Generative Adversarial Neural Architecture Search

Seyed Saeed Changiz Rezaei[1*], **Fred X. Han**[1*], **Di Niu**[2], **Mohammad Salameh**[1], **Keith Mills**[2], **Shuo Lian**[3], **Wei Lu**[1] and **Shangling Jui**[3]

[1]Huawei Technologies Canada Co., Ltd.
[2]Department of Electrical and Computer Engineering, University of Alberta
[3]Huawei Kirin Solution, Shanghai, China

Abstract

Despite the empirical success of neural architecture search (NAS) in deep learning applications, the optimality, reproducibility and cost of NAS schemes remain hard to assess. In this paper, we propose Generative Adversarial NAS (GA-NAS) with theoretically provable convergence guarantees, promoting stability and reproducibility in neural architecture search. Inspired by importance sampling, GA-NAS iteratively fits a generator to previously discovered top architectures, thus increasingly focusing on important parts of a large search space. Furthermore, we propose an efficient adversarial learning approach, where the generator is trained by reinforcement learning based on rewards provided by a discriminator, thus being able to explore the search space without evaluating a large number of architectures. Extensive experiments show that GA-NAS beats the best published results under several cases on three public NAS benchmarks. In the meantime, GA-NAS can handle ad-hoc search constraints and search spaces. We show that GA-NAS can be used to improve already optimized baselines found by other NAS methods, including EfficientNet and ProxylessNAS, in terms of ImageNet accuracy or the number of parameters, in their original search space.

1 Introduction

Neural architecture search (NAS) improves neural network model design by replacing the manual trial-and-error process with an automatic search procedure, and has achieved state-of-the-art performance on many computer vision tasks [Elsken *et al.*, 2018]. Since the underlying search space of architectures grows exponentially as a function of the architecture size, searching for an optimum neural architecture is like looking for a needle in a haystack. A variety of *search algorithms* have been proposed for NAS, including random search (RS) [Li and Talwalkar, 2020], differentiable architecture search (DARTS) [Liu *et al.*, 2018], Bayesian optimization (BO) [Kandasamy *et al.*, 2018], evolutionary algorithm (EA) [Dai *et al.*, 2020], and reinforcement learning (RL) [Pham *et al.*, 2018].

Despite a proliferation of NAS methods proposed, their sensitivity to random seeds and reproducibility issues concern the community [Li and Talwalkar, 2020], [Yu *et al.*, 2019], [Yang *et al.*, 2019]. Comparisons between different search algorithms, such as EA, BO, and RL, etc., are particularly hard, as there is no shared search space or experimental protocol followed by all these NAS approaches. To promote fair comparisons among methods, multiple NAS benchmarks have recently emerged, including NAS-Bench-101 [Ying *et al.*, 2019], NAS-Bench-201 [Dong and Yang, 2020], and NAS-Bench-301 [Siems *et al.*, 2020], which contain collections of architectures with their associated performance. This has provided an opportunity for researchers to fairly benchmark search algorithms (regardless of the search space in which they are performed) by evaluating how many queries to architectures an algorithm needs to make in order to discover a top-ranked architecture in the benchmark set [Luo *et al.*, 2020; Siems *et al.*, 2020]. The number of queries converts to an indicator of how many architectures need be evaluated in reality, which often forms the bottleneck of NAS.

In this paper, we propose Generative Adversarial NAS (GA-NAS), a provably converging and efficient search algorithm to be used in NAS based on adversarial learning. Our method is first inspired by the *Cross Entropy* (CE) method [Rubinstein and Kroese, 2013] in importance sampling, which iteratively retrains an architecture generator to fit to the distribution of winning architectures generated in previous iterations so that the generator will increasingly sample from more important regions in an extremely large search space. However, such a generator cannot be efficiently trained through back-propagation, as performance measurements can only be obtained for discretized architectures and thus the model is not differentiable. To overcome this issue, GA-NAS uses RL to train an architecture generator network based on RNN and GNN. Yet unlike other RL-based NAS schemes, GA-NAS does not obtain rewards by evaluating generated architectures, which is a costly procedure if a large number of architectures are to be explored. Rather, it learns a discriminator to distinguish the winning architectures from randomly generated ones in each iteration. This enables the generator to be efficiently trained based on the rewards provided by the discriminator, without many true evaluations. We further establish the convergence of GA-NAS

*Equal Contribution. Correspondence to: Di Niu <dniu@ualberta.ca>, Seyed Rezaei <seyeds.rezaei@huawei.com>.

in a finite number of steps, by connecting GA-NAS to an importance sampling method with a symmetric Jensen–Shannon (JS) divergence loss.

Extensive experiments have been performed to evaluate GA-NAS in terms of its convergence speed, reproducibility and stability in the presence of random seeds, scalability, flexibility of handling constrained search, and its ability to improve already optimized baselines. We show that GA-NAS outperforms a wide range of existing NAS algorithms, including EA, RL, BO, DARTS, etc., and state-of-the-art results reported on three representative NAS benchmark sets, including NAS-Bench-101, NAS-Bench-201, and NAS-Bench-301—it consistently finds top ranked architectures within a lower number of queries to architecture performance. We also demonstrate the flexibility of GA-NAS by showing its ability to incorporate ad-hoc hard constraints and its ability to further improve existing strong architectures found by other NAS methods. Through experiments on ImageNet, we show that GA-NAS can enhance EfficientNet-B0 [Tan and Le, 2019] and ProxylessNAS [Cai *et al.*, 2018] in their respective search spaces, resulting in architectures with higher accuracy and/or smaller model sizes.

2 Related Work

A typical NAS method includes a *search phase* and an *evaluation phase*. This paper is concerned with the search phase, of which the most important performance criteria are robustness, reproducibility and search cost. DARTS [Liu *et al.*, 2018] has given rise to numerous optimization schemes for NAS [Xie *et al.*, 2018]. While the objectives of these algorithms may vary, they all operate in the same or similar search space. However, [Yu *et al.*, 2019] demonstrates that DARTS performs similarly to a random search and its results heavily dependent on the initial seed. In contrast, GA-NAS has a convergence guarantee under certain assumptions and its results are reproducible.

NAS-Bench-301 [Siems *et al.*, 2020] provides a formal benchmark for all 10^{18} architectures in the DARTS search space. Preceding NAS-Bench-301 are NAS-Bench-101 [Ying *et al.*, 2019] and NAS-Bench-201 [Dong and Yang, 2020]. Besides the cell-based searches, GA-NAS also applies to macro-search [Cai *et al.*, 2018; Tan *et al.*, 2019], which searches for an ordering of a predefined set of blocks.

On the other hand, several RL-based NAS methods have been proposed. ENAS [Pham *et al.*, 2018] is the first Reinforcement Learning scheme in weight-sharing NAS. TuNAS [Bender *et al.*, 2020] shows that guided policies decisively exceed the performance of random search on vast search spaces. In comparison, GA-NAS proves to be a highly efficient RL solution to NAS, since the rewards used to train the actor come from the discriminator instead of from costly evaluations.

Hardware-friendly NAS algorithms take constraints such as model size, FLOPS, and inference time into account [Cai *et al.*, 2018; Tan *et al.*, 2019], usually by introducing regularizers into the loss functions. Contrary to these methods, GA-NAS can support ad-hoc search tasks, by enforcing customized hard constraints in importance sampling instead of resorting to approximate penalty terms.

Algorithm 1 GA-NAS Algorithm

1: **Input:** An initial set of architectures $\mathcal{X}_0$; Discriminator D; Generator $G(x;\theta_0)$; A positive integer k;
2: **for** $t = 1, 2, \ldots, T$ **do**
3: $\quad\mathcal{T} \leftarrow$ top k architectures of $\bigcup_{i=0}^{t-1} \mathcal{X}_i$ according to the performance evaluator $S(.)$.
4: $\quad$Train G and D to obtain their parameters

$$(\theta_t, \phi_t) \longleftarrow \arg\min_{\theta_t}\max_{\phi_t} V\left(G_{\theta_t}, D_{\phi_t}\right) = \mathbb{E}_{x\sim p_{\mathcal{T}}}\left[\log D_{\phi_t}(x)\right] + \mathbb{E}_{x\sim p(x;\theta_t)}\left[\log\left(1 - D_{\phi_t}(x)\right)\right].$$

5: $\quad$Let $G(x;\theta_t)$ generate a new set of architectures $\mathcal{X}_t$.
6: $\quad$Evaluate the performance $S(x)$ of every $x \in \mathcal{X}_t$.
7: **end for**

3 The Proposed Method

In this section, we present Generative Adversarial NAS (GA-NAS) as a search algorithm to discover top architectures in an extremely large search space. GA-NAS is theoretically inspired by the importance sampling approach and implemented by a generative adversarial learning framework to promote efficiency.

We can view a NAS problem as a combinatorial optimization problem. For example, suppose that x is a Directed Acyclic Graph (DAG) connecting a certain number of operations, each chosen from a predefined operation set. Let $S(x)$ be a real-valued function representing the performance, e.g., accuracy, of x. In NAS, we optimize $S(x)$ subject to $x \in \mathcal{X}$, where $\mathcal{X}$ denotes the underlying search space of neural architectures.

One approach to solving a combinatorial optimization problem, especially the NP-hard ones, is to view the problem in the framework of *importance sampling* and *rare event simulation* [Rubinstein and Kroese, 2013]. In this approach we consider a family of probability densities $\{p(.;\theta)\}_{\theta\in\Theta}$ on the set $\mathcal{X}$, with the goal of finding a density $p(.;\theta^*)$ that assigns higher probabilities to optimal solutions to the problem. Then, with high probability, the sampled solution from the density $p(.;\theta^*)$ will be an optimal solution. This approach for importance sampling is called the *Cross-Entropy* method.

3.1 Generative Adversarial NAS

GA-NAS is presented in Algorithm 1, where a discriminator D and a generator G are learned over iterations. In each iteration of GA-NAS, the generator G will generate a new set of architectures $\mathcal{X}_t$, which gets combined with previously generated architectures. A set of top performing architectures $\mathcal{T}$, which we also call *true* set of architectures, is updated by selecting top k from already generated architectures and gets improved over time.

Following the GAN framework, originally proposed by [Goodfellow *et al.*, 2014], the discriminator D and the generator G are trained by playing a two-player minimax game whose corresponding parameters ϕ_t and θ_t are optimized by Step 4 of Algorithm 1. After Step 4, G learns to generate architectures to fit to the distribution of $\mathcal{T}$, the true set of architectures.

Specifically, after Step 4 in the t-th iteration, the generator $G(.;\theta_t)$ parameterized by θ_t learns to generate architectures from a family of probability densities $\{p(.;\theta)\}_{\theta\in\Theta}$ such that $p(.;\theta_t)$ will approximate $p_{\mathcal{T}}$, the architecture distribution in true set $\mathcal{T}$, while $\mathcal{T}$ also gets improved for the next iteration by reselecting top k from generated architectures.

We have the following theorem which provides a theoretical convergence guarantee for GA-NAS under mild conditions.

Theorem 3.1. *Let $\alpha > 0$ be any real number that indicates the performance target, such that $\max_{x\in\mathcal{X}} S(x) \geq \alpha$. Define γ_t as a level parameter in iteration t ($t = 0, \ldots, T$), such that the following holds for all the architectures $x_t \in \mathcal{X}_t$ generated by the generator $G(.;\theta_t)$ in the t-th iteration:*

$$S(x_t) \geq \gamma_t, \quad \forall x_t \in \mathcal{X}_t.$$

Choose k, $|\mathcal{X}_t|$, and the γ_t defined above such that $\gamma_0 < \alpha$, and

$$\gamma_t \geq \min(\alpha, \gamma_{t-1} + \delta), \text{ for some } \delta > 0, \ \forall\, t \in \{1, \ldots, T\}.$$

Then, GA-NAS Algorithm can find an architecture x with $S(x) \geq \alpha$ in a finite number of iterations T.

Proof. Refer to Appendix* for a proof of the theorem.

Remark. This theorem indicates that as long as there exists an architecture in $\mathcal{X}$ with performance above α, GA-NAS is guaranteed to find an architecture with $S(x) \geq \alpha$ in a finite number of iterations. From [Goodfellow *et al.*, 2014], the minimax game of Step 4 of Algorithm 1 is equivalent to minimizing the Jensen-Shannon (JS)-divergence between the distribution $p_{\mathcal{T}}$ of the currently top performing architectures $\mathcal{T}$ and the distribution $p(x;\theta_t)$ of the newly generated architectures. Therefore, our proof involves replacing the arguments around the Cross-Entropy framework in importance sampling [Rubinstein and Kroese, 2013] with minimizing the JS divergence, as shown in the Appendix.

3.2 Models

We now describe different components in GA-NAS. Although GA-NAS can operate on any search space, here we describe the implementation of the Discriminator and Generator in the context of cell search. We evaluate GA-NAS for both cell search and macro search in experiments. A *cell* architecture $\mathcal{C}$ is a Directed Acyclic Graph (DAG) consisting of multiple nodes and directed edges. Each intermediate node represents an operator, such as convolution or pooling, from a predefined set of operators. Each directed edge represents the information flow between nodes. We assume that a cell has at least one input node and only one output node.

Architecture Generator. Here we generate architectures in the discrete search space of DAGs. Particularly, we generate architectures in an autoregressive fashion, which is a frequent technique in neural architecture generation such as in ENAS. At each time step t, given a partial cell architecture $\mathcal{C}_t$ generated by the previous time steps, GA-NAS uses an encoder-decoder architecture to decide what new operation to insert and which previous nodes it should be connected to. The encoder is a multi-layer k-GNN [Morris *et al.*, 2019]. The decoder consists of an MLP that outputs the operator probability distribution and a Gated Recurrent Unit (GRU) that recursively determines the edge connections to previous nodes.

*See https://arxiv.org/abs/2105.09356 for the Appendix.

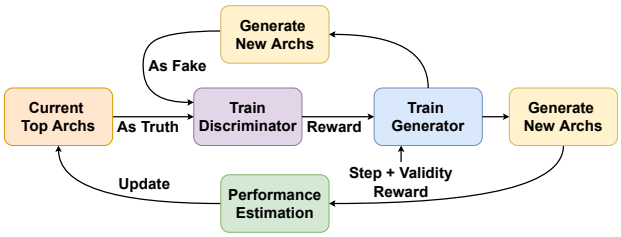

Figure 1: The flow of the proposed GA-NAS algorithm.

Pairwise Architecture Discrimator. In Algorithm 1, $\mathcal{T}$ contains a limited number of architectures. To facilitate a more efficient use of $\mathcal{T}$, we adopt a relativistic discriminator [Jolicoeur-Martineau, 2018] D that follows a Siamese scheme. D takes in a pair of architectures where one is from $\mathcal{T}$, and determines whether the second one is from the same distribution. The discriminator is implemented by encoding both cells in the pair with a shared k-GNN followed by an MLP classifier with details provided in the Appendix.

3.3 Training Procedure

Generally speaking, in order to solve the minimax problem in Step 4 of Algorithm 1, one can follow the minibatch stochastic gradient descent (SGD) training procedure for training GANs as originally proposed in [Goodfellow *et al.*, 2014]. However, this SGD approach is not applicable to our case, since the architectures (DAGs) generated by G are discrete samples, and their performance signals cannot be directly back-propagated to update θ_t, the parameters of G. Therefore, to approximate the JS-divergence minimization between the distribution of top k architectures $\mathcal{T}$, i.e., $p_{\mathcal{T}}$, and the generator distribution $p(x;\theta_t)$ in iteration t, we replace Step 4 of Algorithm 1 by alternately training D and G using a procedure outlined in Algorithm 2. Figure 1 illustrates the overall training flow of GA-NAS.

We first train the GNN-based discriminator using pairs of architectures sampled from $\mathcal{T}$ and $\mathcal{F}$ based on supervised learning. Then, we use Reinforcement Learning (RL) to train G with the reward defined in a similar way as in [You *et al.*, 2018], including a step reward that reflects the validity of an architecture during each step of generation and a final reward that mainly comes from the discriminator prediction D. When the architecture generation terminates, a *final* reward R_{final} penalizes the generated architecture x according to the total number of violations of validity or rewards it with a score from the discriminator $D(x)$ that indicates how similar it is to the current true set $\mathcal{T}$. Both rewards together ensure that G generates valid cells that are structurally similar to top cells from the previous time step. We adopt Proximal Policy Optimization (PPO), a policy gradient algorithm with generalized advantage estimation to train the policy, which is also used for NAS in [Tan *et al.*, 2019].

Algorithm 2 Training Discriminator D and the Generator G

1: **Input:** Discriminator D; Generator $G(x;\theta_{t-1})$; Data set $\mathcal{T}$.
2: **for** $t' = 1, 2, \ldots,\ T'$ **do**
3: Let $G(x;\theta_{t-1})$ generate k random architectures to form the set $\mathcal{F}$.
4: Train discriminator D with $\mathcal{T}$ being positive samples and $\mathcal{F}$ being negative samples.
5: Using the output of $D(x)$ as the main reward signal, train $G(x;\theta_t)$ with Reinforcement Learning.
6: **end for**

Remark. The proposed learning procedure has several benefits. *First,* since the generator must sample a discrete architecture x to obtain its discriminator prediction $D(x)$, the entire generator-discriminator pipeline is not end-to-end differentiable and cannot be trained by SGD as in the original GAN. Training G with PPO solves this differentiability issue. *Second,* in PPO there is an entropy loss term that encourages variations in the generated actions. By tuning the multiplier for the entropy loss, we can control the extent to which we explore a large search space of architectures. *Third,* using the discriminator outputs as the reward can significantly reduce the number of architecture evaluations compared to a conventional scheme where the fully evaluated network accuracy is used as the reward. Ablation studies in Section 4 verifies this claim. Also refer to the Appendix for a detailed discussion.

4 Experimental Results

We evaluate the performance of GA-NAS as a search algorithm by comparing it with a wide range of existing NAS algorithms in terms of convergence speed and stability on NAS benchmarks. We also show the capability of GA-NAS to improve a given network, including already optimized strong baselines such as EfficientNet and ProxylessNAS.

4.1 Results on NAS Benchmarks with or without Weight Sharing

To evaluate search algorithm and decouple it from the impact of search spaces, we query three NAS benchmarks: NAS-Bench-101 [Ying *et al.*, 2019], NAS-Bench-201 [Dong and Yang, 2020], and NAS-Bench-301 [Siems *et al.*, 2020]. The goal is to discover the highest ranked cell with as few queries as possible. The number of queries to the NAS benchmark is used as a reliable measure for search cost in recent NAS literature [Luo *et al.*, 2020; Siems *et al.*, 2020], because each query to the NAS benchmark corresponds to training and evaluating a candidate architecture from scratch, which constitutes the major bottleneck in the overall cost of NAS. By checking the rank an algorithm can reach in a given number of queries, one can also evaluate the convergence speed of a search algorithm. More queries made to the benchmark would indicate a longer evaluation time or higher search cost in a real-world problem.

To further evaluate our algorithm when the true architecture accuracies are unknown, we train a weight-sharing supernet on NAS-Bench-101 and compare GA-NAS with a range of NAS schemes based on weight sharing. **NAS-Bench-101** is the first publicly available benchmark for evaluating NAS algorithms. It consists of 423,624 DAG-style cell-based architectures, each trained and evaluated for 3 times. Metrics for each run include training time and accuracy. Querying NAS-Bench-101 corresponds to evaluating a cell in reality. We provide the results on two setups. In the first setup, we set $|\mathcal{X}_0| = 50$, $|\mathcal{X}_t| = |\mathcal{X}_{t-1}| + 50,\ t \geq 1$, and $k = 25$, In the second setup, we set $|\mathcal{X}_0| = 100$, $|\mathcal{X}_t| = |\mathcal{X}_{t-1}| + 100,\ t \geq 1$, and $k = 50$. For both setups, the initial set $\mathcal{X}_0$ is picked to be a random set, and the number of iterations T is 10. We run each setup with 10 random seeds and the search cost for a run is 8 GPU hours on GeForce GTX 1080 Ti.

Algorithm	Acc (%)	#Q
Random Search †	93.66	2000
RE †	93.97	2000
NAO †	93.87	2000
BANANAS	**94.23**	800
SemiNAS †	94.09	2100
SemiNAS †	93.98	300
GA-NAS-setup1	**94.22**	**150**
GA-NAS-setup2	**94.23**	**378**

Table 1: The best accuracy values found by different search algorithms on NAS-Bench-101 without weight sharing. Note that 94.23% and 94.22% are the accuracies of the 2nd and 3rd best cells. †: taken from [Luo *et al.*, 2020]

Table 1 compares GA-NAS to other methods for the best cell that can be found by querying NAS-Bench-101, in terms of the accuracy and the rank of this cell in NAS-Bench-101, along with the number of queries required to find that cell. Table 2 shows the average performance of GA-NAS in the same experiment over multiple random seeds. Note that Table 1 does not list the average performance of other methods except Random Search, since all the other methods in Table 1 only reported their single-run performance on NAS-Bench-101 in their respective experiments.

In both tables, we find that GA-NAS can reach a higher accuracy in fewer number of queries, and beats the best published results, i.e., BANANAS [White *et al.*, 2019] and SemiNAS [Luo *et al.*, 2020] by an obvious margin. Note that 94.22 is the 3rd best cell while 94.23 is the 2nd best cell in NAS-Bench-101. From Table 2, we observe that GA-NAS achieves superior stability and reproducibility: GA-NAS-setup1 consistently finds the 3rd best in 9 runs and the 2nd best in 1 run out of 10 runs; GA-NAS-setup2 finds the 2nd best in 5 runs and the 3rd best in the other 5 runs.

To evaluate GA-NAS when true accuracy is not available, we train a weight-sharing supernet on the search space of NAS-Bench-101 (with details provided in Appendix) and report the true test accuracies of architectures found by GA-NAS. We use the supernet to evaluate the accuracy of a cell on a validation set of 10k instances of CIFAR10 (see Appendix). Search time including supernet training is around 2 GPU days.

We report results of 10 runs in Table 3, in comparison to other weight-sharing NAS schemes reported in [Yu *et al.*, 2019]. We observe that using a supernet degrades the search performance in general as compared to true evaluation, because weight-sharing often cannot provide a completely reli-

Algorithm	Mean Acc (%)	Mean Rank	Average #Q
Random Search	93.84 ± 0.13	498.80	648
Random Search	93.92 ± 0.11	211.50	1562
GA-NAS-Setup1	**94.22** ± 4.45e-5	2.90	647.50 ± 433.43
GA-NAS-Setup2	**94.23** ± 7.43e-5	2.50	1561.80 ± 802.13

Table 2: The average statistics of the best cells found on NAS-Bench-101 without weight sharing, averaged over 10 runs (with std shown). Note that we set the number of queries (#Q) for Random Search to be the same as the average number of queries incurred by GA-NAS.

Algorithm	Mean Acc	Best Acc	Best Rank
DARTS †	92.21 ± 0.61	93.02	57079
NAO †	92.59 ± 0.59	93.33	19552
ENAS †	91.83 ± 0.42	92.54	96939
GA-NAS	**92.80 ± 0.54**	**93.46**	**5386**

Table 3: Searching on NAS-Bench-101 with weight-sharing, with the mean test accuracy of the best cells from 10 runs, and the best accuracy/rank found by a single run. †: taken from [Yu *et al.*, 2019]

able performance for the candidate architectures. Nevertheless, GA-NAS outperforms other approaches.

NAS-Bench-201

This search space contains 15,625 evaluated cells. Architecture cells consist of 6 searchable edges and 5 candidate operations. We test GA-NAS on NAS-Bench-201 by conducting 20 runs for CIFAR-10, CIFAR-100, and ImageNet-16-120 using the true test accuracy. We compare against the baselines from the original NAS-Bench-201 paper [Dong and Yang, 2020] *that are also directly querying the benchmark data.* Since no information on the rank achieved and the number of queries is reported for these baselines, we also compare GA-NAS to Random Search (RS-500), which evaluates 500 unique cells in each run. Table 4 presents the results. We observe that GA-NAS outperforms a wide range of baselines, including Evolutionary Algorithm (REA), Reinforcement Learning (REINFORCE), and Bayesian Optimization (BOHB), on the task of finding the most accurate cell with much lower variance. Notably, on ImageNet-16-120, GA-NAS outperforms REA by nearly 1.3% on average.

Compared to RS-500, GA-NAS finds cells that are higher ranked while only exploring less than 3.2% of the search space in each run. It is also worth mentioning that in the 20 runs on all three datasets, GA-NAS can find the best cell in the entire benchmark more than once. Specifically for CIFAR-10, it found the best cell in 9 out of 20 runs.

NAS-Bench-301

This is another recently proposed benchmark based on the same search space as DARTS. Relying on surrogate performance models, NAS-Bench-301 reports the accuracy of 10^{18} unique cells. We are especially interested in how the number of queries (#Q) needed to find an architecture with high accuracy scales in a large search space. We run GA-NAS on NAS-Bench-301 v0.9. We compare with Random (RS) and Evolutionary (EA) search baselines. Figure 2 plots the average best accuracy along with the accuracy standard deviations versus the number of queries incurred under the three methods. We observe that GA-NAS outperforms RS at all query budgets and outperforms EA when the number of queries exceeds 3k.

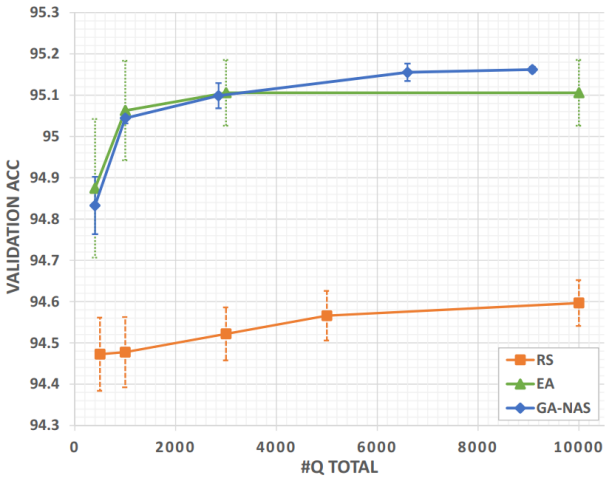

Figure 2: NAS-Bench-301 results comparing the means/standard deviations of the best accuracy found at various total query limits.

The results on NAS-Bench-301 confirm that for GA-NAS, the number of queries (#Q) required to find a good performing cell scales well as the size of the search space increases. For example, on NAS-Bench-101, GA-NAS usually needs around 500 queries to find the 3rd best cell, with an accuracy $\approx 94\%$ among 423k candidates, while on the huge search space of NAS-Bench-301 with up to 10^{18} candidates, it only needs around 6k queries to find an architecture with accuracy approximately equal to 95%.

In contrast to GA-NAS, EA search is less stable and does not improve much as the number of queries increases over 3000. GA-NAS surpasses EA for #Q $\geq$ 3000 in Figure 2. What is more important is that the variance of GA-NAS is much lower than the variance of the EA solution over all ranges of #Q. Even though for #Q < 3000, GA-NAS and EA are close in terms of average performance, EA suffers from a huge variance.

Ablation Studies

A key question one might raise about GA-NAS is how much the discriminator contributes to the superior search performance. Therefore, we perform an ablation study on NAS-Bench-101 by creating an *RL-NAS* algorithm for comparison. RL-NAS removes the discriminator in GA-NAS and directly queries the accuracy of a generated cell from NAS-Bench-101 as the reward for training. We test the performance of RL-NAS under two setups that differ in the total number of queries made to NAS-Bench-101. Table 5 reports the results.

Compared to GA-NAS-Setup2, RL-NAS-1 makes $3\times$ more

	CIFAR-10			CIFAR-100			ImageNet-16-120		
Algorithm	**Mean Acc**	**Rank**	**#Q**	**Mean Acc**	**Rank**	**#Q**	**Mean Acc**	**Rank**	**#Q**
REA †	93.92 ± 0.30	-	-	71.84 ± 0.99	-	-	45.54 ± 1.03	-	-
RS †	93.70 ± 0.36	-	-	71.04 ± 1.07	-	-	44.57 ± 1.25	-	-
REINFORCE †	93.85 ± 0.37	-	-	71.71 ± 1.09	-	-	45.24 ± 1.18	-	-
BOHB †	93.61 ± 0.52	-	-	70.85 ± 1.28	-	-	44.42 ± 1.49	-	-
RS-500	94.11 ± 0.16	30.81	500	72.54 ± 0.54	30.89	500	46.34 ± 0.41	34.18	500
GA-NAS	$\mathbf{94.34 \pm 0.05}$	**4.05**	444	$\mathbf{73.28 \pm 0.17}$	**3.25**	444	$\mathbf{46.80 \pm 0.29}$	**7.40**	445

† The results are taken directly from NAS-Bench-201 [Dong and Yang, 2020].

Table 4: Searching on NAS-Bench-201 without weight sharing, with the mean accuracy and rank of the best cell found reported. #Q represents the average number of queries per run. We conduct 20 runs for GA-NAS.

Algorithm	**Avg. Acc**	**Avg. Rank**	**Avg. #Q**
RL-NAS-1	94.14 ± 0.10	20.8	7093 ± 3904
RL-NAS-2	93.78 ± 0.14	919.0	314 ± 300
GA-NAS-Setup1	94.22± 4.5e-5	2.9	648 ± 433
GA-NAS-Setup2	94.23± 7.4e-5	2.5	1562 ± 802

Table 5: Results of ablation study on NAS-Bench-101 by removing the discriminator and directly queries the benchmark for reward.

queries to NAS-Bench-101, yet cannot outperform either of the GA-NAS setups. If we instead limits the number of queries as in RL-NAS-2 then the search performance deteriorates significantly. Therefore, we conclude that the discriminator in GA-NAS is crucial for reducing the number of queries, which converts to the number of evaluations in real-world problems, as well as for finding architectures with better performance. Interested readers are referred to Appendix for more ablation studies as well as the Pareto Front search results.

4.2 Improving Existing Neural Architectures

We now demonstrate that GA-NAS can improve existing neural architectures, including ResNet and Inception cells in NAS-Bench-101, EfficientNet-B0 under hard constraints, and ProxylessNAS-GPU [Cai *et al.*, 2018] in unconstrained search.

For ResNet and Inception cells, we use GA-NAS to find better cells from NAS-Bench-101 under a lower or equal training time and number of weights. This can be achieved by enforcing a hard constraint in choosing the truth set $\mathcal{T}$ in each iteration. Table 6 shows that GA-NAS can find new, dominating cells for both cells, showing that it can enforce ad-hoc constraints in search, a property not enforceable by regularizers in prior work. We also test Random Search under a

ResNet			
Algorithm	**Best Acc**	**Train Seconds**	**#Weights (M)**
Hand-crafted	93.18	2251.6	20.35
Random Search	93.84	1836.0	10.62
GA-NAS	**93.96**	1993.6	11.06
Inception			
Algorithm	**Best Acc**	**Train Seconds**	**#Weights (M)**
Hand-crafted	93.09	1156.0	2.69
Random Search	93.14	1080.4	2.18
GA-NAS	**93.28**	1085.0	2.69

Table 6: Constrained search results on NAS-Bench-101. GA-NAS can find cells that are superior to the ResNet and Inception cells in terms of test accuracy, training time, and the number of weights.

Network	**#Params**	**Top-1 Acc**
EfficientNet-B0 (no augment)	5.3M	76.7
GA-NAS-ENet-1	**4.6**M	76.5
GA-NAS-ENet-2	**5.2**M	**76.8**
GA-NAS-ENet-3	5.3M	**76.9**
ProxylessNAS-GPU	4.4M	75.1
GA-NAS-ProxylessNAS	4.9M	**75.5**

Table 7: Results on the EfficientNet and ProxylessNAS spaces.

similar number of queries to the benchmark under the same constraints, which is unable to outperform GA-NAS.

We now consider well-known architectures found on ImageNet i.e., EfficientNet-B0 and ProxylessNAS-GPU, which are already optimized strong baselines found by other NAS methods. We show that GA-NAS can be used to improve a given architecture in practice by searching in its original search space.

For EfficientNet-B0, we set the constraint that the found networks all have an equal or lower number of parameters than EfficientNet-B0. For the ProxylessNAS-GPU model, we simply put it in the starting truth set and run an unconstrained search to further improve its top-1 validation accuracy. More details are provided in the Appendix. Table 7 presents the improvements made by GA-NAS over both existing models. Compared to EfficientNet-B0, GA-NAS can find new single-path networks that achieve comparable or better top-1 accuracy on ImageNet with an equal or lower number of trainable weights. We report the accuracy of EfficientNet-B0 and the GA-NAS variants *without data augmentation*. Total search time including supernet training is around 680 GPU hours on Tesla V100 GPUs. It is worth noting that the original EfficientNet-B0 is found using the MNasNet [Tan *et al.*, 2019] with a search cost over 40,000 GPU hours.

For ProxylessNAS experiments, we train a supernet on ImageNet and conduct an unconstrained search using GA-NAS for 38 hours on 8 Tesla V100 GPUs, a major portion of which, i.e., 29 hours is spent on querying the supernet for architecture performance. Compared to ProxylessNAS-GPU, GA-NAS can find an architecture with a comparable number of parameters and a better top-1 accuracy.

5 Conclusion

In this paper, we propose Generative Adversarial NAS (GA-NAS), a provably converging search strategy for NAS prob-

lems, based on generative adversarial learning and the importance sampling framework. Based on extensive search experiments performed on NAS-Bench-101, 201, and 301 benchmarks, we demonstrate the superiority of GA-NAS as compared to a wide range of existing NAS methods, in terms of the best architectures discovered, convergence speed, scalability to a large search space, and more importantly, the stability and insensitivity to random seeds. We also show the capability of GA-NAS to improve a given practical architecture, including already optimized ones either manually designed or found by other NAS methods, and its ability to search under ad-hoc constraints. GA-NAS improves EfficientNet-B0 by generating architectures with higher accuracy or lower numbers of parameters, and improves ProxylessNAS-GPU with enhanced accuracies. These results demonstrate the competence of GA-NAS as a stable and reproducible search algorithm for NAS.

References

[Bender *et al.*, 2020] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14323–14332, 2020.

[Cai *et al.*, 2018] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[Dai *et al.*, 2020] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. Fbnetv3: Joint architecture-recipe search using neural acquisition function. *arXiv preprint arXiv:2006.02049*, 2020.

[Dong and Yang, 2020] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[Elsken *et al.*, 2018] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[Goodfellow *et al.*, 2014] 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.

[Jolicoeur-Martineau, 2018] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

[Kandasamy *et al.*, 2018] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in neural information processing systems*, pages 2016–2025, 2018.

[Li and Talwalkar, 2020] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR, 2020.

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

[Luo *et al.*, 2020] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *arXiv preprint arXiv:2002.10389*, 2020.

[Morris *et al.*, 2019] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[Pham *et al.*, 2018] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[Rubinstein and Kroese, 2013] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[Siems *et al.*, 2020] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

[Tan and Le, 2019] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[Tan *et al.*, 2019] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[White *et al.*, 2019] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *arXiv preprint arXiv:1910.11858*, 2019.

[Xie *et al.*, 2018] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[Yang *et al.*, 2019] Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. *arXiv preprint arXiv:1912.12522*, 2019.

[Ying *et al.*, 2019] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114, 2019.

[You *et al.*, 2018] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in neural information processing systems*, pages 6410–6421, 2018.

[Yu *et al.*, 2019] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.