Learning Where To Look – Generative NAS is Surprisingly Efficient

Jovita Lukasik[1,2*], Steffen Jung[2*], and Margret Keuper[2,3]

[1] University of Mannheim
[2] Max Planck Institute for Informatics, Saarland Informatics Campus
[3] University of Siegen

Abstract. The efficient, automated search for well-performing neural architectures (NAS) has drawn increasing attention in the recent past. Thereby, the predominant research objective is to reduce the necessity of costly evaluations of neural architectures while efficiently exploring large search spaces. To this aim, surrogate models embed architectures in a latent space and predict their performance, while generative models for neural architectures enable optimization-based search within the latent space the generator draws from. Both, surrogate and generative models, have the aim of facilitating query-efficient search in a well-structured latent space. In this paper, we further improve the trade-off between query-efficiency and promising architecture generation by leveraging advantages from both, efficient surrogate models and generative design. To this end, we propose a generative model, paired with a surrogate predictor, that iteratively learns to generate samples from increasingly promising latent subspaces. This approach leads to very effective and efficient architecture search, while keeping the query amount low. In addition, our approach allows in a straightforward manner to jointly optimize for multiple objectives such as accuracy and hardware latency. We show the benefit of this approach not only w.r.t. the optimization of architectures for highest classification accuracy but also in the context of hardware constraints and outperform state-of-the-art methods on several NAS benchmarks for single and multiple objectives. We also achieve state-of-the-art performance on ImageNet. The code is available at `https://github.com/jovitalukasik/AG-Net`.

Keywords: neural architecture search, generative model

1 Introduction

The first image classification network [20] applied to the large-scale visual recognition challenge ImageNet [8] achieved unprecedented results. Since then, the main driver of improvement on this challenge are new architecture designs [38,40], [41,14] that, ultimately, lead to architectures surpassing human performance [13]. Since manual architecture design requires good intuition and a huge amount of

[*]Authors contributed equally.

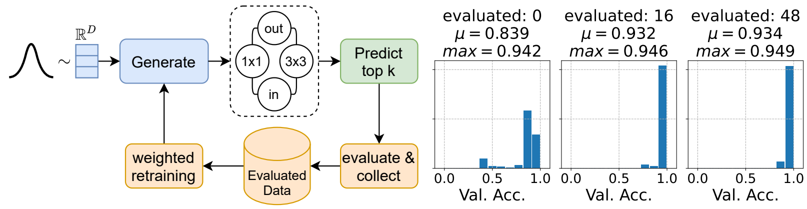

Fig. 1: (**left**) Our search method generates architectures from points in an architecture representation space that is iteratively optimized. (**right**) The architecture representation space is biased towards better-performing architectures with each search iteration. After only 48 evaluated architectures, our generator produces state-of-the-art performing architectures on NAS-Bench-101.

trial-and-error, the automated approach of neural architecture search (NAS) receives growing interest [32,58,54,10,18,21]. Well-performing architectures can be found by applying common search practices like random search [2], evolutionary search [32,31], Bayesian optimization (BO) [16,35,45], or local search [46] on discrete architecture search spaces, such as NAS-Bench-101, NAS-Bench-201, DARTS and NAS-Bench-NLP [54,10,25,18]. However, these methods are inefficient because they require to evaluate thousands of architectures, resulting in impracticable search times. Recent approaches avoid this problem of immense computation costs by either training surrogate models to approximate the performance of an architecture [25,3] or by generating architectures based on learned architecture representation spaces [56,26]. Both methods aim to improve the query efficiency, which is crucial in NAS, since every query implies a full training and evaluation of the neural architecture on the underlying target dataset.

This trade-off between query efficiency and resulting high-scoring architectures is an active research field. Yet, no attempts were made so far to leverage the advantages of both search paradigms. Therefore, we propose a model that incorporates the focus of promising architectures already in the architecture generation process by optimizing the latent space *directly*: We let the generator learn in which areas of the data distribution to look for promising architectures. This way, we reduce the query amount even further, resulting in a query efficient and very effective NAS method. Our proposed method is inspired by a latent space optimization (LSO) technique [42], originally used in the context of variational autoencoders [17] to optimize generated images or arithmetic expressions using BO. We adapt this concept to NAS and pair it with an architecture performance predictor in an end-to-end learning setting, so that it allows us to iteratively reshape the architecture representation space. Thereby, we promote desired properties of generated architectures in a highly query-efficient way, i.e. by learning expert generators for promising architectures. Since we couple the generation process with a surrogate model to predict desired properties such as high accuracy or low latency of generated architectures, there is no need in our method for BO in the generated latent space, making our method even more efficient.

In practice, we pretrain, on a target space of neural architectures, a GNN-based generator network, which does not rely on any architecture evaluation and is therefore fast and query-free. The generator is trained in a novel generative setting that directly compares generated architectures to randomly sampled architectures using a reconstruction loss without the need of a discriminator network as in generative adversarial networks (GANs) [12] or an encoder as in variational autoencoders (VAEs) [17]. We use an MLP as a surrogate to rank performances and hardware properties of generated architectures. In contrast, previous generative methods either rely on training and evaluating supernets [15], which are expensive to train and dataset specific, or pretrain a latent space and search within this space directly using BO [56,53,26], reinforcement learning (RL) [33] or gradient based methods [27]. These methods incorporate either GANs, which can be hard to train or VAEs, which are biased by the regularization, whereas our plain generative model is easy to train. In addition we enable backpropagation from the performance predictor to the generator. Thereby, the generator can efficiently learn which part of the architecture search space is promising with only few evaluated architectures.

By extensive experiments on common NAS benchmarks [54,10,37,18,21] as well as ImageNet [8], we show that our method is effective and sample-efficient. It reinforces the generator network to produce architectures with improving validation accuracy (see Figure 1), as well as in improving on hardware-dependent latency constraints (see Figure 4) while keeping the number of architecture evaluations small. In summary, we make the following contributions:

- We propose a simple model that learns to focus on promising regions of the architecture space. It can thus learn to generate high-scoring architectures from only a few queries.
- We learn architecture representation spaces via a *novel generative design* that is able to generate architectures stochastically while being trained with a simple reconstruction loss. Unlike VAEs [17] or GANs [12], no encoder network nor discriminator network is necessary.
- Our model allows sample-efficient search and achieves state-of-the-art results on several NAS benchmarks as well as on ImageNet. It allows joint optimization w.r.t. hardware properties in a straightforward way.

2 Related Work

Neural Architecture Search Intrinsically, Neural Architecture Search (NAS) is a discrete optimization problem seeking the optimal configuration of operations (such as convolutions, poolings and skip connections) in a constrained *search space* of computational graphs. To enable benchmarking within the NAS community, different search spaces have been proposed. The tabular benchmarks NAS-Bench-101 [54] and NAS-Bench-201 [10] provide both an exhaustive covering of metrics and performances. NAS-bench-NLP [18] provides a search space for natural language processing. In addition to these tabular benchmarks NAS-Bench-301 [37] provides a surrogate benchmark, which allows for fast evaluation of NAS methods on the DARTS [25] search space by querying the validation accuracy. NAS-Bench-x11 [52] is another surrogate benchmark. It outputs full training information for each architecture in all four mentioned benchmarks. NAS-Bench-Suite [28] facilitates reproducible search on these NAS benchmarks.

Early NAS approaches are based on discrete encodings of search spaces, such as in the form of adjacency matrices, and can be distinguished by their *search strategy*. Examples are random search [2,22], reinforcement learning (RL) [57,23], evolutionary methods [32,31], local search [46], and Bayesian optimization (BO) [16,35]. Recent NAS methods shift from discrete optimization to faster weight-sharing approaches, resulting in differentiable optimization methods [30,25,1,3,49,55]. Several approaches map the discrete search space into a continuous architecture representation space [27,56,53,26] and search or optimize within this space using for example BO (e.g. [53]) or gradient-based point operation [27]. In this paper, we also learn continuous architecture representation spaces. However, in contrast to former works, we propose to optimize the representation space, instead of performing point optimization within a fixed space such as e.g. [27]. A survey of different strategies can be found in [11].

All NAS approaches are dependent on *performance estimation* of intermediate architectures. To avoid the computation heavy training and evaluation of queries on the target dataset, methods to approximate the performance have been explored [47]. Common approaches include neural predictors that take path encodings [45] or graph embeddings learned by GNNs [36,43] as input. Weak-NAS [48] proposes to progressively evaluate the search space towards finding high-performing architectures using a set of weak predictors. In our method, we integrate a weak expert predictor with a generator to yield an efficient interplay between predicting for high-performing architectures and generating them.

Graph Generative Models Most graph generation models in NAS employ variational autoencoders (VAE) [17]. [27] uses an LSTM-based VAE, coupled with performance prediction for gradient-based architecture optimization. Note that [27] optimizes the latent point in a fixed latent space while our approach optimizes the latent space itself. [56] use GNNs with asynchronous message-passing to train a VAE for BO. [15] combines a generator with a supernet and searches for neural architectures for different device information. [53] facilitates [50] with an MLP decoder. [26] proposes smooth variational graph embeddings (SVGe) using two-sided GNNs to capture the information flow within a neural architecture.

Our proposed model's generator is inspired by SVGe with the aim to inherit its flexible applicability to various search spaces. Yet, similar to [53], due to the intrinsic discretization and training setting, SVGe does not allow for backpropagation. Recently, [33] facilitates GNNs in a GAN [12] setting, where the backpropagation issue is circumvented using reinforcement learning. In contrast, our proposed GNN generator circumvents the intermediate architecture discretization and can therefore be trained by a single reconstruction loss using backpropagation. Its iterative optimization is inspired by [42], who proposes to use a VAE with weighted retraining w.r.t. a target function to adapt the latent space for the optimization of images and arithmetic functions using BO. Our

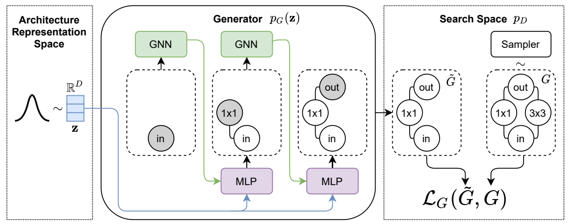

Fig. 2: Representation of the training procedure for our generator in AG-Net. The input is a randomly sampled latent vector $\mathbf{z} \in \mathbb{R}^d$. First, the input node is generated, initialized and input to a GNN to generate a partial graph representation. The learning process iteratively generates node scores and edge scores using $\mathbf{z}$ and the partial graph representation until the output node is generated. The target for this generated graph is a randomly sampled architecture.

model transfers the idea of weighted retraining to NAS. It uses our plain generator and improves sample efficiency by employing a differentiable surrogate model on the target function such that, in contrast to [42], no further black-box optimization step is needed. Next, we describe the proposed generator network.

3 Architecture Generative Model

Preliminaries We aim to generate neural networks represented as directed acyclic graphs (DAG). This representation is in line with the cell-based architecture search spaces commonly used as tabular benchmarks [54,10]. Each cell is a DAG $G = (V, E)$, with nodes $v \in V$ and edges $e \in E$. The graph representations differ between the various benchmarks in terms of their labeling of operations. For example in NAS-Bench-101 [54] each node is associated with an operation, whereas in NAS-Bench-201 [10] each edge is associated with an operation.

Generative Network Commonly used graph generative networks are based on variational autoencoders (VAE) [17]. In contrast, our proposed network is a *purely generative* network, p_G (see Figure 2). To generate valid graphs, we build our model similar to the graph decoder from the VAE approach SVGe [26]. The generator takes a randomly sampled variable $\mathbf{z} \sim \mathcal{N}(0, 1)$ as input and reconstructs a randomly sampled graph from the cell-based search space. The model iteratively builds the graph: it starts with generating the input node v_0, followed by adding subsequent nodes v_i and their labels and connecting them with edges $e_{(j,i)}, j < i$, until the end node v_T with the label *output* is generated. Additionally, we want to learn a surrogate for performance prediction on the generated data and allow for end-to-end training of both. To allow for backpropagation, we need to adapt several details of the generator model. We initialize the node-attributes for each node by one-hot encoded vectors, which are initialized during training using a 2-layer MLP to replace the learnable look-up table proposed in SVGe. The output of our generator is a vector graph representation consisting of a concatenation of generated node scores and edge scores. It is important to note that the iterative generation process is independent of the ground truth data, which are only used as a target for the reconstruction loss. Note that the end-to-end trainability of the proposed generator is a prerequisite for our model: It allows to pair the generator with a learnable performance predictor such that information on the expected architectures' accuracy can be learned by the generator. This enables a stronger coupling with the predictor's target for the generation process and higher query efficiency (see subsection 4.4). In contrast, previous models such as [15,26,53] are not fully differentiable and do not allow such optimization. Our generative model is pretrained on the task of reconstructing neural architectures, where for each randomly drawn latent space sample, we evaluate the reconstruction loss to a randomly drawn architecture. This simple procedure is facilitated by the heavily constrained search spaces of neural architectures, making it easy for the model to learn to generate valid architectures without being supported by a discriminator model as in generative adversarial networks (GANs) [12]. An evaluation of the generation ability of our model and implementation details are provided in the supp. mat. section D.

Performance Predictor This generative model is coupled with a simple surrogate model, a 4-layer MLP with ReLU non-linearities, for target predictions C. These targets can be validation or test accuracy of the generated graph, or the latency with respect to a certain hardware. For comparison, we also include a tree-based method, XGBoost (XGB) [4] as an alternative prediction model. XGB[4] is used as a surrogate model in NAS-Bench-301 [37] and shows high prediction abilities. The input to XGB is the vector representation of the architectures. Since this method is non-differentiable, we additionally include a gradient estimation for rank-based metrics [34]. This way, we are able to include gradient information to the generator. Yet, it is important to note, that this approach is not fully differentiable. This comparison will allow us to measure the trade-off between using supposedly stronger predictors over the capability to allow for full end-to-end learning.

Training Objectives The generative model p_G learns to reconstruct a randomly sampled architecture G from search space p_D given a randomly sampled latent vector $\mathbf{z} \sim \mathcal{N}(0,1)$. The objective function for this generation process can be formulated as the sum of node-level loss $\mathcal{L}_V$ and edge-level loss $\mathcal{L}_E$:

$$\mathcal{L}_G(\tilde{G}, G) = \mathcal{L}_V + \mathcal{L}_E;\ \tilde{G} \sim p_G(\mathbf{z});\ G \sim p_D, \tag{1}$$

where $\mathcal{L}_V$ is the Cross-Entropy loss between the predicted and the ground truth nodes and $\mathcal{L}_E$ is the Binary-Cross Entropy loss between the predicted and ground truth edges of the generated graph $\tilde{G}$. This training step is *completely unsupervised*. Figure 2 presents an overview of the training process. To include the training of the surrogate model, the objective function is reformulated to:

$$\mathcal{L}(\tilde{G}, G) = (1-\alpha)\mathcal{L}_G(\tilde{G}, G) + \alpha\mathcal{L}_C(\tilde{G}, G), \tag{2}$$

where α is a hyperparameter to trade-off generator loss $\mathcal{L}_G$ and prediction loss $\mathcal{L}_C$ for the prediction targets C of graph G. We set the predictor loss as an MSE. Furthermore, each loss is optimized using mini-batch gradient descent.

Generative Latent Space Optimization (LSO) To facilitate the generation process, we optimize the architecture representation space via weighted retraining [42], resulting in a sample efficient search algorithm. The intuition of this approach is to place more probability mass on high-scoring latent points, (e.g. high performing or low latency architectures) and less mass on low-scoring points. Thus, this strategy does not discard low-scoring architectures completely, which would be inadequate for proper learning. The generative model is therefore trained on a data distribution that systematically increases the probability of high-scoring latent points. This can be done by simply assigning a weight w_i to each data point $G_i \sim p_D$, indicating its likelihood to occur during batch-wise training. In addition, the training objective is weighted via a weighted empirical mean $\sum_{G_i \sim p_D} w_i \, \mathcal{L}$ for each data point. As for the weights itself, [42] proposed a rank-based weight function

$$\begin{aligned} w(G; p_D, k) &\propto \frac{1}{kN + \mathrm{rank}_{f,p_D}(G)} \\ \mathrm{rank}_{f,p_D}(x) &= |\{G_i : f(G_i) > f(G), G_i \sim p_D\}|, \end{aligned} \tag{3}$$

where $f(\cdot)$ is the evaluation function of the architecture G_i; for NAS-Bench-101 [54] and NAS-Bench-201 [10] it is the tabular benchmark entry, for NAS-Bench-301 [37] and NAS-Bench-NLP [18] it is the surrogate benchmark prediction. Similar to [42], we set $k = 10e - 3$. The retraining procedure itself then consists of finetuning the pretrained generative model coupled with the surrogate model, where loss functions and data points are both weighted by $w(G; p_D, k)$.

4 Experiments

We evaluate the proposed simple architecture generative network (AG-Net) on the two commonly used tabular benchmarks NAS-Bench-101 [54] and NAS-Bench-201 [10], the surrogate benchmarks NAS-Bench-301 [37] evaluated on the DARTS search space [25], NAS-Bench-NLP [18] and the first hardware device induced benchmark [21]. Additionally we perform experiments on the ImageNet [8] classification task and show state-of-the-art performance on the DARTS search space. In our experiments in subsection 4.3 for the Hardware-Aware Benchmark we consider the latency information on the NAS-Bench-201 search space. Details about all hyperparameters are given in the supp. mat. section E.

4.1 Experiments on Tabular Benchmarks

NAS-Bench-101 For our experiments on NAS-Bench-101, we first pretrain our generator for generating valid graphs on the NAS-Bench-101 search space. This step does not require information about the performance of architectures and is therefore inexpensive. The pretrained generator is then used for all experiments on NAS-Bench-101. Our NAS algorithm is initialized by randomly sampling 16 architectures from the search space, which are then weighted by the weighting function $\mathcal{W} = w(G)_{G \sim p_D}$. Then, latent space optimized architecture search is performed by iteratively retraining the generator coupled with the MLP surrogate model for 15 epochs and generating 100 architectures of which the top 16 (according to their accuracy prediction) are evaluated and added to the training data. This step is repeated until the desired number of queries is reached. When generating architectures, we sample from a grid, containing the 99%-quantiles from $\mathcal{N}(0, 1)$ uniformly distributed. This way, we sample more distributed latent variables for better latent space coverage. We compare our method to the VAE-based search method Arch2vec [53] and predictor based model WeakNAS [48], as well as state-of-the-art methods, such as NAO [27]‡, random search [22], local search [46], Bayesian optimization [39], regularized evolution [31] and BANANAS [45]†. Additionally, we compare the proposed AG-Net to the model using an XGBoost Predictor (see section C). The results of this comparison are listed in Table 1. Here, we report the mean over 10 runs. Results including the standard deviation can be found in the supp. mat. Note, we search for the architecture with the best validation accuracy and report the corresponding test accuracy. Furthermore, we plot the search progress in Figure 3 (bottom left). As we can see, our model AG-Net improves over all state-of-the-art methods, not only at the last query of 300 data points, reaching a top 1 test accuracy of 94.2%, but is also almost any time better during the search process.

A direct comparison to the recently proposed GANAS [33] on NAS-Bench-101 is difficult, since GANAS searches on NAS-Bench-101 until they find the best architecture in terms of validation accuracy, whereas we limit our search to a maximal amount of 192 queries and are able to find high-performing architectures already in this small query setting. The comparison of AG-Net to the generator paired with an XGBoost [4] predictor shows that our end-to-end learnable approach is favorable even over potentially stronger predictors.

NAS-Bench-201 This benchmark contains three different image classification tasks: CIFAR-10, CIFAR-100 [19] and ImageNet16-120 [7]. For the experiments on NAS-Bench-201[10] we retrain AG-Net in the weighted manner for 30 epochs. In this setting, we also compare AG-Net to two recent generative models [33,15]. SGNAS [15] trains a supernet by uniform sampling, following SETN [9]. Additionally a CNN based architecture generator is trained to search architectures on the supernet. When comparing with [53], we also adopt their evaluation scheme of adding only the best-performing architecture (top-1) to the training data instead of top-16 as in our other experiments.

We report the search results for different numbers of queries for the NAS-Bench-201 dataset in Table 2. In addition, we plot the search progress in terms of queries in Figure 3 (top). Our method provides state-of-the-art results on all datasets for a varying number of queries. Most importantly, AG-Net shows strong performance in the few-query regime compared to [53] with the exception of CIFAR-100, proving its high query efficiency.

---

‡We reran this experiment using the implementation from [47].

†We reran these experiments using the official implementation from [44,45,46], with the same initial training data and amount of top k architectures as for AG-Net.

Table 1: Results on NAS-Bench-101 for the search of the best architecture in terms of validation accuracy on CIFAR-10 to state-of-the-art methods (mean over 10 trials).

NAS Method	Val. Acc (%)	Test Acc (%)	Queries
Optimum*	95.06	94.32	
Arch2vec + RL [53]	-	94.10	400
Arch2vec + BO [53]	-	94.05	400
NAO ‡[27]	94.66	93.49	192
BANANAS† [45]	94.73	94.09	192
Bayesian Optimization† [39]	94.57	93.96	192
Local Search† [46]	94.57	93.97	192
Random Search†[22]	94.31	93.61	192
Regularized Evolution†[31]	94.47	93.89	192
WeakNAS [48]	-	**94.18**	200
XGB (ours)	94.61	94.13	192
XGB + ranking (ours)	94.60	94.14	192
AG-Net (ours)	**94.90**	**94.18**	192

4.2 Experiments on Surrogate Benchmarks

We furthermore apply our search method on larger search spaces as DARTS [25] and NAS-Bench-NLP [18] without ground truth evaluations for the whole search space, making use of surrogate benchmarks as NAS-Bench-301 [37], NAS-Bench-X11 [52] and NAS-Bench-Suite [28].

NAS-Bench-301 Here, we report experiments on the cell-based DARTS [25] search space using the surrogate benchmark NAS-Bench-301 [37] for the CIFAR-10 [19] image classification task. The exact search procedure using the cells individually is described in the supp. mat. subsection C.5. The results are described in Table 3 (left) and visualized in Figure 3 (bottom middle). Our method is comparable to other state-of-the-art methods in this search space.

NAS-Bench-NLP Next, we evaluate AG-Net on NAS-Bench-NLP [18] for the language modeling task on Penn TreeBank [29]. We retrain AG-Net coupled with the surrogate model for 30 epochs to predict the validation perplexity. Note, since the search space considered in NAS-Bench-NLP is too large for a full tabular benchmark evaluation, we make use of the surrogate benchmark NAS-Bench-X11 [52] and NAS-Bench-Suite [28] instead of tabular entries.

For fair comparison we compare our methods to the same state-of-the-art methods as in previous experiments. The results are reported in Table 3 (right) and visualized in Figure 3 (bottom right). Our AG-Net improves over all state-of-the-art methods by a substantial margin and using XGB as a predictor even improves the search further.

ImageNet Experiments The previous experiment on NAS-Bench-301 [37] shows the ability of our generator to generate valid architectures and to perform well in the DARTS [25] search space. This allows for searching a well-performing architecture on ImageNet [8]. Yet evaluating up to 300 different found architec-

Table 2: Architecture Search on NAS-Bench-201. We report the mean over 10 trials for the search of the architecture with the highest validation accuracy.

NAS Method	CIFAR-10		CIFAR-100		ImageNet16-120		Queries	Search Method
	Val. Acc	Test Acc	Val. Acc	Test Acc	Val. Acc	Test Acc		
Optimum*	91.61	94.37	73.49	73.51	46.77	47.31		
SGNAS [15]	90.18	93.53	70.28	70.31	44.65	44.98		Supernet
Arch2vec + BO [53]	91.41	94.18	**73.35**	**73.37**	46.34	46.27	100	Bayesian Optimization
AG-Net (ours)	**91.55**	**94.24**	73.2	73.12	46.31	46.2	96	Generative LSO
AG-Net (ours, topk=1)	91.41	94.16	73.14	73.15	**46.42**	**46.43**	100	Generative LSO
BANANAS† [45]	91.56	94.3	**73.49***	73.50	**46.65**	**46.51**	192	Bayesian Optimization
BO† [39]	91.54	94.22	73.26	73.22	46.43	46.40	192	Bayesian Optimization
RS † [22]	91.12	93.89	72.08	72.07	45.87	45.98	192	Random
XGB (ours)	91.54	94.34	73.10	72.93	46.48	46.08	192	Generative LSO
XGB + Ranking (ours)	91.48	94.25	73.20	73.24	46.40	46.16	192	Generative LSO
AG-Net (ours)	**91.60**	**94.37***	**73.49***	**73.51***	46.64	46.43	192	Generative LSO
GANAS [33]	-	94.34	-	73.28	-	**46.80**	444	Generative Reinforcement Learning
AG-Net (ours)	**91.61***	**94.37***	**73.49***	**73.51***	**46.73**	46.42	400	Generative LSO

Table 3: Results on: (**left**) NAS-Bench-301 (mean validation accuracy over 50 trials). (**right**) NAS-Bench-NLP (mean validation perplexity over 100 trials).

NAS Method	NAS-Bench-301		NAS-Bench-NLP	
	Val. Acc (%)	Queries	Val. Perplexity (%)	Queries
BANANAS† [45]	94.77	192	95.68	304
Bayesian Optimization† [39]	94.71	192	-	-
Local Search† [46]	**95.02**	192	95.69	304
Random Search†[22]	94.31	192	95.64	304
Regularized Evolution†[31]	94.75	192	95.66	304
XGB (ours)	94.79	192	**95.95**	304
XGB + Ranking (ours)	94.76	192	95.92	304
AG-Net (ours)	94.79	192	95.86	304

tures on ImageNet is extremely expensive. Our first approach is to retrain the best found architectures on the CIFAR-10 [19] image classification task from the previous experiment on NAS-Bench-301 (AG-Net and the XGBoost adaptions) on ImageNet [8]. Our second approach is based on a training-free neural architecture search approach. The recently proposed TE-NAS [5] provides a training-free neural architecture search approach, by ranking architectures by analysing the neural tangent kernel (NTK) and the number of linear regions (NLR) of each architecture. These two measurements are training free and do not need any labels. The intuition between those two measurements is their implication towards trainability and expressivity of a neural architecture and also their correlation with the neural architecture's accuracy; NTK is negatively correlated and NLR positively correlated with the architecture's test accuracy. We adapt this idea for our search on ImageNet and search architectures in terms of their NTK value and their number of linear regions instead of their validation accuracy. We describe the detailed search process in the supp. mat. subsection C.5.

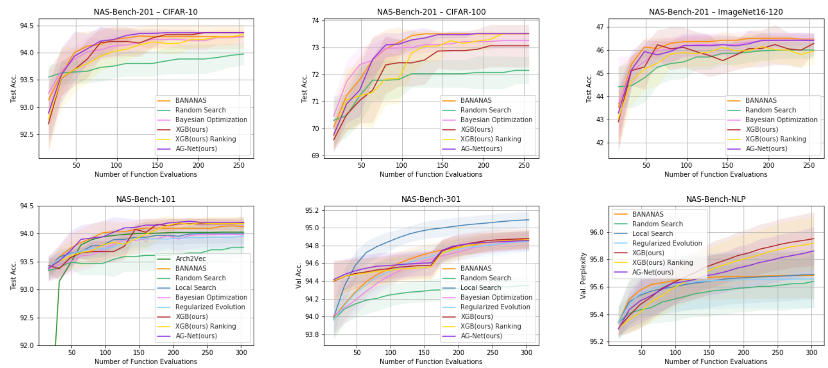

Fig. 3: Architecture search evaluations on NAS-Bench-201, NAS-Bench-101, NAS-Bench-301 and NAS-Bench-NLP for different search methods.

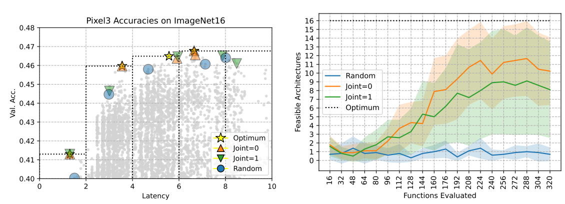

Fig. 4: (**left**) Exemplary searches on HW-NAS-Bench for image classification on ImageNet16 with 192 queries on Pixel 3 and latency conditions $L \in \{2, 4, 6, 8, 10\}$ (y-axis zoomed for visibility). (**right**) Amount of architectures generated and selected in each search iteration (at most 16) that satisfy the latency constraint. In this example we searched on Edge GPU with $L = 2$.

Table 4 shows the results. Note that our latter described search method on ImageNet is **training-free** (as TE-NAS [5]) and the amount of queries displays the amount of data we evaluated for the zero cost measurements. Other query information include the amount of (partly) trained architectures. Furthermore, the displayed differentiable methods are based on training supernets which can lead to expensive training times. The best found architectures on NAS-Bench-301 [37] (CIFAR-10) result in comparable error rates on ImageNet to former approaches. As a result, our search method approach is highly efficient and outperforms previous methods in terms of needed GPU days. The result in terms of top-1 and top-5 error rates are even improving over the one from previous approaches when using the training free approach.

4.3 Experiments on Hardware-Aware Benchmark

Next, we apply AG-Net to the Hardware-Aware NAS-Benchmark [21]. We demonstrate in two settings that AG-Net can be used for multi-objective learning. The first setting (*Joint=1*) is formulated as constrained joint optimization:

Table 4: ImageNet **error** of neural architecture search on DARTS.

NAS Method	Top-1↓	Top-5↓	# Queries	Search GPU days
Mixed Methods				
NASNET-A (CIFAR-10) [58]	26.0	8.4	20000	2000
PNAS (CIFAR-10) [24]	25.8	8.1	1160	225
NAO (CIFAR-10) [27]	24.5	7.8	1000	200
Differentiable Methods				
DARTS (CIFAR-10) [25]	26.7	8.7	-	4.0
SNAS (CIFAR-10)[49]	27.3	9.2	-	1.5
PDARTS (CIFAR-10) [6]	**24.4**	**7.4**	-	0.3
PC-DARTS (CIFAR-10) [51]	25.1	7.8	-	**0.1**
PC-DARTS (ImageNet) [51]	**24.2**	**7.3**	-	3.8
Predictor Based Methods				
WeakNAS (ImageNet) [48]	**23.5**	**6.8**	800	2.5
XGB (NB-301)(CIFAR-10) (ours)	24.1	7.4	304	**0.02**
XGB + Ranking (NB-301)(CIFAR-10) (ours)	24.1	7.2	304	**0.02**
AG-Net (NB-301)(CIFAR-10) (ours)	24.3	7.3	304	0.21
Training-Free Methods				
TE-NAS (CIFAR-10)[5]	26.2	8.3	-	0.05
TE-NAS (ImageNet)[5]	24.5	7.5	-	0.17
AG-Net (CIFAR-10) (ours)	**23.5**	7.1	208	**0.02**
AG-Net (ImageNet) (ours)	**23.5**	6.9	208	**0.09**

$$\max_{G\sim p_D} f(G) \wedge \min_{G\sim p_D,} g_h(G) \qquad \text{s.t. } g_h(G) \leq L, \exists\ h \in H, \tag{4}$$

where $f(\cdot)$ evaluates architecture G for accuracy and $g_h(\cdot)$ evaluates for latency given a hardware $h \in H$ and a user-defined latency constraint L. The second setting (*Joint=0*) is formulated as constraint objective:

$$\max_{G\sim p_D} f(G) \qquad \text{s.t. } g_h(G) \leq L, \exists\ h \in H, \tag{5}$$

where we drop the optimization on latency and only optimize accuracy given the latency constraint. The loss function to train our generator in these settings is updated from Equation 2 to:

$$\mathcal{L}(\tilde{G}, G) = (1-\alpha)\mathcal{L}_G(\tilde{G}, G) + \alpha\big[\lambda\mathcal{L}_{C_1}(\tilde{G}, G) + (1-\lambda)\mathcal{L}_{C_2}(\tilde{G}, G)\big], \tag{6}$$

where α is a hyperparameter trading off generation and prediction loss, and λ is a hyperparameter trading off both prediction targets C_1 (accuracy) and C_2 (latency).

To perform LSO in the joint objective setting from Equation 4, we rank the training data D for both accuracy and latency jointly by summing both

Table 5: Results for searches with at most 200 queries on HW-NAS-Bench [21] with varying devices and latency (Lat.) constraints in two multi-objective settings: *Joint=0* optimizes accuracy under latency constraint, while *Joint=1* optimizes for accuracy and latency jointly. We report the best found architecture out of 10 runs with their corresponding latency, as well as the mean of these runs. We compare to random search as a strong baseline [22]. Feasibility (Feas.) is the proportion of evaluated architectures during the search that satisfy the latency constraint (larger is better). The optimal architecture (*) is the architecture with the highest accuracy satisfying the latency constraint.

Settings		**Best out of 10 runs**						**Mean**							
Constraint		**Joint=0**		**Joint=1**		**Random**		**Joint=0**		**Joint=1**		**Random**		**Optimum***	
Device	Lat.↓	Acc.↑	Lat.↓	Acc.↑	Lat.↓	Acc.↑	Lat.↓	Acc.↑	Feas.↑	Acc.↑	Feas.↑	Acc.↑	Feas.↑	Acc.↑	Lat.↓
Edge GPU	2	**0.406***	1.90	**0.406***	1.90	0.397	1.78	**0.397**	0.29	*0.391*	0.31	0.372	0.05	0.406	1.90
Edge GPU	4	**0.448***	3.49	**0.448***	3.49	0.437	3.35	*0.428*	0.29	**0.433**	0.43	0.417	0.22	0.448	3.49
Edge GPU	6	*0.458*	5.29	**0.464***	5.96	*0.458*	5.29	**0.453**	0.64	*0.450*	0.79	0.449	0.72	0.464	5.96
Edge GPU	8	*0.465*	6.81	**0.468***	6.81	0.464	7.44	**0.463**	0.98	*0.462*	0.99	0.457	1.00	0.468	6.81
Raspi 4	2	**0.355***	1.58	**0.355***	1.58	0.348	1.60	*0.346*	0.28	**0.347**	0.30	0.339	0.08	0.355	1.58
Raspi 4	4	*0.431*	3.83	**0.436***	3.79	0.427	3.85	*0.420*	0.47	**0.428**	0.50	0.419	0.37	0.436	3.79
Raspi 4	6	*0.449*	5.95	**0.452***	5.29	0.445	5.95	*0.440*	0.56	**0.441**	0.57	0.432	0.55	0.452	5.29
Raspi 4	8	*0.456*	6.33	0.455	7.96	**0.457**	7.97	**0.451**	0.69	*0.449*	0.79	0.447	0.76	0.465	7.43
Raspi 4	10	**0.466**	8.66	*0.465*	8.62	0.464	8.72	**0.464**	0.77	*0.454*	0.94	*0.454*	0.90	0.468	8.83
Raspi 4	12	**0.468***	8.83	0.463	9.05	*0.464*	8.72	**0.465**	0.91	*0.457*	0.98	0.456	0.96	0.468	8.83
Edge TPU	1	**0.468***	0.96	*0.466*	0.97	0.464	1.00	**0.464**	0.74	*0.457*	0.82	0.454	0.79	0.468	0.96
Pixel 3	2	**0.413***	1.30	**0.413***	1.30	0.400	1.50	**0.409**	0.48	*0.405*	0.59	0.388	0.30	0.413	1.30
Pixel 3	4	**0.460***	3.55	0.446	3.01	*0.447*	3.23	**0.453**	0.69	*0.441*	0.77	0.438	0.64	0.460	3.55
Pixel 3	6	*0.464*	5.92	**0.465***	5.95	0.458	4.68	**0.457**	0.77	*0.452*	0.94	0.451	0.88	0.465	5.57
Pixel 3	8	**0.468***	6.65	*0.465*	7.88	0.461	7.13	**0.464**	0.87	*0.457*	0.99	0.454	0.97	0.468	6.65
Pixel 3	10	**0.466**	6.70	0.461	8.48	*0.464*	8.01	**0.464**	0.96	0.455	1.00	*0.456*	0.99	0.468	6.65
Eyeriss	1	**0.452***	0.98	*0.449*	0.98	0.447	0.98	**0.445**	0.49	*0.436*	0.53	0.433	0.23	0.452	0.98
Eyeriss	2	**0.465**	1.65	**0.465**	1.65	0.464	1.65	**0.463**	0.87	*0.457*	0.99	*0.457*	0.95	0.468	1.65
FPGA	1	**0.440**	1.00	**0.440**	0.97	0.438	0.97	**0.433**	0.65	**0.433**	0.80	0.429	0.58	0.444	1.00
FPGA	2	**0.465***	1.60	0.460	1.60	*0.463*	1.97	**0.462**	0.82	0.451	0.99	*0.453*	0.97	0.465	1.60

individual rankings. To fulfill the optimization constraint, we further penalize the ranks via a multiplicative penalty if the latency does not fulfill the constraint. This overall ranking is then used for the weight calculation in Equation 3. The LSO for the constraint objective setting from Equation 5 only ranks architectures by accuracy and penalizes architectures with infeasible latency property. We choose random search as a baseline in this setting as it is generally regarded as a strong baseline in NAS [22]. Figure 4 depicts searches with our model in both optimization settings on Pixel 3 with different latency conditions. More results on different hardware and latency constraints are shown in Table 5. We observe that either optimization setting outperforms the random search baseline in almost all tasks. Additionally, our method is able to find the optimal architecture for a task regularly (in 15 out of 20 tasks), which random search was not able to provide. When considering mean accuracy and feasibility of the best architectures of all runs, we see that *Joint=1* is able to improve the ratio of feasible architectures found during the search substantially. This is to be expected given that the latent space is explicitly optimized for latency in this setting. Consequently, *Joint=1* is able to find better-performing architectures compared to *Joint=0* if the constraint restricts the space of feasible architectures strongly (see results on Raspi 4). The feasibility ratio of random search is an indicator on how restricted

Table 6: Ablation: Search results on NAS-Bench-101 and NAS-Bench-201 using AG-Net (mean over 10 trials with a maximal query amount of 192).

	NAS-Bench-101		**NAS-Bench-201**					
	CIFAR-10		CIFAR-10		CIFAR-100		ImageNet16-120	
	Val. Acc	Test Acc	Val. Acc	Test Acc	Val. Acc	Test Acc	Val. Acc	Test Acc
Optimum*	95.06	94.32	91.61	94.37	73.49	73.51	46.77	47.31
AG-Net (ours) w/o LSO	94.38	93.78	91.15	93.84	71.72	71.83	45.33	45.04
AG-Net (ours) w/o backprop	94.71	94.12	**91.60**	94.30	73.38	73.22	46.62	46.13
AG-Net (ours)	**94.90**	**94.18**	**91.60**	**94.37***	**73.49***	**73.51***	**46.64**	**46.43**

the space is. In most cases, the latency penalization seems to be sufficient to find enough well-performing and feasible architectures, as can be seen by the feasibility of *Joint=0* which is greatly improved compared to random search. We show the development of feasibility over time from Table 5 in Figure 4.

4.4 Ablation Studies

In this section we analyse the impact of the LSO technique and the backpropagation ability to the search efficiency. Therefore, we compare our AG-Net with the latter named adaptions on the tabular benchmarks NAS-Bench-101 [54] and NAS-Bench-201 [10]. The results of our ablation study are reported in Table 6. As we can see, the lack of weighted retraining decreases the search substantially. In addition the results without backpropagation support that the coupling of the predictor's target and the generation process enables a more efficient architecture search over different search spaces. Thus, the combination of LSO and a fully differentiable approach improves the effectiveness of the search.

5 Conclusion

We propose a simple architecture generative network (AG-Net), which allows us to directly generate architectures without any additional encoder or discriminator. AG-Net is fully differentiable, allowing to couple it with surrogate models for different target predictions. In contrast to former works, it enables to backpropagate the target information from the surrogate predictor into the generator. By iteratively optimizing the latent space of the generator, our model learns to focus on promising regions of the architecture space, so that it can generate high-scoring architectures directly in a query and sample-efficient manner. Extensive experiments on common NAS benchmarks demonstrate that our model outperforms state-of-the-art methods at almost any time during architecture search and achieves state-of-the-art performance on ImageNet. It also allows for multi-objective optimization on the Hardware-Aware NAS-Benchmark.

Acknowledgments. JL and MK acknowledge the German Federal Ministry of Education and Research Foundation via the project DeToL.

References

1. Bender, G., Kindermans, P., Zoph, B., Vasudevan, V., Le, Q.V.: Understanding and simplifying one-shot architecture search. In: ICML (2018)
2. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(10), 281–305 (2012)
3. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. In: ICLR (2019)
4. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016)
5. Chen, W., Gong, X., Wang, Z.: Neural architecture search on imagenet in four GPU hours: A theoretically inspired perspective. In: ICLR (2021)
6. Chen, X., Xie, L., Wu, J., Tian, Q.: Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In: ICCV (2019)
7. Chrabaszcz, P., Loshchilov, I., Hutter, F.: A downsampled variant of imagenet as an alternative to the CIFAR datasets. CoRR **abs/1707.08819** (2017)
8. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: CVPR (2009)
9. Dong, X., Yang, Y.: One-shot neural architecture search via self-evaluated template network. In: ICCV (2019)
10. Dong, X., Yang, Y.: Nas-bench-201: Extending the scope of reproducible neural architecture search. In: ICLR (2020)
11. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. J. Mach. Learn. Res. (2019)
12. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. arXiv preprint arXiv:1406.2661 (2014)
13. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: ICCV (2015)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
15. Huang, S., Chu, W.: Searching by generating: Flexible and efficient one-shot NAS with architecture generator. In: CVPR (2021)
16. Kandasamy, K., Neiswanger, W., Schneider, J., Póczos, B., Xing, E.P.: Neural architecture search with bayesian optimisation and optimal transport. In: NIPS (2018)
17. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: ICLR (2014)
18. Klyuchnikov, N., Trofimov, I., Artemova, E., Salnikov, M., Fedorov, M., Burnaev, E.: Nas-bench-nlp: Neural architecture search benchmark for natural language processing. CoRR **abs/2006.07116** (2020)
19. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)
20. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS (2012)
21. Li, C., Yu, Z., Fu, Y., Zhang, Y., Zhao, Y., You, H., Yu, Q., Wang, Y., Hao, C., Lin, Y.: Hw-nas-bench: Hardware-aware neural architecture search benchmark. In: ICLR (2021)
22. Li, L., Talwalkar, A.: Random search and reproducibility for neural architecture search. In: UAI (2019)

23. Li, Y., Vinyals, O., Dyer, C., Pascanu, R., Battaglia, P.W.: Learning deep generative models of graphs. CoRR **abs/1803.03324** (2018)
24. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A.L., Huang, J., Murphy, K.: Progressive neural architecture search. In: ECCV (2018)
25. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search (2019)
26. Lukasik, J., Friede, D., Zela, A., Hutter, F., Keuper, M.: Smooth variational graph embeddings for efficient neural architecture search. In: IJCNN (2021)
27. Luo, R., Tian, F., Qin, T., Chen, E., Liu, T.: Neural architecture optimization. In: NeurIPS (2018)
28. Mehta, Y., White, C., Zela, A., Krishnakumar, A., Zabergja, G., Moradian, S., Safari, M., Yu, K., Hutter, F.: Nas-bench-suite: NAS evaluation is (now) surprisingly easy. CoRR **abs/2201.13396** (2022)
29. Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., Khudanpur, S.: Recurrent neural network based language model. In: Kobayashi, T., Hirose, K., Nakamura, S. (eds.) INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010 (2010)
30. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: ICML (2018)
31. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: AAAI (2019)
32. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: ICML (2017)
33. Rezaei, S.S.C., Han, F.X., Niu, D., Salameh, M., Mills, K.G., Lian, S., Lu, W., Jui, S.: Generative adversarial neural architecture search. In: IJCAI (2021)
34. Rolínek, M., Musil, V., Paulus, A., P., M.V., Michaelis, C., Martius, G.: Optimizing rank-based metrics with blackbox differentiation. In: CVPR (2020)
35. Ru, B., Wan, X., Dong, X., Osborne, M.: Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels (2021)
36. Shi, H., Pi, R., Xu, H., Li, Z., Kwok, J.T., Zhang, T.: Multi-objective neural architecture search via predictive network performance optimization. CoRR **abs/1911.09336** (2019)
37. Siems, J., Zimmer, L., Zela, A., Lukasik, J., Keuper, M., Hutter, F.: Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. CoRR **abs/2008.09777** (2020)
38. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2014)
39. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M.M.A., Prabhat, Adams, R.P.: Scalable bayesian optimization using deep neural networks. In: ICML (2015)
40. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: CVPR (2015)
41. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the Inception Architecture for Computer Vision. In: CVPR (2016)
42. Tripp, A., Daxberger, E., Hernández-Lobato, J.M.: Sample-efficient optimization in the latent space of deep generative models via weighted retraining. In: NeurIPS (2020)
43. Wen, W., Liu, H., Chen, Y., Li, H.H., Bender, G., Kindermans, P.: Neural predictor for neural architecture search. In: ECCV (2020)

44. White, C., Neiswanger, W., Nolen, S., Savani, Y.: A study on encodings for neural architecture search. In: NeurIPS (2020)
45. White, C., Neiswanger, W., Savani, Y.: Bananas: Bayesian optimization with neural architectures for neural architecture search. In: AAAI (2021)
46. White, C., Nolen, S., Savani, Y.: Exploring the loss landscape in neural architecture search (2021)
47. White, C., Zela, A., Ru, B., Liu, Y., Hutter, F.: How powerful are performance predictors in neural architecture search? (2021)
48. Wu, J., Dai, X., Chen, D., Chen, Y., Liu, M., Yu, Y., Wang, Z., Liu, Z., Chen, M., Yuan, L.: Stronger nas with weaker predictors. In: NeurIPS (2021)
49. Xie, S., Zheng, H., Liu, C., Lin, L.: SNAS: stochastic neural architecture search. In: ICLR (2019)
50. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2019)
51. Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G., Tian, Q., Xiong, H.: PC-DARTS: partial channel connections for memory-efficient architecture search. In: ICLR (2020)
52. Yan, S., White, C., Savani, Y., Hutter, F.: Nas-bench-x11 and the power of learning curves
53. Yan, S., Zheng, Y., Ao, W., Zeng, X., Zhang, M.: Does unsupervised architecture representation learning help neural architecture search? In: NeurIPS (2020)
54. Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: Nas-bench-101: Towards reproducible neural architecture search. In: ICML (2019)
55. Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., Hutter, F.: Understanding and robustifying differentiable architecture search. In: ICLR (2020)
56. Zhang, M., Jiang, S., Cui, Z., Garnett, R., Chen, Y.: D-vae: A variational autoencoder for directed acyclic graphs. In: NIPS (2019)
57. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: ICLR (2017)
58. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: CVPR (2018)