EAGAN: Efficient Two-stage Evolutionary Architecture Search for GANs

Guohao Ying†[2], Xin He†[1], Bin Gao[3], Bo Han[1], and Xiaowen Chu‡[4,1,5]*

[1] Hong Kong Baptist University, Hong Kong SAR, China
[2] University of Southern California, USA
[3] National University of Singapore, Singapore
[4] The Hong Kong University of Science and Technology (Guangzhou), China
[5] The Hong Kong University of Science and Technology, Hong Kong SAR, China

Abstract. Generative adversarial networks (GANs) have proven successful in image generation tasks. However, GAN training is inherently unstable. Although many works try to stabilize it by manually modifying GAN architecture, it requires much expertise. Neural architecture search (NAS) has become an attractive solution to search GANs automatically. The early NAS-GANs search only generators to reduce search complexity but lead to a sub-optimal GAN. Some recent works try to search both generator (G) and discriminator (D), but they suffer from the instability of GAN training. To alleviate the instability, we propose an efficient two-stage evolutionary algorithm-based NAS framework to search GANs, namely **EAGAN**. We decouple the search of G and D into two stages, where stage-1 searches G with a fixed D and adopts the many-to-one training strategy, and stage-2 searches D with the optimal G found in stage-1 and adopts the one-to-one training and weight-resetting strategies to enhance the stability of GAN training. Both stages use the non-dominated sorting method to produce Pareto-front architectures under multiple objectives (e.g., model size, Inception Score (IS), and Fréchet Inception Distance (FID)). EAGAN is applied to the unconditional image generation task and can efficiently finish the search on the CIFAR-10 dataset in 1.2 GPU days. Our searched GANs achieve competitive results (IS=8.81±0.10, FID=9.91) on the CIFAR-10 dataset and surpass prior NAS-GANs on the STL-10 dataset (IS=10.44±0.087, FID=22.18). Source code: https://github.com/marsggbo/EAGAN.

1 Introduction

Generative adversarial networks (GANs) [11] have obtained remarkable achievements on image generation tasks. A GAN consists of two networks (i.e., generator (G) and discriminator (D)) that contest with each other in a zero-sum game. G learns to generate semantic images from real data distributions, while D distinguishes real data from generated data. Since G and D have conflicting optimization objectives, GAN training is unstable and prone to collapse. Therefore, many

* †: Equal contributions. ‡: Corresponding author (xwchu@ust.hk).

efforts have been made to manually enhance architectures of GANs [29,3], but this requires much professional knowledge. Recently, neural architecture search (NAS) has proven to be effective in automatically finding superior models in various tasks [8,14], including GANs. The early NAS-GAN works [10,35] search only generator with a fixed discriminator to reduce search difficulty, but this may lead to a sub-optimal GAN. Although some recent works have searched both G and D, they suffer from the instability of GAN training. For example, AdversarialNAS [9], which is the first gradient-based NAS-GAN, proposes an adversarial loss function to search G and D simultaneously, but the architectures of G and D are deeply coupled, which increases search complexity and the instability of GAN training. A subsequent gradient-based NAS-GAN work [32] also demonstrates that simultaneously searching both G and D hampers the search of optimal GANs. DGGAN [25] alleviates instability by progressively growing G and D but takes 580 GPU days to search on the CIFAR-10 dataset [20].

In this paper, we propose an efficient two-stage **E**volutionary **A**rchitecture search framework for **G**enerative **A**dversarial **N**etworks (**EAGAN**) on the unconditional image generation task. First, to alleviate the instability of GAN training during the search, we decouple the search of G and D into two stages. In stage-1, we fix the architecture of discriminator and search only generators. All generators are paired with the same discriminator, i.e., the candidate generators and the fixed discriminator are in a *many-to-one* relationship. In stage-2, the best generator of stage-1 is used to provide supervision signals for searching discriminators. Specifically, in stage-2, we create multiple copies of the best generator architecture of stage-1, and each generator copy is paired with a different discriminator and trained independently. Thus, the generators and candidate discriminators of stage-2 are in a *one-to-one* relationship. Because we indirectly evaluate the discriminators of stage-2 via IS (Inception Score [31]) and FID (Fréchet Inception Distance [15]) based on generators, the one-to-one strategy has a potential problem, i.e., if some generators have mode collapse at some time, then subsequently searched discriminators paired with these generators will be evaluated unfairly. To solve this problem, we propose the *weight-resetting* strategy, where all generators inherit the weights of the best generator of the previous search round before a new search round starts. The results in Sec. 5.3 show that our simple yet effective weight-resetting strategy can stabilize GAN searching. We summarize our contributions as follows.

1. We greatly reduce the instability of GAN training by decoupling the search of generator and discriminator into two stages, where stage-1 and stage-2 adopt the *many-to-one* and *one-to-one* training strategy, respectively.
2. We propose the *weight-resetting* strategy, which is simple yet effective to avoid mode collapse when searching discriminators in stage-2 and ensure fair evaluations of different discriminators.
3. EAGAN is efficient and takes 1.2 GPU days on the CIFAR-10 dataset to finish searching GANs. EAGAN achieves competitive results on the CIFAR-10 dataset and outperforms the prior NAS-GANs on the STL-10 dataset [4].

2 Related Work

2.1 Generative Adversarial Network (GAN)

Generative Adversarial Networks (GANs) are first proposed in [11] and have been widely used in the various generation and synthesis tasks. A GAN comprises a generator (G) that generates plausible new data and a discriminator (D) that distinguishes the generator's fake data from real data. Suppose D and G are parameterized by θ and ϕ, respectively, their loss functions are defined as

$$L^D(\phi,\theta) = -E_{x\sim p_{data}(x)}[\log D_\theta(x)] - E_{z\sim p(z)}[\log(1 - D_\theta(G_\phi(z)))] \quad (1)$$

$$L^G(\phi,\theta) = E_{z\sim p(z)}[\log(1 - D_\theta(G_\phi(z)))] \quad (2)$$

where p_{data} is the real data distribution and p_z is a prior distribution. In other words, G and D play a min-max game with value function V, formulated below

$$\min_G \max_D V(G,D) = E_{x\sim p_{\text{data}}}[\log D(x)] + E_{z\sim p_z}[\log(1 - D(G(z)))] \quad (3)$$

The mix-max optimization incurs that GAN training suffers from multiple instability issues, such as mode collapse and gradient vanishing. To alleviate these problems, many efforts have been made [2] from the perspective of loss functions [1,36,16], normalization and constraint [12,26], conditional techniques [27,18], and validation methods [31,15]. Besides, architecture enhancements have been proven effective to improve GANs performance in many works [29,3,17].

2.2 Neural Architecture Search (NAS)

NAS aims at automatic architecture design and has achieved remarkable results in various fields [8,14]. It can be formulated as a bilevel optimization problem as below

$$\begin{aligned} \alpha^* &= \arg\min_\alpha L_{\text{val}}\ (\alpha|w^*) \\ \text{s.t.}\ \ w^* &= \arg\min_w L_{\text{train}}\ (w|\alpha) \end{aligned} \quad (4)$$

where L_{train} and L_{val} indicate the training and validation loss; w and α indicate the weight and architecture of neural network. This process aims to select the architecture α^* performing best on the validation set, conditioned on the optimal network weights w on the training set. There are mainly four approaches in NAS: 1) Reinforcement learning (RL) [39,28] based methods train an RNN controller to generate neural networks; 2) Gradient-based methods [24] apply softmax function to relax the discrete search space, allowing differential optimization of architectures; 3) Surrogate model-based optimization (SMBO) [23] builds a surrogate model of the objective function to predict the searched model's performance, which can substantially improve search efficiency; 4) Evolutionary algorithm (EA) based methods [30,38] maintain and evolve a large population of neural architectures to produce the Pareto-front architectures.

Method	Type	search D?	Multi-objective?	Evaluation Metric(s)
AGAN [35]	RL	×	×	IS
AutoGAN [10]		×	×	IS
E2GAN [33]		×	√	IS+FID†
DEGAS [7]	Gradient	×	×	Loss
AdversarialNAS [9]		√	×	Loss
AlphaGAN [32]		√	×	Loss
EGAN [34]	EA	×	√	Loss
EAS-GAN [22]		x	x	Loss
COEGAN [5]		√	×	FID (G); Loss (D)
EAGAN		√	√	Pareto-front(IS,FID,#size)‡

Table 1. Comparison of our EAGAN and the existing NAS-GAN methods. The third column indicates whether the method supports searching discriminators. † indicates a linear combination of metrics. ‡ indicates the Pareto-front of multiple metrics.

2.3 NAS for GANs

Due to the great success of NAS in searching neural networks, many works have also applied NAS to search GANs, summarized in Table. 1. AGAN [35] and AutoGAN [10] are among the first RL-based NAS methods to search GANs, but they only use IS as the reward to guide the search. E2GAN [33] is rewarded by a linear combination of IS and FID. However, to avoid the notorious instability of GAN training, these early NAS-GAN methods only search generator (G) with a fixed discriminator (D) architecture, resulting in a sub-optimal GAN. AdversarialNAS [9] proposes to search G and D simultaneously in a differentiable way. However, it results in highly coupled architectures of G and D. The ablation study in [32] has demonstrated that simultaneously searching G and D would potentially increase the negative impact of inferior discriminators and hinder finding the optimal GANs. Liu et al. [25] propose to progressively grow the architectures of G and D in an alternating fashion, but this is only a remedy to alleviate the issue of architecture coupling and causes huge computational costs (580 GPU days on the CIFAR-10 [20] dataset). COEGAN [5] is very relevant to our work, which also uses an evolutionary algorithm to search G and D in two separate groups of architectures (called populations), but the two populations' architectures are coupled during the search. To reduce the search difficulty, COEGAN only explores a simple search space and experiments on a small dataset (MNIST [21]). The final results show that COEGAN fails to outperform the previous human-designed GANs. In summary, since coupling G and D is not conducive to searching for the optimal GAN, we decouple them into two stages.

3 Preliminary

3.1 Weight-sharing based Neural Architecture Search

The early NAS methods first retrain the searched models from scratch and then evaluate their performance [39,30], which obtains accurate evaluation but con-

sumes huge resources, e.g., [30] took 3,150 GPU days to search. To improve search efficiency, the weight-sharing strategy [28] was proposed to allow all subnets to share weights within a super network, so they can be evaluated without retraining by inheriting the weights from SuperNet. In our work, we also adopt the weight-sharing method to search generators and discriminators from SuperNet-G $\mathcal{N}_G$ and SuperNet-D $\mathcal{N}_D$, respectively. To simplify the notations, we use $\mathcal{N}$ to refer to both $\mathcal{N}_G$ and $\mathcal{N}_D$. Denote the loss of the i-th subnet $\mathcal{N}_i$ as L_i, and the weights of $\mathcal{N}$ as W. The gradients of SuperNet loss L with respect to W is

$$\nabla_W L = \frac{1}{N}\sum_{i=1}^{N} \nabla_{W_i} L_i = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial L_i}{\partial W_i} \tag{5}$$

where W_i is the weights of $\mathcal{N}_i$, and N is the total number of subnets. However, it is not practical to accumulate all subnets' gradients in each batch. An alternative way is to use mini-batch subnets to update weights W. In our experiments, we find that randomly sampling one subnet (i.e., $N = 1$) per batch can also work.

3.2 Search Space

To ensure a fair comparison, we use the same search space as in [9] since it also searches both generators and discriminators. The search space is given in Fig. 1.

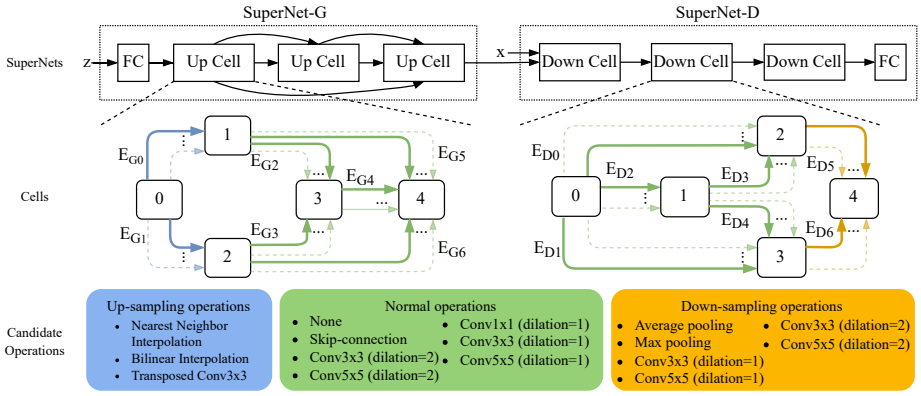


Fig. 1. Overview of search space. E_{G0} and E_{G1} are up-sampling operations, E_{D5} and E_{D6} are down-sampling operations, and the other edges are normal operations.

SuperNet-G $\mathcal{N}_G$ comprises a fully-connected (FC) layer and three Up-Cells. Each cell contains five ordered nodes (0-4), where node 0 is the output of the previous cell. There are multiple candidate operations between two nodes, each represented by an edge, and only one operation will be activated (solid edge). The edges E_{G0} and E_{G1} indicate up-sampling operations. The rest edges (E_{G2}

to E_{G6}) are normal operations, where "None" indicates no connection between two nodes. We encode each edge by a one-hot sequence. For example, [0,1,0] for edge E_{G0} indicates that the bilinear interpolation operation is activated. **SuperNet-D** $\mathcal{N}_D$ comprises three Down-Cells and an FC layer. The Down-Cell is the inverted structure of the Up-Cell. The edges E_{D0} to E_{D4} are normal operations, and E_{D5} and E_{D6} are down-sampling operations. Thus, searching the architecture of G and D is transformed into searching a set of one-hot sequences.

4 Methods

EAGAN comprises two stages, each having two steps: *weights training* and *architecture evolution*. The *many-to-one* and *one-to-one* training strategies tailored for two stages are detailed in Sec. 4.1 and Sec. 4.2, respectively. Sec. 4.3 describes the steps for evolving architectures, which is the same in both stages.

4.1 Stage-1: Searching Generator

Many-to-One GAN Training. As shown in Fig. 2 (left), in stage-1, we search generators (G) with a fixed discriminator (D) that has 0.91M parameters and the same architecture as that of [9]. We adopt the *many(G)-to-one(D)* training strategy. Specifically, the fixed discriminator $\bar{D}$ is denoted by architecture and weights variables, i.e., $\bar{D} \sim (\bar{\beta}, w_{\bar{D}})$. During each round, we produce P candidate generators to form the *population-G* $\mathcal{A}_G$, where all candidate generators share the weights W_G of SuperNet-G, and each candidate G_i is parameterized with architecture and weights variables, i.e., $G_i \sim (\alpha_i, w_{G_i})$, where $w_{G_i} = W_G(\alpha_i)$. We then pair each candidate generator with the fixed discriminator $\bar{D}$ to form P GANs, i.e., $\{(G_1, \bar{D}), ..., (G_P, \bar{D})\}$. Stage-1 can be formalized as below

$$\alpha^* = \arg\min_{\alpha_i} \{V_{val}\left(\alpha_i \mid w^*_{G_i}, w^*_{\bar{D}}, \bar{\beta}\right), i \in \{1, ..., P\}\} \tag{6}$$

$$\text{s.t.} \quad w^*_{G_i} = \arg\min_{w_{G_i}} E_{z\sim p(z)}\left[\log\left(1 - \bar{D}\left(G_i(z)\right)\right)\right] \tag{7}$$

$$w^*_{\bar{D}} = \arg\max_{w_{\bar{D}}} \sum_{i=1}^{P} E_{x\sim p_{\text{data}}\,(x)}[\log \bar{D}(x)] + E_{z\sim p(z)}[\log(1 - D(G_i(z)))] \tag{8}$$

where the inner (Eq. (7)$\sim$(8)) is to optimize weights of P GANs on the training set via the many-to-one strategy, and the outer (Eq. (6)) is to obtain the optimal architecture of G according to the value function on the validation set (i.e., V_{val}). The inner and outer optimizations are solved by iterative procedures, outlined in Alg. 1. These P GANs share the same discriminator and are trained for multiple epochs for each round. To get a fair comparison between generators, for each training batch, we uniformly draw a generator from P candidate generators and train it with the fixed discriminator (lines 4 to 10 in Alg. 1). The many-to-one

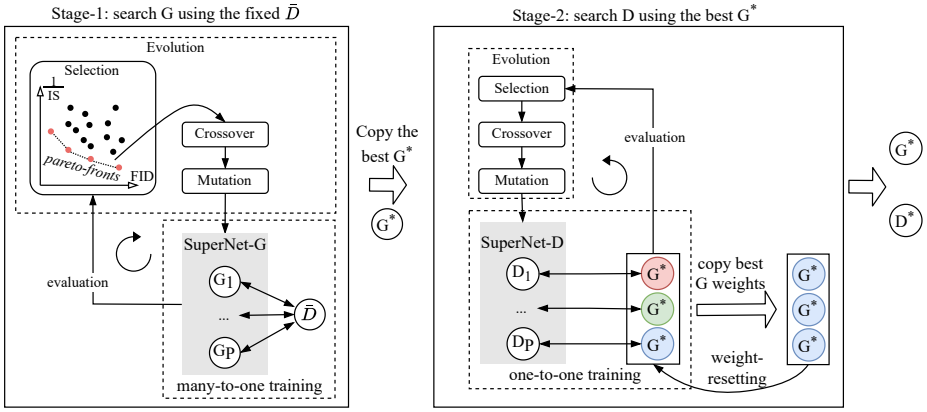


Fig. 2. Two-stage pipeline of EAGAN.

training mechanism can bring two benefits. First, the fixed discriminator $\bar{D}$ is trained with various generators, which can be viewed as an ensemble method to some extent, avoiding that $\bar{D}$ is over-fitted and much stronger than generators. Second, different generators are trained with the same discriminator, so we can fairly compare the performance of these generators to find the optimal one. Besides, a generator with mode collapse will not interfere with other generators because the selection step will eliminate it from the population (see Sec. 4.3).

4.2 Stage-2: Searching Discriminator

After stage-1, we obtain an optimal generator G^* with architecture α^*. In stage-2, we use it to guide searching discriminators (D). There are two major challenges in searching D: the lack of evaluation metrics for discriminators and the instability of GAN training. Next, we describe our approaches to these two challenges.

One-to-One GAN Training. Unlike generators, discriminators are difficult to be assessed directly. For example, the accuracy of discriminators does not reflect the overall performance of GANs, as high accuracy may indicate that generators are too weak to fool discriminators, and low accuracy may indicate that generator has mode collapse, with no way to analyze the real cause. Some works [9,32,5] use the reconstructed loss (e.g., Eq. (1)) to monitor discriminator, but the loss is not a reliable monitor metric as GAN training is a dynamic equilibrium process. An alternative solution is to *indirectly* assess the discriminator via IS and FID metrics calculated based on a generator, so we cannot simply imitate the training strategy of stage-1 (e.g., many(D)-to-one(G)) in stage-2; otherwise, all discriminators are paired with the same generator and not comparable. To this end, we propose the *one-to-one* training strategy. Specifically, we create P copies of G^*, each paired with a candidate discriminator from *population-D* $\mathcal{A}_D$. Thus, we obtain P GANs, i.e., $\{(G_i, D_i), i \in \{1, ..., P\}\}$, where $G_i \sim (\alpha^*, w_{G_i})$

and $D_i \sim (\beta_i, w_{D_i})$. Each GAN is independently trained as a regular GAN via Eq. (1)∼(3). Therefore, stage-2 can be formalized as follows

$$\beta^* = \arg\min_{\beta_i}\{V_{val}\left(\beta_i \mid w^*_{G_i}, w^*_{D_i}, \alpha^*\right), i \in \{1, ..., P\}\} \tag{9}$$

$$\text{s.t. } w^*_{G_i}, w^*_{D_i} = \min_{G_i}\max_{D_i} E_{x\sim p_{\text{data}}\,(x)}[\log D_i(x)] + E_{z\sim p(z)}[\log(1 - D_i(G_i(z)))] \tag{10}$$

Weight-resetting. The second challenge of stage-2 is that the one-to-one training strategy does not fully guarantee a fair comparison between different discriminators. Since P generators are trained independently, each generator will have different weights after a round of one-to-one training, presented with different colors (see Fig. 2 (right)). If some generators have mode collapse due to combination with unsuitable discriminators, then subsequent discriminators paired with these generators will obtain unfair and biased estimation. To alleviate this problem, we propose the *weight-resetting* strategy, which is to first copy the weights of best generator in the current round, and then initialize all generators in the next round with the copied weights. In the first round, all generators are initialized with the weights of G^* found in stage-1. In summary, the one-to-one training strategy allows each discriminator to be paired with an independent generator, and the weight-resetting strategy ensures a fair comparison between different discriminators and alleviates the instability of GAN training.

4.3 Architecture Evolution

As shown in Fig. 2, after weights training, stage-1 and stage-2 perform the same steps to evolve generators and discriminators, respectively. To simplify notations, we use $\mathcal{N}$, $\mathcal{N}_i$, and $\mathcal{A}$ to denote the SuperNet, the i-th subnet, and population, of candidate generators (stage-1) and discriminators (stage-2), respectively.

Selection. This step is equivalent to Eq. (6) of stage-1 and Eq. (9) of stage-2. In our work, we use IS [31] and FID [15] metrics to evaluate the performance of individual (i.e., subnet). FID is inversely correlated with IS, so we adopt the *non-dominated sorting strategy* [6] as the value function to produce the Pareto-front individuals during each round. An individual $\mathcal{N}_i$ is said to be dominated by another individual $\mathcal{N}_j$ when Eq. (11) satisfies.

$$\begin{aligned} &\mathcal{F}_k\left(\mathcal{N}_i\right) \geq \mathcal{F}_k\left(\mathcal{N}_j\right) \forall k \in \{1, \ldots, K\} \\ &\mathcal{F}_k\left(\mathcal{N}_i\right) > \mathcal{F}_k\left(\mathcal{N}_j\right) \exists k \in \{1, \ldots, K\} \end{aligned} \tag{11}$$

where $\mathcal{F}_k$ indicates the objective (e.g., FID, and $\frac{1}{IS}$[6]). We split the population with P individuals into a number of disjoint subsets (or ranks) $\Omega = \{\Omega_0, \Omega_1, ...\}$ by comparing the number of times each individual being dominated by other individuals, where the length of Ω and each subset may be different for each search round. After non-dominated sorting, individuals in the same subset are

[6] The higher the IS value, the better the GAN performance.

Algorithm 1 EAGAN.

Input: SuperNet-G $\mathcal{N}_G$, SuperNet-D $\mathcal{N}_D$, population-G $\mathcal{A}_G$, population-D $\mathcal{A}_D$, population size $P = |\mathcal{A}_G| = |\mathcal{A}_D|$, multi-objective set $\mathcal{F}$, total search rounds R, each round contains E epochs of training.

Output: G^* and D^*

$\bar{D} \sim (\bar{\beta}, w_{\bar{D}}) \leftarrow$ Initialize a discriminator with weights $w_{\bar{D}}$ and fixed architecture $\bar{\beta}$;
$\mathcal{A}_G^{(0)} = \{G_1^{(0)}, ..., G_P^{(0)}\} \leftarrow$ Warm-up($\mathcal{N}_G, \bar{D}$);
$\{(G_i^{(0)}, \bar{D}\}), i \in \{1, ..., P\}\} \leftarrow$ Initialize P GANs that share the same discriminator;
for *r=0:R* -1 **do**
  for *e=0:E* -1 **do**
    for *batch* $x = \{x_1, ..., x_m\}$ *in training set* **do**
      Sample noise data $z = \{z_1, ..., z_m\}$;
      Uniformly sample $G_i^{(r)}$ from $\mathcal{A}_G^{(r)}, i \in \{1, ..., P\}$;
      Update weights of $\bar{D}$ via Eq. (8);
      Update weights of $G_i^{(r)}$ via Eq. (7);
    end
  end
  $\mathcal{A}_G^{(r)} \leftarrow$ Select Pareto-front generators under $\mathcal{F}$ based on validation set;
  $\mathcal{A}_G^{(r)} \leftarrow$ Crossover&Mutation($\mathcal{A}_G^{(r)}$);
end
$G^* \sim (\alpha^*, w_{G^*}) \leftarrow$ the best generator with architecture α^* and weights w_{G^*};
$\mathcal{A}_D^{(0)} = \{D_1^{(0)}, ..., D_P^{(0)}\} \leftarrow$ Warm-up($G^*, \mathcal{N}_D$);
$\{(G_i, D_i^{(0)}), i \in \{1, ..., P\}\} \leftarrow$ Initialize P GANs, where G_i is a copy of G^*;
for *r=0:R* -1 **do**
  for *e=0:E* -1 **do**
    for *batch* $x = \{x_1, ..., x_m\}$ *in training set* **do**
      Sample noise data $z = \{z_1, ..., z_m\}$;
      Uniformly sample a GAN $(G_i, D_i^{(r)})$ from P GANs;
      Update weights of G_i and $D_i^{(r)}$ via Eq. (10);
    end
  end
  $\mathcal{A}_D^{(r)} \leftarrow$ Select Pareto-front discriminators under $\mathcal{F}$ based on validation set;
  $\mathcal{A}_D^{(r)} \leftarrow$ Crossover&Mutation($\mathcal{A}_D^{(r)}$);
  $w_{G^*} \leftarrow$ the generator weights of the best GAN;
  $w_{G_1} = ... = w_{G_P} = w_{G^*} \leftarrow$ Weight-resetting;
end
$D^* \sim (\beta^*, w_{D^*}) \leftarrow$ the best discriminator with architecture β^* and weights w_{D^*};

regarded as equally important and better than those in a larger rank. For example, the individuals in the subset Ω_0 outperform all other subsets of individuals. Finally, we sequentially select $\frac{P}{2}$ individuals from lower to higher ranks.

Crossover&Mutation. As detailed in Sec. 3.2, the architecture of each subnet is encoded by a set of one-hot sequences, where the one-hot sequence indicates an edge and the position of 1 indicates the candidate operation acti-

vated on that edge. Thus, the basic unit of crossover and mutation is the one-hot sequence. We set $\frac{P}{2}$ Pareto-front individuals obtained from the selection step as parents. Then, we repeatedly perform crossover and mutation on these parents with probabilities of 0.3 and 0.5, respectively, until we generate $\frac{P}{2}$ new individuals. For crossover, we randomly choose two parents and exchange a single one-hot sequence (i.e., an edge). For mutation, we also randomly choose the one-hot sequence of an edge and change the position of 1 on it.

5 Experiments

5.1 Implementation Settings

Datasets. Following the previous NAS-GANs [10,9,34], we search on the CIFAR-10 [20] and evaluate on both CIFAR-10 and STL-10 [4] datasets. CIFAR-10 has 50,000 training images and 10,000 test images with 32×32 resolutions. STL-10 has 100,500 images with 96×96 resolutions, but we resize them to 48×48.

Warm-up Stage. We set up a warm-up stage before the start of stage-1 and stage-2 to ensure a fair competition for all candidate subnets. Specifically, all candidate operations in search space are activated uniformly and trained equally. The warm-up stage has 50 epochs. After that, we randomly sample P subnets to form the first round of population.

Two-stage Search. For both stage-1 and stage-2, we use the hinge loss [26] and Adam optimizer [19] with an initial learning rate of 0.0002. The total number of search rounds is 18, each containing 10 epochs. The noise data is sampled from the Gaussian distribution. A population of $P = 32$ individuals is trained and evolved during each round. The batch sizes for generator and discriminator are 40 and 80, respectively. Besides, we adopt a low-fidelity evaluation strategy, i.e., the number of images used to calculate FID and IS is reduced to 5,000, which greatly reduces the evaluation time and keeps the performance of the searched architectures. Stage-1 and stage-2 take 0.8 and 0.4 GPU days, respectively.

Fully-train Stage. After the two-stage search, we fully train the best-performing GAN (G^*, D^*) from scratch. For the CIFAR-10 dataset, the batch size and learning rate are the same as the search stage, but the total number of training epochs is 600. For the STL-10 dataset, the batch size and the learning rate are 128 and 0.0003 for the generator, and 64 and 0.0002 for the discriminator, respectively. Following the previous NAS-GAN works [9,10], we generate 50,000 images to calculate IS and FID metrics.

5.2 Results and Analysis

Search only Generator (EAGAN-G). Our searched generator G^* is shown in Fig. 3. Note that the generators for the CIFAR-10 (G_C with 7.14M parameters) and STL-10 (G_S with 11.55M parameters) datasets have the same architecture but different input channels, so their sizes are different. We can see that 1) bilinear operation is preferred for up-sampling, which is also observed in previous

NAS-GANs [9,33]; 2) there are 6 "None" operations and 3 "skip-connect" operations among 15 total normal operations, and the normal convolution with kernel size 3×3 is preferred, which is probably because the low-resolution images do not need complicated convolutions to generate. The results in Table. 2 show that, compared with AdversarialNAS [9], our EAGAN can find a better generator with similar time overhead, given the same search space and fixed discriminator. Specifically, our discovered generator achieves a highly competitive FID (10.14) and IS (8.76±0.09) on the CIFAR-10 dataset. In terms of IS, there is a certain gap between NAS-GANs and BigGAN [3] because BigGAN additionally introduces category information as input into the generator's architecture, while NAS-GANs only receive noise data as input. Besides, our generator G_S achieves remarkable results (IS 10.02±0.11, FID=23.34) on the STL-10 dataset, showing an excellent transferability.

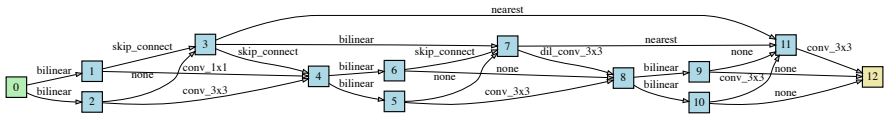


Fig. 3. The architecture of the searched generator ($G_C = G_S = G^*$).

Search both Generator and Discriminator (EAGAN-GD1). In stage-2, we use the best generator G^* found in stage-1 to help search a set of Pareto-front discriminators, from which we select the optimal discriminators for the CIFAR-10 (D_C with 0.91M parameters) and STL-10 (D_S with 1.58M parameters) datasets, respectively, shown in Fig. 4. We can see a subtle difference (marked in red) between them, i.e., D_S prefers convolutions with a larger kernel size (5×5), while D_C selects skip-connection and a smaller convolution. A possible reason is that the resolution of STL-10 (48×48) is larger than CIFAR-10 (32×32), so it needs a larger kernel size to obtain larger receptive fields.

After two-stage search, we retrain two GANs (i.e., (G_C, D_C) and (G_C, D_S)) on the CIFAR-10 and STL-10 datasets, respectively, and report their results in Table. 2. We can see that none of existing NAS-GANs can guarantee to find excellent GANs in both search scenarios: (a) searching only generators; and (b) searching both generators and discriminators. For example, AdverearialNAS [9] performs poorly (IS=7.86±0.08, FID=24.04) in scenario (a), and AlphaGAN [32] suffers from instability in scenario (b), as its performance drops significantly from (IS=8.89±0.09, FID=10.35) in scenario (a) to (IS=8.70±0.11, FID=15.56) in scenario (b). However, our EAGAN performs well in both search scenarios, and the discriminators searched in stage-2 can further improve the performance of the optimal generator discovered in stage-1. Specifically, we achieve a competitive IS value (8.81±0.10) and the best FID (9.91) on the CIFAR-10 dataset. Besides,

Method	Search Method	GPU Days	CIFAR-10		STL-10	
			IS↑	FID↓	IS↑	FID↓
DCGANs [29]	Manual	–	6.64±0.14	37.7	–	–
WGAN-GP [12]			7.86±0.07	29.3	–	–
Progressive GAN [17]			8.80±0.05	18.33	–	–
SN-GAN [26]			8.22±0.05	21.7	9.16±0.12	40.1
ProbGAN [13]			7.75	24.60	8.87±0.09	46.74
Improv MMD GAN[36]			8.29	16.21	9.34	37.63
BigGAN [3]			**9.22**	14.73	-	-
AGAN [35]	RL	1200	8.29±0.09	30.5	9.23±0.08	52.7
AutoGAN [10]		2	8.55±0.10	12.42	9.16±0.12	31.01
E2GAN [33]		0.3	8.51±0.13	11.26	9.51±0.09	25.35
DEGAS [7]	Gradient	1.167	8.37±0.08	12.01	9.71±0.11	28.76
AlphaGAN [32]		0.13	8.98±0.09	10.35	10.12±0.13	22.43
AlphaGAN [32]†		-	8.70±0.11	15.56	-	-
AdversarialNAS [9]		1	7.86±0.08	24.04	8.52±0.05	38.85
AdversarialNAS [9]†		1	8.74±0.07	10.87	9.63±0.19	26.98
DGGAN [25]	Heuristic	580	8.64±0.06	12.10	-	-
EGAN [34]	EA	1.25	6.9±0.09	-	-	-
EAS-GAN [22]		1	7.45±0.08	33.2	-	38.84
EAGAN-G	EA	0.8	8.76±0.09	10.14	10.02±0.11	23.34
EAGAN-GD1†		0.8+0.4	8.81±0.10	**9.91**	**10.44±0.08**	**22.18**
EAGAN-GD2†		0.75+0.37	8.63±0.09	12.84	9.76±0.06	26.52
EAGAN-GD3†		1.55+0.73	8.69±0.10	10.53	10.14±0.11	24.22

Table 2. Results on the CIFAR-10 and STL-10 datasets. † indicates searching both generators (G) and discriminators (D).

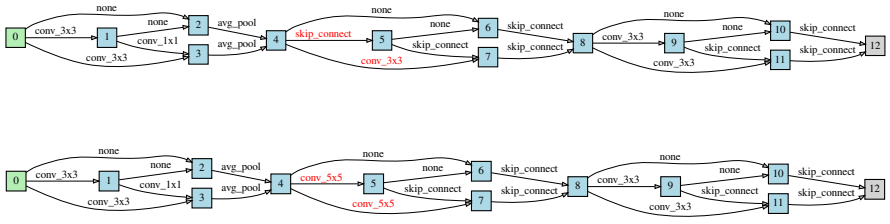
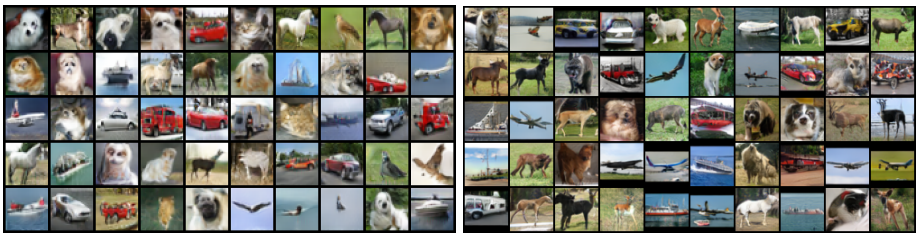


Fig. 4. The searched discriminators on CIFAR-10 (top) and STL-10 (bottom).

our EAGAN achieves remarkable performance (IS=10.44±0.08, FID=22.18) on the STL-10 dataset, which outperforms the existing NAS-searched GANs. In Fig. 5, we present 50 images randomly generated by generators trained on the CIFAR-10 and the STL-10 datasets without cherry-picking, respectively. The generated images are of rich diversity and high quality.



(a) CIFAR-10 (b) STL-10

Fig. 5. The generated images by EAGAN in random without cherry-picking.

5.3 Ablation Study

Search G or D first? EAGAN searches G first and then searches D. *What about search D first?* Our experiments show that searching D first in stage-1 will make the searched D much stronger than candidate G in stage-2, which in turn causes the gradients of G to vanish. Thus, we should search G first.

Initialize different D in stage-1. Our above experiment (i.e., EAGAN-GD1) uses the discriminator of [9] in stage-1. We further implement two experiments to explore the effect of initializing different D in stage-1. EAGAN-GD2 uses a simple network with 0.92M parameters, comprising five normal convolutions and a linear layer, as the initial D in stage-1. EAGAN-GD3 is to repeat the two-stage search several times, i.e., the optimal D of the previous stage-2 is set as the initial D of the next stage-1. From Table. 2, we can see that both EAGAN-GD2 and EAGAN-GD3 achieve competitive results on the CIFAR-10 and STL-10 datasets, indicating that EAGAN does not require strong prior knowledge to design the initial state of D and that searching once is sufficient to find good models, balancing search overhead and model performance.

Decoupled vs. Coupled. To validate the effectiveness of our decoupled search method, we perform a coupled search experiment as the baseline, i.e., the architectures of G and D are evolved simultaneously for each search round. Fig. 6 presents the learning curves of the baseline and our EAGAN, which shows that coupled search is unstable as it fluctuates throughout the search. In contrast, the overall performance of our decoupled search is better and significantly improved, especially in stage-2 of searching discriminators. Besides, the decoupled search also fluctuates in stage-1 due to the competition among candidate generators incurred by the weight-sharing strategy, and how to address the negative impact of weight-sharing is still an open problem [37].

Weight-resetting Strategy. We conduct another experiment on the CIFAR-10 dataset, which differs from our EAGAN only in that the weights of P generators in stage-2 are continuously and independently trained without weight-resetting (WR) strategy. Fig. 7 presents the learning curves with and without the WR strategy in stage-2, which shows that our proposed WR strategy can effectively enhance the stability of GAN training and obtain better IS and FID scores in stage-2 of searching discriminators.

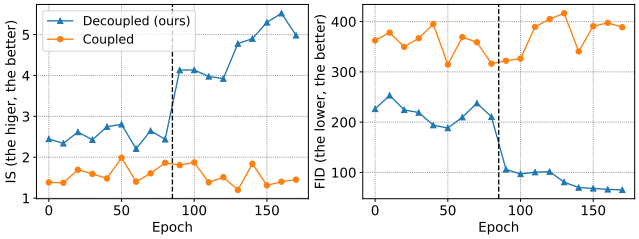


Fig. 6. Learning curves when generators and discriminators are coupled/decoupled. The dashed line indicates the boundary between the two decoupled stages of EAGAN.

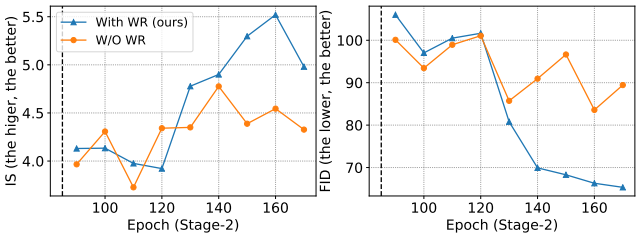


Fig. 7. Learning curves with and without (W/O) the weight-resetting (WR) strategy in stage-2.

6 Conclusion & Future Work

This paper proposes an efficient two-stage evolutionary algorithm-based NAS framework to search GANs, namely EAGAN. We demonstrate that decoupling the search of the generator and discriminator into two stages can significantly improve the stability of searching GANs via the GAN training strategies (many-to-one and one-to-one) tailored for both stages and the weight-resetting strategy. EAGAN is very efficient and takes 1.2 GPU days to finish the search on CIFAR-10. Our searched GANs achieve competitive performance (IS and FID) on the CIFAR-10 dataset and outperform previous NAS-GANs on the STL-10 dataset.

We believe our work deserves more in-depth study and may benefit other potential fields. For example, our decoupled paradigm and tailored training strategies are well suited for large-scale parallel search when architectures require adversarial training. Further, we shall investigate reducing the interference of weight-sharing in search and explore high-resolution generative tasks.

Acknowledgements. Thanks to the NVIDIA AI Technology Center (NVAITC) for providing the GPU cluster to support our work. BH was supported by the NSFC Young Scientists Fund No. 62006202, Guangdong Basic and Applied Basic Research Foundation No. 2022A1515011652, RGC Early Career Scheme No. 22200720, RGC Research Matching Grant Scheme No. RMGS2022_11_02 and HKBU CSD Departmental Incentive Grant.

References

1. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein generative adversarial networks. In: International conference on machine learning. pp. 214–223. PMLR (2017)
2. Bissoto, A., Valle, E., Avila, S.: The six fronts of the generative adversarial networks. arXiv preprint arXiv:1910.13076 (2019)
3. Brock, A., Donahue, J., Simonyan, K.: Large scale GAN training for high fidelity natural image synthesis. In: ICLR (2019)
4. Coates, A., Ng, A., Lee, H.: An analysis of single-layer networks in unsupervised feature learning. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics (2011)
5. Costa, V., Lourenço, N., Machado, P.: Coevolution of generative adversarial networks. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar). pp. 473–487. Springer (2019)
6. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In: International conference on parallel problem solving from nature. pp. 849–858. Springer (2000)
7. Doveh, S., Giryes, R.: Degas: Differentiable efficient generator search. arXiv preprint arXiv:1912.00606 (2019)
8. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. arXiv preprint arXiv:1808.05377 (2018)
9. Gao, C., Chen, Y., Liu, S., Tan, Z., Yan, S.: Adversarialnas: Adversarial neural architecture search for gans. In: Proceedings of the CVPR (2020)
10. Gong, X., Chang, S., Jiang, Y., Wang, Z.: Autogan: Neural architecture search for generative adversarial networks. In: Proceedings of the ICCV (2019)
11. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Advances in neural information processing systems **27** (2014)
12. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. Advances in neural information processing systems **30** (2017)
13. He, H., Wang, H., Lee, G.H., Tian, Y.: Probgan: Towards probabilistic gan with theoretical guarantees. In: ICLR (2018)
14. He, X., Zhao, K., Chu, X.: Automl: A survey of the state-of-the-art. Knowledge-Based Systems **212**, 106622 (2021)
15. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: Proceedings of the NeurIPS (2017)
16. Hjelm, R.D., Jacob, A.P., Che, T., Trischler, A., Cho, K., Bengio, Y.: Boundary-seeking generative adversarial networks. arXiv preprint arXiv:1702.08431 (2017)
17. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)
18. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4401–4410 (2019)
19. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
20. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Tech. rep. (2009)

21. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
22. Lin, Q., Fang, Z., Chen, Y., Tan, K.C., Li, Y.: Evolutionary architectural search for generative adversarial networks. IEEE Transactions on Emerging Topics in Computational Intelligence (2022)
23. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European conference on computer vision (ECCV). pp. 19–34 (2018)
24. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
25. Liu, L., Zhang, Y., Deng, J., Soatto, S.: Dynamically grown generative adversarial networks. Proceedings of the AAAI Conference on Artificial Intelligence **35**(10), 8680–8687 (May 2021)
26. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. arXiv preprint arXiv:1802.05957 (2018)
27. Odena, A., Olah, C., Shlens, J.: Conditional image synthesis with auxiliary classifier gans. In: International conference on machine learning. pp. 2642–2651. PMLR (2017)
28. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)
29. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
30. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI. vol. 33 (2019)
31. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Proceedings of the NeurIPS (2016)
32. Tian, Y., Shen, L., Su, G., Li, Z., Liu, W.: Alphagan: Fully differentiable architecture search for generative adversarial networks. arXiv preprint arXiv:2006.09134 (2020)
33. Tian, Y., Wang, Q., Huang, Z., Li, W., Dai, D., Yang, M., Wang, J., Fink, O.: Off-policy reinforcement learning for efficient and effective gan architecture search. In: Proceedings of the ECCV (2020)
34. Wang, C., Xu, C., Yao, X., Tao, D.: Evolutionary generative adversarial networks. IEEE Transactions on Evolutionary Computation **23**(6), 921–934 (2019)
35. Wang, H., Huan, J.: Agan: Towards automated design of generative adversarial networks. arXiv preprint arXiv:1906.11080 (2019)
36. Wang, W., Sun, Y., Halgamuge, S.: Improving MMD-GAN training with repulsive loss function. In: ICLR (2019)
37. Xie, L., Chen, X., Bi, K., Wei, L., Xu, Y., Wang, L., Chen, Z., Xiao, A., Chang, J., Zhang, X., et al.: Weight-sharing neural architecture search: A battle to shrink the optimization gap. ACM Computing Surveys (CSUR) **54**(9), 1–37 (2021)
38. Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., Xu, C.: Cars: Continuous evolution for efficient neural architecture search. In: Proceedings of the CVPR (2020). https://doi.org/10.1109/CVPR42600.2020.00190
39. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)