Learning Where To Look – Generative NAS is Surprisingly Efficient: Supplementary Material

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

Section A provides an overview about the graph representations for each search space, we consider in the main paper. In section B we show additional ablation studies. In section C, we provide more details about the experimental settings. In subsection C.5 we provide additional details for our search method on the DARTS search space. In section D we describe details about the generator network, and in section E we list all hyperparameter settings of our experiments. Lastly, we include a visual intuition of the latent space optimization technique in section F.

A Search Space Representations

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

A.1 NAS-Bench-101

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 [17] for image classification. The cell topology is limited to the number of nodes $|V| \leq 7$ (including input and output nodes) and edges $|E| \leq 9$. The nodes represent the architecture layers and intermediate nodes can take any operation from the operation set $\mathcal{O} = \{1 \times 1 \text{ conv.}, 3 \times 3 \text{ conv.}, 3 \times 3 \text{ max pooling}\}$. For visualization purposes, we present in Figure 5 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.

A.2 NAS-Bench-201

NAS-Bench-201 [12] is another cell-structured search space, which consists of $15,625$ architectures. Each architecture is trained for 200 training epochs on

*Authors contributed equally.

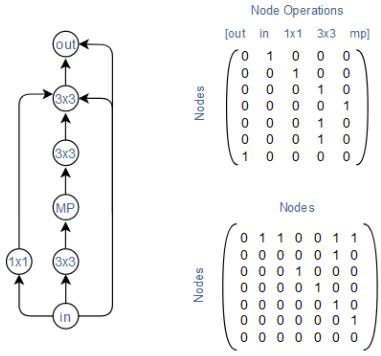

Fig. 5: Exemplary cell representation from the NAS-Bench-101 search space. (**left**) DAG representation of a graph with 7 nodes. (**right**) The top part shows the node attribute matrix to the DAG and the bottom part shows its adjacency matrix.

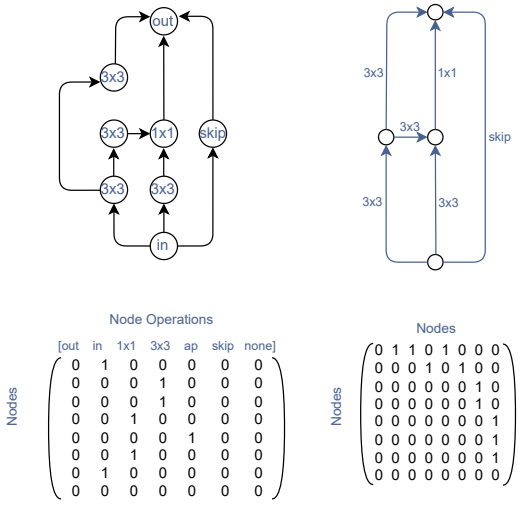

Fig. 6: Exemplary cell representation from the NAS-Bench-201 search space. (**top**) The left part visualizes the DAG representation with node attributes instead of edge attributes. The right part shows the true DAG representation in the NAS-Bench-201 search space. (**bottom**) The left part shows the node attribute matrix to the DAG and the right part shows its adjacency matrix.

CIFAR-10 [17], CIFAR-100 [17], and ImageNet16-120 [10]. 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 $\mathcal{O} = \{1 \times 1 \text{ conv.}, 3 \times 3 \text{ conv.}, 3 \times 3 \text{ avg pooling}, \text{skip}, \text{zero}\}$.

Figure 6 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 [36]. We show experiments on NAS-Bench-101 and NAS-Bench-201 in subsection 4.1.

A.3 DARTS Search Space

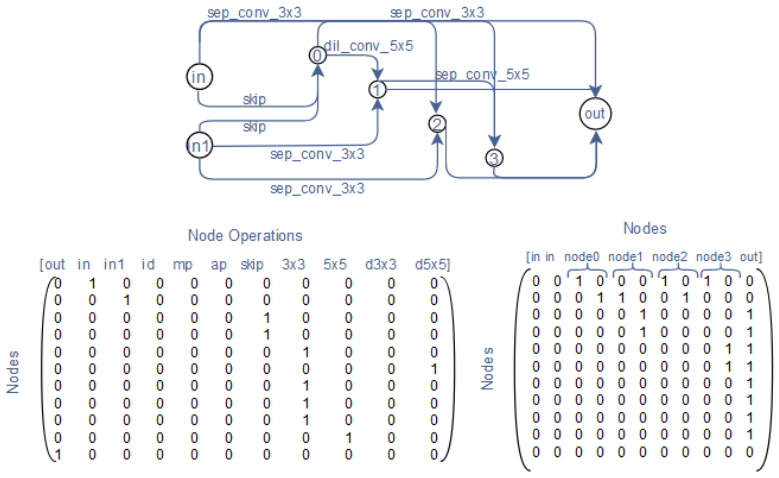

Fig. 7: Exemplary cell representation from the DARTS search space. (**top**) Visualization of the DAG representation in the DARTS search space. (**bottom**) The left part shows the node attribute matrix to the DAG and the right part shows its adjacency matrix.

NAS-Bench-301 [29] is the first surrogate benchmark, which evaluates several surrogate models on in total $60,000$ sampled architectures from the DARTS [20] search space on the CIFAR-10 [17] 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 $\mathcal{O} = \{3 \times 3$ sep. conv., 5×5 sep. conv., 3×3 dil. conv., 5×5 dil. conv., 3×3 avg pooling, 3×3 max pooling, identity, 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 [20]. This is similar to the representation in [36].

A.4 NAS-Bench-NLP

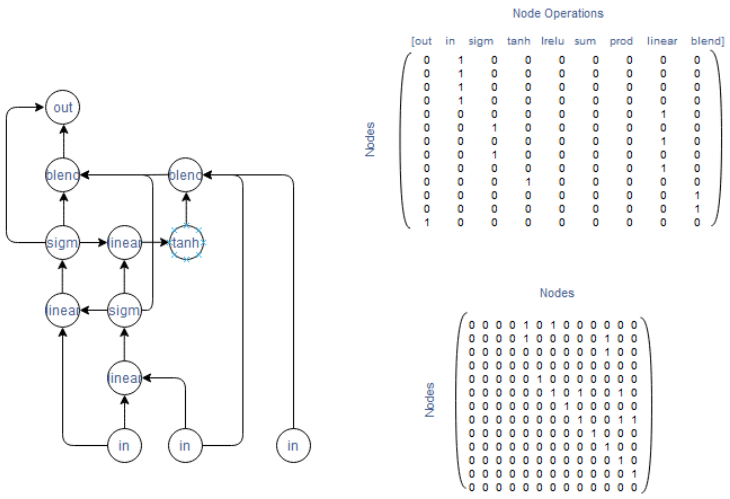

Fig. 8: Exemplary cell representation from the NAS-Bench-NLP search space. (**left**) DAG representation of a graph with 12 nodes. (**right**) The top part shows the node attribute matrix to the DAG and the bottom part shows its adjacency matrix.

NAS-Bench-NLP [16] 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 [24] (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 $\mathcal{O} = \{\text{linear}, \text{ element wise blending}, \text{ element wise product}, \text{ element wise sum}, \text{ Tanh activation}, \text{ Sigmoid activation}, \text{ LeakyReLU activation}\}$.

For the experiments on NAS-Bench-NLP [16] we make use of the surrogate benchmark NAS-Bench-X11 [35] and the additional implementation in NAS-Bench-Suite [23]. 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 [35] with the modification that each hidden node is connected to the output node. An exemplary architecture representation is visualized in Figure 8. A next step is to analyse the $14,332$ provided architectures on uniqueness, which leads to $12,107$ unique architectures. Furthermore, since [35] and [23] 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.

We show experiments in the DARTs search space and on NAS-Bench-NLP in subsection 4.2.

A.5 Hardware-Aware-NAS-Bench

The recently introduced HW-NAS-Bench [18] is the first public dataset for hardware NAS. It extends two representative NAS search spaces, NAS-Bench-201 [12] and FBNet [33], 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* [3], *Raspi 4* [4], *Edge TPU* [1], *Pixel 3* [2], *ASIC-Eyeriss* [9] and *FPGA* [5,6].

In our experiments in subsection 4.3 we consider the latency information on the NAS-Bench-201 search space.

B Additional Ablation Studies

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

B.1 Oracle Ablation

As we have seen in the previous section, our model AG-Net is able to find high-scoring architectures in various search spaces of different sizes and with different objectives. In addition, including the supposedly stronger predictor XGB [7] leads to improvements for the search on NAS-Bench-NLP [16]. In this section, we include an even stronger architecture accuracy evaluation model, i.e. the benchmark query input itself (oracle).

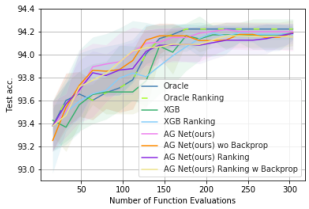

Fig. 9: Architecture search on NAS-Bench-101. Reported is the mean over 10 trials for the search of the best architecture in terms of validation accuracy on the CIFAR-10 image classification task compared to strong predictor models.

Table 7: 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).

	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 [14] + 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

The comparison of the oracle benchmark (also including the ranking metric as for XGB in the main paper) to our AG-Net and XGB modifications are visualized in Figure 9. This figure demonstrates the high performance of our model in the low query area. The more queries are evaluated for the search, the better the oracle becomes, outperforming all other methods after 150 queries.

B.2 Latent Space Ablations

As we have seen in subsection 4.1, 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 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[30] 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

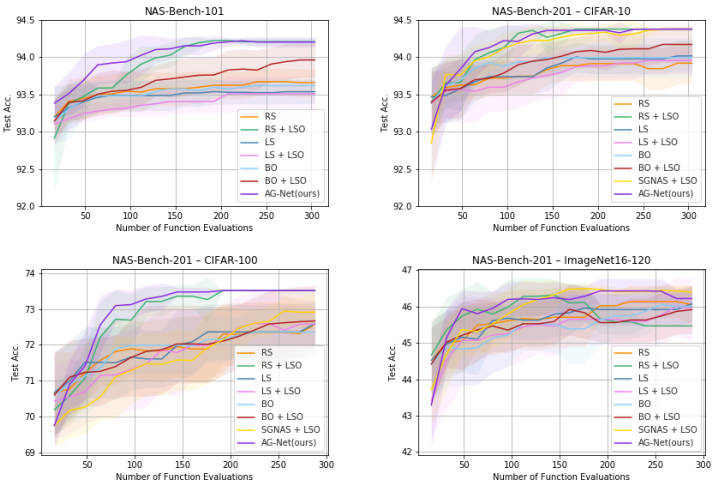

Fig. 10: Ablation: neural architecture search on NAS-Bench-101 and NAS-Bench-201 over 10 trials.

for 100 epochs and expected improvement (EI) [25] 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 [19] and local search investigation. Recently, [32] show that local search is a powerful NAS baseline, resulting in competitive results. Local search [32] 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 [14] 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 7. 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 numbers 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.

B.3 Predictor Ablation – Local Solution

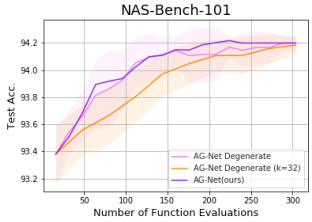

Fig. 11: Architecture search on NAS-Bench-101 in the degenerate setting. Reported is the mean over 10 trials.

Our proposed method, consisting of the generative and surrogate model combined with the latent space optimization, makes the architecture search focus on promising regions in the search space. This method could be trapped in local solutions, which we investigate experimentally in the following. First, the previous section already points out that our proposed method AG-Net improves over both local search methods with and without the latent space optimization approach. Thus, we assume that the latent space optimization learns properties of high-scoring architectures without being easily trapped in poor local solutions. The amount of samples drawn in each search iteration also provides a trade-off between diversity versus specificity. To investigate further how easily AG-Net could be trapped in a local solution, we test our method when it only uses in total the best k (predicted) architectures from our test samples and the training data as a new training set for the next search iteration (degenerative) and is thereby encouraged to forget about worse performing architectures. Figure 11 shows the search behaviour of the degenerative model with $k = 16$ and $k = 32$. Even in this case, AG-Net is not easily trapped in poor solutions.

C Experiments: Implementation Details

C.1 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, see section A for visualizations. 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.

We consider as an alternative surrogate model the XGB [7] prediction model. The input to this prediction model is the vector representation of the architecture.

C.2 Search Algorithm

High-level descriptions of the unconstrained (subsection 4.1) and constrained (subsection 4.3) versions of our search algorithm are depicted in algorithm 1 and algorithm 2 respectively.

C.3 NAS-Bench-101

In this section, we give more information about the NAS-Bench-101 experiments from the main paper.

Table 8 is the detailed version of Table 1 including the standard deviation.

C.4 NAS-Bench-201

Table 9 is the detailed version of Table 2 including the standard deviation.

C.5 DARTS Search Space

Additional Results Table 10 is the detailed version of Table 3 including the standard deviation.

Search Process using NAS-Bench-301 For experiments in the DARTS [20] 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 cell representations in the DARTS search space allows for searching well-performing architectures. Here we describe the search process for architectures evaluated on CIFAR-10 using the surrogate benchmark NAS-Bench-301 [29]. Since the

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

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 between the DAG. 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 then follows the same steps as for the tabular benchmarks and stops after we reach a query amount of 155. Now, we can 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

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 accuracies 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

a random reduction cell is analogous. In the main paper, we report the search outcome for NAS-Bench-301 [29] starting with a random reduction cell.

Search Process using TENAS As we described in the previous section, the search in the DARTS [20] 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 [8], which ranks architectures by analysing the neural

Table 8: 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	**Val. Acc** (%)	**StD** (%)	**Test Acc** (%)	**StD**	(%)**Queries**
Optimum*	95.06	-	94.32	-	
Arch2vec + RL [36]	-	-	94.10	-	400
Arch2vec + BO [36]	-	-	94.05	-	400
NAO ‡[22]	94.66	0.14	93.49	0.59	192
BANANAS† [31]	94.73	0.17	94.09	0.19	192
Bayesian Optimization† [30]	94.57	0.2	93.96	0.21	192
Local Search† [32]	94.57	0.15	93.97	0.13	192
Random Search†[19]	94.31	0.15	93.61	0.27	192
Regularized Evolution*[27]	94.47	0.11	93.89	0.2	192
WeakNAS [34]	-	-	**94.18**	0.14	200
XGB (ours)	94.61	0.04	94.13	0.11	192
XGB + ranking (ours)	94.60	0.08	94.14	0.19	192
AG-Net (ours)	**94.90**	0.22	**94.18**	0.10	192

tangent kernel, by its condition number (KN), and the number of linear regions (NLR) of each architecture. Concretely, for the search on ImageNet [11] 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 KN and NLR measurements. 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 a 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 as before. The search stops after a total of 208 queries and outputs an overall normal and reduction cell combination, leading to a DARTS [20] architecture, which we train on ImageNet [11] using the same training pipeline as [8].

C.6 NAS-Bench-NLP

Table 11 is the detailed version of Table 3 including the standard deviation.

C.7 Hardware-Aware NAS-Bench

In comparison to the experiments for NAS-Bench-101 [37] and NAS-Bench-201 [12] image benchmarks, the search on the Hardware-Aware NAS-Bench [18]

Table 9: 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 the previous state of the art.

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 [14]	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 [36]	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† [31]	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† [30]	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 † [19]	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
XGB (ours)	91.54	0.09	94.34	0.10	73.10	0.51	72.93	0.74	46.48	0.13	46.08	0.79	192
XGB + Ranking (ours)	91.48	0.12	94.25	0.15	73.20	0.36	73.24	0.34	46.40	0.28	46.16	0.64	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 [28]	-	-	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 10: 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.

NAS Method	**Val. Acc** (%)	**StD** (%)	**Queries**
BANANAS† [31]	94.77	0.10	192
Bayesian Optimization† [30]	94.71	0.10	192
Local Search† [32]	**95.02**	0.10	192
Random Search†[19]	94.31	0.12	192
Regularized Evolution†[27]	94.75	0.11	192
XGB (ours)	94.79	0.13	192
XGR + Ranking (ours)	94.76	0.14	192
AG-Net (ours)	94.79	0.12	192

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 section E. 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 11: 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.

NAS Method	**Val. Perplexity** (%)	**StD** (%)	**Queries**
BANANAS† [31]	95.68	0.16	304
Local Search† [32]	95.69	0.18	304
Random Search†[19]	95.64	0.19	304
Regularized Evolution†[27]	95.66	0.21	304
XGB (ours)	**95.95**	0.20	304
XGR + Ranking (ours)	95.92	0.19	304
AG-Net (ours)	95.86	0.18	304

Exemplary Searches for Other Devices In Figure 4 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 12. 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.

Search Progress and Baselines Local search [32] 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 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 [19] in this constrained search space. Consequently, we only show random search as the baseline in Table 5 to improve readability. In Figure 13 we show the progress of our algorithms (*Joint=0* and *Joint=1*) compared to random search and local search in settings (a) and (b).

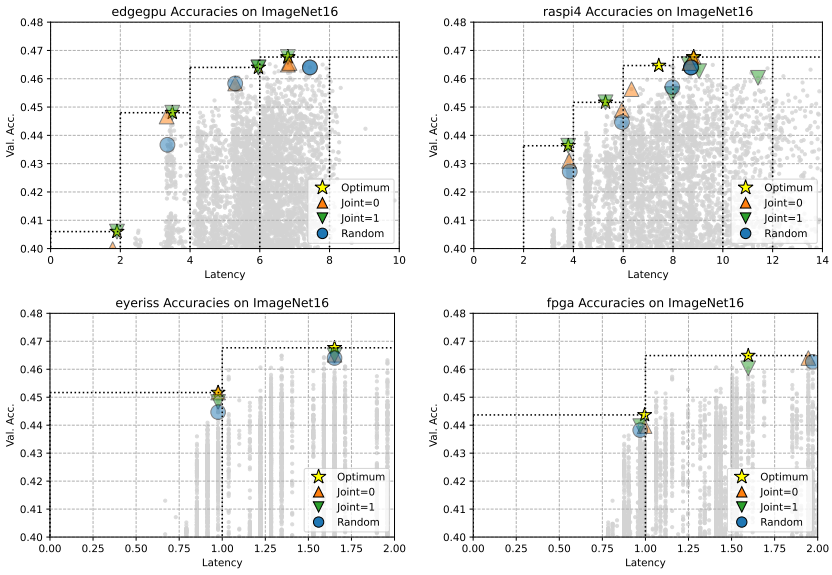

Fig. 12: Exemplary searches on HW-NAS-Bench for image classification on ImageNet16 with 192 queries on Edge GPU, Raspi4, Eyeriss, FPGA and latency conditions $L \in \{2, 4, 6, 8, 10\}$, $L \in \{2, 4, 6, 8, 10, 12, 14\}$ and $L \in \{1, 2\}$ (y-axis zoomed for visibility).

D Generator Details

D.1 Generator Evaluation

Based on an investigation of autoencoder abilities from [36] and [21], 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 12. 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.

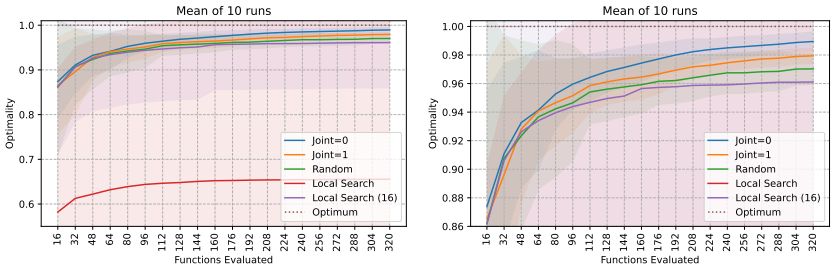

Fig. 13: (**left**) Optimality for all search parameters in Table 5 at any time during the search progress in terms of the number of evaluated architectures (up to 320). Optimality is the mean validation accuracy of 10 runs per algorithm, normalized by the optimal value for each parameter setting (hence, optimum is at 1.0). (**right**) zoomed y-axis

Table 12: Generator Abilities and training costs. The proposed generator generates architectures with high validity and uniqueness scores. The novelty scores are in a similar range as for previous methods [21].

Search Space	Validity (in %)	Uniqueness in (%)	Novelty in (%)	Training (in GPU days)
NAS-Bench-101	71.69	97.92	62.30	0.4
NAS-Bench-201	99.97	73.61	10.03	0.3
NAS-Bench-301	42.27	100	100	0.9
NAS-Bench-NLP	57.95	100	100	0.7

Table 12 also reports the training costs of the generative model on the complete dataset as described in section A on a single Tesla V100. We used for the experiments the OMNI cluster from the University of Siegen.

D.2 Generator Implementation Details

In this section we present more details about the generation model SVGe from [21]. The pseudo algorithm is described in algorithm 3. The modules f_{initNode}, $f_{\text{addNode}}, f_{\text{addEdges}}, f_{\text{Embedding}}$ 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.

E Hyperparameters

In this section we give a detailed overview about the hyperparameter for our generative network. We use pytorch [26] and pytorch geometric [13] for all our implementations.

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 $\mathbf{h}_0 \leftarrow f_{\text{initNode}}(\mathbf{z}, f_{\text{Embedding}})[\text{InputType}])$
$V \leftarrow \{v_0\}$, $E \leftarrow \emptyset$, $\mathbf{h}_G \leftarrow \mathbf{z}$,
while $|V| \leq$ *Max Number of Nodes* **do**
  $v_{t+1} \leftarrow f_{\text{addNode}}(\mathbf{z}, \mathbf{h}_G)$
  $V \leftarrow V \cup \{v_{t+1}\}$
  $\mathbf{h}_{t+1} \leftarrow f_{\text{initNode}}(\mathbf{z}, \mathbf{h}_G, f_{\text{Embedding}}(v_{t+1})])$
  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`
$\widetilde{G} = (V, E)$

E.1 Generator

Table 13 presents all used hyperparameters for the generation training. We train our generator in a ticked 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.

E.2 Surrogate Model

The overall surrogate is an MLP with ReLU activations. Table 14 and Table 15 list all hyperparameters 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. The hyperparameters for XGB [7] are the same as in [23].

Table 13: Hyperparameters of the generator model.

Hyperparameter	Default Value
Node Embedding	32
Latent Vector	32
MLP Node Embedding layer	2
GNN layer	2
Batch Size	32
Optimizer	Adam [15]
Learning Rate	0.0002
Betas	(0.5, 0.999)
Ticks	500
Tick Size	5,000

Table 14: Hyperparameters for the performance surrogate model $f(\cdot)$

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 [15]			
LR	0.001			
Betas	(0.5, 0.999)			
weight factor	10 e-3			
batch size	16			
loss	L2			

F 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 14.

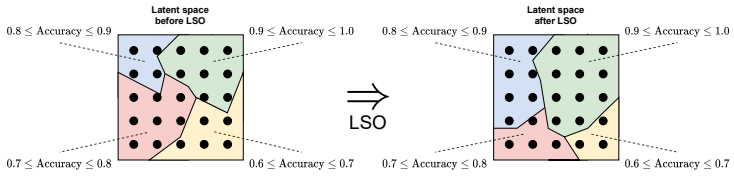

Fig. 14: The latent space is reshaped in a way that promotes desired properties of generated architectures (in this example: accuracy). Consequently, it becomes more likely for the generator to generate architectures satisfying this property.

Table 15: Hyperparameters for both surrogate models $f(\cdot)$ and $g(\cdot)$ for the multi-objective search in the Hardware-Aware Benchmark

Hyperparameter	Hardware-Aware NASBench
α	0.95
λ	0.5
MLP Layers	4
MLP Hidden	82
Epochs	30
Optimizer	Adam [15]
LR	0.002
Betas	(0.5, 0.999)
weight factor	10 e-3
penalty term	1000
batch size	16
loss	L2

References

1. Google llc. edge tpu compiler. `https://coral.ai/docs/dev-board/get-started/`, accessed: 2021-11-17
2. Google llc. pixel 3. `https://g.co/kgs/pVRc1Y`, accessed: 2021-11-17
3. Nvidia jetson tx2. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/`, accessed: 2021-11-17
4. Raspberry pi limited. `https://www.raspberrypi.org/products/raspberry-pi-4-model-b/`, accessed: 2021-11-17
5. Xilinx inc. vivado high-level synthesis. `https://https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html`, accessed: 2021-11-17
6. Xilinx zynq-7000 soc zc706 evaluation kit. `https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html`, accessed: 2021-11-17
7. 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)
8. Chen, W., Gong, X., Wang, Z.: Neural architecture search on imagenet in four GPU hours: A theoretically inspired perspective. In: ICLR (2021)
9. Chen, Y., Krishna, T., Emer, J.S., Sze, V.: 14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In: International Solid-State Circuits Conference, ISSCC (2016)
10. Chrabaszcz, P., Loshchilov, I., Hutter, F.: A downsampled variant of imagenet as an alternative to the CIFAR datasets. CoRR **abs/1707.08819** (2017)
11. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: CVPR (2009)
12. Dong, X., Yang, Y.: Nas-bench-201: Extending the scope of reproducible neural architecture search. In: ICLR (2020)
13. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
14. Huang, S., Chu, W.: Searching by generating: Flexible and efficient one-shot NAS with architecture generator. In: CVPR (2021)
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)

16. 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)
17. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)
18. 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)
19. Li, L., Talwalkar, A.: Random search and reproducibility for neural architecture search. In: UAI (2019)
20. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search (2019)
21. Lukasik, J., Friede, D., Zela, A., Hutter, F., Keuper, M.: Smooth variational graph embeddings for efficient neural architecture search. In: IJCNN (2021)
22. Luo, R., Tian, F., Qin, T., Chen, E., Liu, T.: Neural architecture optimization. In: NeurIPS (2018)
23. 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)
24. 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)
25. Mockus, J.: On bayesian methods for seeking the extremum. In: Optimization Techniques, IFIP Technical Conference, Novosibirsk, USSR, July 1-7, 1974. Springer (1974)
26. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: NeurIPS, pp. 8024–8035 (2019)
27. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: AAAI (2019)
28. 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)
29. 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)
30. 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)
31. White, C., Neiswanger, W., Savani, Y.: Bananas: Bayesian optimization with neural architectures for neural architecture search. In: AAAI (2021)
32. White, C., Nolen, S., Savani, Y.: Exploring the loss landscape in neural architecture search (2021)
33. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: CVPR (2019)
34. 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)
35. Yan, S., White, C., Savani, Y., Hutter, F.: Nas-bench-x11 and the power of learning curves

36. Yan, S., Zheng, Y., Ao, W., Zeng, X., Zhang, M.: Does unsupervised architecture representation learning help neural architecture search? In: NeurIPS (2020)
37. Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: Nas-bench-101: Towards reproducible neural architecture search. In: ICML (2019)