is shown in figure 5. Specifically, it can be divided into several stages. Next we will illustrate it thoroughly.

First of all, sample $[\eta c]$ sub-networks, N_1, N_2, ...,$N_{[\eta c]}$ according to the output of the controller[2]. Then test these sub-networks and get their IS, IS_1, IS_2, ..., $\mathrm{IS}_{[\eta c]}$. Optimize the controller by using these data as rewards.

Secondly, feed N_1, N_2, ..., $N_{[\eta c]}$ into the predictor and get the predictive performance, $\hat{\mathrm{IS}}_1$, $\hat{\mathrm{IS}}_2$, ..., $\hat{\mathrm{IS}}_{[\eta c]}$. Calculate the systematic bias by

$$b = \frac{1}{[\eta c]}\sum_{i=1}^{[\eta c]}\left(\mathrm{IS}_i - \hat{\mathrm{IS}}_i\right), \quad (4)$$

[2]Here N_i includes $\boldsymbol{V}$, $\boldsymbol{A}$ and p.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems

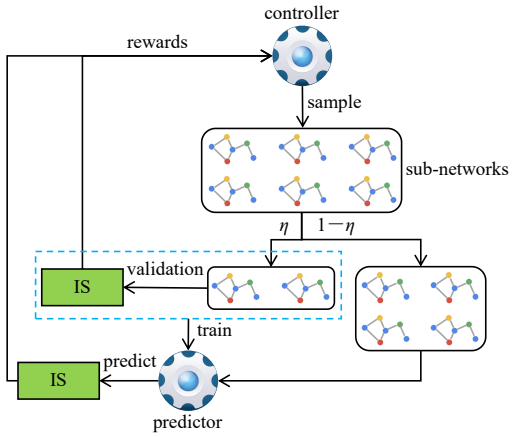


Fig. 5. The training flow of the controller. First, the controller samples a batch of sub-networks according to the current probability distribution. Then the sub-networks are divided into two parts according to a certain proportion. The IS of one part are obtained using standard methods (validating the corresponding GANs), and these sub-networks and their IS are used as training data of the predictor. The IS of the other parts are predicted by the predictor. Finally, all sub-networks and their IS are collected for training the controller.

and then obtain the normalized IS,

$$\mathrm{IS}_i^{\circ} = \mathrm{IS}_i - b, i = 1, 2, \cdots, [\eta c]\,. \tag{5}$$

Afterwards, train the predictor by using (network, performance) pair, $(N_1, \mathrm{IS}_1^{\circ})$, $(N_2, \mathrm{IS}_2^{\circ})$, …, $\left(N_{[\eta c]}, \mathrm{IS}_{[\eta c]}^{\circ}\right)$ as the training data.

Finally, sample $c - [\eta c]$ sub-networks $N_{[\eta c]+1}$, $N_{[\eta c]+2}$, …, N_c again, and predict their performance using the predictor, denoted as $\hat{\mathrm{IS}}_{[\eta c]+1}^{\circ}$, $\hat{\mathrm{IS}}_{[\eta c]+2}^{\circ}$, …, $\hat{\mathrm{IS}}_c^{\circ}$. Then calculate

$$\hat{\mathrm{IS}}_i = \hat{\mathrm{IS}}_i^{\circ} + b, i = [\eta c] + 1, [\eta c] + 2, \cdots, c, \tag{6}$$

and use $\hat{\mathrm{IS}}_{[\eta c]+1}$, $\hat{\mathrm{IS}}_{[\eta c]+2}$, …, $\hat{\mathrm{IS}}_c$ as rewards to optimize the controller.

As a summary, here we list the pseudo-code of EfficientAutoGAN in Algorithm 1.

The search framework is briefly analyzed below. It can be seen that the task of the predictor is to predict the relationship between network performance and network architecture, which can be viewed as building a mapping $\boldsymbol{r} = f(\boldsymbol{a})$ between feedback $\boldsymbol{r}$ and action $\boldsymbol{a}$ from an reinforcement learning perspective. This is actually a value-based reinforcement learning method. Therefore, in essence, **the proposed framework is an ensemble learning model containing policy gradient-based reinforcement learning and value-based reinforcement learning**, which has a higher convergence rate than employing a single learning method. For this reason, we can get the expected results with less training, resulting in less time consumption.

Algorithm 1 The search process of EfficientAutoGAN

Input: shared GAN, controller, predictor, train dataset.
Parameter: the max iteration number iters_m, sampling times in one iteration c
Output: discovered GAN

for iters = 1 **to** iters_m **do**
  train(shared GAN);
  η = get_eta(iters);
  for i = 1 : $[\eta c]$ **do**
    N_i = sample(controller);
    IS_i = evaluate_IS(N_i);
    train(controller, N_i, IS_i);
    $\hat{\mathrm{IS}}_i$ = predict_IS(predictor, N_i);
  end for
  $b = \frac{1}{[\eta c]} \sum_{i=1}^{[\eta c]} \left(\mathrm{IS}_i - \hat{\mathrm{IS}}_i\right)$;
  for iters = 1 **to** $[\eta c]$ **do**
    $\mathrm{IS}_i^{\circ} = \mathrm{IS}_i - b$;
  end for
  train(predictor, $(N_1, \mathrm{IS}_1^{\circ})$, $(N_2, \mathrm{IS}_2^{\circ})$, ..., $(N_{[\eta c]}, \mathrm{IS}_{[\eta c]}^{\circ})$);
  for iters = $[\eta c] + 1$ **to** c **do**
    N_i = sample(controller);
    $\hat{\mathrm{IS}}_i^{\circ}$ = predict_IS(predictor, N_i);
    $\hat{\mathrm{IS}}_i = \hat{\mathrm{IS}}_i^{\circ} + b$;
    train(controller, N_i, $\hat{\mathrm{IS}}_i$);
  end for
end for
discovered GAN = generate_result(controller);
return discovered GAN;

V. Experiment

A. Dataset

The datasets involved in our experiments are CIFAR-10 and STL-10. CIFAR-10 has a total of 60,000 color images with the size of 32×32, which can be divided into 10 categories according to their content. Amongst the 60,000 pictures, 50,000 were used for training and 10,000 for testing. STL-10 has a total of 13,000 color images with the size of 96×96, and they can also be classified into 10 categories, of which 5,000 are used for training and 8,000 for testing. In addition to images with category labels, there are 100,000 images without category information. In order to make a fair comparison with the existing search algorithms [8], [9], we only resized STL-10 here, and did not conduct any other processing on the images.

B. The validation of predictor

In EfficientAutoGAN, the accuracy of the predictor in predicting IS is the key to success. For this reason, we first evaluated the prediction error of the predictor. We conducted a complete search for GAN following the flow in [9], and recorded the IS of each sub-network being tested during the search process. The parameter settings during the search are consistent with [9]. However, we reduced the iterations of the three stages to 15, 20 and 30 respectively to simplify the search process. Then, these IS are used to simulate the predictor's training and prediction behavior during the search process, and the prediction results are compared with the actual IS.

The main feature of our designed predictor is that it employs a GCN network and contains information about candidate networks. In order to prove the appropriateness of the above

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems

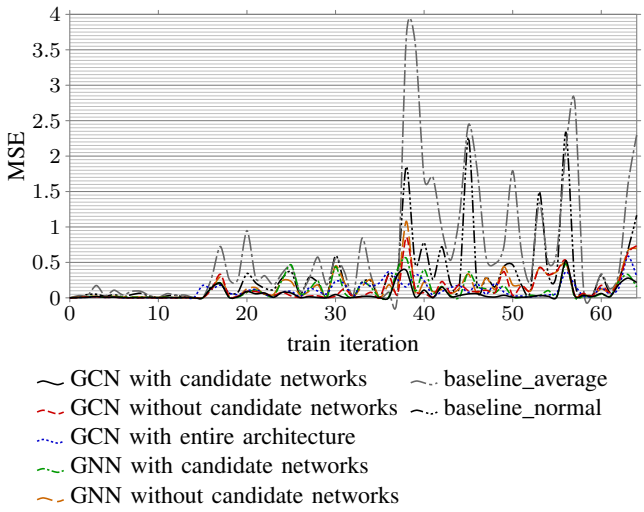


Fig. 6. The MSE of different predictors in the simulated searching process. The smaller the better.

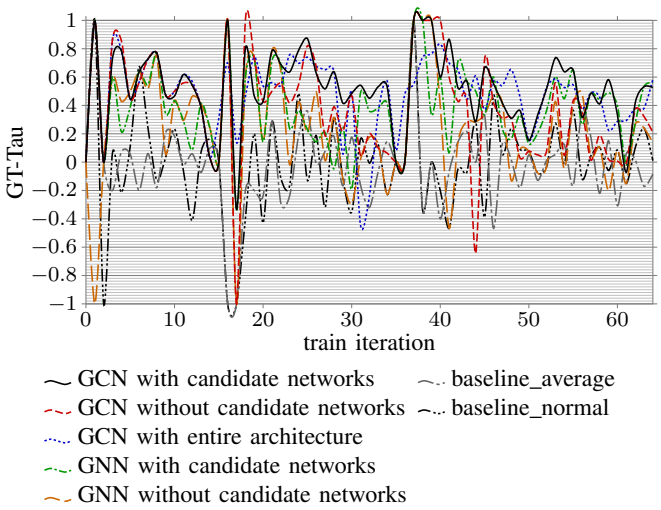


Fig. 7. The GT-Tau of different predictors in the simulated searching process. The value of GT-Tau is between -1 and 1. When the value is closed 1, it represents the predictor can almost rank all the sub-networks correctly, and vice visa.

two features, a comparative experiment is carried out herein first. GCN implies the idea of "convolution", which bridges the gap between spectral-based approaches and spatial-based approaches [49]. In order to verify the necessity of this idea, we also adopted GNN, which is free from the imitation of convolution, as the predictor. For GCN predictor, we set the dimension of the latent variable in the GCN layer to 144, in the linear layer to 128, and in the embed layer to 36. To be fair, the state variables and the latent variables of linear layer in GNN predictor are also set to 144 and 128 respectively. Besides, as mentioned in IV-A, feeding the entire architecture with preceding cells seems to be improper, so we have also investigated it in this experiment. Furthermore, we set up two trivial predictors as the baseline, named baseline_average and baseline_normal respectively. Assuming that in the current iteration the mean and variance of IS of all training data are μ and σ respectively, baseline_average uses μ as for all the prediction results, and baseline_normal generates the prediction results randomly from the normal distribution, $N(\mu, \sigma)$. Every time the predictor is trained, all the training data shaped like the pair (network architecture, IS) will be exploited as a batch to train the predictor for 300 epochs.

In terms of parameter settings, we set the value of η_T to $\frac{1}{3}$. Since the sampling times in each iteration, c, is 30, it can be seen from the equation (3) that the times for testing IS in the three stages are 283, 373, 556, respectively, accounting for 62.9%, 62.2% and 61.8% of all sampling times.

In terms of the selection of evaluation indexes, we adopt Mean Square Error (MSE) and Ground Truth-Tau (GT-Tau) to reflect the theoretical accuracy and practical value respectively. It should be pointed out that in order to observe the variation of accuracy with the iteration, all evaluations are in units of one iteration.

Figure 6 and figure 7 display the MSE and GT-Tau in each iteration. For comparison purposes, we calculate the mean values of MSE and GT-Tau for each predictor respectively, displayed in table III. It can be seen that on the whole the values of MSE and GT-Tau of GCN predictor and GNN predictor are better than those of baseline_average and baseline_normal no matter the candidate networks are integrated into the predictor or not. This proves that all the non-trivial predictors can learn something useful from the training data. Furthermore, from figure 6 and table III, it can be seen that the two predictors with the smallest fluctuations are GCN with candidate networks and GNN with candidate networks. As for figure 7, the curves representing different predictors are intricate and the conclusion seems not so apparent, but it can be found that GCN with candidate networks can maintain a high ranking in most iterations. The only exception was the 16th $\sim$ 17th iterations. However, through further analysis, it can be found that at these iterations shared GAN has just undergone an expansion and it lied in the early period of the second stage. It was when the predictor only predicted very few IS symbolically, so the poor results in the 16th $\sim$ 17th iterations would hardly influence the search result. Last but not least, GCN with candidate networks performs the best of all predictors both in MSE and in GT-Tau. Therefore, obeying the analysis in IV-A, GCN with candidate networks is the most appropriate predictor in this scenario.

TABLE III
AVERAGE PERFORMANCE OF DIFFERENT PREDICTORS

predictor	MSE↓	GT-Tau↑
GCN with candidate networks	**0.056**	**0.511**
GCN without candidate networks	0.121	0.376
GCN with entire architecture	0.106	0.481
GNN with candidate networks	0.107	0.379
GNN without candidate networks	0.157	0.230
baseline_average	0.654	-0.017
baseline_normal	0.320	-0.004

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems

C. Search results

From the section above, it can be seen that using GCN with candidate networks is expected to obtain highly accurate IS and can guide the training of the controller to the maximum extent. Therefore, we will use this network as a predictor to search GAN on CIFAR-10 in this section. All parameter settings, including the dynamic resetting mechanism, are still the same as [9] except that the iterations of the three stages are reduced to 15, 20 and 30 respectively.

We searched on a single GPU, GTX 1080 Ti, which took 57 hours. Meanwhile, AutoGAN with the same parameter settings needs 87 hours on our machine. We did not validate the time consuming of AGAN [8] due to the high time cost, as according to the original research, it took 6 days using 200 Titan X GPUs. Therefore, compared with other RL-based NAS for GAN, efficiency of EfficientAutoGAN is significantly improved.

The search result are shown in figure 8(b). It can be found from figure 8(b) that the architecture is similar to the optimal one of AutoGAN (as shown in figure 8(a)). The main differences are as follows: (a) The second cell uses post-activation rather than pre-activation convolution; (b) In the third cell, deconvolution is used for up-sampling; (c) There is a short cut in the third cell. Among the three differences, (c) is consistent with the analysis in [9] that GAN tends to use nonlinear connections. On the whole, analyzing the search results of EfficientAutoGAN yields similar results to AutoGAN's. This reflects that our search process has achieved the goal similar to AutoGAN.

We retrained the GAN in figure 8(b) for 320 epochs and tested the IS and FID scores of the generated images. In addition, to investigate the transferability of the network, we also trained the network on STL-10 and performed the same test. We also compare the test results with the network obtained by other NAS method for GAN and the manually designed GAN, which are shown in table IV. **Please note that some classical GAN models, such as SinGAN [50] and SAGAN [25], conduct experiments on other datasets or other tasks rather than unconditional image generation tasks on CIFAR-10 and STL-10, so they are not included in table IV.** We listed the random search results given in [38] as the baseline. From table IV, we can see that EfficientAutoGAN's results on CIFAR-10 are better than the baseline and are considered to have achieved comparable performance with the state-of-the art NAS method. It should be noted that there is still a gap between our method and AdversarialNAS-GAN as for performance metrics. Anyhow, as a gradient-based NAS, AdversarialNAS-GAN is beyond the scope of our discussion in this paper. Figure 9 shows some images generated by AutoGAN and EffcientAutoGAN. It can be seen intuitively that the quality of images generated by EffcientAutoGAN is similar to that by AutoGAN. Combined with the satisfying efficiency, EfficientAutoGAN can be considered very practical. **From a broader perspective, these results indicate that it is beneficial to design better GAN architectures by simulating human's cognitive processes and integrating an intuitive model into delicate search methods.**

D. Ablation study

The core idea of the proposed algorithm is to use a predictor to predict some IS to moderate the time complexity problem. But a crucial research challenge is: does the use of predictor really improve search results? In other words, is it possible that simply reducing the times of training controller without using the predicted IS will yield similar search results?

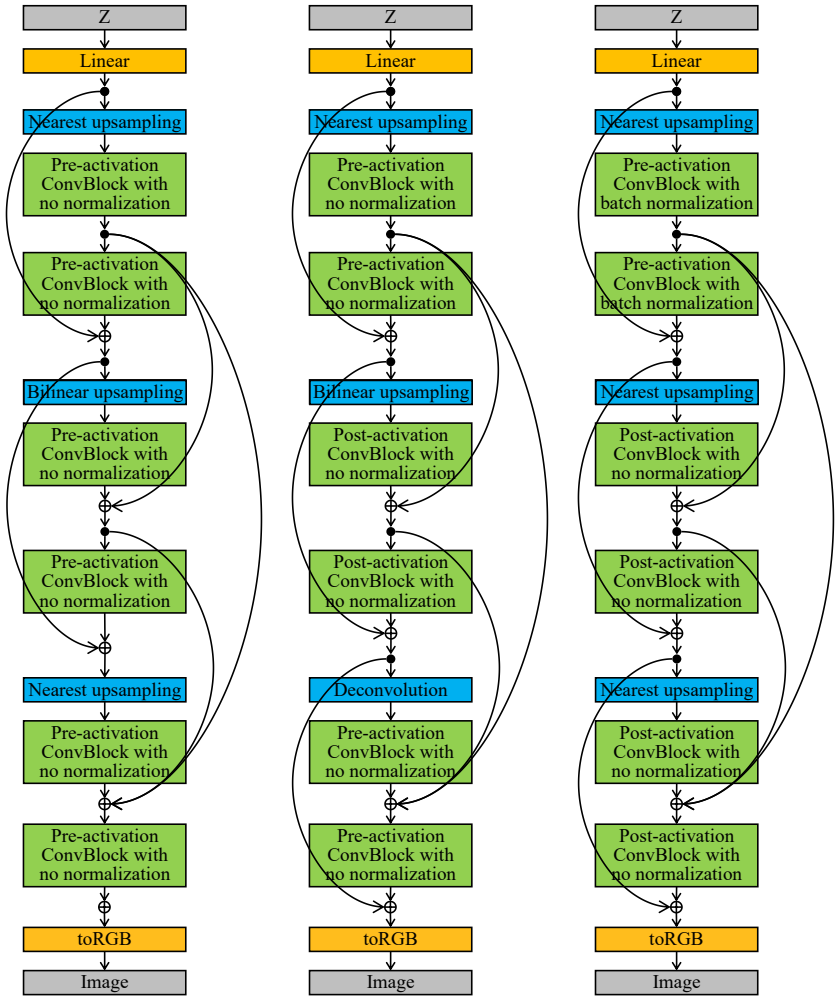
To this end, we carried out a contrast experiment. In this experiment, we removed the predictor and only trained the controller $\eta\,(i)\,c$ times using the conventional method in the i-th iteration. The search result is shown in figure 8(c). Test results show that the generated image has an IS of 8.33 and an FID score of 15.38, which is worse than the counterpart in EfficientAutoGAN. This suggests that using predictors can really improve search results.

E. Discussion

Is the search time measured in the experiment reasonable? Subsection V-C has shown that AutoGAN and EfficientAutoGAN's search times are 84 hours and 54 hours respectively. Here we further explain it from the micro prospective. Additional experiments found that it took an average of 146 seconds to calculate an IS using a single GPU on our machine. In AutoGAN, the whole search process contains 65 iterations, in which 30 IS are calculated, attaching two network expansion and a final determination, in which 10 IS are calculated accordingly, so the time for calculating IS in the search process is $146 \times (30 \times 65 + 10 \times 3) = 289080\text{s} = 80.3\text{h}$. This means 95.6% of the search time are used to calculate IS, and only 3.7 hours are devoted to the training of shared GAN and controller. Applying the results in Subsection V-B, the time to calculate IS is $146 \times (283 + 373 + 556 + 10 \times 3) = 181332\text{s} = 50.4\text{h}$ in EfficientAutoGAN. This does not only demonstrate the rationality of time measured in our experiments, but also indicates to us that the search time is almost linearly related to the times of calculating IS. So, on the premise of ensuring the search quality, reducing the time used in IS calculation is the key to efficient search.

Can we use RNN as the predictor? Different from GCN, a commonly used network for learning network architecture is RNN. So a crucial question is whether we can use RNN as the predictor. To investigate this question, we use LSTM network as the predictor and repeat the experiment in Subsection V-B. Here the state variables and the latent variables of linear layer in LSTM predictor are also set to 144 and 128 respectively. Due to the special characteristic of LSTM network, the indexes of the candidate networks are directly used as elements in the input sequence rather than integrated with other data via embedding. The experimental results show that the average MSE and average GT-Tau are 0.035 and 0.514 respectively, slightly higher than GCN. This seems to suggest that LSTM network is a better predictor than GCN. **However, it should be noted that often controller is also a LSTM network. If LSTM is also used by the predictor, the similarity between the controller and the predictor is high, resulting in that the relationship between GANs' architecture and performance learnt by them is**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems



(a) AutoGAN (b) EfficientAutoGAN (c) Ablation study

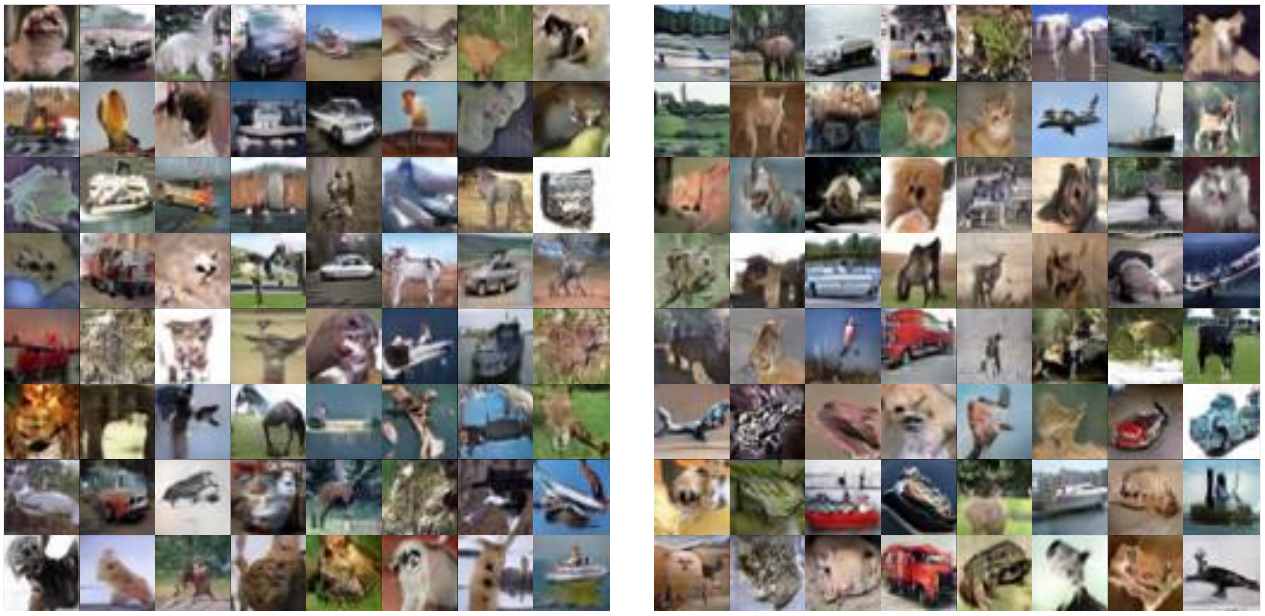
Fig. 8. The discovered architectures of AutoGAN, EfficientAutoGAN (ours) and ablation study (see Subsection V-D).

homogeneous. This is not conducive to multi-perspective mining of the relationship between GANs' architecture and performance. As mentioned in Subsection IV-B, the proposed framework is an ensemble learning model containing policy gradient-based reinforcement learning and value-based reinforcement learning. The key of ensemble learning lies in the cooperation and complementation among multiple weak learners to optimize the overall performance, rather than the optimization of a certain learner. In the proposed method, GCN predictor and LSTM controller play such a role. In this sense, we believe that using the GCN network is a wise choice.

Will the predictor trap into over-fitting? As to predictor, there are two types of over-fitting that may occur in EfficientAutoGAN. Firstly, the predictor over-fitted the training data; Secondly, the predictor and the controller are homogenized, that is, they learn similar content and the controller can replace the predictor. For the first type, our main measurement is controlling the number of epochs in the training process of the predictor. The number of iterations set in Subsection V-B, 300, is an ideal value on this occasion. For the second type, our strategy is to use a heterogeneous architecture as the predictor to learn something different from what the controller learns, which has been discussed in detail in the last question.

Can we expand the search space? The predictor used in this paper treat the sub-network architecture as a directed acyclic graph, which is represented by the operation type $\boldsymbol{V}$ and the adjacency matrix $\boldsymbol{A}$ of the graph. In fact, all the architectures which can be expressed by $\boldsymbol{V}$, $\boldsymbol{A}$ are well beyond the search space. In figure 10 the architecture is just outside the search space. An intuitive question is: can we expand the search space to the range that includes all the architectures, which can be represented by $\boldsymbol{V}$ and $\boldsymbol{A}$? Here, we maintain this expansion would not be justified. On the one hand, our search space continues the search space of AutoGAN. Its design principle derives from some artificially designed GAN [24], which contains certain prior knowledge. On the other hand, since all the training samples of the predictor are generated by evaluating sub-network, when the search space is too large, the distribution of the training samples will be very sparse. This situation is not conducive to the convergence of the predictor under the condition that the search object, GAN, has greater instability. So blindly expanding the search space is not desirable. Based on our analysis, if we want to get richer architectures by expanding the search space, we should

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems



(a) AutoGAN (b) EfficientAutoGAN

Fig. 9. The picture generated by AutoGAN and EfficientAutoGAN (ours). It is hard to see the difference bewteen them visually.

TABLE IV
PERFORMANCE COMPARISON

Method	Type	CIFAR-10		STL-10	
		IS↑	FID↓	IS↑	FID↓
DCGAN [15]	Manual	6.64 ± 0.14	–[1]	–	–
Improved GAN [10]		6.86 ± 0.06	–	–	–
LRGAN [51]		7.17 ± 0.17	–	–	–
DFM [52]		7.72 ± 0.13	–	8.51 ± 0.13	–
ProbGAN [53]		7.75	24.6	8.87 ± 0.09	46.74
WGAN-GP, ResNet [19]		7.86 ± 0.07	–	–	–
Splitting GAN [54]		7.90 ± 0.09	–	–	–
MGAN [55]		8.33 ± 0.10	26.7	–	–
Dist-GAN [56]		–	17.61	–	36.19
Progressive GAN [18]		8.80 ± 0.08	–	–	–
Improving MMD-GAN [57]		8.29	16.21	9.23 ± 0.12	37.64
SN-GAN [24]		8.22 ± 0.05	21.7	9.16 ± 0.05	40.1
BigGANs [16]		**9.22**	14.73	–	–
AGAN [8]	RL	8.29 ± 0.09	30.5	9.23 ± 0.08	52.7
AutoGAN [9]		8.55 ± 0.10	12.42	9.16 ± 0.12	31.01
EfficientAutoGAN **(ours)**		8.43 ± 0.14	13.28	9.06 ± 0.06	34.67
AdversarialNAS-GAN [38] **(beyond the scope of our discussion)**[2]	Gradient	8.74 ± 0.07	**10.87**	$\mathbf{9.63 \pm 0.19}$	**26.98**

[1] For the reliability, we only extract the experimental results in the corresponding paper, and write down "–" if the result had not been measured in the paper. A similar approach was also adopted in [9] [38].

[2] Note that although the performance of AdversarialNAS-GAN is much better than other methods including ours, it is a gradient-based NAS method and beyond our discussion. The predictor is dedicated to solving efficiency problems in RL-based NAS methods.

follow the principle by including more artificial architecture, and avoid the waste of computing resources caused by the unreasonable architecture similar to the network displayed in figure 10.

Does the proposed method have the possibility of failure? In general, any method is not perfect and has its scope of application, especially the heuristic algorithm proposed in this paper, which requires the support of data set and hyperparameters, though failure cases have not been found in our experiment yet. Therefore, we will theoretically analyze

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2020.3040796, IEEE Transactions on Cognitive and Developmental Systems

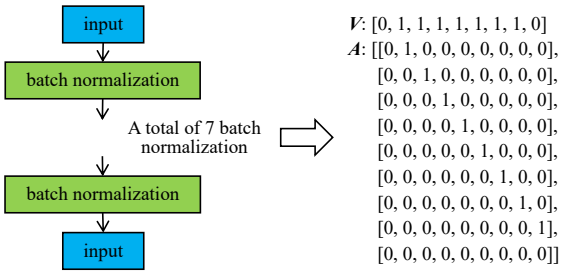


Fig. 10. An improper architecture. The architecture can be represented by the input of GCN, but is obviously a ridiculous one. For simplicity, we use index directly rather than one-hot vector to represent operations in $\boldsymbol{V}$.

under which circumstances our method will lead to training failure. The first case is the network architecture in the search space is too deformed, that is, too much linear architecture and too little 'skip out' and 'short cut'. Since GCN plays the role of predictor, it specialize in learning different topological structures of GANs. When topological structures is no longer the key points of training, the performance of the predictor will be greatly reduced, thus affecting the search results. The second case is there are too many hyperparameters to be optimized. The reason is that the increase of hyperparameters means a more complex relationship between network architecture and performance, resulting in the obstacle for GCN to grasp the relationship between architecture and performance on the condition that the number of training samples remains unchanged, which will also reduce the performance of the predictor. Another case is the capacity of the controller and predictor is much greater than the complexity of the model. The performance of GAN will decrease instead of rise late in the training process due to overfitting.

What does the proposed method bring to us in terms of human cognitive processes? We have mentioned that our method can be seen as a combination of policy gradient-based reinforcement learning and value-based reinforcement learning. In fact, the reason human think faster on some problems is essentially a combination of multiple reasoning processes, and they subconsciously choose the processes that have expectation of producing results faster [58]. This paper simulates the "shortcut" thinking mode of human brain, selects reasonable performance feedback method according to different stages of search, and combines the relatively intuitive method (supervised learning) with the highly accurate method (reinforcement learning). We believe that in future research, the superior thinking mode of human brain needs to be further explored and reasonably applied to accelerate the "thinking" speed of machines.

VI. Conclusion

The existing RL-based NAS methods for GAN are all confronted with the problem of enormous time consumption, which is mainly because the evaluation of sub-networks takes too much time. In order to alleviate this problem, EfficientAutoGAN is proposed in this paper. In this algorithm, we designed a GCN based predictor incorporating the information about candidate networks to learn the relationship between architecture and performance of sub-networks, and to predict the performance of partial networks instead of calculating all IS. Experimental results show that our method can save nearly half of the search time and attain comparable effectiveness with the existing state-of-the-art RL-based method for GAN.

In the future, we will work further on this algorithm from the following aspects:

- The stability of GAN has always been the concern for many researchers. Although this paper does not propose a totally different GAN, but it also faces the problem of stability resulting from involving GAN in terms of its search object. We plan to test the proposed approach on more datasets and more applications, and to see if there are less adaptive situations. The failure cases will be analyzed and then the improvement plan will be put forward pertinently.
- In this algorithm, the proportion of calculating IS by standard algorithm in all sampled sub-networks, η, is prefixed. We will further consider making η adaptive to the current prediction accuracy.
- The training of the predictor in this algorithm still depends on the real IS obtained through testing sub-networks. If the amount of real IS is reduced to a certain extent, the performance of the predictor will be seriously affected. This prevents the further improvement in the efficiency of the algorithm. We will consider various methods to obtain the training data with universal significance, and pre-train the predictor before beginning the search.
- Also, this algorithm represents a class of possible improvements for RL-based NAS for GAN rather than an isolated idea so far, by which AutoGAN algorithm is chosen to implement in this paper. In the future, we will consider applying this approach to other scenarios, for example, when the architecture of the generator network and the discriminator network are searched simultaneously, and when the search object is a conditional GAN.

Appendix
The Configuration in Our Experiments

In order to make a fair comparison of the experimental results, we use the configuration in [9] as much as possible. It is explained in detail below.

Data preparation phase. We used 8 CPU threads to generate a batch of images. There is no image processing operation other than regularization.

The architecture of the discriminator. The architecture of the discriminator is also composed of several cells, the number of which is equal to that in the generator. The architecture of cell is similar to the cells in generator. The main difference is that the upsampling operation is replaced by stride 2 convolution.

Network hyper-parameters. For the generator, the dimensionality of the latent space (input of the generator) is 128, and the channel number of feature map is 128. For discriminator, the channel number of feature map is 64, and spectral normalization is required after convolution. For the

controller, the dimension of hidden vector is 100. As to rewards, the historical average performance accounts for 90%, and the performance of the current network accounts for 10%. Moreover, the performance of the current network includes the entropy output by the controller with a coefficient of 1×10^{-3}.

Search phase. This phase is used to find GAN with excellent performance. The parameters in all the networks are initialized using Xavier uniform. The batch sizes for the generator and discriminator are 128 and 64 respectively. In each iteration, the generator is trained once and the discriminator is trained five times. For the parameter optimization method, Adam algorithm was used for both generator, discriminator and controller. Decay of first order momentum and second order momentum of gradient were 0 and 0.9 respectively. The learning rate of generator and discriminator is both 0.0002, and that of controller is 3.5×10^{-4}. There are a total of 65 iterations in the search, and the three stages contain 15, 20 and 30 iterations accordingly. In an iteration, 15 epochs were trained by the GAN, 30 epochs were trained by the controller and 30 rounds were trained by the predictor. After a stage is completed, the controller samples 10 networks and 5 of them are selected for the next round through performance evaluation. In order to prevent model collapse, when the loss of the generated network changes less than 1×10^{-3} within latest 500 updates of the parameters, the GAN will be dynamically reset.

Training and evaluation phase. In this phase, GAN found in the search phase will be trained from scratch and its performance will be tested. Instead of specifying the training epoch, we specify the total iteration to be 50000, so the epoch will depend on the size of the dataset. For example, GAN will be trained 320 epochs when working on CIFAR-10 and 153 on STL-10. The batch size, training times in each iteration, and parameter optimization methods of the generator and discriminator are consistent with the search phase. After every 20 rounds of training, the network was evaluated by calculating IS and FID in batches of 100 images.

ACKNOWLEDGMENT

The authors are grateful to the College of Computer Science, Nanjing University of Posts and Telecommunications for kindly providing the required computational resources. We also thank Baofeng Zhang for his help with this project. This work was supported by NSF of China 2019-2022 (grants 61877051 and 61872079). Associate Professor Jun Shen was also supported by research exchange program funded University Global Partnership Network.

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[2] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[3] 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*, 2018, pp. 8697–8710.

[4] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *ICML*, 2018.

[5] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2423–2432.

[6] 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)*, 2018, pp. 19–34.

[7] J. Perezrua, M. Baccouche, and S. Pateux, "Efficient progressive neural architecture search." *British Machine Vision Conference* p. 150, 2018.

[8] H. Wang and J. Huan, "Agan: Towards automated design of generative adversarial networks," *arXiv preprint arXiv:1906.11080*, 2019.

[9] X. Gong, S. Chang, Y. Jiang, and Z. Wang, "Autogan: Neural architecture search for generative adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3224–3234.

[10] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems (NeurPIS)*, 2016, pp. 2234–2242.

[11] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems*, 2017, pp. 6626–6637.

[12] Y. Zhang, Z. Lin, J. Jiang, Q. Zhang, Y. Wang, H. Xue, C. Zhang, and Y. Yang, "Deeper insights into weight sharing in neural architecture search," *arXiv preprint arXiv:2001.01431*, 2020.

[13] A. Pourchot, A. Ducarouge, and O. Sigaud, "To share or not to share: A comprehensive appraisal of weight-sharing," *arXiv preprint arXiv:2002.04289*, 2020.

[14] S. Niu, J. Wu, Y. Zhang, Y. Guo, P. Zhao, J. Huang, and M. Tan, "Disturbance-immune weight sharing for neural architecture search," *arXiv preprint arXiv:2003.13089*, 2020.

[15] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks,", *https://arxiv.org/abs/1511.06434* 2016.

[16] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," in *International Conference on Learning Representations*, 2019.

[17] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He, "Attngan: Fine-grained text to image generation with attentional generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1316–1324.

[18] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," in *International Conference on Learning Representations*, 2018.

[19] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, 2017, pp. 5767–5777.

[20] X. Zhang, Z. Wang, D. Liu, and Q. Ling, "Dada: Deep adversarial data augmentation for extremely low data regime classification," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 2807–2811.

[21] S. Martin Arjovsky and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia*, 2017.

[22] J. Zhao, M. Mathieu, and Y. Lecun, "Energy-based generative adversarial network," in *International Conference on Learning Representations (ICLR)*, 2017.

[23] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Neural photo editing with introspective adversarial networks," in *International Conference on Learning Representations*, 2017.

[24] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *International Conference on Learning Representations*, 2018.

[25] H. Zhang, I. Goodfellow, D. N. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *International Conference on Machine Learning* pp. 7354–7363, 2019.

[26] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional gans," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8798–8807.

[27] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE, 1992, pp. 1–37.

[28] 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*. JMLR. org, 2017, pp. 2902–2911.

[29] 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, 2019, pp. 4780–4789.

[30] D. So, Q. V. Le, and C. Liang, "The evolved transformer," in *International Conference on Learning Representations*, 2019, pp. 5877–5886.

[31] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[32] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 7816–7827.

[33] X. Zhang, Z. Huang, and N. Wang, "You only search once: Single shot neural architecture search via direct sparse optimization," *arXiv preprint arXiv:1811.01567*, 2018.

[34] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," in *International Conference on Learning Representations*, 2019.

[35] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *ICLR, https://arxiv.org/abs/1812.00332* 2019.

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

[37] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," *arXiv preprint arXiv:1905.01392*, 2019.

[38] C. Gao, Y. Chen, S. Liu, Z. Tan, and S. Yan, "Adversarialnas: Adversarial neural architecture search for gans," *arXiv preprint arXiv:1912.02037*, 2019.

[39] V. Costa, N. Lourenço, and P. Machado, "Coevolution of generative adversarial networks," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 2019, pp. 473–487.

[40] V. Costa, N. Lourenço, J. Correia, and P. Machado, "Coegan: evaluating the coevolution effect in generative adversarial networks," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 374–382.

[41] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "Gan compression: Efficient architectures for interactive conditional gans," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5284–5294.

[42] 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.

[43] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Practical neural network performance prediction for early stopping," *arXiv preprint arXiv:1705.10823*, vol. 2, no. 3, p. 6, 2017.

[44] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," in *International Conference on Learning Representations*, 2017.

[45] Z. Zhong, Z. Yang, B. Deng, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Blockqnn: Efficient block-wise neural network architecture generation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[46] W. Wen, H. Liu, H. Li, Y. Chen, G. Bender, and P.-J. Kindermans, "Neural predictor for neural architecture search," *arXiv preprint arXiv:1912.00848*, 2019.

[47] C. Wei, C. Niu, Y. Tang, and J. Liang, "Npenas: Neural predictor guided evolution for neural architecture search," *arXiv preprint arXiv:2003.12857*, 2020.

[48] 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*, 2016, pp. 770–778.

[49] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[50] T. R. Shaham, T. Dekel, and T. Michaeli, "Singan: Learning a generative model from a single natural image," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 4570–4580.

[51] J. Yang, A. Kannan, D. Batra, and D. Parikh, "Lr-gan: Layered recursive generative adversarial networks for image generation," in *International Conference on Learning Representations*, 2017.

[52] D. Wardefarley and Y. Bengio, "Improving generative adversarial networks with denoising feature matching," in *International Conference on Learning Representations*, 2017.

[53] H. He, H. Wang, G. Lee, and Y. Tian, "Probgan: Towards probabilistic gan with theoretical guarantees," in *International Conference on Learning Representations*, 2019.

[54] G. L. Grinblat, L. C. Uzal, and P. M. Granitto, "Class-splitting generative adversarial networks," *arXiv preprint arXiv:1709.07359*, 2017.

[55] Q. Hoang, T. D. Nguyen, T. Le, and D. Phung, "Mgan: Training generative adversarial nets with multiple generators," in *International Conference on Learning Representations*, 2018.

[56] N.-T. Tran, T.-A. Bui, and N.-M. Cheung, "Dist-gan: An improved gan using distance constraints," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 370–385.

[57] W. Wang, Y. Sun, and S. K. Halgamuge, "Improving mmd-gan training with repulsive loss function," in *International Conference on Learning Representations*, 2019.

[58] Tibbetts and Paul, "Evolution of nervous systems, vol 3, the nervous systems of non-human primates," *Quarterly Review of Biology*, 2017.

Yi Fan is a research candidate with College of Computer Science, Nanjing University of Posts and Telecommunications. His main research interests include machine learning, recommendation system and computer vision.

Xiulian Tang is a research candidate with College of Computer Science, Nanjing University of Posts and Telecommunications. Her main research interests include machine learning, natural language processing and privacy preservation.

Guoqiang Zhou is an Associate Professor with College of Computer Science, Nanjing University of Posts and Telecommunications. He is a member of CCF. His main research interests include data mining, machine learning, recommendation system and information security. A/Prof Zhou has published extensively in the above areas and advised 10-plus research students to completion.

Jun Shen (M'98, SM'06) was awarded Ph.D. in 2001 at Southeast University, China. He held positions at Swinburne University of Technology in Melbourne and University of South Australia in Adelaide before 2006. He is an Associate Professor in School of Computing and Information Technology at University of Wollongong in Wollongong, NSW of Australia, where he had been Head of Postgraduate Studies, and Chair of School Research Committee since 2014. He is a senior member of three institutions: IEEE, ACM and ACS. He has published more than 200 papers in journals and conferences in CS/IT areas. His expertise includes computational intelligence, bioinformatics, Cloud computing and learning technologies including MOOC. He has been Editor, PC Chair, Guest Editor, PC Member for numerous journals and conferences published by IEEE, ACM, Elsevier and Springer. A/Prof Shen was also member of ACM/AIS Task Force on Curriculum MSIS 2016. His publications appeared at IEEE Transactions on Learning Technologies, Briefs in Bioinformatics, International Journal of Information Management and many others.