PINAT: A Permutation INvariance Augmented Transformer for NAS Predictor - Supplementary Materials

Shun Lu[1, 2], **Yu Hu**[1,2*], **Peihao Wang**[3], **Yan Han**[3], **Jianchao Tan**[4], **Jixiang Li**[4], **Sen Yang**[5], **Ji Liu**[6]

[1] Research Center for Intelligent Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences
[2] School of Computer Science and Technology, University of Chinese Academy of Sciences
[3] University of Texas at Austin [4] Kuaishou Technology. [5] Snap Inc. [6] Meta Platforms, Inc.
{lushun19s, huyu}@ict.ac.cn, {peihaowang, yh9442}@utexas.edu, {jianchaotan, lijixiang}@kuaishou.com,
syang3@snap.com, ji.liu.uwisc@gmail.com

A Overview

In the following, we first introduce the social impact of our work in Section B. Then we present more ablation studies to clarify the necessity of the positional encodings, the chosen hidden dimension of PIM, different combinations of SA and PISA heads number, comparison with TNASP (Lu et al. 2021) about the performance of our search architectures, and the application of the ranking loss in Section C. Next, we give a detailed description of our implementation and explored search spaces in Section D. Finally, we provide more numerical performance and structure visualization of all searched architectures in Section E.

B Social impact

In this work, we mainly focus on extracting more representative features for discrete architectures based upon the permutation invariance of graph features in this graph regression task. With our proposed predictor, we can efficiently search for effective architectures, saving a large number of computing resources and facilitating NAS searching. Moreover, this work may inspire the general transformer-like model designs for processing graph data in the future. It may take additional resources like GPU and electric energies to train the predictor. However, those resources are pretty less than those spent by previous NAS-related works.

C More ablation studies

Discussion about the positional encodings

TNASP (Lu et al. 2021) also used the Laplacian matrix as the positional encodings and pointed out that a significant performance drop emerges if removing the positional encodings. We verify this conclusion in Tab.6, omitting the positional encodings does lead to a performance drop for both TNASP and our PINAT. However, our PIM has encoded node features using partial permutation invariance augmentation from neighborhoods, **also containing one kind of positional information for the topology, thus PINAT w/o PE is still better than TNASP with PE.**

Method	PE	S_1	S_2	S_3	S_4	S_5
TNASP	×	0.386	0.419	0.456	0.426	0.458
TNASP	✓	0.600	0.669	0.752	0.705	0.820
PINAT	×	0.647	0.702	0.772	0.769	0.832
PINAT	✓	0.679	0.715	0.801	0.772	0.846

Table 6: Ablation study of the positional encodings on NAS-Bench-101.

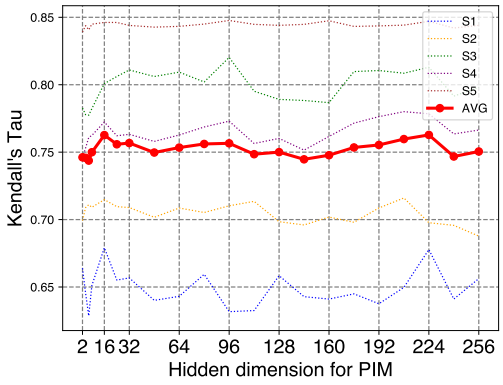


Figure 4: Ablation study for hidden dimensions of PIM. Dashed lines with various colors correspond to different data splits. Red solid line represents the average ranking performance of five data splits with different hidden dimensions.

Different hidden dimensions of PIM

Different hidden dimensions of PIM (serving as the PITE and the PISA module) lead to great changes in model parameters. Thus we decided to investigate the effect of various hidden dimensions of PIM on our PINAT. We conduct the ablation study on NAS-Bench-101 (Ying et al. 2019) with the hidden dimension of PIM ranging from 2 to 256. As shown in Fig. 4, there are three small peaks of performance around 16, 96, and 224, but the average ranking performance changes little. Therefore, we choose hidden dimension 16 of PIM to get a lightweight predictor.

*Corresponding author.

Config	#P(M)	S_1	S_2	S_3	S_4	S_5
(0, 4)	0.723	0.515	0.647	0.737	0.713	0.823
(1, 3)	0.604	0.602	0.686	0.776	0.742	0.837
(2, 2)	0.554	**0.679**	**0.715**	**0.801**	**0.772**	**0.846**
(3, 1)	**0.553**	0.631	0.699	0.790	0.760	0.838
(4, 0)	0.582	0.603	0.697	0.782	0.732	0.842

Table 7: Ablation study for the number of self-attention (SA) heads and the number of permutation invariant self-attention (PISA) heads on NAS-Bench-101. The first item in brackets is the number of SA heads and the second refers to the PISA heads. We also report the number of our predictor model's parameters with different configurations.

Method	S_1	S_2	S_3	S_4	S_5
TNASP	93.19	**93.29**	93.56	93.56	93.36
PINAT	**93.54**	93.14	**93.68**	**93.68**	**93.52**

Table 8: Comparison with TNASP (Lu et al. 2021) about the average performance of searched top-5 architectures on NAS-Bench-101.

Different number of heads for SA and PISA

The encoder block of our proposed predictor totally adopts 4 heads of attention computation, which can be divided into self-attention(SA) heads and permutation invariant self-attention (PISA) heads. We conduct an empirical study to explore the most reasonable configuration. Note that the number of the PISA heads will affect the hidden dimension of the PISA module, making its parameters not linearly increase with the head number. As shown in Tab. 7, when the attention heads are purely SA heads or purely PISA heads, the worst two results emerge. When SA heads and PISA heads share the same number, we get the most lightweight model with the top performance. Therefore, we choose two SA heads and two PISA heads for each encoder layer in our experiments.

Performance of our searched architectures

We list the performance of our searched architectures on NAS-Bench-101 (Ying et al. 2019) and compare with the recent state-of-the-art method TNASP (Lu et al. 2021) in Tab.8. To avoid cherry-picking results, we report the average performance of the searched top-5 architectures ranked by the pre-trained predictor. We can observe that the average performance of top-5 architectures ranked by PINAT surpasses TNASP in most data splits. Even when the training data is extremely scarce i.e. in the data split S_1, the average performance of the searched top-5 architectures from PINAT can still achieve 93.54, outperforming TNASP by 0.35, clearly showing the superiority of our method.

Application of the ranking loss.

Recently, by comparing the ranking of the predicted performance of the encoded architecture, the ranking loss (Ning et al. 2020; Chen et al. 2021) requires fewer training samples for the predictor training to achieve a reasonable ranking ability, thus becoming more and more popular. Note that our PINAT with PIM module is orthogonal to this training technique of using ranking loss. As shown in Tab.9, we combine two forms of ranking loss mentioned in GATES (Ning et al. 2020) with our PINAT, i.e., binary cross entropy loss (BCE) and hinge loss. We can see that our method still achieves higher ranking results than TNASP and GATES, clearly showing the effectiveness of our predictor design.

Method	Loss	S_1	S_2	S_3	S_4	S_5
TNASP	MSE	0.600	0.669	0.752	0.705	0.820
PINAT	MSE	**0.679**	**0.715**	**0.801**	**0.772**	**0.846**
GATES	Hinge	0.605	0.659	0.666	0.691	0.822
TNASP	Hinge	0.655	0.680	0.789	0.740	0.819
PINAT	Hinge	**0.716**	**0.762**	**0.812**	**0.807**	**0.853**
TNASP	BCE	0.676	0.691	0.791	0.747	0.824
PINAT	BCE	**0.717**	**0.764**	**0.820**	**0.812**	**0.854**

Table 9: Ablation of the loss type on NAS-Bench-101. MSE refers to the mean squared error and BCE denotes the binary cross entropy. More details of BCE and Hinge loss can be found in GATES.

D Implementation details

Predictor construction and training

We set the feature dimension as 80 for the embedding features, the positional encodings, and the features from the PITE module. We aggregate them together by simple addition to get the input features to our network backbone, which is comprised of 3 repeated encoder blocks. Each encoder block consists of 4 attention heads altogether, 2 of which are PISA attention heads and the other 2 are self-attention heads. The encoding dimension of query, key, and value in self-attention heads is 64 and we also choose the hidden dimension of 64 for the PISA. Among each encoder block, the hidden dimension of the Feed-Forward Network is set to be 512. We use a regressor behind the backbone structure consisting of a 2-layer fully-connected network with hidden dimension 96.

We adopt the MSE loss to train our predictor with 300 epochs on each dataset. Adam optimizer is used with the initial learning rate 1e-4, weight decay 1e-3, and a cosine decay strategy. For the experiments on NAS-Bench-101 (Ying et al. 2019) and NAS-Bench-201 (Dong and Yang 2020), we adopt the same data splits as TNASP (Lu et al. 2021), detailed in Tab.10. We use 1000 training samples on CIFAR-10, ImageNet, and PPI datasets to train our predictor when conducting the open-domain search. To save the search cost on ImageNet, we use the same data split as PC-DARTS (Xu et al. 2019) to get a subset for the supernet training. Moreover, for the experiments on ModelNet (Wu et al. 2015), we only adopt 500 training samples to train the predictor, which ensures a lower search cost than SGAS (Li et al. 2020).

NAS-Bench-101	S_1	S_2	S_3	S_4	S_5
Train Samples	100	172	424	424	4236
Train Ratio(%)	0.02	0.04	0.1	0.1	1
Test Samples	all	all	100	all	all
NAS-Bench-201	S'_1	S'_2	S'_3	S'_4	S'_5
Train Samples	78	156	469	781	1563
Train Ratio(%)	0.05	1	3	5	10
Test Samples	all	all	all	all	all

Table 10: Data splits from TNASP (Lu et al. 2021) on NAS-Bench-101 and NAS-Bench-201.

Evolutionary search

We apply the widely-used evolutionary algorithm (Deb et al. 2002) to search for effective architectures in a huge search space. Specifically, the whole search process lasts for 100 epochs and we maintain a population of 100 individuals at each epoch. During the first 10 epochs, we randomly generate various architectures to warm-up our population to ensure a reasonable population diversity. In the subsequent epochs, we apply the crossover and mutation operation to generate new architectures. After the whole search, we directly choose the best architecture from the Pareto front.

Computation of the search cost on CIFAR-10

We follow CTNAS (Chen et al. 2021) to get a proxy dataset on DARTS search space. It took us 0.2 GPU days to train a supernet on CIFAR-10 and 0.125 GPU days to collect 1k architecture-accuracy training pairs. Then we spent 7 minutes training predictor on these samples for 300 epochs, and finally, we conducted the evolutionary search on the CPU device. Our predictor only has 0.55M parameters and can be efficiently inferred on the CPU device. Therefore, we report the total GPU search cost as 0.3 GPU days.

Search space

CIFAR-10 We mainly search for repeated cells as proposed in DARTS (Liu et al. 2019). Each cell has 7 nodes, including 2 input nodes from previous layers, 4 intermediate nodes, and 1 output node. Each intermediate node can randomly select 2 inputs from the previous, resulting in 8 edges in a searched cell. Our goal is to select an optimal operation from 7 candidate operations for each edge and choose the inputs for intermediate nodes.

ImageNet We adopt the chain-like search space proposed in ProxylessNAS (Cai, Zhu, and Han 2019). We use 21 searching layers, each of which has 7 candidate choices except the down-sampling layer, specifically skip connection, MobileNetV2 Bottlenecks (Sandler et al. 2018) with convolution kernel sizes $\{3, 5, 7\}$ and expansion ratios $\{3, 6\}$. We aim to search for an optimal operation for each layer.

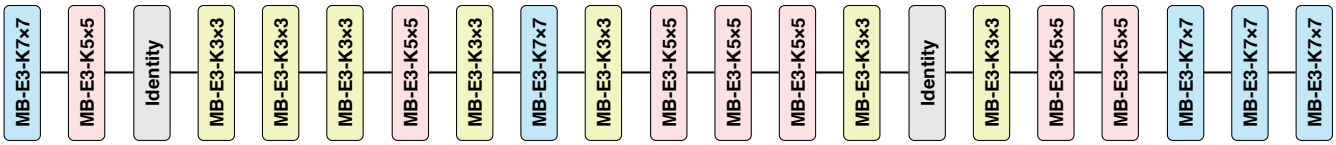
PPI and ModelNet Similar to the DARTS (Liu et al. 2019) search space, SGAS (Li et al. 2020) also proposed a cell-like search space on PPI and ModelNet to search for GCN architectures. Each GCN cell contains 2 input nodes, 3 intermediate nodes, and 1 output node. The main goal is to search for the optimal operation from 10 candidate graph operations for each edge and the best two inputs for the intermediate nodes.

E Details of searched architectures

We summarize the detailed performance of our searched architectures on CIFAR-10 (Krizhevsky et al. 2009), PPI, and ModelNet (Wu et al. 2015) datasets in Tab.11. All the searched architectures of CIFAR-10, ImageNet (Krizhevsky et al. 2017), PPI, and ModelNet are shown in the Fig.6, Fig.5, Fig.7, and Fig.8, respectively.

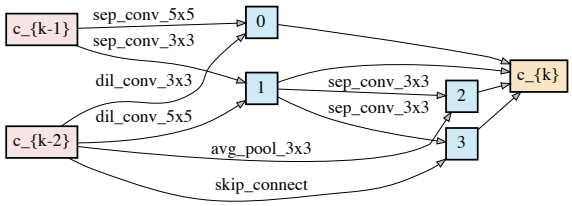
Dataset	**Cell**	**Params(M)**	**Performance(%)**
CIFAR-10	C0	3.47	97.39
	C1	3.80	97.42
	C2	3.65	**97.58**
	avg.	3.64	97.46 $\pm$ 0.08
PPI	P0	24.48	99.475
	P1	23.17	99.471
	P2	23.17	99.472
	P3	21.86	99.480
	P4	24.48	99.473
	P5	24.48	99.471
	P6	24.48	99.481
	P7	21.87	**99.485**
	P8	24.48	99.471
	P9	24.48	99.470
	avg.	23.70	99.47$\pm$0.01
ModelNet	M0	3.76	92.747
	M1	3.91	92.787
	M2	3.90	92.828
	M3	3.95	92.909
	M4	4.00	92.747
	M5	3.95	92.787
	M6	4.10	92.787
	M7	3.95	**93.071**
	M8	3.95	93.031
	M9	3.95	93.031
	avg.	3.94	92.87$\pm$0.12

Table 11: Detailed performance of our searched architectures on CIFAR-10, PPI, and ModelNet datasets.

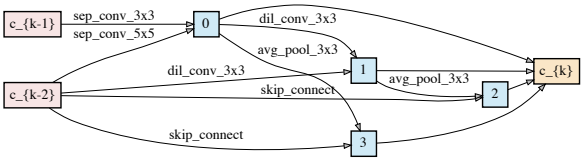


(a) PINAT

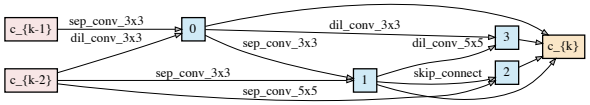
Figure 5: Our searched architecture on the ImageNet dataset.



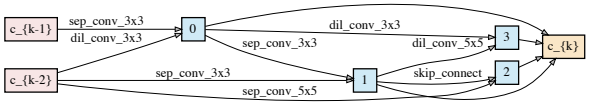
(a) Normal cell of C0.



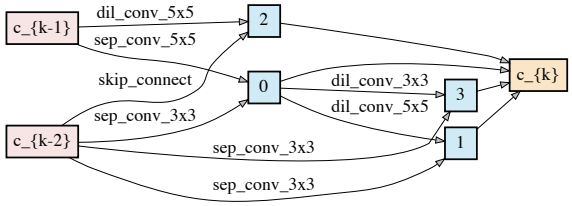
(b) Reduction cell of C0.



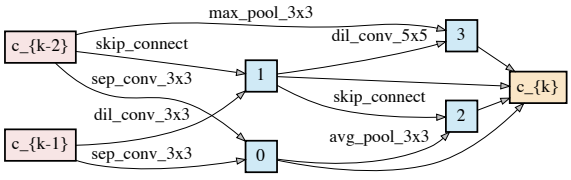
(c) Normal cell of C1.



(d) Reduction cell of C1.

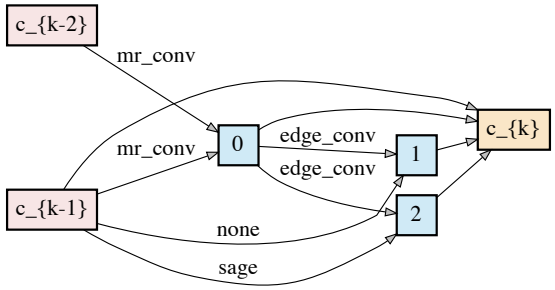


(e) (**Best**) Normal cell of C2 .

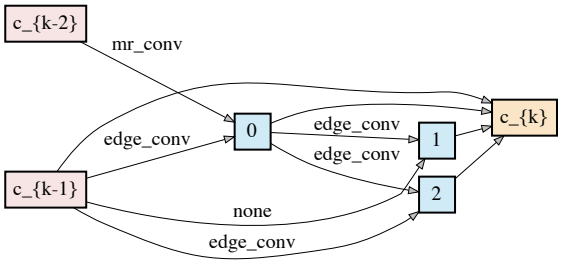


(f) (**Best**) Reduction cell of C2.

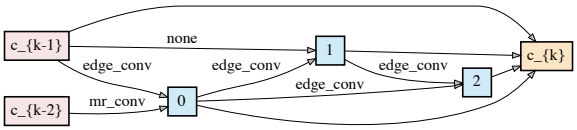
Figure 6: Our searched cells on the CIFAR-10 dataset.



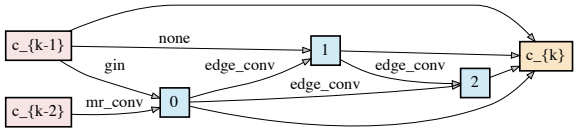
(a) Searched cell of P0.



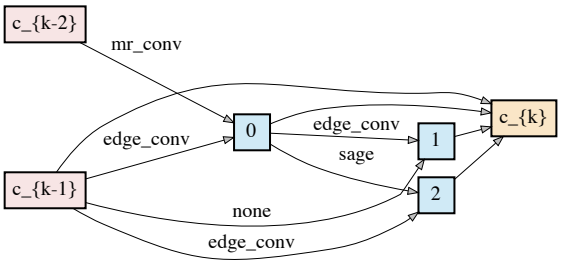
(b) Searched cell of P1.



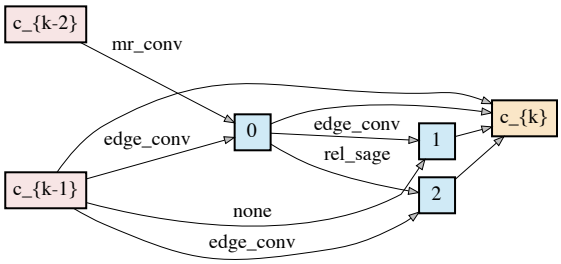
(c) Searched cell of P2.



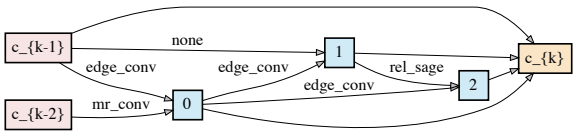
(d) Searched cell of P3.



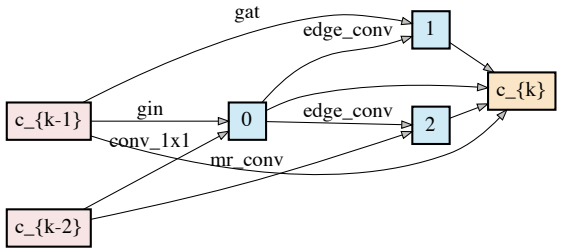
(e) Searched cell of P4.



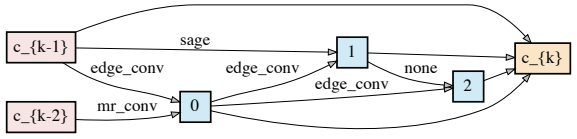
(f) Searched cell of P5.



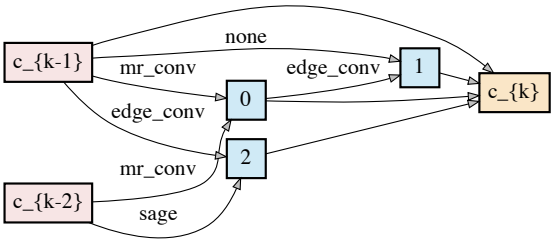
(g) Searched cell of P6.



(h) (**Best**) Searched cell of P7.

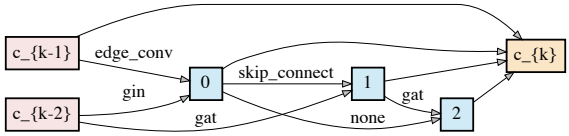


(i) Searched cell of P8.

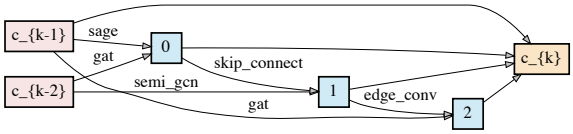


(j) Searched cell of P9.

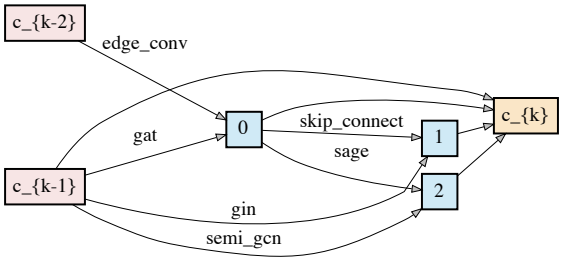
Figure 7: Our searched cells on the PPI dataset.



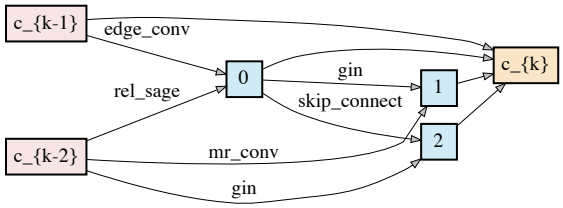
(a) Searched cell of M0.



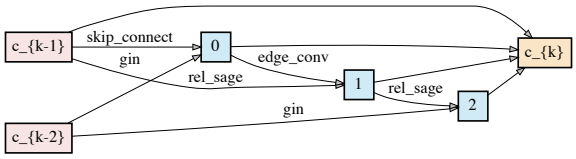
(b) Searched cell of M1.



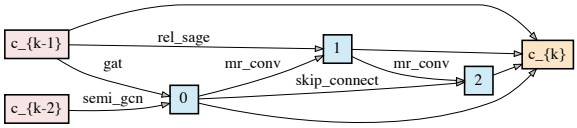
(c) Searched cell of M2.



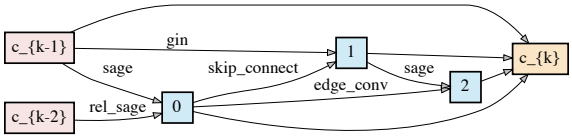
(d) Searched cell of M3.



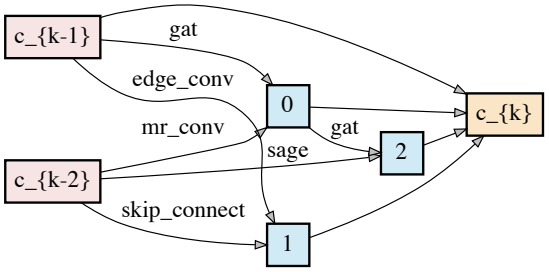
(e) Searched cell of M4.



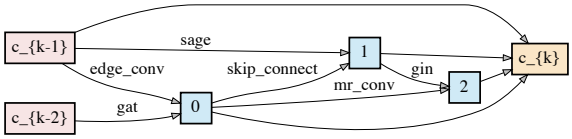
(f) Searched cell of M5.



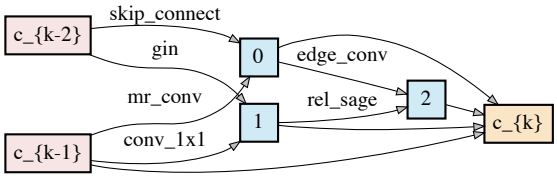
(g) Searched cell of M6.



(h) Searched cell of M7.



(i) (**Best**) Searched cell of M8.



(j) Searched cell of M9.

Figure 8: Our searched cells on the ModelNet dataset.

References

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

Chen, Y.; Guo, Y.; Chen, Q.; Li, M.; Wang, Y.; Zeng, W.; and Tan, M. 2021. Contrastive Neural Architecture Search with Neural Architecture Comparators. In *CVPR*.

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2): 182–197.

Dong, X.; and Yang, Y. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *ICLR*.

Krizhevsky, A.; et al. 2009. Learning multiple layers of features from tiny images.

Krizhevsky, A.; et al. 2017. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6): 84–90.

Li, G.; Qian, G.; Delgadillo, I. C.; Muller, M.; Thabet, A.; and Ghanem, B. 2020. Sgas: Sequential Greedy Architecture Search. In *CVPR*.

Liu, H.; et al. 2019. DARTS: Differentiable Architecture Search. In *ICLR*.

Lu, S.; Li, J.; Tan, J.; Yang, S.; and Liu, J. 2021. TNASP: A Transformer-based NAS Predictor with a Self-evolution Framework. In *NeurIPS*.

Ning, X.; Zheng, Y.; Zhao, T.; Wang, Y.; and Yang, H. 2020. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *ECCV*.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*.

Wu, Z.; Song, S.; Khosla, A.; Yu, F.; Zhang, L.; Tang, X.; and Xiao, J. 2015. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*.

Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Qi, G.-J.; Tian, Q.; and Xiong, H. 2019. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *ICLR*.

Ying, C.; Klein, A.; Christiansen, E.; Real, E.; Murphy, K.; and Hutter, F. 2019. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*.