Loss Functions for Predictor-based Neural Architecture Search

Supplementary Material

1. Descriptions and Implementation Details

In this section, we provide implementation details for loss functions and descriptions for search spaces used in experiments.

1.1. Loss Functions

We first present the formulation and implementation details of eight loss functions evaluated in this paper.

MSE. MSE is a popular regression loss for predictors [11–13, 20, 21], which calculates the squared error between prediction scores and GTs:

$$\mathcal{L}_{MSE} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

We directly use the official implementation from the PyTorch package [16].

HR. HR is a pairwise hinge ranking loss widely used in prior works [6, 14, 15]. We formulate it as:

$$\mathcal{L}_{pair} = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\phi((\hat{y}_i - \hat{y}_j) * sign(y_i - y_j)) \tag{2}$$

where $\phi(z) = max(0, a - z)$ with a margin parameter a. The loss is 0 only when the pair is correctly ranked with a margin larger than a. We use the implementation from the open source of FlowerFormer [6].

LR. LR is a pairwise ranking loss used in ReNAS [26]. LR is similar to HR except that it uses a logistic function $\phi(z) = log(1 + e^{-z})$, alternatively. We use the implementation from the open source of ReNAS.

MSE+SR. MSE+SR is composed of MSE and a pairwise sequential ranking loss. It can be formulated as:

$$\mathcal{L}_{MSE+SR} = \mathcal{L}_{MSE} + \lambda\sum_{i=1}^{n}(y_{\hat{I(i)}} - \hat{y}_i) - (y_{I(i)} - y_i) \tag{3}$$

where $I(\cdot)$ is randomly sampled from 1 to n and λ is a weight coefficient ranging from $(0, 1]$. We use the implementation from the open source of NAR-Former [6].

ListMLE. ListMLE [25] is a listwise ranking loss introduced by DCLP [29]. The predictor first outputs a list of prediction scores $\hat{Y}$. A sorted list $\hat{Y}_g = \{\hat{y_{g_i}}\}_{i=1}^{N}$ is then obtained by sorting $\hat{Y}$ according to GTs of architectures in descending order. For instance, the first item in $\hat{Y}_g$ denotes the prediction score of the architecture with the highest GT. ListMLE maximizes the probability of $\hat{Y}_g$ to make the predicted ranking list close to the actual ranking of architectures, which can be formulated as:

$$\mathcal{L}_{ListMLE} = -\sum_{i=1}^{n}\log\left(\frac{\exp(\hat{y_{g_i}})}{\sum_{j=i}^{n}\exp(\hat{y_{g_j}})}\right) \tag{4}$$

We use the implementation from the open source of DCLP [29].

EW. EW is a weighted regression loss that adopts an exponential level weight for each architecture used in BONAS [29] and CATE [27]. It can be formulated as:

$$\mathcal{L}_{EW} = -\sum_{i=1}^{n}(\exp(y_i) - 1)(\hat{y}_i - y_i)^2 \tag{5}$$

We opt for the implementation version of BONAS [19].

MAPE. MAPE is also a weighted regression loss that employs the percentage between the GT of each architecture and the GT of the global best as the weight. It was introduced by BANANAS [23], which can be formulated as:

$$\mathcal{L}_{MAPE} = -\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y^* - y_i}\right| \tag{6}$$

We use the implementation from the open source of BANANAS [23]. Given that DARTS does not provide the accuracy of the best architecture, we use 99% to approximate the global optima.

WARP. WARP [22] is a weighted ranking loss designed for the multilabel image annotation task. We adapt it to train the predictor for its success in the multi-label image annotation task. WARP uses a stochastic sample strategy to optimize the top-K ranking precision of the predictor. For each architecture in the batch, we randomly sample architectures until we find a violated architecture x_m such that the relative ranking of the architecture pair (x_i, x_m) is incorrectly predicted. It can be formulated as:

$$\mathcal{L}_{WARP} = -\sum_{i=1}^{n}J(i)(\hat{y}_i - \hat{y_m}) \cdot sign(y_i - y_m) \tag{7}$$

where $J(i)$ is a weighting function and M is the number of sampling trials before finding a violated architecture. A small M means that it is easy to find a violated architecture for the architecture x_i, implying that x_i is ranked far away from its actual position. In this case, the predictor should assign a large loss to x_i to place it in the correct position. Hence, we define the weighting function as:

$$J(i) = log\left[\frac{n-1}{M}\right] \cdot (e^{y_i} - 1) \tag{8}$$

where $[\cdot]$ is the flooring function. We also use an exponential level of weight for each architecture. We adapt WARP

into the performance predictor based on the implementation of WSABIE [22].

1.2. Search Spaces

Below, we discuss the search spaces we used. The URLs of search spaces are reported in Table 1.

NAS-Bench-101. NAS-Bench-101 [28] is a cell-based search space composed of over 423k architectures. An operation is represented by a node in the cell. Each architecture cell includes at most seven nodes and nine edges. The search space provides the accuracy of architectures on the CIFAR-10 dataset.

NAS-Bench-201. NAS-Bench-201 [3] is also a cell-based search space which contains 15625 architectures. The operation type is represented by the edge on NAS-Bench-201. There exist four nodes and six edges in the architecture cell. NAS-Bench-201 provides the accuracy of architectures on the CIFAR-10, CIFAR-100, and ImageNet-16-120 datasets.

TransNAS-Bench-101. TransNAS-Bench-101 [4] is composed of a **Micro** (cell-based) search space containing 4096 architectures and a **Macro** (skeleton-based) search space containing 3256 architectures. The constitution of the architecture cell in the micro search space is the same as NAS-Bench-201. As for the macro space, we use the same encoding in Arch-Graph [5]. The search space provides architecture performance across seven tasks. In our experiments, we test each loss function on four tasks: Class-Object, Class-Scene, Jigsaw, and Autoencoder.

NAS-Bench-Graph. NAS-Bench-Graph [17] contains 26206 architectures and a cell-based search space. There are six nodes and at most eight edges in each cell. NAS-Bench-Graph provides the accuracy of architectures on nine node classification datasets. The Cora dataset is used in our experiments.

DARTS. DARTS [10] is an open-domain cell-based search space that does not provide architecture performance. It is a much larger search space with around 10^{18} architectures. Each architecture consists of a normal cell and a reduction cell, each containing seven nodes and eight edges.

2. Details of PWLNAS

PWLNAS is based on the prevalent predictor-guided evolution framework [7] for NAS. This framework first randomly samples a subset of architectures from the search space to compose the initial population and dataset. Then, a set of candidate architectures is generated by mutating high-accuracy architectures in the current population in an iterative way. These candidate architectures are estimated by a predictor (trained on the entire dataset), and those with top predicted scores are added to the population and dataset. The architecture with the highest GT within the dataset is selected as the searching result. However, this framework utilizes a fixed loss function during the iterative training of the predictor. Instead, we opt for a piecewise loss function that combines effective ones to enhance the predictor-guided evolution framework. We use a warm-up loss function (ranking/regression loss) in the early iterations and change it to a weighted loss function in the following iterations. The choice of loss function and the number of warm-up samples depend on specific tasks. The pipeline of PWLNAS is demonstrated in Algorithm 1.

On all tasks, we set the initial size n_0 to 20, the number of offspring n_o to 10, and the number of mutated architectures n_{mu} to 5. Other hyperparameters of PWLNAS on different tasks are presented in Table 9. Allowing for DARTS does not provide the accuracy of the best architecture, we use 100% as the global optimal GT when training the predictor with MAPE loss.

Algorithm 1 PWLNAS

Input: Search space S, training dataset D_t, population D_p, initial size n_0, query budget n_{max}, number of candidate architectures n_c, number of offspring n_o, number of individuals to mutate n_{mu}, number of warm-up samples n_w, loss function used in warm-up stage L_w, main loss function L_m, performance predictor P.

Initialize D_t and D_p with n_0 architectures randomly sampled in S and their GT performance;
for n from n_0 to n_{max} **do**
  if $n < n_w$ **then**
    Train P with D_t using the warm-up loss L_w;
  else
    Train P with D_t using the main loss L_m;
  end if
  Choose n_{mu} architectures based on their validation accuracy via tournament;
  Mutate the selected architectures and generate n_c candidate architectures;
  Utilize P to predict the performance of candidate architectures;
  Generate offsprings M_o that consist of candidate architectures with top-n_o highest prediction scores.
  Append M_o to D_t and D_p;
  Remove the n_o oldest architectures from D_p;
  $n = n + n_o$
end for
return Architecture x^* with the highest GT in D_t.

Table 1. Basic information of search spaces used in our experiments.

Search Space	Size	Is Cell-based?	Task	URL
NAS-Bench-101	423k	✓	Image Classification	https://github.com/google-research/nasbench
NAS-Bench-201	15625	✓	Image Classification	https://github.com/D-X-Y/NAS-Bench-201
TransNAS-Bench-101 Micro	4096	✓	Class Scene, Class Object, Jigsaw, Autoencoder	https://github.com/yawen-d/TransNASBench
TransNAS-Bench-101 Macro	3256	×		
NAS-Bench-Graph	26206	✓	Graph Node Classification	https://github.com/THUMNLab/NAS-Bench-Graph
DARTS	10^{18}	✓	Image Classification	https://github.com/quark0/darts

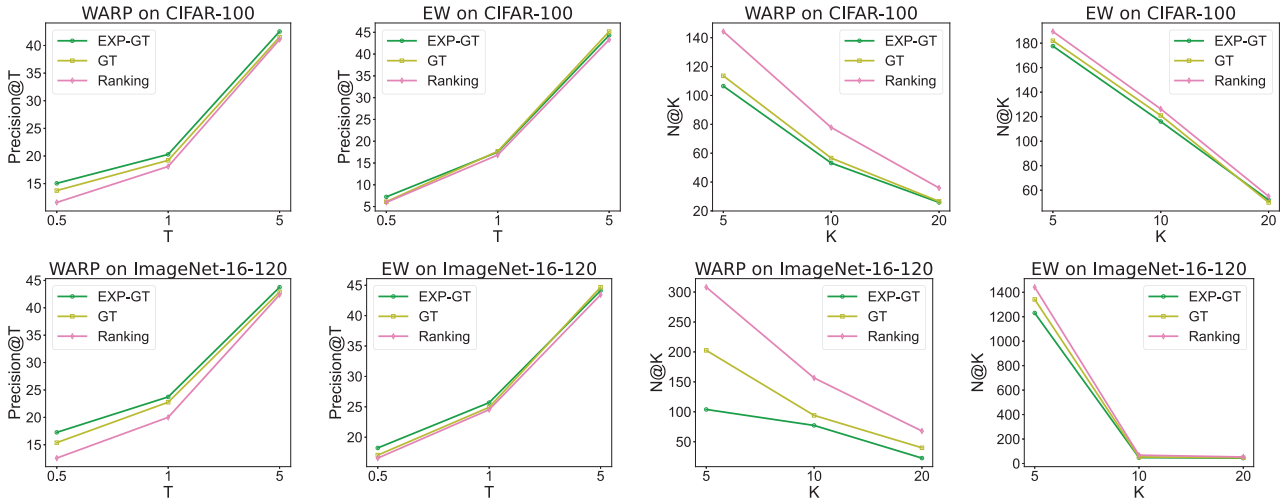


Figure 1. Precision@T and N@K of different weighting types for WARP and EW on NAS-Bench-201 CIFAR-100 and ImageNet-16-120 with 1% training data. Note that a higher Precision@T and a lower N@K are preferred.

3. Additional Experimental Results

3.1. Full Evaluation of performance Predictor

In this part, we provide the full results of different loss functions with more assessment metrics. All the results are averaged over 30 runs. Additionally, we use the common Normalized Discounted Cumulative Gain@K (NDCG@K) metric in the field of information retrieval. It measures the quality of the top-K predicted architectures. Assuming the GT of the ith best architecture in the search space is y_{g_i}. We use $X_K = \{x_i | \hat{r}_i <= K\}$ to represent the architectures with top-K highest prediction scores. NDCG@K can be calculated as:

$$\begin{aligned} NDCG@K &= \frac{DCG@K}{IDCG@K}, \\ DCG@K &= \sum_{x_i \in X_K} \frac{y_i}{\log_2(i+1)}, \\ IDCG@K &= \sum_{i=1}^{K} \frac{y_{g_i}}{\log_2(i+1)}. \end{aligned} \tag{9}$$

where Discounted Cumulative Gain@K (DCG@K) computes the scores of the top-K predicted architectures and gives larger weights to architectures with higher rankings. Ideal Discounted Cumulative Gain@K (IDCG@K) is the highest value for NDCG@K. NDCG@K is scaled up by a factor of 100 in our results.

Different search spaces and training portions. We plot the results on Precision@0.5, N@10, NDCG@10, and Kendall's Tau on 13 tasks in Figure 3, Figure 4, and Figure 5. The trend is largely similar across all search spaces. In the region of small training portion, ranking loss functions like HR and ListMLE roughly perform well in top-ranking metrics. When the training portion increases, weighted loss functions tend to identify well-performing architectures better. The optimal loss function varies according to specific search spaces and tasks. For instance, with enough training data, MAPE, EW, and WARP perform best on NAS-Bench-201, TransNAS-Bench-101 Micro Jigsaw, and TransNAS-Bench-101 Macro, respectively. Among these weighted loss functions, WARP is the most stable on all tasks and is recommended as a reliable choice for unseen tasks with sufficient training data. As for the overall ranking performance, ranking loss functions generally take the lead.

Different weighting types. We further investigate the impact of weighting types of WARP and EW on another two datasets on NAS-Bench-201. Figure 1 demonstrates the ad-

Table 2. Results of the MLP-based AP with different loss functions on NAS-Bench-101 with 0.1% training data and NAS-Bench-201 with 1% training data. Precision@0.5 is short for Ptop@0.5. **Bold** indicates the best.

Tasks	NB101 CIFAR-10			NB201 CIFAR-10			NB201 CIFAR-100			NB201 ImageNet-16-120		
Metrics	Ptop@0.5$^{\uparrow}$	N@10$^{\downarrow}$	$\tau^{\uparrow}$	Ptop@0.5$^{\uparrow}$	N@10$^{\downarrow}$	$\tau^{\uparrow}$	Ptop@0.5$^{\uparrow}$	N@10$^{\downarrow}$	$\tau^{\uparrow}$	Ptop@0.5$^{\uparrow}$	N@10$^{\downarrow}$	$\tau^{\uparrow}$
MSE	2.46	11832.40	0.18	4.41	250.94	0.43	5.77	163.80	0.54	7.69	75.10	0.57
HR	**19.58**	1172.60	**0.65**	22.15	23.58	0.65	15.90	76.60	0.66	19.74	**32.40**	0.64
ListMLE	16.85	**325.80**	**0.65**	**24.15**	**22.74**	**0.66**	**16.79**	**53.20**	**0.67**	**22.05**	34.10	**0.65**
WARP	11.34	470.20	0.57	9.36	113.20	0.43	7.56	98.70	0.49	7.95	79.60	0.45

Table 3. Results of the Transformer-based PINAT with different loss functions on NAS-Bench-101 with 0.1% training data and NAS-Bench-201 with 1% training data. Precision@0.5 is short for Ptop@0.5. **Bold** indicates the best.

Tasks	NB101 CIFAR-10			NB201 CIFAR-10			NB201 CIFAR-100			NB201 ImageNet-16-120		
Metrics	Ptop@0.5$^{\uparrow}$	N@10$^{\downarrow}$	$\tau^{\uparrow}$	Ptop@0.5$^{\uparrow}$	N@10$^{\downarrow}$	$\tau^{\uparrow}$	Ptop@0.5$^{\uparrow}$	N@10$^{\downarrow}$	$\tau^{\uparrow}$	Ptop@0.5$^{\uparrow}$	N@10$^{\downarrow}$	$\tau^{\uparrow}$
MSE	18.52	1956.80	0.75	8.62	146.60	0.62	14.96	67.17	0.66	17.18	64.00	0.67
HR	23.12	635.14	0.76	29.32	8.44	0.67	19.23	33.33	0.68	25.32	16.75	0.67
ListMLE	25.73	236.70	**0.78**	23.56	21.78	**0.68**	24.87	28.40	**0.69**	27.35	13.20	**0.68**
WARP	**29.98**	**140.13**	0.71	**38.71**	**3.78**	0.65	**27.78**	**18.33**	0.67	**31.54**	**12.67**	0.66

Table 4. Searching results on TransNAS-Bench-101 Macro with a query budget of 50. **Bold** indicates the best.

Tasks		Cls.O.	Cls.S.	Auto.	Jigsaw
Predictor	Loss	Acc.$^{\uparrow}$	Acc.$^{\uparrow}$	SSIM$^{\uparrow}$	Acc.$^{\uparrow}$
Arch-Graph [5]	BCE	47.35	56.77	71.32	96.70
WeakNAS [24]	MSE	47.40	56.88	72.54	96.86
PWLNAS (ours)	MSE	47.11	56.80	73.72	96.83
	HR	47.11	56.92	73.69	96.84
	ListMLE	47.09	56.94	73.84	96.84
	WARP	47.27	56.79	73.87	96.82
	PW	**47.46**	**57.11**	**74.02**	**96.90**
Global Best		47.96	57.48	76.88	97.02

vantage of 'EXP-GT' and 'GT' over 'Ranking' for both loss functions in Precision@T and N@K. This implies that the top-ranking ability of weighted loss functions is sensitive to the choice of weight, and GT-based weights are more effective than ranking-based weights.

Different predictors. We test different categories of loss functions using two representative predictors: the simple MLP-based AP [2] and the advanced Transformer-based PINAT [13] on NAS-Bench-101 and NAS-Bench-201. As shown in Table 2 and Table 3, we observe that ListMLE yields the best results with AP, and WARP beats other loss functions with PINAT. These results reveal that ranking loss functions are suitable for simple predictors, while weighted loss functions fit better with advanced predictors.

3.2. Searching on TransNAS-Bench-101 Macro

We use a PW loss composed of HR and WARP on TransNAS-Bench-101 Macro. Table 4 shows the superior performance of PW loss on all tasks compared with the single loss and prior works, which is consistent with the results on the micro space. This suggests that combining specific loss functions can empower the predictor-based NAS.

3.3. Searching on NAS-Bench-NLP

We validate our PWLNAS on an additional benchmark: NAS-Bench-NLP [9], with a PW loss composed of HR and WARP. As shown in Table 5, PWLNAS achieves the lowest log PPL complexity among all competitors, indicating its strong generalizability in the challenging NLP tasks.

Table 5. Searching results on NAS-Bench-NLP with a query budget of 100. **Bold** indicates the best.

Method	**PWLNAS** (ours)	**BANANAS** [23]	**NASBOT** [8]	**NPENAS** [20]
Log PPL↓	**4.575**	4.595	4.579	4.587

3.4. Searching with Different Strategies

We compare our PWLNAS using random search [1] and evolutionary search [18] strategy on NAS-Bench-101. Table 6 demonstrates that PWLNAS improves both baselines by a large margin, which demonstrates its broad applicability with different search strategies.

Table 6. Search results on NAS-Bench-101 using different search strategies with a query budget of 150.

Strategy	**Random Search**	**Evolutionary Search**
Test error (%) w/o PWLNAS	6.43	6.36
Test error (%) w/ PWLNAS	**6.06** (↓ 0.37%)	**5.80** (↓ 0.56%)

3.5. Computational Cost

We calculate the computational cost of each loss function on NAS-Bench-201 with 1% training data in Table 7 using a single RTX 3090 GPU. Among them, WARP is the most time-consuming option because of its iterative sampling in mini-batch to construct data pairs. Meanwhile, ranking loss functions take more time than MSE due to their higher com-

putational complexity in processing pairwise or listwise relationships.

Table 7. Training time on NAS-Bench-201 with a training portion of 1%.

Loss	MSE	LR	HR	MSE+SR	ListMLE	EW	MAPE	WARP
Time (Sec)	79.26	89.81	90.21	84.45	81.86	75.66	76.86	380.92

3.6. Visualization Results on DARTS

We visualize the architecture cells searched by different loss functions in Figure 2.

4. Hyperparameter Tuning

We present the details of hyperparameter tuning for loss functions in Table 8. The baseline predictor is based on GCN. We use different hyperparameters for different loss functions for two reasons. First, each loss function is sensitive to the choice of hyperparameter. For example, listwise ranking loss prefers a higher learning rate, and pairwise ones require a lower learning rate. Hence, it is fairer to compare these loss functions with different levels of hyperparameter tuning. Besides, it is common practice for predictor-based NAS to modify the hyperparameter for better search results because the major computational cost lies in training the architectures for GT performance instead of training the predictor. As a result, it makes sense to tune the hyperparameters for the predictor-based NAS in practice.

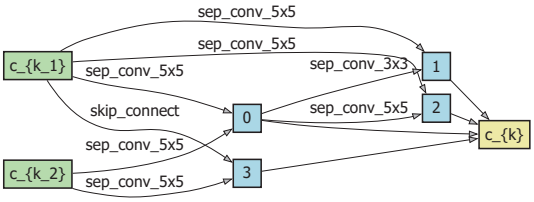
Then, we give the hyperparameters of three performance predictors in our experiments. The GCN-based predictor is composed of a four-layer GCN encoder and a three-layer MLP regressor. As for the MLP-based predictor, we use the implementation from AP [2] without changing its structure. Similarly, we directly employ the PINAT [13] as the Transformer-based predictor. The batch size has a range of [16, 32, 64, 128], and we choose the maximum in this range that does not exceed the number of training data. For instance, if we train the predictor with 100 architectures (0.02% portion), the batch size is set to 64. All experiments are conducted on a single RTX 3090 GPU.

Table 8. Hyperparameters of loss functions for architecture performance prediction.

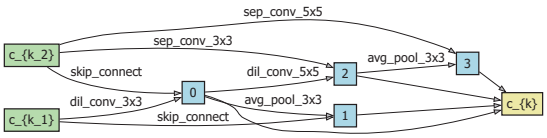
Loss	Hyperparameter	NB101	NB201	TB101 Micro	TB101 Macro	NB-Graph
MSE	Learning rate	0.0005	0.0001	0.0001	0.0001	0.0001
	Weight decay	0.001	0.0005	0.0005	0.0005	0.0005
LR	Learning rate	0.0005	0.0001	0.0001	0.0001	0.0001
	Weight decay	0.001	0.0005	0.0005	0.0005	0.0005
HR	Learning rate	0.0005	0.0001	0.0001	0.0001	0.0001
	Weight decay	0.001	0.0005	0.0005	0.0005	0.0005
	Margin a	0.5	0.5	0.5	0.5	0.5
MSE+SR	Learning rate	0.0005	0.0001	0.0001	0.0001	0.0001
	Weight decay	0.001	0.0005	0.0005	0.0005	0.0005
	Auxiliary coefficient λ	0.5	0.6	0.6	0.6	0.5
ListMLE	Learning rate	0.001	0.0005	0.0005	0.0005	0.0005
	Weight decay	0.001	0.001	0.001	0.001	0.001
EW	Learning rate	0.0005	0.0001	0.0001	0.0001	0.0001
	Weight decay	0.001	0.0005	0.0005	0.0005	0.0005
MAPE	Learning rate	0.0005	0.0001	0.0001	0.0001	0.0001
	Weight decay	0.001	0.0005	0.0005	0.0005	0.0005
WARP	Learning rate	0.001	0.005	0.0005	0.0005	0.0005
	Weight decay	0.001	0.001	0.001	0.001	0.001

Table 9. Hyperparameter of PWLNAS on different tasks.

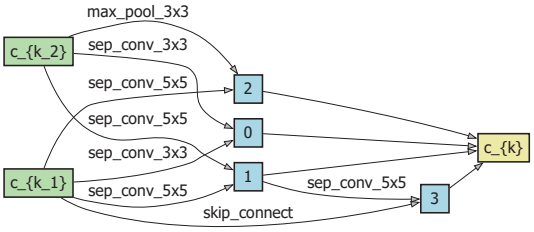
Search Space	Task	Loss	n_w	n_{max}	n_c
NB101	CIFAR-10	HR (warm-up), WARP (main)	100	150	200
NB201	CIFAR-10	HR (warm-up), MAPE (main)	40	100	200
	CIFAR-100	HR (warm-up), MAPE (main)	40	100	200
	ImageNet16-120	HR (warm-up), MAPE (main)	40	100	200
TB101 Micro	Class Scene	HR (warm-up), WARP (main)	30	50	100
	Class Object	HR (warm-up), WARP (main)	30	50	100
	Jigsaw	MSE (warm-up), EW (main)	30	50	100
	Autoencoder	HR (warm-up), WARP (main)	30	50	100
TB101 Macro	Class Scene	HR (warm-up), WARP (main)	30	50	100
	Class Object	HR (warm-up), WARP (main)	30	50	100
	Jigsaw	HR (warm-up), WARP (main)	30	50	100
	Autoencoder	HR (warm-up), WARP (main)	30	50	100
DARTS	CIFAR-10	HR (warm-up), MAPE (main)	40	100	200



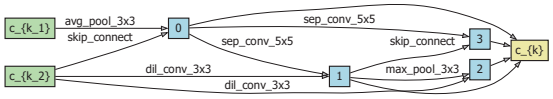
(a) Normal cell found by MSE loss.



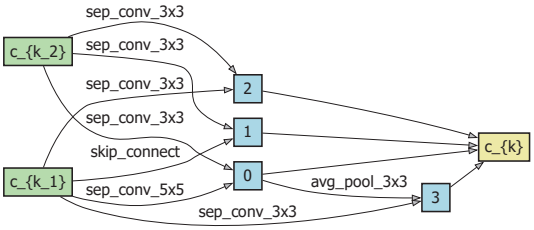
(b) Reduction cell found by MSE loss.



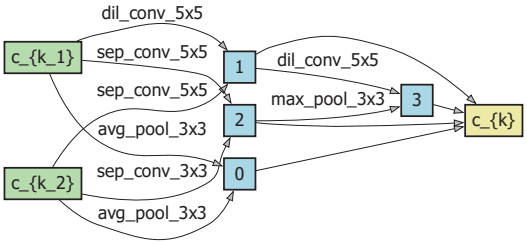
(c) Normal cell found by HR loss.



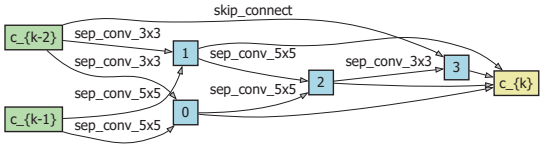
(d) Reduction cell found by HR loss.



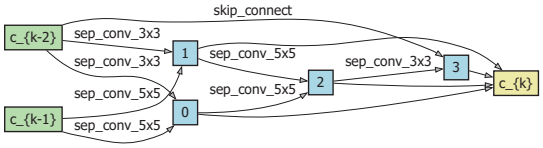
(e) Normal cell found by ListMLE loss.



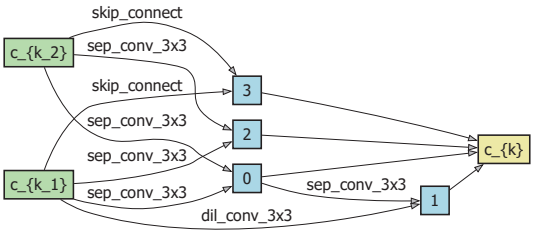
(f) Reduction cell found by ListMLE loss.



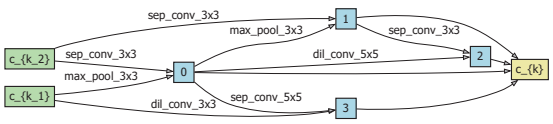
(g) Normal cell found by MAPE loss.



(h) Reduction cell found by MAPE loss.



(i) Normal cell found by PW loss.



(j) Reduction cell found by PW loss.

Figure 2. Architectures found by different loss functions on the CIFAR-10 dataset.

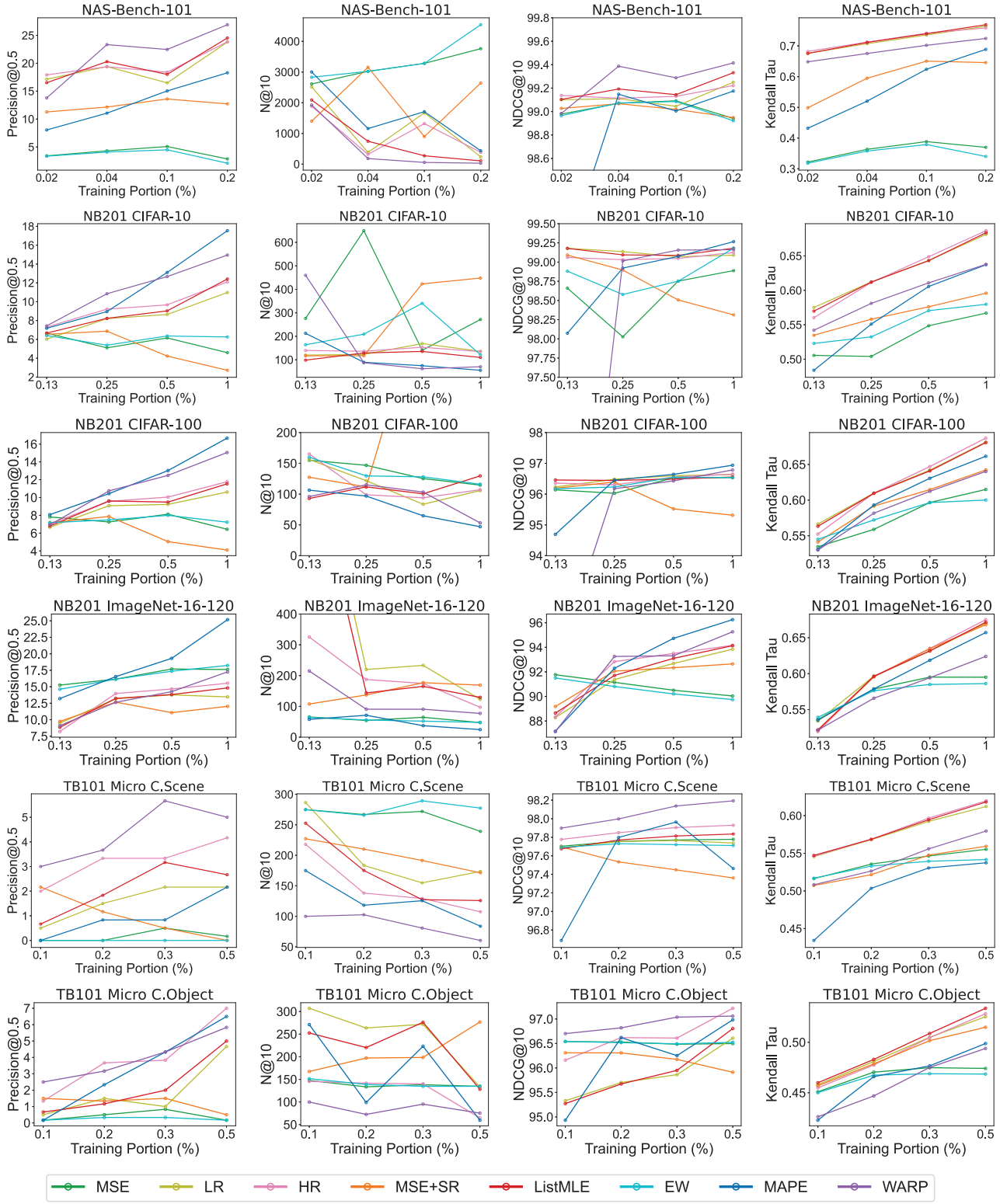


Figure 3. Precision@0.5, N@10, NDCG@10 and Kendall's Tau (*Left to Right*) of different loss functions on NAS-Bench 101, NAS-Bench-201, and TransNAS-Bench-101 Micro. Note that a higher metric indicates better performance except for N@10.

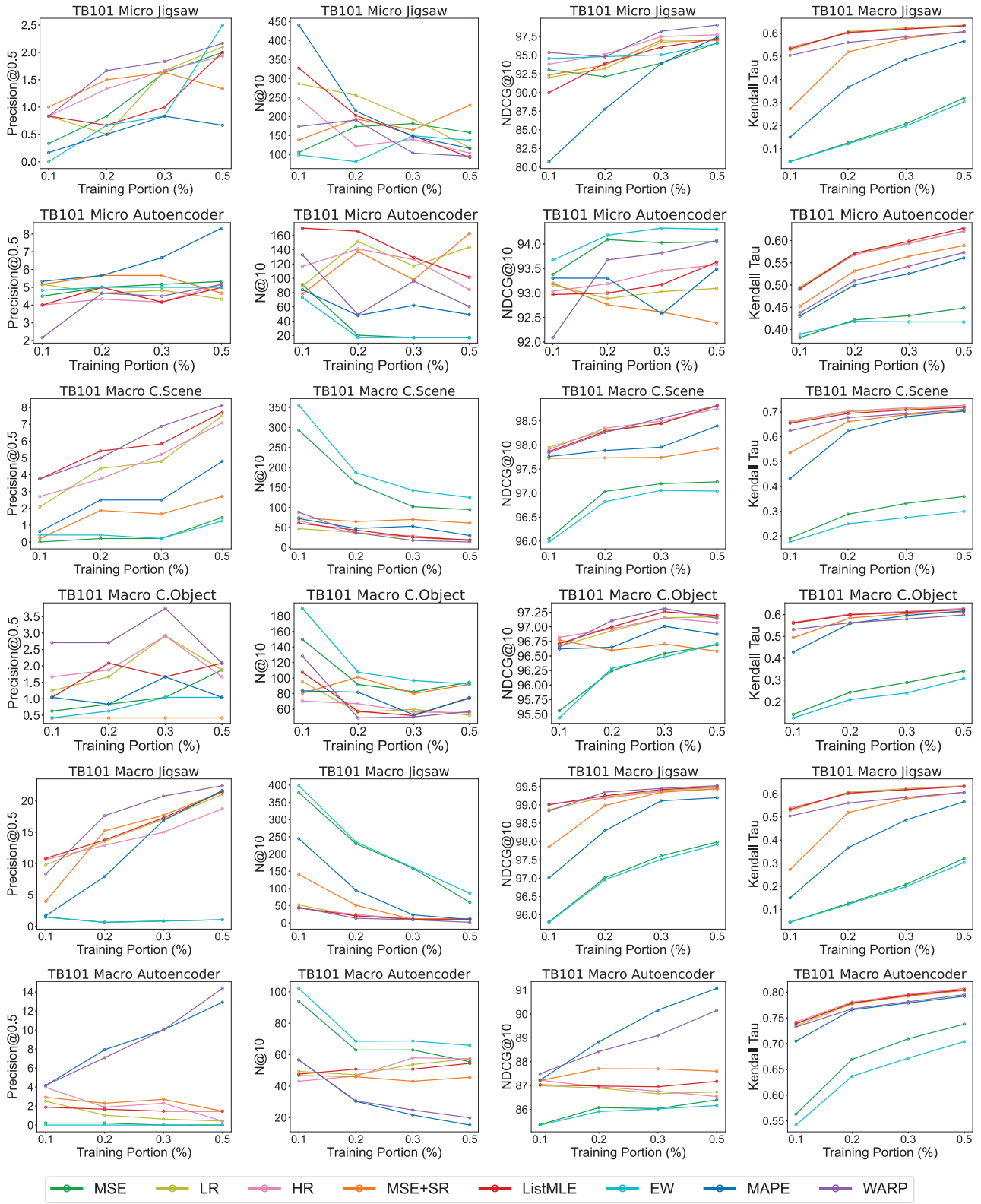


Figure 4. Precision@0.5, N@10, NDCG@10 and Kendall's Tau (*Left to Right*) of different loss functions on TransNAS-Bench-101 Micro/Macro. Note that a higher metric indicates better performance except for N@10.

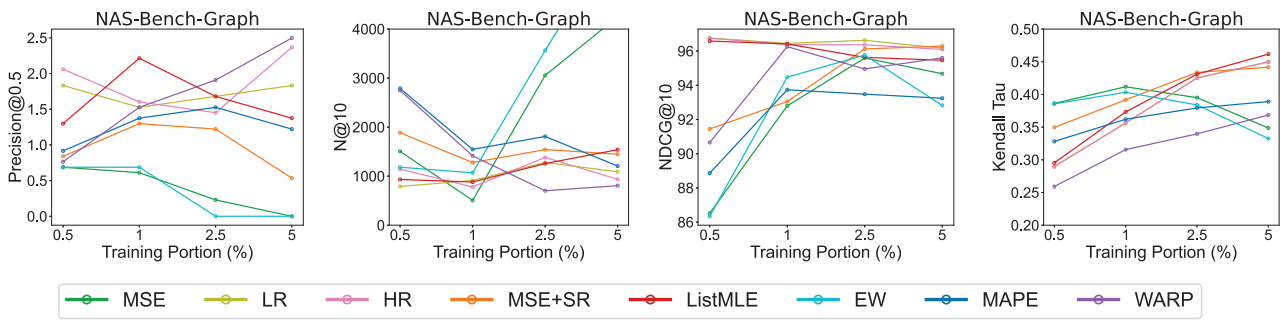


Figure 5. Precision@0.5, N@10, NDCG@10 and Kendall's Tau (*Left to Right*) of different loss functions on NAS-Bench-Graph. Note that a higher metric indicates better performance except for N@10.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012. 4

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 4, 5

[3] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *Proc. of ICLR*, 2019. 2

[4] Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5251–5260, 2021. 2

[5] Minbin Huang, Zhijian Huang, Changlin Li, Xin Chen, Hang Xu, Zhenguo Li, and Xiaodan Liang. Arch-graph: Acyclic architecture relation predictor for task-transferable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11881–11891, 2022. 2, 4

[6] Dongyeong Hwang, Hyunju Kim, Sunwoo Kim, and Kijung Shin. Flowerformer: Empowering neural architecture encoding using a flow-aware graph transformer. In *Proc. of CVPR*, pages 6128–6137, 2024. 1

[7] Kun Jing, Jungang Xu, and Pengfei Li. Graph masked autoencoder enhanced predictor for neural architecture search. In *Proc. of IJCAI*, 2022. 2

[8] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31, 2018. 4

[9] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov, and Evgeny Burnaev. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747, 2022. 4

[10] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *Proc. of ICLR*, 2018. 2

[11] Yuqiao Liu, Yehui Tang, and Yanan Sun. Homogeneous architecture augmentation for neural predictor. In *Proc. of ICCV*, 2021. 1

[12] Shun Lu, Jixiang Li, Jianchao Tan, Sen Yang, and Ji Liu. Tnasp: A transformer-based nas predictor with a self-evolution framework. *Advances in Neural Information Processing Systems*, 34:15125–15137, 2021.

[13] Shun Lu, Yu Hu, Peihao Wang, Yan Han, Jianchao Tan, Jixiang Li, Sen Yang, and Ji Liu. Pinat: A permutation invariance augmented transformer for nas predictor. In *Proc. of AAAI*, 2023. 1, 4, 5

[14] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *Proc. of ECCV*, 2020. 1

[15] Xuefei Ning, Zixuan Zhou, Junbo Zhao, Tianchen Zhao, Yiping Deng, Changcheng Tang, Shuang Liang, Huazhong Yang, and Yu Wang. Ta-gates: An encoding scheme for neural network architectures. *Advances in Neural Information Processing Systems*, 35:32325–32339, 2022. 1

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 1

[17] Yijian Qin, Ziwei Zhang, Xin Wang, Zeyang Zhang, and Wenwu Zhu. Nas-bench-graph: Benchmarking graph neural architecture search. *Advances in neural information processing systems*, 35:54–69, 2022. 2

[18] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, pages 4780–4789, 2019. 4

[19] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. *Advances in Neural Information Processing Systems*, 33:1808–1819, 2020. 1

[20] Chen Wei, Chuang Niu, Yiping Tang, Yue Wang, Haihong Hu, and Jimin Liang. Npenas: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8441–8455, 2022. 1, 4

[21] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *Proc. of ECCV*, 2020. 1

[22] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Twenty-Second International Joint Conference on Artificial Intelligence*. Citeseer, 2011. 1, 2

[23] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, pages 10293–10301, 2021. 1, 4

[24] 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. *Advances in Neural Information Processing Systems*, 34: 28904–28918, 2021. 4

[25] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *Proc. of ICML*, 2008. 1

[26] Yixing Xu, Yunhe Wang, Kai Han, Yehui Tang, Shangling Jui, Chunjing Xu, and Chang Xu. Renas: Relativistic evaluation of neural architecture search. In *Proc. of CVPR*, 2021. 1

[27] Shen Yan, Kaiqiang Song, Fei Liu, and Mi Zhang. Cate: Computation-aware neural architecture encoding with transformers. In *International Conference on Machine Learning*, pages 11670–11681. PMLR, 2021. 1

[28] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards

reproducible neural architecture search. In *Proc. of ICML*, 2019. 2

[29] Shenghe Zheng, Hongzhi Wang, and Tianyu Mu. Dclp: Neural architecture predictor with curriculum contrastive learning. In *Proc. of AAAI*, 2024. 1