Learning Where To Look – Generative NAS is Surprisingly Efficient

Jovita Lukasik[1]* Steffen Jung[2]* 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 effective and efficient architecture search, while keeping the query amount low.

1. Introduction

The first image classification network [13] applied to the large scale visual recognition challenge ImageNet [4] achieved unprecedented results. Since then, the main driver of improvement on this challenge are new architecture designs [7, 22] that, ultimately, lead to architectures surpassing human performance [6]. Since manual architecture design requires good intuition and a huge amount of trial-and-error, the automated approach of neural architecture search (NAS) receives growing interest [5, 11, 14, 19, 27, 29]. Well-performing architectures can be found by applying common search practices like random search [1], evolutionary search [18, 19], Bayesian optimization (BO) [9, 20, 24], or local search [25] on discrete architecture search spaces, such as NAS-Bench-101 and NAS-Bench-201 [5, 27]. 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 [2, 16] or by generating architectures based on learned architecture representation spaces [17, 28].

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 effective NAS method. Our proposed method is inspired by a latent space optimization (LSO) technique [23], originally used in the context of variational autoencoders [10] to optimize generated images or arithmetic expression 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, there is no need for BO in the generated latent space, making our method even more efficient.

By extensive experiments on common NAS benchmarks [5, 14, 27] we show that our method is effective and sample-efficient. It reinforces the generator network to produce architectures with improving validation accuracy, as well as in improving on hardware-dependent latency constraints (see Figure 2) while keeping the number of architecture evaluations small.

*Authors contributed equally.

2. 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 [5, 27].

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.

Generative Network Our proposed network is a *purely generative* network, p_G, which does not rely on any architecture evaluation and is therefore fast and query free. To generate valid graphs, we build our model similar to the graph decoder from the VAE approach SVGe [17]. 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 in an iterative manner. Furthermore, we adapt the generator to allow for backpropagation. Thereby, the generator can efficiently learn which part of the architecture search space is promising with only few evaluated architectures. 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. In contrast, previous models such as [8, 17, 26] 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.

Performance Predictor Our 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.

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*. 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 [23], 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) than on low-scoring points. Note that this strategy does not discard low-scoring architectures completely, which would be inadequate for proper learning.

We assign 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, [23] 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 [27] and NAS-Bench-201 [5] it is the tabular benchmark entry. Similar to [23], 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 datapoints are both weighted by $w(G; p_D, k)$.

3. Experiments

We evaluate the proposed simple architecture generative network (AG-Net) on the two commonly used tabular benchmarks NAS-Bench-101 [27] and NAS-Bench-201 [5] and the first hardware device induced benchmark [14], for which we consider the latency information on the NAS-Bench-201 search space.

3.1. 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)

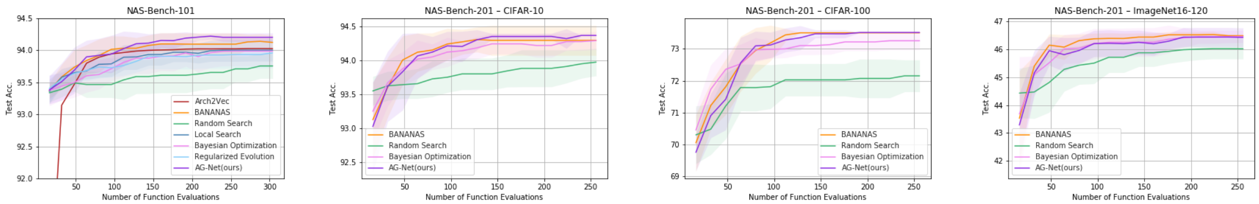

Figure 1. Architecture search evaluations on NAS-Bench-101 and NAS-Bench-201 for different search methods.

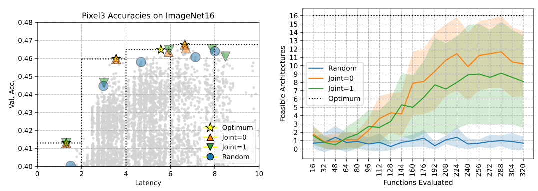

Figure 2. (**left**) Exemplary searches on HW-NAS-Bench for image classification on ImageNet16 with 192 queries on Pixel 3 and latency conditions $L \in \{2, 4, 6, 8, 10\}$ (y-axis zoomed for visibility). (**right**) Amount of architectures generated and selected in each search iteration (at most 16)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

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 [26], as well as state-of-the-art methods, random search [15], local search [25], Bayesian optimization [21], regularized evolution [18] and BANANAS [24]. Note that we search for the architecture with the best validation accuracy and report the corresponding test accuracy. We plot the search progress in Figure 1 (left) of this comparison. 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%. It is also almost any time better during the search process.

NAS-Bench-201 covers three image classification tasks: CIFAR-10, CIFAR-100 [12] and ImageNet16-120 [3]. For NAS-Bench-201 [5] we retrain AG-Net in the weighted manner for 30 epochs. We plot the search progress over queries in Figure 1 (right). Our method provides state-of-the-art results on all datasets for varying number of queries.

3.2. Experiments on Hardware-Aware Benchmark

Next, we apply AG-Net to the Hardware-Aware NAS-Benchmark [14]. 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:

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

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:

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

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 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 [15]. Figure 2 depicts searches with our model in both optimization settings on Pixel 3 with different latency conditions. We observe that either optimization setting outperforms the random search baseline given different latency constraints. 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.

We can see, *Joint=1* is able to find better-performing architectures compared to *Joint=0* if the constraint restricts the space of feasible architectures strongly.

The feasibility ratio of random search is an indicator on how restricted 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 in Figure 2.

4. Conclusion

We propose a simple architecture generative network (AG-Net), which allows 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. It also allows for multi objective optimization on the Hardware-Aware NAS-Benchmark.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. 1

[2] H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 1

[3] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017. 3, 5

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 6, 9, 10

[5] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020. 1, 2, 3, 5, 10, 11

[6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 1

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[8] Sian-Yao Huang and Wei-Ta Chu. Searching by generating: Flexible and efficient one-shot NAS with architecture generator. In *CVPR*, 2021. 2, 10, 11, 13

[9] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *NIPS*, 2018. 1

[10] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014. 1

[11] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, and E. Burnaev. Nas-bench-nlp: Neural architecture search benchmark for natural language processing. *CoRR*, abs/2006.07116, 2020. 1, 6, 7

[12] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 3, 5, 8, 9

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 1

[14] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, C. Hao, and Y. Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. In *ICLR*, 2021. 1, 2, 3, 5, 10, 12

[15] L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. In *UAI*, 2019. 3, 8, 9, 11, 12

[16] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. 2019. 1, 6, 8, 9, 10

[17] J. Lukasik, D. Friede, A. Zela, F. Hutter, and M. Keuper. Smooth variational graph embeddings for efficient neural architecture search. In *IJCNN*, 2021. 1, 2, 13, 14

[18] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 1, 3, 8, 9, 11

[19] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017. 1

[20] Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. 2021. 1

[21] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, Md. M. A. Patwary, Prabhat, and R. P. Adams. Scalable bayesian optimization using deep neural networks. In *ICML*, 2015. 3, 9, 11, 12

[22] C. Szegedy, Wei L., Yangqing J., P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1

[23] A. Tripp, E. Daxberger, and J. Hernández-Lobato. Sample-efficient optimization in the latent space of deep generative models via weighted retraining. In *NeurIPS*, 2020. 1, 2

[24] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *AAAI*, 2021. 1, 3, 8, 9, 11

[25] C. White, S. Nolen, and Y. Savani. Exploring the loss landscape in neural architecture search. 2021. 1, 3, 8, 9, 10, 11, 12, 13

[26] S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang. Does unsupervised architecture representation learning help neural architecture search? In *NeurIPS*, 2020. 2, 3, 5, 8, 10, 11, 13

[27] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*, 2019. 1, 2, 10, 11

[28] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen. D-vae: A variational autoencoder for directed acyclic graphs. In *NeurIPS*, 2019. 1

[29] B. Zoph, V. Vasudevan, J. Shlens, and Q. V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 9

Appendices

Section A provides an overview about the overall implementation details, by first introducing graph representations for each search space, which we consider in the main paper, second detailed information about the surrogate prediction model and last the search algorithm. Appendix B shows results on additional surrogate benchmarks as well as results on the DARTS search space in the mobile setting. In Appendix C, we provide more details about the experimental results from the main paper. In Appendix D we show additional ablation studies. In Appendix E we describe details about the generator network, and in Appendix F we list all hyperparameter settings of our experiments. Lastly, we include a visual intuition of the latent space optimization technique in Appendix G.

A. Implementation Details

A.1. Search Space Representations

In this section we provide more details about the search spaces, we consider in the main paper.

A.1.1 NAS-Bench-101

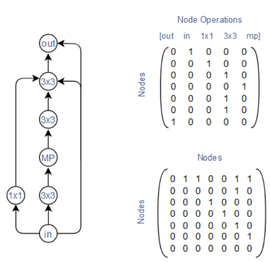

Figure 3. Exemplary cell representation from the NAS-Bench-101 search space. (**left**) DAG representation of a graph with 7 nodes. (**right**) The top part shows the node attribute matrix to the DAG and the bottom part shows its adjacency matrix.

NAS-Bench-101 is the first tabular benchmark designed for benchmarking NAS methods. This search space is a cell-based search space and contains $423,624$ unique neural networks. Each architecture is trained 3 times on CIFAR-10 [12] for image classification. The cell topology is limited to number of nodes $|V| \leq 7$ (including input and output node) and edges $|E| \leq 9$. The nodes represent the architecture layers and intermediate nodes can take any operation from the operation set $\mathcal{O} = \{1 \times 1$ conv., 3×3 conv., 3×3 max pooling$\}$. For visualization purposes, we present in Figure 3 exemplary a DAG from the NAS-Bench-101 search space, with its corresponding node attribute matrix and its adjacency matrix. Note, a concatenation of the flatted node attribute matrix and the flatted upper triangular adjacency matrix is the representation our generator model is trained to learn; this holds for all search spaces.

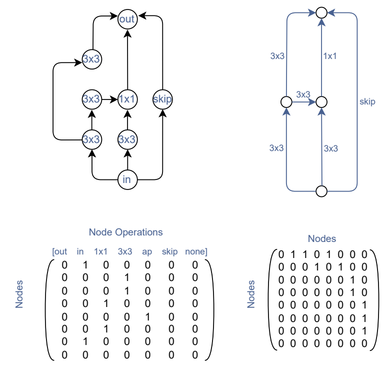

Figure 4. Exemplary cell representation from the NAS-Bench-201 search space. (**top**) The left part visualizes the DAG representation with node attributes instead of edge attributes. The right part shows the true DAG representation in the NAS-Bench-201 search space. (**bottom**) The left part shows the node attribute matrix to the DAG and the right part shows its adjacency matrix.

A.1.2 NAS-Bench-201

NAS-Bench-201 [5] is another cell-structured search space, which consists of $15,625$ architectures. Each architecture is trained for 200 training epochs on CIFAR-10 [12], CIFAR-100 [12], and ImageNet16-120 [3]. This benchmark provides validation and test accuracy information for each of the three datasets. The cell structure is different compared to NAS-Bench-101: Each cell has $|V| = 4$ nodes and $|E| = 6$ edges, where the former represent feature maps and the latter denote operations chosen from the set $\mathcal{O} = \{1\times1$ conv., 3×3 conv., 3×3 avg pooling, skip, zero$\}$.

Figure 4 visualizes a DAG in the true variant in the NAS-Bench-201 search space with edge attributes, as well as our adapted representation, where the edge attributes are changed to node attributes. This is similar to the representation in [26]. We show experiments on NAS-Bench-101 and NAS-Bench-201 in subsection 3.1.

A.1.3 Hardware-Aware-NAS-Bench

The recently introduced HW-NAS-Bench [14] is the first public dataset for hardware NAS. It extends two representative NAS search spaces, NAS-Bench-201 [5] and FBNet [51], by providing measured and estimated hardware costs (i.e. latency and/or energy) for each device for all architectures in both search spaces. For this, HW-NAS-Bench considers six hardware devices: *Edge GPU* [32], *Raspi 4* [33], *Edge TPU* [30], *Pixel 3* [31], *ASIC-Eyeriss* [38] and *FPGA* [34, 35].

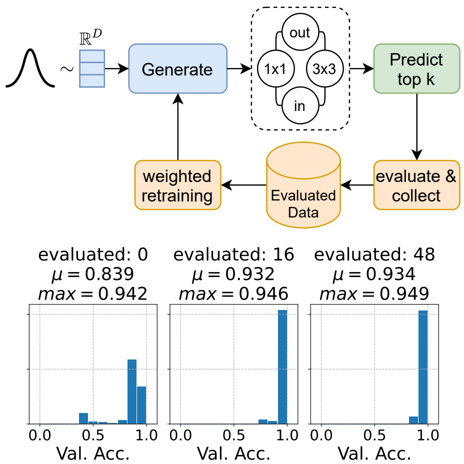

Figure 5. (**Top**) Our search method generates architectures from points in an architecture representation space that is iteratively optimized. (**bottom**) The architecture representation space is biased towards better-performing architectures with each search iteration. After only 48 evaluated architectures, our generator produces state-of-the-art performing architectures on NAS-Bench-101.

In our experiments in subsection 3.2 we consider the latency information on the NAS-Bench-201 search space.

A.2. Surrogate Model

In this section, we present details about the surrogate models used in the main paper. The MLP surrogate model used for our AG-Net is a 4-layer MLP with ReLU non-activation functions. The hidden size equals the input size. The input to the MLP surrogate model is the vector representation $\in \mathbb{R}^n$ of our graphs: a concatenation of the flatted node attribute matrix and flatted upper triangular matrix of the adjacency matrix, which presents the edge scores. Note, the vector dimension n differs across the search spaces due to the different maximal amount of nodes. Our AG-Net passes the output of our generator, i.e. a generated vector representation, as the direct input to our MLP surrogate model.

A.3. Search Algorithm

Figure 5 visualizes the search process of our method, which is effective and sample-efficient. It reinforces the generator network to produce architectures with improving validation accuracy. High-level descriptions of the unconstrained (subsection 3.1) and constrained (subsection 3.2) versions of our search algorithm are depicted in algorithm 1 and algorithm 2 respectively.

B. Additional Studies

In the main paper we evaluate our proposed simple architecture generative network (AG-Net) on two tabular benchmarks. Here, we additionally evaluate AG-Net on the surrogate benchmarks NAS-Bench-NLP [11] and NAS-Bench-301 [50] evaluated on the DARTS search space [16]. Additionally we perform experiments on the ImageNet [4] classification task and show state-of-the-art performance on the DARTS search space.

Algorithm 1: Unconstrained Search Algorithm

Input: (i) Search space p_D
Input: (ii) Pretrained generator G
Input: (iii) Untrained performance predictor P
Input: (iv) Query budget b
Input: (v) e epochs to train G and P
▷ `Initialize training data`
$\mathbf{D} \leftarrow \{\}$
while $|\mathbf{D}| < 16$ **do**
  $\mathbf{D} \leftarrow \mathbf{D} \cup \{d \sim p_D\}$
end
▷ `Evaluate architectures (get accuracies on target image dataset)`
$\mathbf{D} \leftarrow \text{eval}(\mathbf{D})$
▷ `Randomly initialize predictor weights`
$P \leftarrow \text{init}(P)$
▷ `Search loop`
while $|\mathbf{D}| < b$ **do**
  ▷ `Weight training data by performance`
  $\mathbf{D}_{\text{w}} \leftarrow \text{weight}(\mathbf{D})$
  ▷ `Train generator and predictor`
  $\text{train}(G, P, \mathbf{D}_{\text{w}}, e)$
  ▷ `Generate 100 candidates`
  $\mathbf{D}_{\text{cand}} \leftarrow \{\}$
  while $|\mathbf{D}_{cand}| < 100$ **do**
    $\mathbf{z} \sim \mathcal{U}[-3, 3]$
    $\mathbf{D}_{\text{cand}} \leftarrow \mathbf{D}_{\text{cand}} \cup G(\mathbf{z})$
  end
  ▷ `Select top 16 candidates with P`
  $\mathbf{D}_{\text{cand}} \leftarrow \text{select}(\mathbf{D}_{\text{cand}}, P, 16)$
  ▷ `Evaluate and add to data`
  $\mathbf{D} \leftarrow \mathbf{D} \cup \text{eval}(\mathbf{D}_{\text{cand}})$
end

B.1. Experiments on Surrogate Benchmarks

We furthermore apply our search method on larger search spaces as NAS-Bench-NLP [11] and DARTS [16] without ground truth evaluations for the whole search space, making use of surrogate benchmarks as, NAS-Bench-X11 [55], NAS-Bench-Suite [45] and NAS-Bench-301 [50].

Algorithm 2: Constrained Search Algorithm

Input: (i) Search space p_D
Input: (ii) Pretrained generator G
Input: (iii) Untrained performance predictor P_a
Input: (iv) Set of constraint predictors P_c
Input: (v) Query budget b
Input: (vi) e epochs to train G and P
Input: (vii) Set of constraints C

▷ Initialize training data
$\mathbf{D} \leftarrow \{\}$
while $|\mathbf{D}| < 16$ **do**
  $\mathbf{D} \leftarrow \mathbf{D} \cup \{d \sim p_D\}$
end
▷ Evaluate architectures (get accurracies and constraints on target image dataset)
$\mathbf{D} \leftarrow \text{eval}(\mathbf{D})$
▷ Randomly initialize predictor weights
$P_a \leftarrow \text{init}(P_a)$
foreach $P \in P_c$ **do**
  $P \leftarrow \text{init}(P)$
end
▷ Search loop
while $|\mathbf{D}| < b$ **do**
  ▷ Weight train data by performance and constraints
  $\mathbf{D}_{\text{w}} \leftarrow \text{weight}(\mathbf{D}, C)$
  ▷ Train generator and predictors
  $\text{train}(G, P_a, P_c, \mathbf{D}_{\text{w}}, e)$
  ▷ Generate 100 candidates
  $\mathbf{D}_{\text{cand}} \leftarrow \{\}$
  while $|\mathbf{D}_{cand}| < 100$ **do**
    $\mathbf{z} \sim \mathcal{U}[-3, 3]$
    $\mathbf{D}_{\text{cand}} \leftarrow \mathbf{D}_{\text{cand}} \cup G(\mathbf{z})$
  end
  ▷ Select top16 candidates with P_a and P_c
  $\mathbf{D}_{\text{cand}} \leftarrow \text{select}(\mathbf{D}_{\text{cand}}, P_a, P_c, 16)$
  ▷ Evaluate and add to data
  $\mathbf{D} \leftarrow \mathbf{D} \cup \text{eval}(\mathbf{D}_{\text{cand}})$
end

B.2. NAS-Bench-NLP

Here, we report experiments on NAS-Bench-NLP [11] for the language modeling task on Penn TreeBank [46]. We first introduce the search space representation and second display the experimental results.

Search Space Representation NAS-Bench-NLP [11] is the first RNN-derived benchmark for language modeling tasks. From the total 10^{53} possible architectures in the complete search space, $14,322$ architectures are trained on Penn TreeBank [46] (PTB) and provided in this benchmark. The cell search space is constrained by the number of nodes $|V| \leq 24$, the number of hidden states $|H| \leq 3$ and the number of linear input vectors ≤ 3. The nodes represent the architecture operational layer and are chosen from the set $\mathcal{O} = \{\text{linear, element wise blending, element wise product, element wise sum, Tanh activation, Sigmoid activation, LeakyReLU activation}\}$.

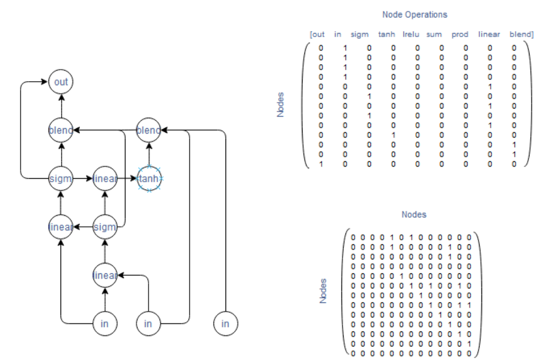

Figure 6. Exemplary cell representation from the NAS-Bench-NLP search space. (**left**) DAG representation of a graph with 12 nodes. (**right**) The top part shows the node attribute matrix to the DAG and the bottom part shows its adjacency matrix.

For the experiments on NAS-Bench-NLP [11] we make use of the surrogate benchmark NAS-Bench-X11 [55] and the additional implementation in NAS-Bench-Suite [45]. Note, for the NAS-Bench-X11 evaluations, each architecture from the NAS-Bench-NLP search space must be trained for three epochs to use the surrogate model, whereas NAS-Bench-Suite provides the surrogate model for NAS-Bench-NLP without learning curve information, but also accompanying a lower Kendall Tau rank correlation. For fast evaluations we use the latter surrogate for our experiments. In order to use the surrogate benchmark, the architecture representation is the same used in [55] with the modification, that each hidden node is connected to the output node. An exemplary architecture representation is visualized in Figure 6. A next step is to analyse the $14,332$ provided architectures on uniqueness, which leads to $12,107$ unique architectures. Furthermore, since [55] and [45] only provide a surrogate model, which only considers architectures with up to 12 nodes, we also restrict our training data to this subset leading to a total of $7,258$ architectures.

Experimental Results We retrain AG-Net coupled with the surrogate model for 30 epochs to predict the validation perplexity. As already mentioned, the search space

NAS Method	Val. Perplexity (%)	StD (%)	Queries
BANANAS† [24]	95.68	0.16	304
Local Search† [25]	95.69	0.18	304
Random Search† [15]	95.64	0.19	304
Regularized Evolution† [18]	95.66	0.21	304
AG-Net (ours)	95.86	0.18	304

Table 1. Results on NAS-Bench-NLP (mean and standard deviation over 100 trials) for the search of the best architecture in terms of validation perplexity compared to state-of-the-art methods.

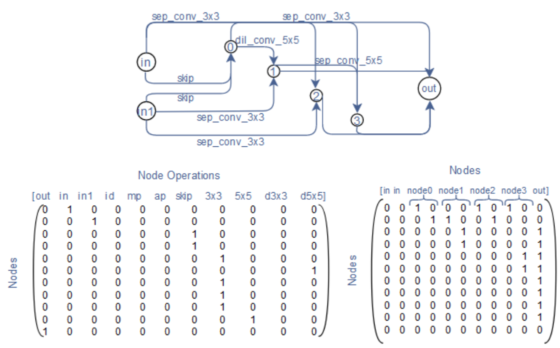

Figure 7. Exemplary cell representation from the DARTS search space. (**top**) Visualization of the DAG representation in the DARTS search space. (**bottom**) The left part shows the node attribute matrix to the DAG and the right part shows its adjacency matrix.

considered in NAS-Bench-NLP is too large for a full tabular benchmark evaluation, thus we make use of the surrogate benchmark NAS-Bench-X11 [55] and NAS-Bench-Suite [45] instead of tabular entries.

For fair comparison we compare our method to the same state-of-the-art-methods as considered in the main paper. The results are reported in Table 1. Our AG-Net improves over all state-of-the-art-methods by a substantial margin.

B.3. NAS-Bench-301

Next, we evaluate AG-Net on the cell-based DARTS [16] search space using the surrogate benchmark NAS-Bench-301 [50] for the CIFAR-10 [12] image classification task. We first introduce the search space representation, second the search process in this cell-based search space, which needed to be adapted and last we display the experimental results.

Search Space Representation NAS-Bench-301 [50] is the first surrogate benchmark, which evaluates several surrogate models on in total $60,000$ sampled architectures from the DARTS [16] search sapce on the CIFAR-10 [12] image classification task. The DARTS search space consists of 10^{18} neural networks, where each network consists of two cells; a normal cell and a reduction cell. Each cell is limited by the number of nodes $|N| = 7$ and the number of edges $|E| = 12$, where 4 of these edges connect the intermediate nodes (excluding the input nodes) to the output node. Each edge denotes an operation from the set $\mathcal{O} = \{3 \times 3 \text{ sep. conv.}, 5 \times 5 \text{ sep. conv.}, 3 \times 3 \text{ dil. conv.}, 5 \times 5 \text{ dil. conv.}, 3 \times 3 \text{ avg pooling}, 3 \times 3 \text{ max pooling}, \text{identity}, \text{zero}\}$. Each intermediate edge is connected to two predecessor nodes. Each cell also contains two input nodes, which are the output nodes from the previous two cells. The overall network is created by stacking the normal and reduction cell.

In order to train our generative model to generate valid cells, we additionally randomly sample 500k architectures from the DARTS search space. We train our generative model to learn to generate valid cells independently of being a normal or reduction cell. In Figure 7 we visualize the adapted node attribute matrix and the adapted adjacency matrix to an exemplary DAG in the DARTS search space [16]. This is similar to the representation in [26].

Search Process using NAS-Bench-301 In order to apply our AG-Net on the DARTS search space, we have to adapt the exact search procedure using the cells individually, which we describe in the following. For experiments in the DARTS [16] search space, we first train our generative model on generating valid cells, as visualized in Figure 7; here we do not distinguish between generating a normal or a reduction cell. Having a pretrained generative model for generating valid cells representations in the DARTS search space allows for searching well-performing architectures. Next, we describe the search process for architectures evaluated on CIFAR-10 using the surrogate benchmark NAS-Bench-301 [50]. Since the DARTS search space is defined by a normal and reduction cell, we have to adapt the search process, compared to the search in the tabular benchmark search spaces, where the architectures differ only between the DAGs. We begin the search by randomly sampling 16 architectures from NAS-Bench-301. Next, we generate one normal cell. This cell is used to search for the best reduction cell in terms of the accuracy given by the surrogate benchmark NAS-Bench-301, in combination with the randomly sampled cell. This search procedure follows then the same steps as for the tabular benchmarks and stops after we reach a query amount of 155. Now, we use the best found reduction cell as a fixed starting point to search for the best normal cell in the same manner as before. The overall search stops after a maximal amount of 310 queries. The search outcome differs between starting with a reduction or the normal cell. The search procedure starting with a random reduction cell is analogous. In the following, we report the search outcome for NAS-Bench-301 [50] starting with a random reduction cell.

Experimental Results The results are described in Table 2. Our method is comparable to other state-of-the-art methods in this search space.

NAS Method	Val. Acc (%)	StD (%)	Queries
BANANAS† [24]	94.77	0.10	192
Bayesian Optimization† [21]	94.71	0.10	192
Local Search† [25]	**95.02**	0.10	192
Random Search† [15]	94.31	0.12	192
Regularized Evolution† [18]	94.75	0.11	192
AG-Net (ours)	94.79	0.12	192

Table 2. Results on NAS-Bench-301 (mean and standard deviation over 50 trials) for the search of the best architecture in terms of validation accuracy compared to state-of-the-art methods.

B.3.1 ImageNet Experiments

The previous experiment on NAS-Bench-301 [50] shows the ability of our generator to generate valid architectures and to perform well in the DARTS [16] search space. This allows for searching a well-performing architecture on ImageNet [4]. Yet evaluating up to 300 different found architectures on ImageNet is extremly expensive. Our first approach is to retrain the best found architectures on the CIFAR-10 [12] image classification task from the previous experiment on NAS-Bench-301 on ImageNet [4]. Our second approach is based on a training-free neural architetures search approach. The recently proposed TE-NAS [36] 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 using TENAS in the next section.

Table 3 shows the results. Note that our latter described search method on ImageNet is **training-free** (as TE-NAS [36]) 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 Method	Top-1↓	Top-5↓	# Queries	Search GPU days
Mixed Methods				
NASNET-A (CIFAR-10) [29]	26.0	8.4	20000	2000
PNAS (CIFAR-10) [43]	25.8	8.1	1160	225
NAO (CIFAR-10) [44]	24.5	7.8	1000	200
Differentiable Methods				
DARTS (CIFAR-10) [16]	26.7	8.7	-	4.0
SNAS (CIFAR-10) [53]	27.3	9.2	-	1.5
PDARTS (CIFAR-10) [37]	**24.4**	**7.4**	-	0.3
PC-DARTS (CIFAR-10) [54]	25.1	7.8	-	**0.1**
PC-DARTS (ImageNet) [54]	**24.2**	**7.3**	-	3.8
Predictor Based Methods				
WeakNAS (ImageNet) [52]	**23.5**	**6.8**	800	2.5
AG-Net (NB-301)(CIFAR-10) (ours)	24.3	7.3	304	0.21
Training-Free Methods				
TE-NAS (CIFAR-10) [36]	26.2	8.3	-	0.05
TE-NAS (ImageNet) [36]	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**

Table 3. ImageNet **error** of neural architecture search on DARTS.

NAS-Bench-301 [50] (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.

Search Process using TENAS As we described in the previous section, the search in the DARTS [16] search space needs adaptions in the search procedure. Here we describe the further adaption of using training free measurements instead of the NAS-Bench-301 prediction. The training free measurements are based on the recent paper TE-NAS [36], which ranks architectures by analysing the neural tangent kernel, by its condition number (KN), and the number of linear regions (NLR) of each architecture. Concretely, for the search on ImageNet [4] we search for architectures in terms of their KN value and their number of linear regions instead of their validation accuracy. In the beginning of our search we generate three random normal cells. These cells are used to search for an optimal reduction cell optimizing both measurements KN and NLR. In each search iteration we generate reduction cells and calculate the KN and NLR for each combination of normal cell and reduction cell. The reduction cells are ranked according to their mean KN and their mean NLR (mean in terms of all three normal cells). The 16 best ranked reduction cells are then used for the next iteration of reduction cell search. The reduction cell search stops, when a maximum of 104 queries is reached. After that we use the best found reduction cell in terms of the lowest KN and the highest NLR for the next search for normal cell. The next steps use this best found reduction cell as a starting point and searches for the best normal cell in the same manner es before. The search stops after a total of 208 queries and outputs an overall normal and reduction cell combination, leading to a DARTS [16] architecture, which we train on ImageNet [4] using the same training pipeline as [36].

C. Experimental Details

C.1. NAS-Bench-101

Here we include tabular results of our NAS-Bench-101 experiments from the main paper in Table 4. We report the mean and standard deviation over 10 runs.

C.2. NAS-Bench-201

We report tabular search results of our NAS-Bench-201 experiments from the main paper for different numbers of queries in Table 5. In this section, we also compare AG-Net to two recent generative models [8, 49]. SGNAS [8] trains a supernet by uniform sampling, following SETN [39]. Additionally a CNN based architecture generator is trained to search architectures on the supernet. When comparing with [26], 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. The results show, ,most importantly, that AG-Net shows strong performance in the few-query regime comparing to [26] with the exception of CIFAR-100, proving its high query efficiency.

C.3. Experiments on Hardware-Aware Benchmark

In comparison to the experiments for NAS-Bench-101 [27] and NAS-Bench-201 [5] image benchmarks, the search on the Hardware-Aware NAS-Bench [14] changes to be a multi objective learning procedure. We compare two different objective settings: i) a joint constrained optimization in Equation 4 and ii) a constrained optimization in Equation 5. For both settings we need to adapt the surrogate model by including an additional predictor $g(\cdot)$ for latency. We implement $g(\cdot)$ equally to the performance predictor $f(\cdot)$, whereas both predictors share weights in our experiments. We give a detailed overview of the hyperparameter settings in Appendix F. Since we include an additional predictor, the training objective needs to be updated, as seen in Equation 6 with multiple targets. The risk of including multiple targets to the training objective is an exploding loss leading to reduced valid generation ability of our generative network. In order to overcome this problem, we scale each loss term by the largest one, such that each term is at most 1. This way, we have a more stable training.

Table 6 shows the tabular results on different hardware and latency constraints.

Exemplary Searches for Other Devices In Figure 2 we showed an exemplary search result comparing random search with both of our constrained algorithm settings in the case of different latency constraints on a Pixel3. In the following, we show more examples on different devices in Figure 8. These plots show that both methods *Joint=1* and *Joint=0* outperform the random search baseline in all different device experiments. The same results as in the main paper holds therefore for all other devices too; *Joint=1* is able to find better performing architectures compared to *Joint=0* if the latency constraint L restricts the feasible search space strongly.

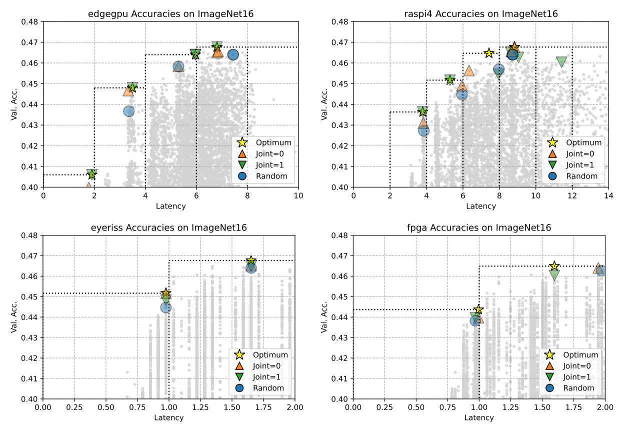

Figure 8. Exemplary searches on HW-NAS-Bench for image classification on ImageNet16 with 192 queries on Edge GPU, Raspi4, Eyeriss, FPGA and latency conditions $L \in \{2, 4, 6, 8, 10\}$, $L \in \{2, 4, 6, 8, 10, 12, 14\}$ and $L \in \{1, 2\}$ (y-axis zoomed for visibility).

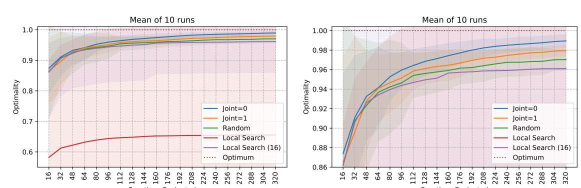

Figure 9. (**left**) Optimality for all search parameters in Table 6 at any time during the search progress in terms of the number of evaluated architectures (up to 320). Optimality is the mean validation accuracy of 10 runs per algorithm, normalized by the optimal value for each parameter setting (hence, optimum is at 1.0). (**right**) zoomed y-axis

Search Progress and Baselines Local search [25] is considered a strong baseline in NAS. In the case of constrained searches (as in HW-NAS-Bench), we noticed that it cannot perform well without adaptation. The vanilla local search algorithm expects as input a single randomly drawn architecture from the search space. However, this architecture is

NAS Method	Val. Acc (%)	StD (%)	Test Acc (%)	StD	(%)Queries
Optimum*	95.06	-	94.32	-	
Arch2vec + RL [26]	-	-	94.10	-	400
Arch2vec + BO [26]	-	-	94.05	-	400
NAO [‡] [44]	94.66	0.14	93.49	0.59	192
BANANAS[†] [24]	94.73	0.17	94.09	0.19	192
Bayesian Optimization[†] [21]	94.57	0.2	93.96	0.21	192
Local Search[†] [25]	94.57	0.15	93.97	0.13	192
Random Search[†] [15]	94.31	0.15	93.61	0.27	192
Regularized Evolution[*] [18]	94.47	0.11	93.89	0.2	192
WeakNAS [52]	-	-	**94.18**	0.14	200
AG-Net (ours)	**94.90**	0.22	**94.18**	0.10	192

Table 4. Architecture search on NAS-Bench-101. Reported is the mean and the standard deviation over 10 trials for the search of the best architecture in terms of validation accuracy on the CIFAR-10 image classification task compared to state-of-the-art methods.

NAS Method	CIFAR-10				CIFAR-100				ImageNet16-120				Queries
	Val. Acc	StD	Test Acc	StD	Val. Acc	StD	Test Acc	StD	Val. Acc	StD	Test Acc	StD	
Optimum*	91.61		94.37		73.49		73.51		46.73		47.31		
SGNAS [8]	90.18	0.31	93.53	0.12	70.28	1.2	70.31	1.09	44.65	2.32	44.98	2.10	
Arch2vec + BO [26]	91.41	0.22	94.18	0.24	**73.35**	0.32	**73.37**	0.30	46.34	0.18	46.27	0.37	100
AG-Net (ours)	**91.55**	0.08	**94.24**	0.19	73.2	0.34	73.12	0.40	46.31	0.33	46.2	0.47	96
AG-Net (ours with topk=1)	91.41	0.30	94.16	0.31	73.14	0.56	73.15	0.54	**46.42**	0.14	**46.43**	0.30	100
BANANAS[†] [24]	91.56	0.14	94.3	0.22	**73.49***	0.00	73.50	0.00	**46.65**	0.13	**46.51**	0.11	192
BO[†] [21]	91.54	0.06	94.22	0.18	73.26	0.19	73.22	0.27	46.43	0.35	46.40	0.35	192
RS [†] [15]	91.12	0.26	93.89	0.27	72.08	0.53	72.07	0.61	45.87	0.39	45.98	0.41	192
AG-Net (ours)	**91.60**	0.02	**94.37***	0.00	**73.49***	0.00	**73.51***	0.00	46.64	0.12	46.43	0.34	192
GANAS [49]	-	-	94.34	0.05	-	-	73.28	0.17	-	-	**46.80**	0.29	444
AG-Net (ours)	**91.61***	0.00	**94.37***	0.00	**73.49***	0.00	**73.51***	0.00	**46.73***	0.00	46.42	0.00	400

Table 5. Architecture Search on NAS-Bench-201. We report the mean and standard deviation over 10 trials for the search of the architecture with the highest validation accuracy. For comparable numbers of queries, AG-Net performs similarly or better than then previous state of the art.

not guaranteed to be feasible in this setting, as its latency can be larger than the latency constraint. To circumvent this, we performed local search in the following settings: (a) local search vanilla setting with one randomly drawn architecture, and (b) local search initialized with 16 randomly drawn architectures. In each setting, local search continues to search the neighborhood of the next best architecture in terms of accuracy that satisfies the latency constraint. We noticed that initializing local search with 16 randomly drawn architectures improves its performance substantially, however, it is still not on par with random search [15] in this constrained search space. Consequently, we only show random search as baseline in Table 6 to improve readability. In Figure 9 we show the progress of our algorithms (*Joint=0* and *Joint=1*) compared to random search and local search in settings (a) and (b).

D. Ablation Studies

In this section we give an overview of different ablation studies with respect to the proposed AG-Net.

D.1. Backpropagation and LSO 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 [27] and NAS-Bench-201 [5]. The results of our ablation study are reported in Table 7. 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.

D.2. Latent Space Ablations

As we have seen in subsection 3.1, AG-Net improves over state-of-the art methods. For additional comparisons, we investigate different search methods in the latent space of the generative model, with samples z from a grid and also include baselines using the LSO approach. For the first

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	**40.6***	1.90	**40.6***	1.90	39.7	1.78	**39.7**	0.29	*39.1*	0.31	37.2	0.05	40.6	1.90
Edge GPU	4	**44.8***	3.49	**44.8***	3.49	43.7	3.35	*42.8*	0.29	**43.3**	0.43	41.7	0.22	44.8	3.49
Edge GPU	6	*45.8*	5.29	**46.4***	5.96	*45.8*	5.29	**45.3**	0.64	*45.0*	0.79	44.9	0.72	46.4	5.96
Edge GPU	8	*46.5*	6.81	**46.8***	6.81	46.4	7.44	**46.3**	0.98	*46.2*	0.99	45.7	1.00	46.8	6.81
Raspi 4	2	**35.5***	1.58	**35.5***	1.58	34.8	1.60	*34.6*	0.28	**34.7**	0.30	33.9	0.08	35.5	1.58
Raspi 4	4	*43.1*	3.83	**436.***	3.79	42.7	3.85	*42.0*	0.47	**42.8**	0.50	41.9	0.37	43.6	3.79
Raspi 4	6	*44.9*	5.95	**45.2***	5.29	44.5	5.95	*44.0*	0.56	**44.1**	0.57	43.2	0.55	45.2	5.29
Raspi 4	8	*45.6*	6.33	45.5	7.96	**45.7**	7.97	**45.1**	0.69	*44.9*	0.79	44.7	0.76	46.5	7.43
Raspi 4	10	**46.6**	8.66	*46.5*	8.62	46.4	8.72	**46.4**	0.77	*45.4*	0.94	*45.4*	0.90	46.8	8.83
Raspi 4	12	**46.8***	8.83	46.3	9.05	*46.4*	8.72	**46.5**	0.91	*45.7*	0.98	45.6	0.96	46.8	8.83
Edge TPU	1	**46.8***	0.96	*46.6*	0.97	46.4	1.00	**46.4**	0.74	*45.7*	0.82	45.4	0.79	46.8	0.96
Pixel 3	2	**41.3***	1.30	**41.3***	1.30	40.0	1.50	**40.9**	0.48	*40.5*	0.59	38.8	0.30	41.3	1.30
Pixel 3	4	**46.0***	3.55	44.6	3.01	*44.7*	3.23	**45.3**	0.69	*44.1*	0.77	43.8	0.64	46.0	3.55
Pixel 3	6	*46.4*	5.92	**46.5***	5.95	45.8	4.68	**45.7**	0.77	*45.2*	0.94	45.1	0.88	46.5	5.57
Pixel 3	8	**46.8***	6.65	*46.5*	7.88	46.1	7.13	**46.4**	0.87	*45.7*	0.99	45.4	0.97	46.8	6.65
Pixel 3	10	**46.6**	6.70	46.1	8.48	*46.4*	8.01	**46.4**	0.96	45.5	1.00	*45.6*	0.99	46.8	6.65
Eyeriss	1	**45.2***	0.98	*44.9*	0.98	44.7	0.98	**44.5**	0.49	*43.6*	0.53	43.3	0.23	45.2	0.98
Eyeriss	2	**46.5**	1.65	**46.5**	1.65	46.4	1.65	**46.3**	0.87	*45.7*	0.99	*45.7*	0.95	46.8	1.65
FPGA	1	**44.0**	1.00	**44.0**	0.97	43.8	0.97	**43.3**	0.65	**43.3**	0.80	42.9	0.58	44.4	1.00
FPGA	2	**46.5***	1.60	46.0	1.60	*46.3*	1.97	**46.2**	0.82	45.1	0.99	*45.3*	0.97	46.5	1.60

Table 6. Results for searches with at most 200 queries on HW-NAS-Bench [14] 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 (in %) 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 [15]. 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.

	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**

Table 7. 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).

experiment we use the generator solely as a data sampler from the generator's latent space without any retraining, for the latter baseline we retrain the generator during the search. For the optimization, we use Bayesian optimization, local search and random search.

Bayesian Optimization We use DNGO [21] as our uncertainty prediction model for the Bayesian optimization search strategy, with the basis regression network being a one-layer MLP with a hidden dimensionality of 128, which is trained for 100 epochs and expected improvement (EI) [47] as our acquisition function, which is mostly used in NAS. We set the best function value for the EI evaluation as the best validation accuracy of the training data. We sample 16 initial random latent space variables $\mathbf{z} \sim \mathcal{U}[-3,3]$ and decode them to graph data using our pretrained generative model. These latent space variables and their corresponding validation architecture performances are then the inputs for the DNGO model for training. Again, the best 16 architectures are selected using EI in each round to be evaluated and added to the training data. This search ends when the total query amount of 300 is reached.

Random and Local Search In addition to Bayesian Optimization as a comparison, we also include a random search [15] and local search investigation. Recently, [25] show

	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
Random Search	94.27	93.65	91.37	93.92	72.55	72.49	46.09	46.05
Local Search	94.31	93.66	91.28	94.01	72.52	72.59	45.89	46.07
Bayesian Optimization	94.27	93.62	91.30	93.99	72.23	72.35	46.09	46.01
Random Search + LSO	94.64	**94.20**	**91.61***	**94.37***	**73.49***	**73.51***	**46.77***	45.47
Local Search + LSO	94.17	93.50	91.30	93.96	72.43	72.58	45.83	45.95
Bayesian Optimization +LSO	94.50	93.96	91.43	94.17	72.64	72.67	46.30	45.91
SGNAS [8] + LSO	-	-	**91.61***	**94.37***	73.04	73.12	46.56	**46.32**
AG-Net (ours)	**94.96**	**94.20**	**91.61***	**94.37***	**73.49***	**73.51***	46.67	46.22

Table 8. Ablation: Search results on NAS-Bench-101 and NAS-Bench-201 on the AG-Net latent space (mean over 10 trials with a maximal query amount of 300).

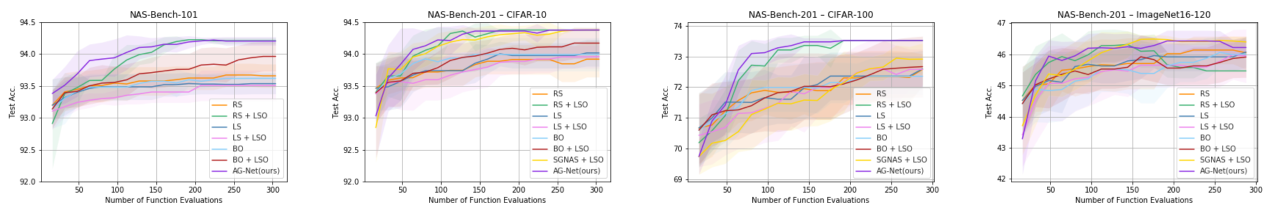

Figure 10. Ablation: neural architecture search on NAS-Bench-101 and NAS-Bench-201 over 10 trials.

that local search is a powerful NAS baseline, resulting in competitive results. Local search [25] evaluates samples and their neighborhood uniformly at random. An option to define the neighborhood is the set of architectures which differ from a sampled architecture by one node or edge. This can be done only in the discrete search space, given for example by the tabular NAS-Benchmarks. We have to adapt the neighborhood definition in our latent space for local search in this space. We sample a latent space variable $\mathbf{z} \sim \mathcal{U}[-3,3]$, decode it and evaluate the generated neural architecture. Here, we define neighborhood as the Euclidean space around the sampled latent variable $U_\epsilon(z) = \{y \sim \mathcal{U}[-3,3] | d(z,y) < \epsilon\}$, with ϵ being sufficiently small. This neighborhood is then investigated until a local optimum in terms of validation accuracy is reached. Furthermore, we include a random search and local search comparison using weighted retraining. Here, we retrain the generative model in each search iteration for 1 epoch with the weighted objective function, ceteris paribus.

To compare with weight-sharing approaches, we also compare to the supernet from [8] for the NAS-Bench-201 search space. To compare our AG-Net with SGNAS, we use the supernet as our surrogate model to predict the architectures performance while retraining the generative model in the weighted manner. The results of our ablation studies are reported in Table 8. AG-Net improves over search methods on the latent space with and without LSO on both benchmarks, demonstrating that our generator in combination with our MLP surrogate model learns to adapt the distribution shift constructed by the weighted retraining best.

For further visualizations we also plot different ablation search methods over different query number in Figure 10 for both benchmarks NAS-Bench-101 and NAS-Bench-201. This figure demonstrates the high any-time performance of our method on both search spaces. For any number of available queries, our model is better in finding high-performing architectures from the latent space than other latent space based methods.

E. Generator Details

Figure 11 presents an overview of the training process of the proposed generative model.

E.1. Generator Evaluation

Based on an investigation of autoencoder abilities from [26] and [17], we can examine the generation ability of our generative model. For that we train our generator on 90% of the overall dataset, and thus have a hold-out dataset of 10% for the tabular benchmarks. The generative model training on the surrogate benchmarks is a priori only on a subset of the overall dataset. Additionally, we sample 10,000 random variables $\mathbf{z} \sim \mathcal{N}(0,1)$ and decode them to graphs. We report the results of this investigation in Table 9.

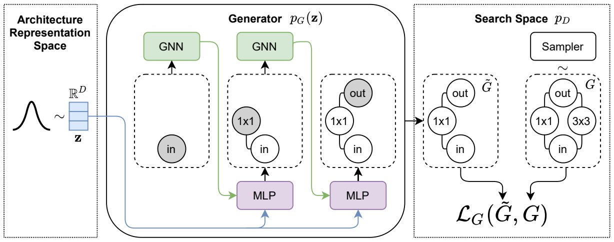

Figure 11. Representation of the training procedure for our generator in AG-Net. The input is a randomly sampled latent vector $\mathbf{z} \in \mathbb{R}^d$. First, the input node is generated, initialized and input to a GNN to generate a partial graph representation. The learning process iterative generates 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.

Search Space	Validity (in %)	Uniqueness in (%)	Novelty in (%)
NAS-Bench-101	71.69	97.92	62.30
NAS-Bench-201	99.97	73.61	10.03
NAS-Bench-301	42.27	100	100
NAS-Bench-NLP	57.95	100	100

Table 9. Generator Abilities. The proposed generator generates architectures with high validity and uniqueness scores. The novelty scores are in a similar range as for previous methods [17].

Here, validity describes the ratio of valid graphs our generator model generates, uniqueness describes the portion of unique graphs from the valid generated graphs, and novelty is the portion of generated graphs *not in the training set*. It is not surprising for the NAS-Bench-301 and NAS-Bench-NLP search spaces, that our model is able to generate 100% unique and novel graphs, given the large size of both search spaces. This demonstrates that our simple generator model is able to generate valid graphs with high novelty and consequently is able to cover a substantial part of the search space.

E.2. Generator Implementation Details

In this section we present more details about the generation model SVGe from [17]. The pseudo agorithm is described in algorithm 3. The modules f_{initNode}, f_{addNode}, f_{addEdges}, f_{Embeding} used in this code are two-layer MLPs with ReLU activation functions. Note, in contrast to SVGe, we don't sample within the generation process, in order to allow for end-to-end learning with the prediction model for AG-Net.

F. Hyperparameters

In this section we give a detailed overview about the hyperparameter for our generative network. We use pytorch [48] and pytorch geometric [40] for all our implementations.

F.1. Generator

Table 10 presents all used hyperparameters for the generation training. We train our generator in a ticks manner; after every 5.000 train data, we evaluate our generator for validity ability. The used pretrained state dict for our search is then, the one, which the highest validation measurement, which is defined by randomly sample $10,000$ latent vectors $\mathbf{z} \in \mathbb{R}^{32}$ and generate architectures. The training is the same for all different search spaces.

F.2. Surrogate Model

The overall surrogate is an MLP with ReLU activations. Table 11 and Table 12 list all used hyperparameter used for the search experiments in the main paper for the simple performance surrogate model and the multi-objective surrogate model for the additional hardware objective.

G. Latent Space Optimization Visualization

A more descriptive visualization of the latent space optimization technique used for our AG-Net neural architecture search is displayed in Figure 12.

Algorithm 3: Graph Generation

Input: $\mathbf{z} \sim \mathcal{N}(0,1)$

Output: random sampled reconstructed graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$

initialize one-hot encoded InputNode v_0, with embedding $\mathbf{h}_0 \leftarrow f_{\text{initNode}}(\mathbf{z}, f_{\text{Embeding}})[\text{InputType}])$

$V \leftarrow \{v_0\}$, $E \leftarrow \emptyset$, $\mathbf{h}_G \leftarrow \mathbf{z}$,

while $|V| \leq$ *Max Number of Nodes* **do**

  $v_{t+1} \leftarrow f_{\text{addNode}}(\mathbf{z}, \mathbf{h}_G)$

  $V \leftarrow V \cup \{v_{t+1}\}$

  $\mathbf{h}_{t+1} \leftarrow f_{\text{initNode}}(\mathbf{z}, \mathbf{h}_G, f_{\text{Embeding}}(v_{t+1})])$

  for $v_j \in V \setminus v_{t+1}$ **do**

    $s_{\text{addEdges}}(j, t+1) \leftarrow f_{\text{addEdges}}(\mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{h}_G, \mathbf{z})$

    $e_{(j,t+1)} \sim \text{Eval}(s_{\text{addEdges}}(j, t+1))$; ▷ `evaluate whether to add edge`

    if $e_{(j,t+1)} = 1$ **then**

      $E \leftarrow E \cup \{e_{(j,t+1)} = (v_j, v_{t+1})\}$

    end

  end

  $\mathbf{h}_t \leftarrow \text{concat}(\mathbf{h}_t, \mathbf{h}_{t+1})$

  $G \leftarrow (V, E)$

  $\mathbf{h}_t \leftarrow (\mathbf{h}_t, G)$; ▷ `update node embeddings`

  $\mathbf{h}_G \leftarrow \text{aggregate}(\mathbf{h}_t)$; ▷ `update graph embedding`

  $t \leftarrow t + 1$

end

$V \sim \text{Categorical}(V)$; ▷ `Sample node types`

$E \sim \text{Ber}(E)$; ▷ `Sample edges`

$\widetilde{G} = (V, E)$

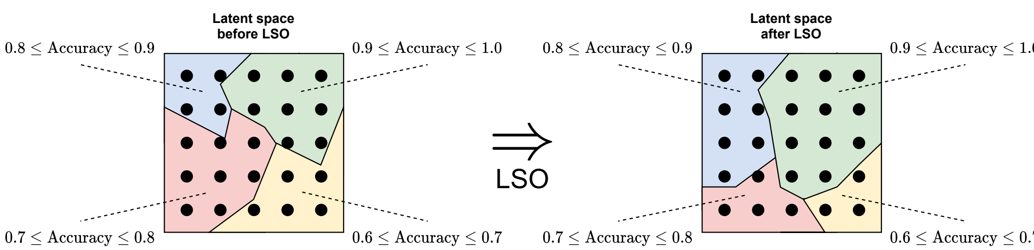

Figure 12. The latent space is reshaped in a way that promotes desired properties of generated architectures (in this example: accuracy). Consequently, it becomes more likely for the generator to generate architectures satisfying this property.

Hyperparameter	Default Value
Node Embedding	32
Latent Vector	32
MLP Node Embedding layer	2
GNN layer	2
Batch Size	32
Optimizer	Adam [42]
Learning Rate	0.0002
Betas	(0.5, 0.999)
Ticks	500
Tick Size	5,000

Table 10. Hyperparameters of the generator model.

Hyperparameter	Dataset			
	NB101	NB201	NB301	NBNLP
α	0.9			
MLP Layers	4			
MLP Hidden	56	84	176	559
Epochs	15	30	15	30
Optimizer	Adam [42]			
LR	0.001			
Betas	(0.5, 0.999)			
weight factor	10 e-3			
batch size	16			
loss	L2			

Table 11. Hyperparameters for the performance surrogate model $f(\cdot)$

Hyperparameter	Hardware-Aware NASBench
α	0.95
λ	0.5
MLP Layers	4
MLP Hidden	82
Epochs	30
Optimizer	Adam [42]
LR	0.002
Betas	(0.5, 0.999)
weight factor	10 e-3
penalty term	1000
batch size	16
loss	L2

Table 12. Hyperparameters for both surrogate models $f(\cdot)$ and $g(\cdot)$ for the multi objective search in the Hardware-Aware Benchmark

References

[30] Google llc. edge tpu compiler. https://coral.ai/docs/dev-board/get-started/. Accessed: 2021-11-17. 5

[31] Google llc. pixel 3. https://g.co/kgs/pVRc1Y. Accessed: 2021-11-17. 5

[32] Nvidia jetson tx2. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/. Accessed: 2021-11-17. 5

[33] Raspberry pi limited. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. Accessed: 2021-11-17. 5

[34] Xilinx inc. vivado high-level synthesis. https://https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html. Accessed: 2021-11-17. 5

[35] Xilinx zynq-7000 soc zc706 evaluation kit. https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html. Accessed: 2021-11-17. 5

[36] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four GPU hours: A theoretically inspired perspective. In *ICLR*, 2021. 9, 10

[37] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019. 9

[38] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *International Solid-State Circuits Conference, ISSCC*, 2016. 5

[39] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *ICCV*, 2019. 10

[40] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 14

[41] Sian-Yao Huang and Wei-Ta Chu. Searching by generating: Flexible and efficient one-shot NAS with architecture generator. In *CVPR*, 2021. 2, 10, 11, 13

[42] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 15, 16

[43] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 9

[44] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018. 9, 11

[45] Yash Mehta, Colin White, Arber Zela, Arjun Krishnakumar, Guri Zabergja, Shakiba Moradian, Mahmoud Safari, Kaicheng Yu, and Frank Hutter. Nas-bench-suite: NAS evaluation is (now) surprisingly easy. *CoRR*, abs/2201.13396, 2022. 6, 7, 8

[46] Tomás Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, 2010. 7

[47] Jonas Mockus. On bayesian methods for seeking the extremum. In *Optimization Techniques, IFIP Technical Conference, Novosibirsk, USSR, July 1-7, 1974*. Springer, 1974. 12

[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035. 2019. 14

[49] Seyed Saeed Changiz Rezaei, Fred X. Han, Di Niu, Mohammad Salameh, Keith G. Mills, Shuo Lian, Wei Lu, and Shangling Jui. Generative adversarial neural architecture search. In *IJCAI*, 2021. 10, 11

[50] 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. *CoRR*, abs/2008.09777, 2020. 6, 8, 9

[51] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019. 5

[52] Junru Wu, Xiyang Dai, Dongdong Chen, Yinpeng Chen, Mengchen Liu, Ye Yu, Zhangyang Wang, Zicheng Liu, Mei Chen, and Lu Yuan. Stronger nas with weaker predictors. In *NeurIPS*, 2021. 9, 11

[53] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019. 9

[54] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: partial channel connections for memory-efficient architecture search. In *ICLR*, 2020. 9

[55] Shen Yan, Colin White, Yash Savani, and Frank Hutter. Nas-bench-x11 and the power of learning curves. 6, 7, 8