AlphaGAN: Fully Differentiable Architecture Search for Generative Adversarial Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Generative Adversarial Networks (GANs) are formulated as minimax game problems, whereby generators attempt to approach real data distributions by virtue of adversarial learning against discriminators. In this work, we aim to boost model learning from the perspective of network architectures, by incorporating recent progress on automated architecture search into GANs. To this end, we propose a fully differentiable search framework for generative adversarial networks, dubbed *alphaGAN*. The searching process is formalized as solving a bi-level minimax optimization problem, where the outer-level objective aims for seeking a suitable network architecture towards pure Nash Equilibrium conditioned on the network parameters optimized in the inner level. The entire optimization performs a first-order method by alternately optimizing the two-level objective in a fully differentiable manner, enabling architecture search to be completed in an enormous search space. Extensive experiments on CIFAR-10 and STL-10 datasets show that our algorithm can obtain high-performing architectures only with 3-GPU hours on a single GPU in the search space comprised of approximate 2×10^{11} possible configurations.

1 Introduction

Generative Adversarial Networks (GANs) [1] have shown promising performance on a variety of generative tasks (e.g., image generation [2], image translation [3, 4], dialogue generation [5], and image inpainting [6]). However, pursuing high-performance generative networks is non-trivial due to the non-convex non-concave property. There is a rich history of research aiming to improve the training stabilization and alleviate mode collapse by introducing generative adversarial functions (e.g., Wasserstein distance [7], Least Squares loss [8], and hinge loss [9]) or regularization (e.g., gradient penalty [10, 11]).

Alongside the direction of improving loss functions, improving architectures has been proven to be important for stabilizing training and improving generalization. Previous works [12, 9, 10, 13, 2] employ deep convolutional networks to boost the performance of GANs. However, such a manual architecture design typically requires domain-specific knowledge from human experts, which is even challenging for GANs due to the minimax formulation that it intrinsically possesses. Recent progress of architecture search on a variety of supervised learning tasks [14, 15, 16, 17] has shown that remarkable achievements can be achieved by automating the architecture search process.

In this paper, we aim to address the problem of GAN architecture search from the perspective of Game theory since it is essentially a minimax problem [1] targeting at finding pure Nash Equilibrium of generator and discriminator [18, 19]. From this perspective, we propose a fully differentiable architecture search framework for GANs, dubbed *alphaGAN*, in which a differentiable evaluation metric is introduced for guiding architecture search towards pure Nash Equilibrium [20]. Motivated by DARTS [15], we formulate the search process of alphaGAN as a bi-level minimax optimization

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

problem, and solve it efficiently via stochastic gradient-type methods. Specifically, the outer level objective aims to optimize the generator architecture parameters towards pure Nash Equilibrium, whereas the inner level constraint targets at optimizing the weight parameters conditioned on the architecture currently searched.

This work is related to several recent methods. GAN architecture search is performed with a reinforcement learning paradigm [21, 22, 23] or a gradient-based paradigm [24].

Extensive experiments including comparison to these methods and analysis of the searched architectures, demonstrate the effectiveness of the proposed algorithm in performance and efficiency. Specially, alphaGAN can discover high-performance architectures while being much faster than the other automated architecture search methods.

2 GAN Architecture Search as Fully Differential Optimization

Differentiable Architecture Search was first proposed in [15], where the problem is formulated as a bi-level optimization. Weight parameters and architecture parameters are optimized in the inner level and outer level, respectively. The objective function in both levels is the cross entropy loss, which can reflect the quality of current models in classification tasks.

However, deploying such a framework to searching architectures of GANs is non-trivial. The training of GANs corresponds to the optimization of a minimax problem. Many previous works [2, 25] have pointed that the adversarial loss cannot refect the quality of GANs. A suitable evaluation metric is essential for a gradient-based NAS-GAN framework.

Evaluation function. Due to the intrinsic minimax property of GANs, the training process of GANs can be viewed as a zero-sum game as in [18, 1]. The universal objective of training GANs consequentially can be regarded as reaching pure equilibrium. Hence, we adopt the primal-dual gap function [25, 26] for evaluating the generalization of vanilla GANs. Given a pair of G and D, the duality-gap function is defined as

$$\mathcal{V}(G,D)=\mathrm{Adv}(G,\overline{D})-\mathrm{Adv}(\overline{G},D):=\max_{D}\mathrm{Adv}(G,D)-\min_{G}\mathrm{Adv}(G,D). \tag{1}$$

The evaluation metric $\mathcal{V}(G,D)$ is non-negative and $V(G,D)=0$ can be only achieved when the pure equilibrium holds. Function (1) provides a quantified measure of describing “how close is current GAN to pure equilibrium", which can be used for assessing model capacity.

The architecture search for GANs can be formulated as a specific bi-level optimization problem:

$$\min_{\alpha}\left\{\mathcal{V}(G,D):(G,D):=\arg\min_{G}\max_{D}\mathrm{Adv}\left(G,D\right)\right\}, \tag{2}$$

where $\mathcal{V}(G,D)$ performs on the validation dataset and supervises seeking the optimal generator architecture as an outer-level problem, and the inner-level optimization on $\mathrm{Adv}\left(G,D\right)$ aims to learn suitable network parameters (including both the generator and discriminator) for GAN on the current architecture.

In this work, we exploit the hinge loss from [9, 27] as the generative adversarial function $\mathrm{Adv}\left(G,D\right)$.

AlphaGAN formulation. By integrating the generative adversarial function (i.e., hinge loss) and evaluation function (1) into the bi-level optimization (2), we can obtain the final objective for the framework as follows,

$$\min_{\alpha}\mathcal{V}_{val}(G,D)=\mathrm{Adv}(G,\overline{D})-\mathrm{Adv}(\overline{G},D) \tag{3}$$

$$s.t.\quad \omega\in\arg\min_{\omega_G}\max_{\omega_D}\mathrm{Adv}_{train}(G,D) \tag{4}$$

where generator G and discriminator D are parameterized with variables (α_G,ω_G) and (ω_D), respectively, $\overline{D}=\arg\max_D \mathrm{Adv}_{val}(G,D)$, and $\overline{G}=\arg\min_G \mathrm{Adv}_{val}(G,D)$. The detailed search algorithm and other details are in the supplementary material.

Table 1: Comparison with state-of-the-art GANs on CIFAR-10. † denotes the results reproduced by us, with the structure released by Auto-GAN and trained under the same setting as AutoGAN.

Architecture	Params (M)	FLOPs (G)	search time (GPU-hours)	search space	search method	IS (↑ is better)	FID (↓ is better)
DCGAN([12])	-	-	-	-	manual	6.64 ± 0.14	-
SN-GAN([9])	-	-	-	-	manual	8.22 ± 0.05	21.7 ± 0.01
Progressive GAN([13])	-	-	-	-	manual	**8.80±0.05**	-
WGAN-GP, ResNet([10])	-	-	-	-	manual	7.86 ± 0.07	-
AutoGAN([22])	5.192	1.77	-	$\sim 10^5$	RL	8.55 ± 0.1	12.42
AutoGAN†	5.192	1.77	82	$\sim 10^5$	RL	8.38 ± 0.08	13.95
AGAN([21])	-	-	28800	~ 20000	RL	8.29 ± 0.09	30.5
Random search([30])	2.701	1.11	40	$\sim 2 \times 10^{11}$	Random	8.46 ± 0.09	15.43
alphaGAN$_{(l)}$	8.618	2.78	22	$\sim 2 \times 10^{11}$	gradient	8.71 ± 0.12	**11.23**
alphaGAN$_{(s)}$	2.953	1.32	3	$\sim 2 \times 10^{11}$	gradient	8.60 ± 0.11	11.85

3 Experiments

In this section, we conduct extensive experiments on CIFAR-10 [28] and STL-10 [29]. First, the generator architecture is searched on CIFAR-10 and the discretized optimal structure is fully re-trained from scratch following [22] in Section 3.1. We compare alphaGAN with the other automated GAN methods in multiple measures to demonstrate its effectiveness. Second, the generalization of the searched architectures is verified by fully training on STL-10 and evaluation in Section 3.2. To further understand the properties of our method, a series of studies on the key components of the framework are shown in Section 3.3. Experiment details are in the supplementary material.

3.1 Searching on CIFAR-10

We first compare our method with recent automated GAN methods. For a fair comparison, we report the performance of best run (over 3 runs) for reproduced baselines and ours in the Table 1 and provide the performance of several representative works with manually designed architectures for reference. We provide a detailed analysis and discussion about the searching process. And the statistic properties of architectures searched by alphaGAN are in the supplementary material.

Performances of alphaGAN with two search configurations are shown In Tab. 1 by adjusting step sizes of the inner loop and the outer loop, where alphaGAN$_{(l)}$ represents passing through every epoch on the training and validation sets for each loop, i.e., $steps = 390$. And alphaGAN$_{(s)}$ represents using smaller interval steps with $steps = 20$.

The results show that our method performs well in the two settings and outperforms the other automated GAN methods in terms of both efficiency and performance. alphaGAN$_{(l)}$ obtains the lowest FID compared to all the baselines. Particularly, alphaGAN$_{(s)}$ can attain the best tradeoff between efficiency and performance, and it can be achieve comparable results by searching in a large search space (significantly larger than RL-based baselines) in a considerably efficient manner (i.e., only 3 GPU hours compared to the baselines with tens to thousands of GPU hours). The architecture obtained by alphaGAN$_{(s)}$ is light-weight and computationally efficient, which reaches a good trade-off between performance and time complexity.

3.2 Transferability on STL-10

To validate the transferability of the architectures obtained by alphaGAN, we directly train models by using the obtained architectures on the STL-10 dataset. The results are shown in Table 2. Both alphaGAN$_{(l)}$ and alphaGAN$_{(s)}$ show remarkable superiority in performance over the baselines with either automated or manually designed architectures. It reveals the benefit that the architecture searched by alphaGAN can be effectively exploited across datasets. It is surprising that alphaGAN$_{(s)}$ is best-behaved, which achieves the best performance in both IS and FID scores. It also shows that compared to increase on model complexity, appropriate selection and composition of operations can contribute to model performance in a more efficient manner which is consistent with the primary motivation of automating architecture search.

Table 2: Results on STL-10. The structures of alphaGAN$_{(l)}$ and alphaGAN$_{(s)}$ are searched on CIFAR-10 and fully trained on STL-10. † denotes the reproduced results, with the architectural configurations released by the original papers.

Architecture	Params (M)	FLOPs (G)	IS	FID
SN-GAN([9])	-	-	9.10 ± 0.04	40.1 ± 0.5
ProbGAN([31])	-	-	8.87 ± 0.095	46.74
Improving MMD GAN([32])	-	-	9.36	36.67
Auto-GAN([22])	5.853	3.98	9.16 ± 0.12	31.01
Auto-GAN†	5.853	3.98	9.38 ± 0.08	27.69
AGAN([21])	-	-	9.23 ± 0.08	52.7
alphaGAN$_{(l)}$	9.279	6.26	9.53 ± 0.12	24.52
alphaGAN$_{(s)}$	3.613	2.97	**10.12±0.13**	**22.43**

3.3 Ablation Study

We conduct ablation experiments on CIFAR-10 to better understand the influence of components when applying different configurations on both alphaGAN$_{(l)}$ and alphaGAN$_{(s)}$, including the studies with the questions: the effect of searching the discriminator architecture and obtaining the optimal generator $\overline{G}$. More experiments are shown in the supplementary material.

Table 3: Ablation studies on CIFAR-10.

Type	Search D?	Obtain $\overline{G}$		IS	FID
		Update α_G	Update ω_G		
alphaGAN$_{(l)}$	✓	×	✓	8.51 ± 0.09	18.07
	×	×	✓	8.51 ± 0.06	11.38
	×	×	✓	8.51 ± 0.06	11.38
	×	✓	×	7.06 ± 0.06	43.99
	×	✓	✓	8.43 ± 0.11	13.91
alphaGAN$_{(s)}$	✓	×	✓	8.70 ± 0.11	15.56
	×	×	✓	8.72 ± 0.11	12.86
	×	×	✓	8.72 ± 0.11	12.86
	×	✓	×	8.45 ± 0.09	15.47
	×	✓	✓	8.18 ± 0.11	18.85

Search D's architecture or not? A problem may arise from alphaGAN: If searching discriminator structures can facilitate the searching and training of generators? The results in Table 3 show that searching the discriminator cannot help the search of the optimal generator. We also conducted the trial by training GANs with the obtained architectures by searching G and D, while the final performance is inferior to the setting of retraining with a given discriminator configuration. Simultaneously searching architectures of both G and D potentially increases the effect of inferior discriminators which may hamper the search of optimal generators conditioned on strong discriminators. In this regard, solely learning generators' architectures may be a better choice.

How to obtain $\overline{G}$? In the definition of duality gap, $\overline{G}$ and $\overline{D}$ denote the optima of G and D, respectively. As both of the architecture and network parameters are variables for $\overline{G}$, we do the experiments of investigating the effect of updating ω_G and α_G for attaining $\overline{G}$. The results in Table 3 show that updating ω_G solely achieves the best performance. Approximating $\overline{G}$ with ω_G update solely means that the architectures of G and $\overline{G}$ are identical, and hence optimizing architecture parameters α_G in (3) can be viewed as the compensation for the gap brought by the weight parameters of ω_G and $\omega_{\overline{G}}$.

4 Conclusion

We presented alphaGAN, a fully differentiable architecture search framework for GANs, which is efficient and effective to seek high-performing generator architectures from vast possible configurations, achieving comparable or superior performance compared to state-of-the-art architectures being either manually designed or automatically searched. In addition, the analysis of tracking the behavior of architecture performance and operation distribution gives some insights about architecture design, which may promote further research on architecture improvement. We mainly focused on vanilla GANs in this work and would like to extend such a framework to conditional GANs, in which extra regularization on the parts of networks is typically imposed for task specialization, as future work.

Broader Impact

AlphaGAN will have potential positive impacts on the tasks of image/video generation, natural language generation, and high fidelity speech synthesis. Researchers can utilize alphaGAN to design a powerful generator adversarial network with superior performance. On the other hand, we would hope that this work can attract more attention in the AI research community to design architectures of generation tasks rather than classification tasks. Moreover, the theory behind alphaGAN is so transferable that it can apply to conditional GANs on several conditionally generative tasks, e.g., style transfer, image-to-image translation, etc.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[3] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[4] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797, 2018.

[5] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*, 2017.

[6] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5505–5514, 2018.

[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[8] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[9] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[10] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[11] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On convergence and stability of gans. *arXiv preprint arXiv:1705.07215*, 2017.

[12] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[13] Shaokai Ye, Xiaoyu Feng, Tianyun Zhang, Xiaolong Ma, Sheng Lin, Zhengang Li, Kaidi Xu, Wujie Wen, Sijia Liu, Jian Tang, et al. Progressive dnn compression: A key to achieve ultra-high weight pruning and quantization rates using admm. *arXiv preprint arXiv:1903.09769*, 2019.

[14] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[16] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[17] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[18] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[19] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

[20] John F Nash et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[21] Hanchao Wang and Jun Huan. Agan: Towards automated design of generative adversarial networks. *arXiv preprint arXiv:1906.11080*, 2019.

[22] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3224–3234, 2019.

[23] Yuan Tian, Qin Wang, Zhiwu Huang, Wen Li, Dengxin Dai, Minghao Yang, Jun Wang, and Olga Fink. Off-policy reinforcement learning for efficient and effective gan architecture search. *arXiv preprint arXiv:2007.09180*, 2020.

[24] Chen Gao, Yunpeng Chen, Si Liu, Zhenxiong Tan, and Shuicheng Yan. Adversarialnas: Adversarial neural architecture search for gans. *arXiv preprint arXiv:1912.02037*, 2019.

[25] Paulina Grnarova, Kfir Y Levy, Aurelien Lucchi, Nathanael Perraudin, Ian Goodfellow, Thomas Hofmann, and Andreas Krause. A domain agnostic measure for monitoring and evaluating gans. In *Advances in Neural Information Processing Systems*, pages 12069–12079, 2019.

[26] Cheng Peng, Hao Wang, Xiao Wang, and Zhouwang Yang. {DG}-{gan}: the {gan} with the duality gap, 2020.

[27] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.

[28] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[29] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[30] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[31] Hao He, Hao Wang, Guang-He Lee, and Yonglong Tian. Probgan: Towards probabilistic gan with theoretical guarantees.

[32] Wei Wang, Yuan Sun, and Saman Halgamuge. Improving mmd-gan training with repulsive loss function. *arXiv preprint arXiv:1812.09916*, 2018.

[33] Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.

[34] Ding-Zhu Du and Panos M Pardalos. *Minimax and applications*, volume 4. Springer Science & Business Media, 2013.

[35] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[36] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Advances in Neural Information Processing Systems*, pages 3086–3094, 2014.

[39] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019.

A Preliminaries

Minimax Games have regained a lot of attraction [33, 34] since they are popularized in machine learning, such as generative adversarial networks (GAN) [18], reinforcement learning [35, 36], etc. Given the function $Adv\colon \mathbb{X}\times\mathbb{Y}\to\mathbb{R}$, we consider a minimax game and its dual form:

$$\min_G \max_D \mathrm{Adv}(G,D) = \min_G \big\{ \max_D \mathrm{Adv}(G,D)\big\}, \max_D \min_G \mathrm{Adv}(G,D) = \max_D \big\{ \min_G \mathrm{Adv}(G,D)\big\}.$$

The pure equilibrium [20] of minimax game can be used to characterize the best decisions of two players G and D for above minmax game.

Definition 1 *$(\overline{G},\overline{D})$ is called a pure equilibrium of game $\min_G \max_D \mathrm{Adv}(G,D)$ if it holds that*

$$\max_D \mathrm{Adv}(\overline{G},D) = \min_G \mathrm{Adv}(G,\overline{D}), \tag{5}$$

where $\overline{G} = \arg\min_G \mathrm{Adv}(G,D)$ and $\overline{D} = \arg\max_D \mathrm{Adv}(G,D)$. When minimax game equals to its dual problem, $(\overline{G},\overline{D})$ is the pure equilibrium of the game. Hence, the gap between the minimax problem and its dual form can be used to measure the degree of approaching pure equilibrium [25].

Generative Adversarial Network (GAN) proposed in [1] is mathematically defined as a minimax game problem with a binary cross entropy loss of competing between the distributions of real and synthetic images generated by the GAN model. Despite remarkable progress achieved by GANs, training high-performance models is still challenging for many generative tasks due to its fragility to almost every factor in the training process. Architectures of GANs have proven useful for stabilizing training and improving generalization [9, 10, 13, 2], and we hope to discover architectures by automating the design process with limited computational resource in a principled differentiable manner.

B Algorithm and Optimization

In this section, we will give a detailed description for the training algorithm and optimization process of alphaGAN. We first describe the entire network structure of the generator and the discriminator, the search space of the generator, and the continuous relaxation of architectural parameters.

Base Backbone of G and D. The illumination of the entire structure for the generator and discriminator is shown in the supplementary material. The generator is constructed by stacking several cells whose topology is identical to those in AutoGAN [22] and SN-GAN [9] (shown in the supplementary material). Each cell, regarded as a directed acyclic graph, is comprised of the nodes representing intermediate feature maps and the edges connecting pairs of nodes via different operations. We apply a fixed network architecture for the discriminator, based on the conventional design as [9].

Search space of G. The search space is compounded from two types of operations, i.e., normal operations and up-sampling operations. The pool of normal operations, denoted as $\mathcal{O}_n$, is comprised of {conv_1x1, conv_3x3, conv_5x5, sep_conv_3x3, sep_conv_5x5, sep_conv_7x7} . The pool of up-sampling operations, denoted as $\mathcal{O}_u$, is comprised of { deconv, nearest, bilinear}, where "deconv" denotes the ConvTransposed_2x2. operation. Our method allows $(6^3\times 3^2)^3\times 3^3 \approx 2\times 10^{11}$ possible configurations for the generator architecture, which is larger than $\sim 10^5$ of AutoGAN [22].

Continuous relaxation. The discrete selection of operations is approximated by using a soft decision with a mutually exclusive function, following [15]. Formally, let $o\in\mathcal{O}_n$ denote some normal operations on node i, and $\alpha^o_{i,j}$ represent the architectural parameter with respect to the operation between node i and its adjacent node j, respectively. Then the node output induced by the input node i can be calculated by

$$O_{i,j}(x) = \sum_{o\in\mathcal{O}_n} \frac{\exp\left(\alpha^o_{i,j}\right)}{\sum_{o'\in\mathcal{O}_n}\exp\left(\alpha^{o'}_{i,j}\right)} o(x), \tag{6}$$

and the final output is summed over all of its preceding nodes, i.e., $x^j = \sum_{i\in Pr(j)} O_{i,j}(x^i)$. The selection on up-sampling operations follows the same procedure.

Solving alphaGAN. We apply an alternating minimization method to solve alphaGAN with respect to variables $\big((\omega_G,\omega_D),(\omega_{\overline{G}},\omega_{\overline{D}}),\alpha_G\big)$ in Algorithm 1, which is a fully differentiable gradient-type

Algorithm 1 Searching the architecture of alphaGAN

Parameters: Initialize weight parameters (ω_G^1,ω_G^1). Initialize generator architecture parameters α_G^1. Initialize base learning rate η, momentum parameter β_1, and exponential moving average parameter β_2 for Adam optimizer.

1: **for** $k = 1, 2, \cdots, K$ **do**
2: Set $(\omega_G^{k,1}, \omega_D^{k,1}) = (\omega_G^k, \omega_D^k)$ and set $\alpha_G^{k,1} = \alpha_G^k$;
3: **for** $t = 1, 2, \cdots, T$ **do**
4: Sample real data $\{x^{(l)}\}_{l=1}^m \sim \mathbb{P}_r$ from training set and noise $\{z^{(l)}\}_{l=1}^m \sim \mathbb{P}_z$; Estimate gradient of Adv loss with $\{x^{(l)}, z^{(l)}\}$ at $(\omega_G^{k,t}, \omega_D^{k,t})$, dubbed $\nabla \mathrm{Adv}(\omega_G^{k,t}, \omega_D^{k,t})$;
5: $\omega_D^{k,t+1} = \mathrm{Adam}\big(\nabla_{\omega_D} \mathrm{Adv}(\omega_G^{k,t}, \omega_D^{k,t}), \omega_D^{k,t}, \eta, \beta_1, \beta_2\big)$;
6: $\omega_G^{k,t+1} = \mathrm{Adam}\big(\nabla_{\omega_G} \mathrm{Adv}(\omega_G^{k,t}, \omega_D^{k,t}), \omega_G^{k,t}, \eta, \beta_1, \beta_2\big)$;
7: **end for**
8: Set $(\omega_G^{k+1}, \omega_D^{k+1}) = (\omega_G^{k,T}, \omega_D^{k,T})$;
9: Receive architecture searching parameter α_G^k and network weight parameters (ω_G^{k+1},ω_D^{k+1}); Estimate neural architecture parameters $(\omega_{\overline{G}}^{k+1}, \omega_{\overline{D}}^{k+1})$ of $(\overline{G}, \overline{D})$ via Algorithm 2;
10: **for** $s = 1, 2, \cdots, S$ **do**
11: Sample real data $\{x^{(l)}\}_{l=1}^m \sim \mathbb{P}_r$ from the validation set and latent variables $\{z^{(l)}\}_{l=1}^m \sim \mathbb{P}_z$. Estimate gradient of the duality gap $\mathcal{V}$ with $\{x^{(l)}, z^{(l)}\}$ at $(\alpha^{k,s})$, dubbed $\nabla V(\alpha_G^{k,s})$;
12: $\alpha_G^{k,s+1} = \mathrm{Adam}(\nabla_{\alpha_G} V(\alpha_G^{k,s}), \alpha_G^{k,s}, \eta, \beta_1, \beta_2)$;
13: **end for**
14: Set α_G^{k+1}=$\alpha_G^{k,S}$;
15: **end for**
16: Return α_G=α_G^K.

Algorithm 2 Solving $\overline{G}$ and $\overline{D}$

Parameters: Receive architecture searching parameter α_G and weight parameter (ω_G,ω_D). Initialize weight parameter $(\omega_{\overline{G}}^1, \omega_{\overline{D}}^1) = (\omega_G, \omega_D)$ for $(\overline{G}, \overline{D})$. Initialize base learning rate η, momentum parameter β_1, and EMA parameter β_2 for Adam optimizer.

for $r = 1, 2, \cdots, R$ **do**
Sample real data $\{x^{(l)}\}_{l=1}^m \sim \mathbb{P}_r$ from validation dataset and noise $\{z^{(l)}\}_{l=1}^m \sim \mathbb{P}_z$; Estimate gradient of Adv loss with $\{x^{(l)}, z^{(l)}\}$ at $(\omega_G, \omega_{\overline{D}}^r)$, dubbed $\nabla \mathrm{Adv}(\omega_G, \omega_{\overline{D}}^r)$;
$\omega_{\overline{D}}^{r+1} = \mathrm{Adam}\big(\nabla_{\omega_{\overline{D}}} \mathrm{Adv}(\omega_G, \omega_{\overline{D}}^r), \omega_{\overline{D}}^r, \eta, \beta_1, \beta_2\big)$;
end for
for $r = 1, 2, \cdots, R$ **do**
Sample noise $\{z^{(l)}\}_{l=1}^m \sim \mathbb{P}_z$; Estimate gradient of the Adv loss with $\{z^{(l)}\}$ at point $(\omega_{\overline{G}}^r, \omega_D)$, dubbed $\nabla \mathrm{Adv}(\omega_{\overline{G}}^r, \omega_D)$;
$\omega_{\overline{G}}^{r+1} = \mathrm{Adam}\big(\nabla_{\omega_{\overline{G}}} \mathrm{Adv}(\omega_{\overline{G}}^r, \omega_D), \omega_{\overline{G}}^r, \eta, \beta_1, \beta_2\big)$;
end for
Return $(\omega_{\overline{G}}, \omega_{\overline{D}}) = (\omega_{\overline{G}}^R, \omega_{\overline{D}}^R)$.

algorithm. Algorithm 1 is composed of three parts. The first part (line 3-8), called “weight_part", aims to optimize weight parameters ω on the training dataset via Adam optimizer [37]. The second part (line 9), called “test-weight_part", aims to optimize the weight parameters $(\omega_{\overline{G}}, \omega_{\overline{D}})$, and the third part (line 10-12), called ’arch_part’, aims to optimize architecture parameters α_G by minimizing the duality gap. Both ’test-weight_part’ and ’arch_part’ are optimized over the validation dataset via Adam optimizer. Algorithm 2 illuminates the detailed process of computing $\overline{G}$ and $\overline{D}$ by updating weight parameters $(\omega_{\overline{G}}, \omega_{\overline{D}})$ with last searched generator network architecture parameters α_G and related network weight parameters (ω_G, ω_D). In summary, the variables $\big((\omega_G, \omega_D), (\omega_{\overline{G}}, \omega_{\overline{D}}), \alpha_G\big)$ are optimized in an alternating fashion.

C Experiment Details

C.1 Searching on CIFAR-10

The CIFAR-10 dataset is comprised of 50000 images for training. The resolution of the images is 32x32. We randomly split the dataset into two sets during searching: one is used as the training set for optimizing network parameters ω_G and ω_D (25000 images), and another is used as the validation set for optimizing architecture parameters α_G (25000 images). The search iterations for alphaGAN$_{(l)}$ and alphaGAN$_{(s)}$ are set to 100. We use a minibatch size of 64 for both generators and discriminators, channel number of 256 for generators and 128 for discriminators. The dimension of the noise vector is 128. For a fair comparison, the discriminator adopted in searching is the same as the discriminator in AutoGAN [22]. Batch sizes of both the generator and the discriminator are set to 64. The learning rates of weight parameters ω_G and ω_D are $2\mathrm{e}-4$ and the learning rate of architecture parameter α_G is $3\mathrm{e}-4$. We use Adam as the optimizer. The hyperparameters for optimizing weight parameters ω_G and ω_D are set as, 0.0 for β_1 and 0.999 for β_2, and 0 for the weight decay. The hyperparameters for optimizing architecture parameters α_G are set as 0.5 for β_1, 0.999 for β_2 and $1\mathrm{e}-3$ for weight decay.

We use the entire training set of CIFAR-10 for retraining the network parameters after obtaining architectures. we use a minibatch size of 128 for generators and 64 for discriminators. The channel number is set to 256 for generators and 128 for discriminators. The dimension of the noise vector is 128. Discriminator exploited in the re-training phase is identical to that during searching. The batch size of the generator is set to 128. The batch size of the discriminator is set to 64. The generator is trained for 50000 iterations. The learning rates of the generator and discriminator are set to $2\mathrm{e}-4$. The hyperparameters for the Adam optimizer are set to 0.0 for β_1, 0.9 for β_2 and 0 for weight decay.

When testing, 50000 images are generated with random noise, and IS [18] and FID [19] are used to evaluate the performance of generators.

C.2 Transferability

The STL-10 dataset is comprised of ~ 105k training images. We resize the images to the size of 48x48 due to the consideration of memory and computational overhead. The dimension of the noise vector is 128. We train the generator for 80000 iterations. The batch sizes for optimizing the generators and the discriminator are set to 128 and 64, respectively. The channel numbers of the generator and the discriminator are set to 256 and 128, respectively. The learning rates for the generator and the discriminator are both set to $2\mathrm{e}-4$. We also use the Adam as the optimizer, where β_1 is set to 0.5, β_2 is set to 0.9 and weight decay is set to 0.

D The structures of the generator and the discriminator

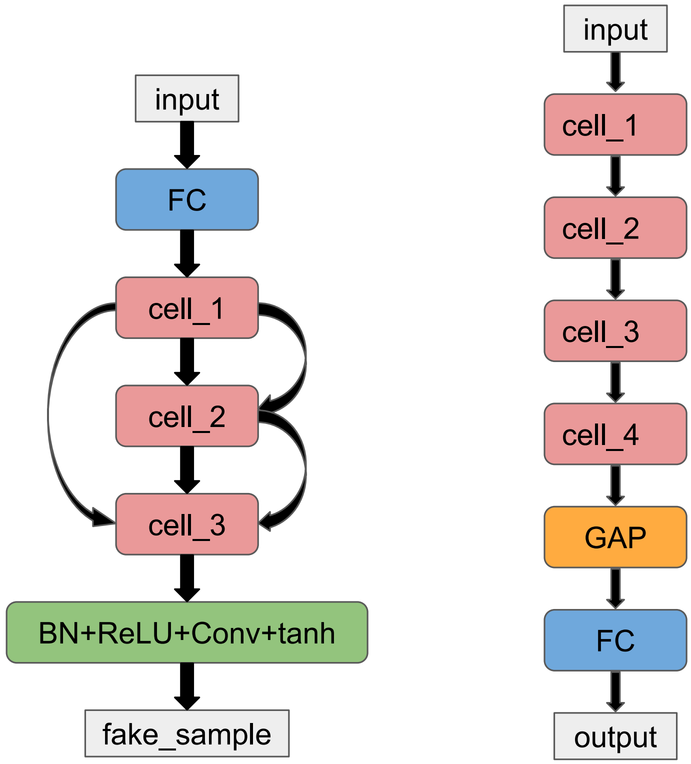
The entire structures of the generator and the discriminator are illustrated in Fig. 1.

The topology of cells in the generator and the discriminator is illustrated in the Fig. 2. In the cell of the generator, the edges from the node 0 to the node 1 and from the node 0 to the node 3 correspond to up-sampling operations, and the rest edges are normal operations. In the cell of the discriminator, the edges from the node 2 to the node 4 and from the node 3 to the node 4 are the operation of avg_pool_2x2 with stride 2, the edges from the node 0 to the node 1 and from the node 1 to the node 2 are the operation of conv_3x3 with stride 1, and the edge from the node 0 to the node 3 is the operation of conv_1x1 with stride 1.

The structures of alphaGAN$_{(l)}$ and alphaGAN$_{(s)}$ are shown in Fig. 4 and Fig. 3.

E Relation between performance and structure

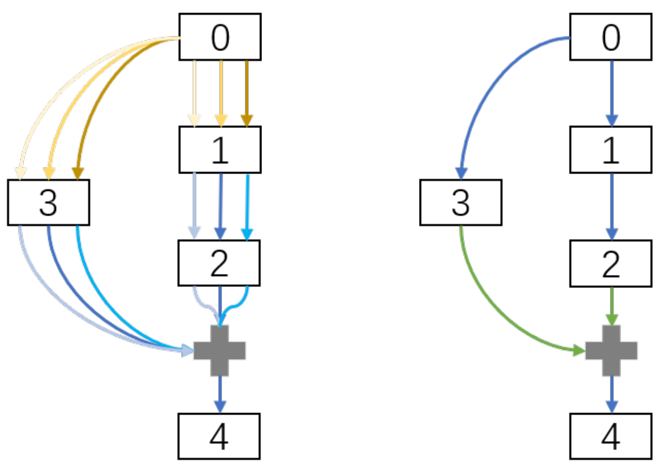
The distributions of operations in 'superior' and 'inferior' are shown in Fig. 5 and Fig. 6, respectively. We get the following observations: first, for up-sampling operations, superior architectures tend to exploit "nearest" or "bilinear" rather than "deconvolution" operations. Second, "conv_1x1" operations dominate in the cell_1 of superior generators, suggesting that convolutions with large kernel sizes may not be optimal when the spatial dimensions of feature maps are relatively small (i.e., 8x8). Finally,



(a) G

(b) D

Figure 1: The topology of the generator and the discriminator.



(a) The cell of G

(b) The cell of D

Figure 2: The topology of the cell in the generator and the discriminator. The topology of the generator and the discriminator is identical to those of AutoGAN [22] and SN-GAN [9].

convolutions with large kernels (e.g., conv_5x5, sep_conv_3x3, and sep_conv_5x5) are preferred on higher resolutions (i.e., cell_3 of superior generators), indicating the benefit of integrating information from relatively large receptive fields for low-level representations on high resolutions.

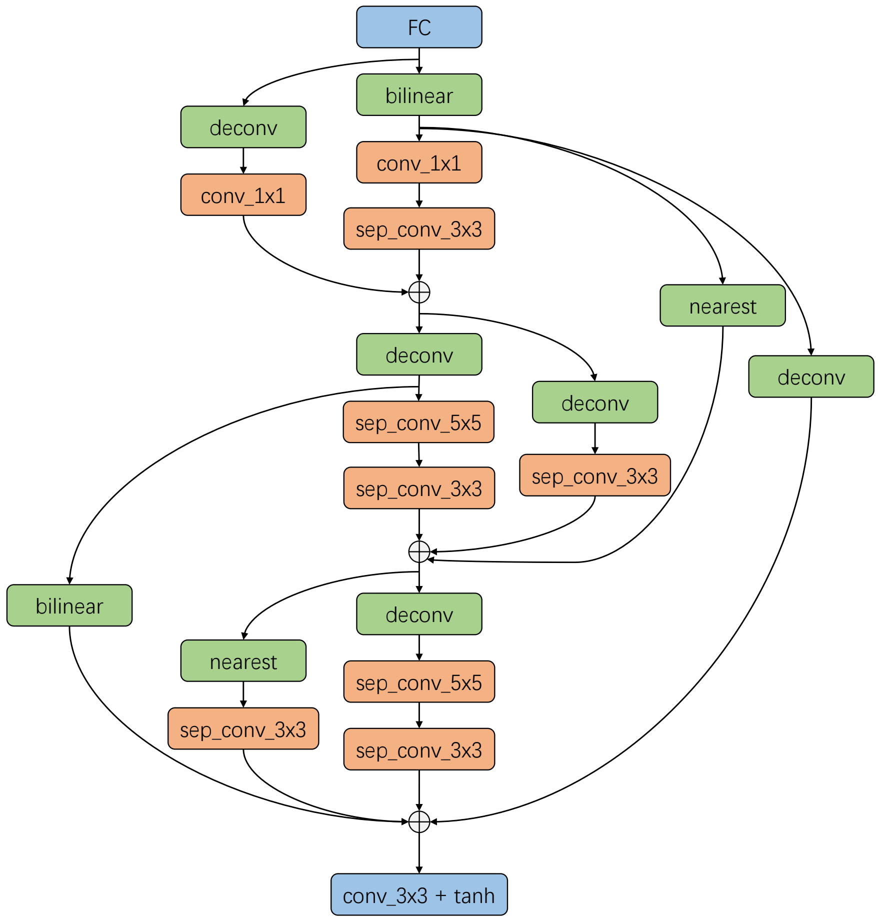


Figure 3: The structure of alphaGAN$_{(s)}$.

F Generated Samples

Generated samples of alphaGAN$_{(s)}$ on STL-10 are shown in Fig. 7.

G Additional Results

In this section, we present the more experimental results and analysis (due to page limit), including model scaling, intermediate architectures in searching, using Gumbel-max trick, warm-up, ablation study on step sizes for 'arch_part', effect of channel numbers for searching, searching on STL-10, and the analysis of failure cases. The 'baseline' in Tab. 4 denotes the structure searched under the default settings of alphaGAN.

G.1 Robustness on Model Scaling

It would be interesting to know how the architecture performs when scaling up/down model complexity. To this regard, we introduce a ratio to simply re-scale the channel dimension of the network configuration for the fully training step. The relation between performance and parameter size is illuminated in Fig. 8. The range of attaining promising performance is relatively narrow for alphaGAN$_{(s)}$, mainly caused by the light-weight property induced by dominated depthwise separable convolutions. Light-weight architectures naturally result in highly sparse connections between network neurons which may be sensitive to the configuration difference between searching and re-training. In contrast, alphaGAN$_{(l)}$ shows acceptable performance in a wide range of parameter

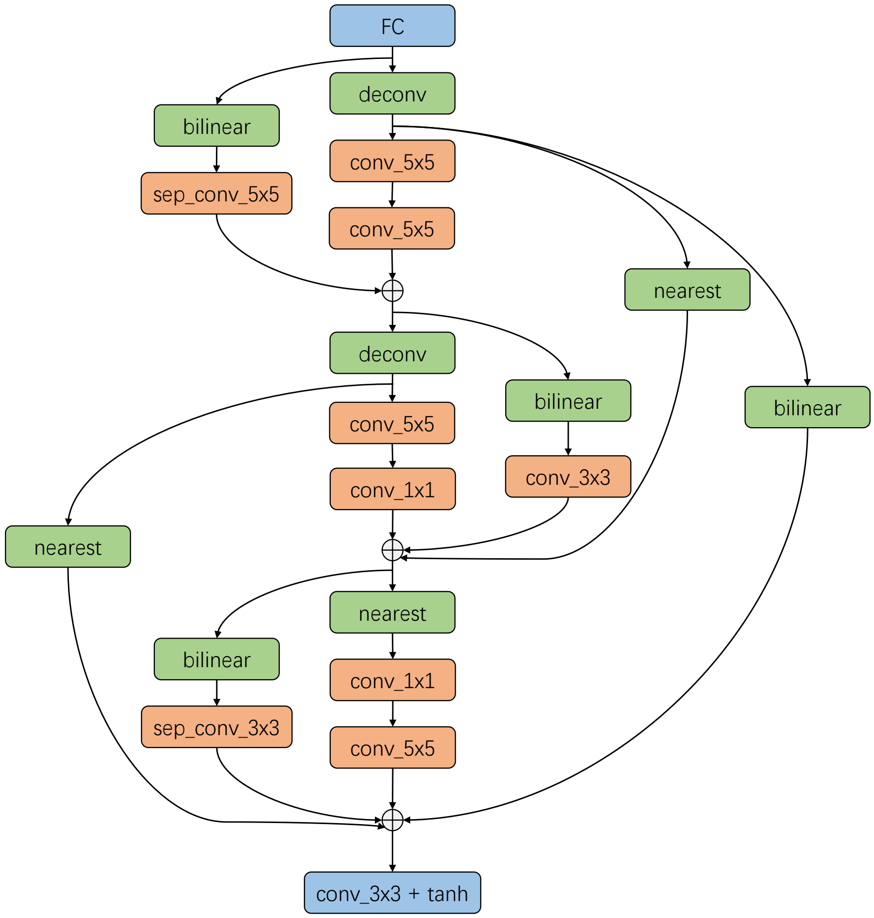
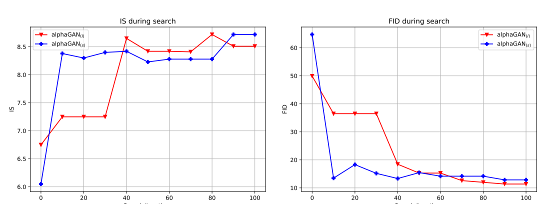


Figure 4: The structure of alphaGAN$_{(l)}$.

sizes (from 2M to 18M). While both of them present some degree of robustness on the scaling of the original searching configuration.

G.2 Intermediate Architectures in Searching

To understand the search process of alphaGAN, we track the intermediate structures of alphaGAN$_{(s)}$ and alphaGAN$_{(l)}$ during searching, and fully train them on CIFAR-10 (in Fig. 9). We observe a clear trend that the architectures are learned towards high performance during searching though slight oscillation may happen. Specially, alphaGAN$_l$ realizes gradual improvement in performance during the process, while alphaGAN$_{(s)}$ displays a faster convergence on the early stage of the process and can achieve comparable results, indicating solving inner-level optimization problem by virtue of rough approximations (as using more steps can always achieve a closer approximation of the optimum) can significantly benefit the efficiency of solving the bi-level problem without sacrifice in accuracy.



(a) `IS in search` (b) `FID in search`

Figure 9: Tracking architectures during searching. alphaGAN$_{(s)}$ is denoted by blue color with plus marker and alphaGAN$_{(l)}$ is denoted by red color with triangle marker.

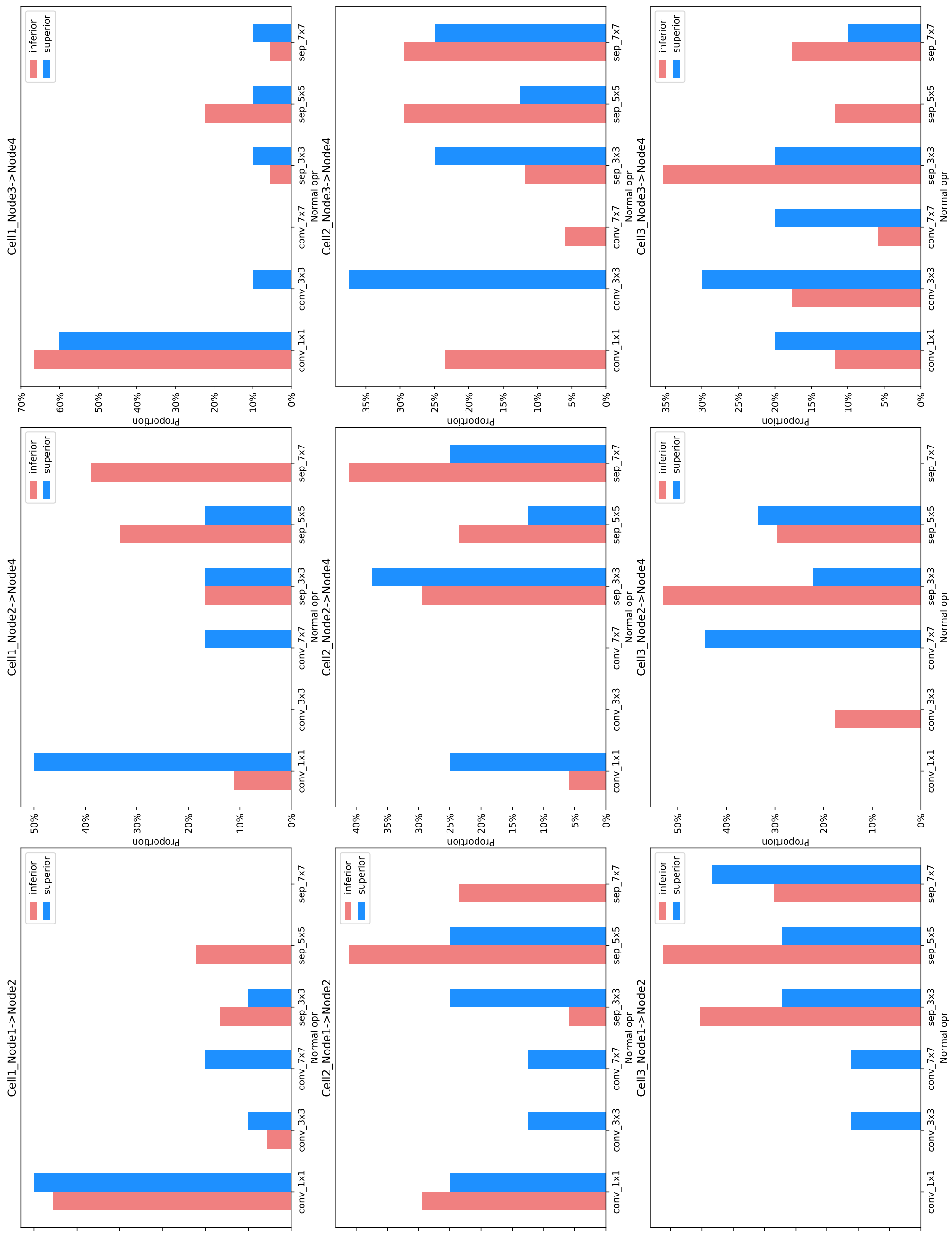


Figure 5: The distributions of normal operations.

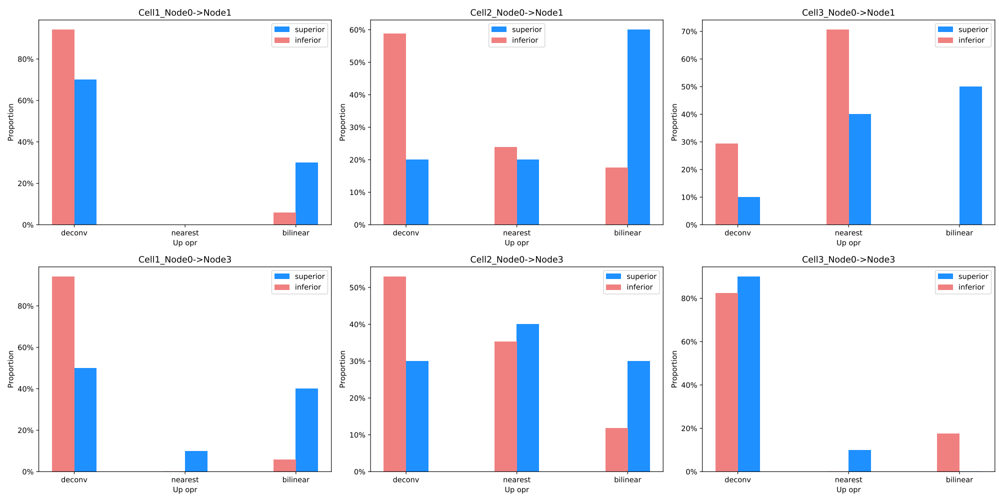


Figure 6: The distributions of up-sampling operations.

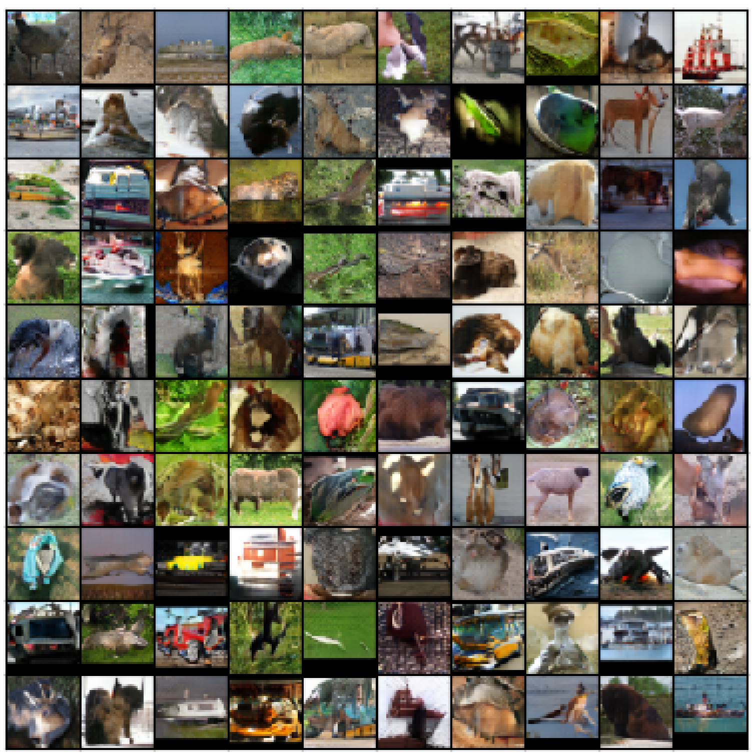


Figure 7: Generated samples of alphaGAN$_{(s)}$ on STL-10.

392 ## G.3 Gumbel-max Trick

393 Gumbel-max trick [38] can be written as,

$$\beta^{o'} = \frac{\exp\left(\left(\alpha^{o'} + g^{o'}\right)/\tau\right)}{\sum_{o \in \mathcal{O}_n} \exp\left(\left(\alpha^{o} + g^{o}\right)/\tau\right)}, \tag{7}$$

394 where $\beta^{o'}$ is the probability of selecting operation o' after Gumbel-max, and $\alpha^{o'}$ represents the
395 architecture parameter of operation o', respectively. $\mathcal{O}_n$ represents the operation search space. g^{o}

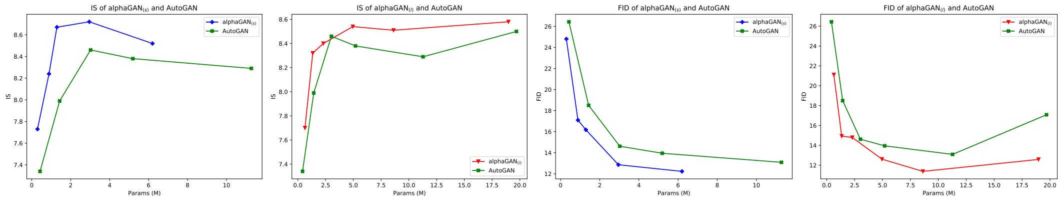


Figure 8: Relation between model capacity and performance. To align the model capacities with AutoGAN, the channels for G and D in alphaGAN$_{(l)}$ are $[64, 96, 128, 192, 256, 384]$, the channels for G and D in alphaGAN$_s$ are $[64, 128, 160, 256, 384]$, and the channels for G and D in AutoGAN are $[64, 128, 192, 256, 384, 512]$.

Table 4: Gumbel-max trick and Warm-up.

Type	Name	Gumbel-max?	Fix alphas?	IS	FID
alphaGAN$_{(l)}$	baseline	$\times$	$\times$	8.51 ± 0.06	11.38
	Gumbel-max	✓	$\times$	8.48 ± 0.10	20.69
	Warm-up	$\times$	✓	8.34 ± 0.07	15.49
alphaGAN$_{(s)}$	baseline	$\times$	$\times$	8.72 ± 0.11	12.86
	Gumbel-max	✓	$\times$	8.56 ± 0.06	15.66
	Warm-up	$\times$	✓	8.25 ± 0.12	19.07

denotes samples drawn from the Gumbel (0,1) distribution, and τ represents the temperature to control the sharpness of the distribution. Instead of continuous relaxation, the trick chooses an operation on each edge, enabling discretization during searching. We compare the results by searching with and without Gumbel-max trick. The results in Tab. 4 show that searching with Gumbel-max may not be the essential factor for obtaining high-performance generator architectures.

G.4 Warm-up protocols

The generator contains two parts of parameters, (ω_G, α_G). The optimization of α_G is highly related to network parameters ω_G. Intuitively, pretraining the network parameters ω_G can benefit the search of architectures since a better initialization may facilitate the convergence. To investigate the effect, we fix α_G and only update ω_G at the initial half of the searching schedule, and then α_G and ω_G are optimized alternately. This strategy is denoted as 'Warm-up' in Table 4.

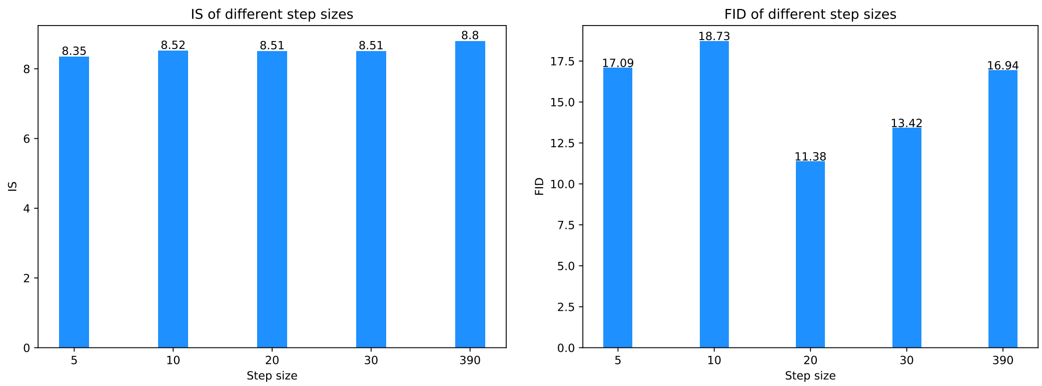
The results show that the strategy may not help performance, i.e., IS of 'Warm-up' is slightly worse than that of the baseline and FID of 'Warm-up' is worse than that of the baseline, while it can benefit the searching efficiency, i.e., it spends ~ 15 GPU-hours for alphaGAN$_{(l)}$ (compared to $\sim$22 GPU-hours via the baseline) , and ~ 1 GPU-hour for alphaGAN$_{(s)}$ (compared to ~ 3 GPU-hours via the baseline).

G.5 Effect of Step Sizes

To analyze the effect of different step sizes on the "arch part", corresponding to the optimization process of the architecture parameters α_G in Algorithm 1 (line 10-13). Since alphaGAN$_{(l)}$ has larger step sizes for 'weight part' and 'test-weight part' compared with alphaGAN$_{(s)}$, the step size of 'arch part' can be adjusted in a wider range. We select the alphaGAN$_{(l)}$ to conduct the experiments and the results are shown in Fig. 10. We can observe that the method perform fair robustness among different step sizes on the IS metric, while network performance based on the FID metric may be hampered with a less proper step.

G.6 Effect of Channels in Searching

As the default settings of alphaGAN, we search and re-train the networks with the same channel dimensions (i.e., G_channels=256 and D_channels=128), which are predefined. To explore the impact



(a) IS (b) FID

Figure 10: The effect of different step sizes of 'arch part'.

Table 5: The channels in searching on the alphaGAN$_{(s)}$.

Search channels	Re-train channels	Params (M)	FLOPs (G)	IS	FID
G_32 D_32	G_32 D_32	0.109	0.02	7.10 ± 0.08	36.22
	G_256 D_128	2.481	1.12	8.61 ± 0.12	14.98
G_64 D_64	G_64 D_64	0.403	0.212	7.97 ± 0.09	22.49
	G_256 D_128	4.658	3.26	8.70 ± 0.17	14.02
G_128 D_128	G_128 D_128	1.967	0.91	8.26 ± 0.08	16.50
	G_256 D_128	7.309	3.64	8.75 ± 0.09	13.02
G_256 D_128	G_256 D_128	2.953	1.32	8.72 ± 0.11	12.86
	G_128 D_128	0.887	0.34	8.36 ± 0.08	17.12
	G_64 D_64	0.296	0.09	7.73 ± 0.08	24.81
	G_32 D_32	0.111	0.025	6.85 ± 0.1	35.6

of the channel dimensions during searching on the final performance of the searched architectures, we adjust the channel numbers of the generator and the discriminator during searching based on the searching configuration of alphaGAN$_{(s)}$. The results are shown in Tab. 5. We observe that our method can achieve acceptable performance under a wide range of channel numbers (i.e., $32 \sim 256$). We also find that using consistent channel dimensions during searching and re-training phases is beneficial to the final performance.

When reducing channels during searching, we observe an increasing trend on the operations of depth-wise convolutions with large kernels (e.g. 7x7), indicating that the operation selection induced by such automated mechanism is adaptive to need of preserving the entire information flow (i.e., increasing information extraction on the spatial dimensions to compensate for the channel limits).

G.7 Searching on STL-10

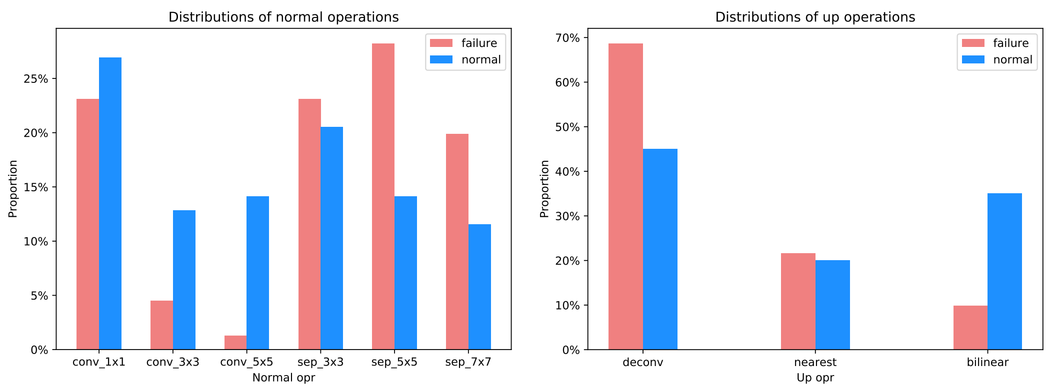
We also search alphaGAN$_{(s)}$ on STL-10. The channel dimensions in the generator and the discriminator are set to 64 (due to the consideration of GPU memory limit). We use the size of 48x48 as the resolution of images. The rest experimental settings are same as the one of searching on CIFAR-10. The settings remain the same as Section C.2 when retraining the networks.

The results of three runs are shown in Tab. 6. Our method achieves high performance on both STL-10 and CIFAR-10, demonstrating the effectiveness and transferability of alphaGAN are not confined to a certain dataset. alphaGAN$_{(s)}$ remains efficient which can obtain the structure reaching the state-of-the-art on STL-10 with only 2 GPU-hours. We also find no failure case exists in the three repeated experiments of alphaGAN$_{(s)}$ compared to that on CIFAR-10, which may be related to

Table 6: Search on STL-10. We search alphaGAN$_{(s)}$ on STL-10 and re-train the searched structure on STL-10 and CIFAR-10. In our repeated experiments, failure cases are prevented.

Name	Search time (GPU-hours)	Dataset of re-training	Params (M)	FLOPs (G)	IS	FID
Repeat_1	~ 2	STL-10	4.552	5.55	9.22 ± 0.08	25.42
Repeat_2	~ 2		2.475	2.01	9.66 ± 0.10	29.28
Repeat_3	~ 2		4.013	3.67	9.47 ± 0.10	26.61
Repeat_1	~ 2	CIFAR-10	3.891	2.47	8.29 ± 0.17	13.94
Repeat_2	~ 2		1.815	0.90	8.20 ± 0.13	16.54
Repeat_3	~ 2		3.352	1.63	8.62 ± 0.11	12.64

multiple latent factors that datasets intrinsically possess (e.g., resolution, categories) and we leave as a future work.



(a) `Distribution of normal operations` (b) `Distribution of up operations`

Figure 11: Distributions of operations in normal cases and failure cases of alphaGAN.

Table 7: Repeated search on CIFAR-10.

Name	Description	Params (M)	FLOPs (G)	IS	FID
alphaGAN$_{(s)}$	normal case	4.475	2.36	8.44 ± 0.13	13.62
		2.953	1.32	8.72 ± 0.11	12.86
	failure case	2.994	1.08	6.77 ± 0.07	45.88
alphaGAN$_{(l)}$	normal case	8.207	2.41	8.55 ± 0.08	15.42
		8.618	2.78	8.51 ± 0.06	11.38
	failure case	4.666	2.36	7.48 ± 0.1	52.58

G.8 Failure cases

As we pointed out in the main paper, the searching of alphaGAN will encounter failure cases, analogous to other NAS methods [39]. For better understanding the method, we present the comparison between normal cases and failure cases in Tab. 7 and the distributions of operations in Fig. 11. We find that deconvolution operations dominate in these failure cases. To validate this, we conduct the experiments on the variant by removing deconvolution operations from the search space under the configuration of alphaGAN$_{(s)}$. The results (with 6 runs) in Tab. 8 show that the failure cases can be prevented in this scenario.

Table 8: Search w\o deconv on alphaGAN$_{(s)}$.

Name	Params (M)	FLOPs (G)	IS	FID
Repeat_1	4.594	2.20	8.29 ± 0.08	15.12
Repeat_2	2.035	0.51	8.34 ± 0.10	14.92
Repeat_3	1.586	0.55	8.24 ± 0.09	18.07
Repeat_4	1.631	0.58	8.32 ± 0.09	15.85
Repeat_5	1.631	0.60	8.43 ± 0.08	17.15
Repeat_6	2.064	1.03	8.26 ± 0.11	16.00

We also test on another setting by integrating conv_1x1 operation with the interpolation operations (i.e., nearest and bilinear) and making them learnable as deconvonvolution, denoted as 'learnable interpolation'. The results (with 6 runs) under the configuration of alphaGAN$_{(s)}$ are shown in Tab. 9, suggesting that the failure cases can also be alleviated by the strategy.

Table 9: The effect of 'learnable interpolation' on alphaGAN$_{(s)}$.

Method	Name	Params (M)	FLOPs (G)	IS	FID
Learnable Interpolation	Repeat_1	2.775	0.99	8.43 ± 0.15	14.8
	Repeat_2	2.243	0.545	8.49 ± 0.12	18.82
	Repeat_3	3.500	0.99	8.35 ± 0.1	18.93
	Repeat_4	3.195	1.53	8.59 ± 0.1	13.22
	Repeat_5	2.968	0.82	8.22 ± 0.11	14.76
	Repeat_6	2.712	0.77	8.41 ± 0.11	13.47