Rank-based Training-Free NAS Algorithm

Hsieh Cheng-Han
emiliaistruelove@gmail.com
Department of Computer Science and Engineering,
National Sun Yat-sen University
Kaohsiung, Taiwan

Chun-Wei Tsai
cwtsai@mail.cse.nsysu.edu.tw
Department of Computer Science and Engineering,
National Sun Yat-sen University
Kaohsiung, Taiwan

ABSTRACT

Although the training-free neural architecture search (NAS) are typically faster than training-based NAS methods, however, the correlation between the measurement and the final accuracy of an architecture is not well enough in most cases. To address this problem, we propose a rank-based training-free NAS algorithm, which combines three complement training-free score functions to evaluate an architecture by ranking. More precisely, noise immunity (NI) and the correlation between binary activation paterns, named as NASWOT, are used as measurement of image recognition ability, and, SynFlow are used as measurement of trainability. With that, a modified version of simulated annealing (SA) algorithm can applies on and obtains a better performance on searching high accuracy architectures. To evaluate the performance of the proposed algorithm, this paper compared it with several NAS algorithms, including weight-sharing methods, non-weight-sharing methods, and several state-of-the-art training-free score functions. The final result shows that in relatively larger search space, like NATS-Bench SSS, the proposed algorithm outperforms most of NAS methods.

ACM Reference Format:
Hsieh Cheng-Han and Chun-Wei Tsai. 2023. Rank-based Training-Free NAS Algorithm. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Neural architecture search (NAS) has recently drawn a big amount of attention, since the ability to automatically design a "good" neural architecture. By leveraging machine learning algorithms [1], NAS algorithms can explore a search space, which is comprised of numerous potential architectures, to find out a good architectures that outperform those designed by human experts. Recently, the use of NAS is widespread, from object detection [2], image recognition [3] and speech recognition [4] [5] to natural language processing. [6] [7] [8]

Despite the promising results of NAS, there are still many challenges to conquer. One major problem is the extremely high computational cost to search for an optimal architecture, which can make NAS impractical for real-world applications, particularly on resource-constrained platforms like embedded system. The reason why NAS is costly is that during the searching, a candidate architecture must be trained to evaluate how good of this architecture.

Table 1: The Kendall correlation between training-free score and test accuracy, evaluated on the three datasets of NATS-Bench [17]. Each metric has been computed three times with different initialisations and the average is taken as final score.

Training-free score function	CIFAR-10	CIFAR-100	ImageNet-16-120
NTK	-0.33	-0.30	-0.39
Snip	0.45	0.47	0.41
Fisher	0.39	0.40	0.36
Grasp	0.28	0.35	0.35
NASWOT	0.61	0.62	0.60
SynFlow	0.57	0.56	0.56
LogSynFlow	0.61	0.60	0.59
Rank (ours)	X	X	X

To overcome this challenge, recent works developed and proposed lots of method which is so called training-free NAS. For example, Mellor et al. [3] proposed the measurement of the correlation between the binary activation paterns, induced by the untrained network at two inputs, named as neural architecture search without training (NASWOT). On the other hand, Lee et al. [9] proposed purning parameters based on a saliency matric, which then extended further by Wang et al. [10] and Tanaka et al. [11]. In [11], Tanaka et al. proposed Iterative Synaptic Flow Pruning (SynFlow) algorithm which intends to deal with the layer collapse problem when purning a network. The score function used in the algorithm is so-called *synaptic saliency* score. Later, Abdelfattah et al. [12] extended SynFlow to score a network architecture by summing synaptic saliency score over all parameters in the model. Cavagnero et al. [13] found that SynFlow is likely to suffer from gradient explosion, then proposed the LogSynFlow score function which prevents the problem. For another example, Chen et al. [14] proposed to compute the condition number of neural tangent kernel (NTK) [15] [16], which is used as the score to estimate the trainability of an architecture.

However, Table 1 shows most of score functions suffer from low correlation between score values and the final accuracy of architectures, leading to a predicament that no matter how good the search method is used, we can hardly find an optimal architecture. The major problem causes the low correlation is that a single score function can only evaluate one perspect/characteristic of an architecture. To address the problem, we propose cooperating three complement score functions by ranking, which allows every score functions evaluate an architecture without losing fairness. The first score function

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

is noise immunity (NI) [18], as an indicator of the ability to distinguish two images. The second one is NASWOT, as score to estimate the ability of expression of an architecture and also a complement of NI. The last one is SNIP used as the measurement of trainability and also as a complement to NI and NASWOT. With these three complement score functions, an evaluation of an architecture is not simply from a single aspect but three different aspects, which is like estimate the weight of an object by not only the length but its height and width.

The remainder of this paper is organized as follows: Section 2 provides the detail about the three complement score functions. Section 3 gives a detailed description about the proposed method. Section 4 begins with parameters setting and provide the simulation results following is the experiment results in different search space. The conclusion and further prospect are gived in Section 5.

2 RELATED WORKS

2.1 Training-free Score Functions

In [3], Mellor et al. proposed a score function without the requirement for training which is named neural architecture search without training (NASWOT). Figure 1 gives a simple example to illustrate the procedure of NASWOT score function. Consider a mini-batch of data $X = \{x_i\}_{i=1}^N$ passing through a neural network architecture. The activated ReLU units in every layer of the architecture form a binary code c_i that define the linear region. The correlation between binary codes for the whole mini-batch can be examined by computing the kernel matrix

$$K_H = \begin{pmatrix} N_A - d_H(c_1, c_1) & \cdots & N_A - d_H(c_1, c_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(c_N, c_1) & \cdots & N_A - d_H(c_N, c_N) \end{pmatrix}, \quad (1)$$

where N_A is the number of ReLU units and $d_H(c_i, c_j)$ is the hamming distance between the binary code c_i and c_j. With the kernel matrix, the score of an architecture can be evaluated as follow:

$$s = \log|K_H|, \quad (2)$$

The rationale behind is that, the more different between the binary codes the better the architecture learns. And the determinant of the kernel matrix measures how "different" they are by calculating the volume formed by the row vector of K_H.

Wu et al. [18] found, in some case, an architecture with high NASWOT score may classify the same kind of input data into different classes. To fix this defect, Wu et al. proposed using noise immunity (NI) to evaluate an architecture. Figure 2 gives an example to illustrate how noise immunity evaluate an architecture. The score function picks a mini-batch of data, denoted X, and then applies Gaussion noise on it. The process can be defined by $X' = X + z$ where z is the Gaussion noise. By passing X and X' through the untrained architecture, then computing the difference of square of each feature maps captured at all pooling layers, denoted η. More specifically, first, calculate the matrix κ defined by

$$\kappa = \begin{pmatrix} \frac{(\tau_{1,1} - \tau'_{1,1})^2}{|\tau_{1,1}|} & \cdots & \frac{(\tau_{1,N} - \tau'_{1,N})^2}{|\tau_{1,N}|} \\ \vdots & \ddots & \vdots \\ \frac{(\tau_{L,1} - \tau'_{L,1})^2}{|\tau_{L,1}|} & \cdots & \frac{(\tau_{L,N} - \tau'_{L,N})^2}{|\tau_{L,N}|} \end{pmatrix}, \quad (3)$$

where L is the total number of pooling layers; N the size of mini-batch, and $\tau_{i,j}$ and $\tau'_{i,j}$ are the feature maps which X passing and the one perturbed by X' at the i-th pooling layer and j-th input data of the mini-batch, respectively. Then η can be calculated by

$$\eta = \ln(\epsilon + e_L \kappa e_N), \quad (4)$$

where ϵ is a small positive number, and e_L and e_N are a row vector of 1's of size L and a column vector of 1's of size N, respectively. According to the fact that the input data are the same kind, the smaller η is, the better the noise immunity of the architecture is.

Besides from these two image-related aspects, there is another aspect to evaluate a network. *The Lottery Ticket Hypothesis* [21], reveals that a network may have a "core" which decides the final accuracy of the network. How to pruning a network correctly is therefore tempting. Lee et al. [9] proposed to use connection sensitivity as a criterion to prune the network which can be briefly viewed as

$$s_j = |\frac{\partial \mathcal{L}}{\partial w_j} w_j|, \quad (5)$$

where $\mathbf{w}$ is the weight of the netwrok, w_j the j-th element of $\mathbf{w}$, and $\mathcal{L}$ is the empirical risk. The meaning behind is to approximate the contribution to the change of the loss function from a specific connection. By pruning the connections which has relatively small contribution, the expensive prune, retrain cycles, can be prevented. Later, Wang et al. [10] noticed that SNIP with a high pruning ratio tends to cause *layer-collapse*, which prunes all parameters in a single weight layer. Therefore, they propose Gradient Signal Preservation (GraSP) algorithm, which aims to preserve the gradient flow at initialization by approximating the change in gradient norm defined as

$$S_p(\theta) = -(\mathbf{H} \frac{\partial \mathcal{L}}{\partial \theta}) \odot \theta, \quad (6)$$

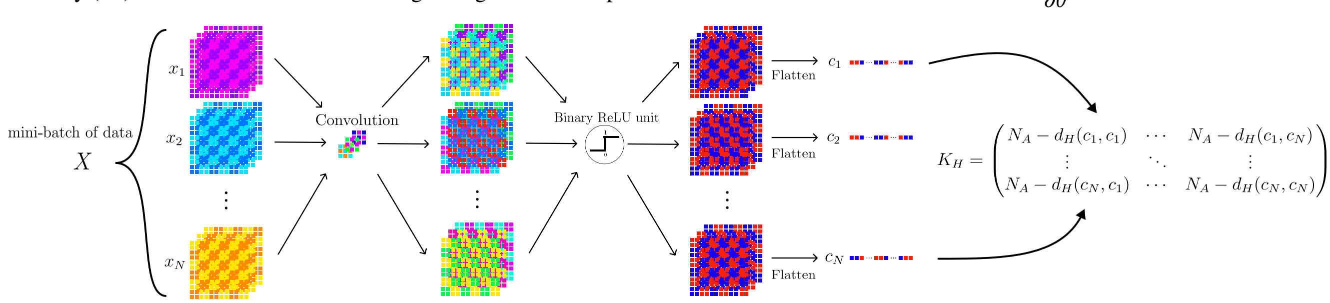


Figure 1: A simple example to illustrate the procedure of NASWOT.

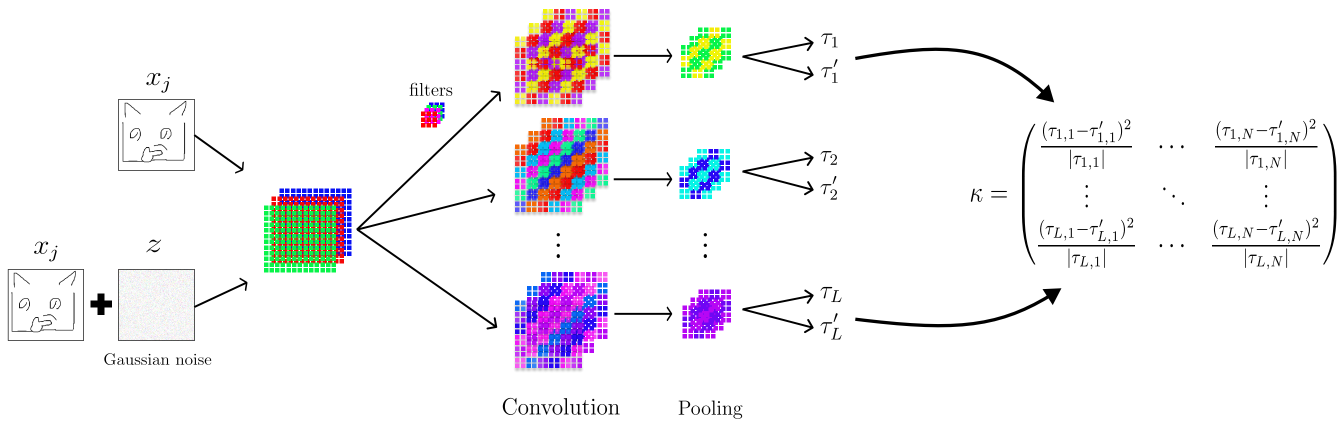


Figure 2: A simple example to illustrate the procedure of noise immunity.

where $\mathbf{H}$ is the Hessian matrix, θ the parameters, and $\odot$ is Hadamard product. In [11], to solve the problem that the existing pruning algorithms, e.g., Magnitude, SNIP, GraSP, using global-masking usually encounter layer-collapse which will make the pruned network untrainable. Tanaka et al. generalized the synaptic saliency scores as

$$S_p(\theta) = \frac{\partial \mathcal{R}}{\partial \theta} \odot \theta, \tag{7}$$

where $\mathcal{R}$ is a scalar loss function, and proposed Iterative Synaptic Flow Pruning (SynFlow) algorithm which is an extension of magnitude pruning algorithm and avoids layer-collapse. Abdelfattah et al. [12] extended the work, proposed to score a network by summing the synaptic saliency score over all parameters in the model, which is defined as

$$S_n = \sum_{i}^{N} S_p(\theta)_i. \tag{8}$$

The rationale behind is to calculate all the contribution to the loss function of parameters. The higher the score of an architecture, the more trainable this architecture is. Finally, Cavagnero et al. [13] imporved SynFlow, which is likely to fall into gradient explosion problem, and proposed LogSynFlow which simply scaling down the gradient. The formula is defined by

$$S(\theta) = \theta \cdot \log(\frac{\partial \mathcal{L}}{\partial \theta} + 1) \tag{9}$$

2.2 Search Algorithms

First assume that input data D are separated into two subset, D_r and D_t. Further more, assume $\mathcal{F}_A$ is the accuracy function, $\mathcal{F}_S$ the score function, $\mathcal{A}_s$ the search algorithm of NAS, and $\mathcal{A}_L$ the learning algorithm. Then, NAS problem can be described as an optimization problem as follows:

$$\mathbb{N}^* = \max_{\mathbb{N}^b \in \mathbb{N}} \mathcal{F}_A(\mathcal{A}_L(\mathbb{N}^b, D_r), D_t) \tag{10}$$

where $\mathbb{N}$ is a set of neural architectures, namely, the search space of NAS. For non-training-free NAS

$$\mathbb{N}^b = \mathcal{A}_S(\mathcal{F}_A(\mathcal{A}_L(\mathbb{N}^s, D_r), D_t), \mathbb{N}) \tag{11}$$

and for training-free NAS

$$\mathbb{N}^b = \mathcal{A}_S(\mathcal{F}_S(\mathbb{N}^s, D_r), \mathbb{N}) \tag{12}$$

It can be seen that training-free NAS is comprised of two part, search algorithm $\mathcal{A}_S$ and score function $\mathcal{F}_S$. There are several search algorithms applied on NAS, including random search, reinforcement learning, and metaheuristic algorithm.

In [3] [22], random search is applied on. The advantage of random search is simple, easy to implement, and the result can be token as a baseline compared to more comprehensive search algorithm. Generally speaking, when applying random search on a samll search space, e.g., nasbench201 [23], the performance is similar to other search algorithms. But when it comes to a relatively larger search space, e.g., nasbench101 [24] and natsbenchsss [17], random search can no longer standout in other search algorithms.

In [25], reinforcement learning is used to find the maximum accuracy of an architecture generated by a network architecture controller. The actions $a_{1:T}$ of the reinforcement learning is equivalent to updating the parameters θ_c for the controller. More specifically, the goal is to maximize its expected reward, defined by

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R] \tag{13}$$

and the gradient of $J(\theta_c)$ is defined by

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t}^{T} = 1 E_{P(a_{1:T};\theta_c)}[\nabla_{\theta_c} \log P(a_t \mid a_{(t-1):1};\theta_c)R] \tag{14}$$

As for an example of metaheuristic algorithm, in [26], Wu et al. leveraged genetic algorithm (GA) as search strategy. By encoding the network architecture, the architecture can be viewed as gene. Therefore, GA can be easily applied on. Later in [18], Wu et al. used search economic (SE) [27] as search strategy. The basic idea of SEs is to first divide the search space into a set of subspaces and investiagte those subspaces based on the expected value of each subspace. The expected value is comprised of

- The number of times the subspace has been investiagted.
- The average objective value of the new candidate solutions in the subspace at current iteration.
- The best solution so far in this subspace.

Based on these design, SE can avoid fall into local optimum, while search for high expected value instead.

3 METHODS

3.1 Motivation

The motivation origins from these three questions:

1. Can we combine multiple score functions to improve the correlation between score values and the final accuracy of architectures?

The answer is yes. It's konwn that if the chosen score functions can complement with each others, the new correlation can eventually beyond the original ones. [26] shows that possibility. When an architecture gain a high NASWOT score, it may classify images, which are in the same class originally, into different classes. NI here comes to rescue. By measuring the noise immunity of an architecture, the miss-classifying can be solved. Thus, if mix NASWOT with NI, the correlation increases. An example is shown in Figure 3.

2. How to combine multiple different kinds of score functions together and prevent one from overwhelming the others numerically?

One possible way is normalization, but the necessary information is known only after evaluating the whole search space, e.g., range, mean or standard deviation, which is practically impossible. In order to achieve the goal, the proposed score function, named rank-based NAS, leverage multiple score functions by ranking, which provides a solution to have equal contribution from each score function.

3. How to choose the score functions?

As mentioned, the key is to choose the complement of a score function. For example, NASWOT and NI. And for NASWOT and NI, which are both description of the ability of the graphic recognition of an architecture. The complement of them can be a measurement of trainability, e.g., SynFlow, NTK, which means whether this architecture is easy to train or not. In this paper, NI, NASWOT, and LogSynFlow are used in the ranking algorithm.

3.2 The Ranking Algorithm

Algorithm 1 The Ranking Algorithm

Input: a subspace, $\mathbb{N}^s$, from search space, $\mathbb{N}^*$

for each $\mathbb{N}^i$ in $\mathbb{N}^s$ **do**
  for each $j \in \{1, \ldots, M\}$ **do**
    $s_{\mathcal{F}_j}(\mathbb{N}^i) = \mathcal{F}_j(\mathbb{N}^i)$
  end for
end for
for each $j \in \{1, \ldots, M\}$ **do**
  $r_{\mathcal{F}_j}(\mathbb{N}^i)$ = index of $\mathbb{N}^i$ in list $[s_{\mathcal{F}_j}(\mathbb{N}^1), \ldots, s_{\mathcal{F}_j}(\mathbb{N}^{|\mathbb{N}^s|})]$ sorted by descending order
end for
$r(\mathbb{N}^i) = \sum_{j=0}^{M} r_{\mathcal{F}_j}(\mathbb{N}^i)$
Return r

The proposed ranking algorithm is outlined in Algorithm 1. The first step is to evaluate the network architectures in the subset, denoted $\mathbb{N}^s$, of search space, denoted $\mathbb{N}^*$. The evaluation of an architecture determined by j-th score function is denoted $s_{\mathcal{F}_j}(\mathbb{N}^i)$. After evaluating the subset of search space, the rank of each score function is calculated, denoted $r_{\mathcal{F}}$. Then the final rank, r, calculated by summing up the ranks. If a network architecture gains higher score in these score functions, the final rank should be higher (smaller index) too. Therefore, the highest (minimum) rank in r shall be the best network architecture. In summary, the rank-based score function makes the contribution of each score function even, and therefore evaluate an architecture from different aspects. In this paper, NI and NASWOT are used as a complement set, which take the ability of images generalization and distinction. Lastly, LogSynFlow complements NI and NASWOT which takes the trainability into account.

3.3 Simulated Annealing with Rank-Based Score function

Algorithm 2 The Simulated Annealing with Ranking Algorithm

Input: search space $\mathbb{N}^*$, initial temperature T, ending temperature τ, a real number between 0 and 1, α, the number of iteration, I

$s^* = s$ = sample a random architecture from $\mathbb{N}^*$
while $T > \tau$ **do**
  for each $i \in [1 \ldots I]$ **do**
    $r = \text{Ranking}(\{s\} \cup U(s))$ ▷ Ranking s with the neighbourhood of s, denoted $U(s)$
    $s^* = \arg\min\{r(\mathbb{N}^i)\}$ ▷ Update the expected best algorithm
    $\Delta E = r(s^*) - r(s)$
    $x \sim \mathcal{U}(0, 1)$ ▷ Sample x from a uniform distribution between 0 and 1
    if $x \geq e^{\frac{\Delta E}{T}}$ **then** ▷ Accept by probability
      $s \leftarrow s^*$
    end if
  end for
  $T \leftarrow T \times \alpha$
end while
Return s^* ▷ Return the best architecture

The modified version of the simulated annealing algorithm with ranking algorithm is outlined in Algorithm 2. The first step is to sample a architecture, denoted s, from search space. Then use s as a "seed" to generate the neighbourhood of it, applying ranking algorithm on the neighbourhood, $\{s\} \cup U(s)$. After ranking, the expected best architecture can be determined and update, but the seed for next iteration will be replaced by s^* according to probability, $e^{\frac{\Delta E}{T}}$. After the I iterations, the temperature scales down by α. The algorithm is done when $T \leq \tau$, and the best architecture will be retruned.

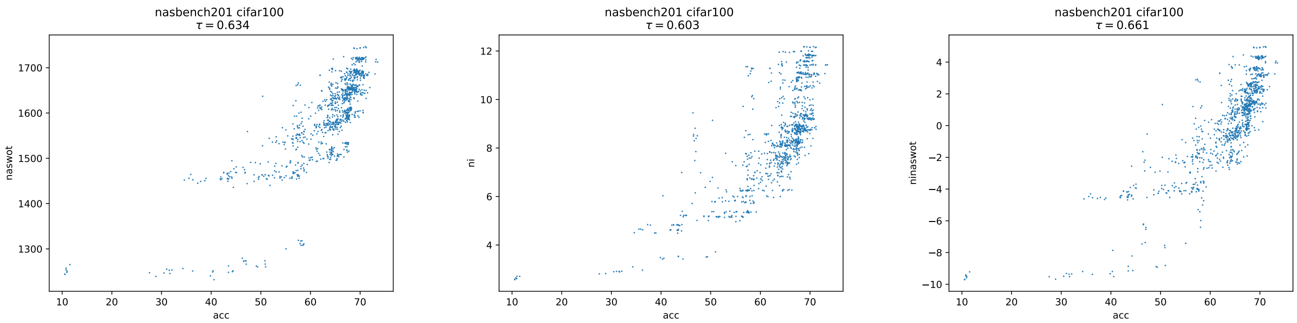
4 EXPERIMENT RESULT

Environment and Parameter Settings

The experiment is conducted on a PC with Intel Core i7-11700K (3.60 GHz, 16-MB Cache, and 16 cores), a single NVIDIA RTX3070 Ti GPU with 8 GB memory, driver version 520.61.05, CUDA version 11.8, and 65 GB available memory running on Ubuntu 20.04.1 with linux kernel 5.15.0-73-generic. All the program is written in Python 3.7.16 with PyTorch 1.7.1+cu110 package.

NATS-bench-SSS

The result of NATS-bench-SSS is shown in Table 2.



(a) (b) (c)

Figure 3: (a) NASWOT score for 1,000 randomly chosen architectures from NAS-Bench-201 in the CIFAR-10 dataset (b) NI score for 1,000 identical architectures from NAS-Bench-201 in the CIFAR-10 dataset. (c) NI score + NASWOT score for 1,000 identical architectures from NAS-Bench-201 in the CIFAR-10 dataset.

Table 2: Comparison of rank-based NAS and all the other NAS algorithms in NATS-Bench-SSS.

Method	Search (s)	CIFAR-10		CIFAR-100		ImageNet-16-120	
		validation	test	validation	test	validation	test
				Non-weight sharing			
AREA	12,000	84.62 ± 0.41	93.16 ± 0.16	59.24 ± 1.11	69.56 ± 0.96	37.58 ± 1.09	45.30 ± 0.91
REA	12,000	90.26 ± 0.22	93.17 ± 0.21	69.48 ± 0.76	69.49 ± 0.94	44.84 ± 0.72	45.47 ± 0.91
RS	12,000	90.08 ± 0.24	93.01 ± 0.29	69.14 ± 0.94	69.17 ± 1.00	44.66 ± 1.02	44.83 ± 1.05
RL	12,000	90.17 ± 0.23	93.08 ± 0.21	69.23 ± 0.87	69.29 ± 1.08	44.68 ± 0.91	45.05 ± 0.93
BOHB	12,000	90.05 ± 0.30	93.03 ± 0.20	69.04 ± 0.76	69.16 ± 0.90	44.71 ± 0.78	44.91 ± 1.05
				Training-free			
NI (N=1,000)	*X*	89.56 ± 0.147	92.55 ± 0.187	*X* ± *X*	*X* ± *X*	*X* ± *X*	*X* ± *X*
NASWOT (N=1,000)	*X*	89.25 ± 0.412	92.21 ± 0.298	*X* ± *X*	*X* ± *X*	*X* ± *X*	*X* ± *X*
LogSynFlow (N=1,000)	*X*	89.61 ± 0.101	92.60 ± 0.188	*X* ± *X*	*X* ± *X*	*X* ± *X*	*X* ± *X*
rk (N=1000)	*X*	89.59 ± 0.136	92.51 ± 0.171	*X* ± *X*	*X* ± *X*	*X* ± *X*	*X* ± *X*
GA-rk	457.54	90.29 ± 0.149	93.27 ± 0.193	69.88 ± 0.497	70.06 ± 0.481	45.57 ± 0.425	46.19 ± 0.846
SA-rk	682.36	90.37 ± 0.204	93.29 ± 0.182	70.11 ± 0.457	70.32 ± 0.458	45.38 ± 0.499	46.45 ± 0.687

NAS-Bench-201

The result of NAS-Bench-201 is shown in Table 3.

NAS-Bench-101

The result of NAS-Bench-101 is shown in Table 4.

5 CONCLUSION

REFERENCES

[1] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016. [Online]. Available: https://arxiv.org/abs/1611.01578

[2] Z. Sun, M. Lin, X. Sun, Z. Tan, H. Li, and R. Jin, "Mae-det: Revisiting maximum entropy principle in zero-shot nas for efficient object detection," 2021. [Online]. Available: https://arxiv.org/abs/2111.13336

[3] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," 2020. [Online]. Available: https://arxiv.org/abs/2006.04647

[4] H. Zheng, K. An, and Z. Ou, "Efficient neural architecture search for end-to-end speech recognition via straight-through gradients," 2020. [Online]. Available: https://arxiv.org/abs/2011.05649

[5] A. Mehrotra, A. G. C. P. Ramos, S. Bhattacharya, Ł. Dudziak, R. Vipperla, T. Chau, M. S. Abdelfattah, S. Ishtiaq, and N. D. Lane, "{NAS}-bench-{asr}: Reproducible neural architecture search for speech recognition," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=CU0APx9LMaL

[6] Y. Jiang, C. Hu, T. Xiao, C. Zhang, and J. Zhu, "Improved differentiable architecture search for language modeling and named entity recognition," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3585–3590. [Online]. Available: https://aclanthology.org/D19-1367

[7] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, and E. Burnaev, "Nas-bench-nlp: Neural architecture search benchmark for natural language processing," 2020. [Online]. Available: https://arxiv.org/abs/2006.07116

[8] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "Hat: Hardware-aware transformers for efficient natural language processing," 2020. [Online]. Available: https://arxiv.org/abs/2005.14187

[9] N. Lee, T. Ajanthan, and P. H. S. Torr, "Snip: Single-shot network pruning based on connection sensitivity," 2019.

[10] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," 2020.

[11] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," 2020.

[12] M. S. Abdelfattah, A. Mehrotra, Łukasz Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight nas," 2021.

[13] N. Cavagnero, L. Robbiano, B. Caputo, and G. Averta, "FreeREA: Training-free evolution-based architecture search," in *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, jan 2023. [Online]. Available: https://doi.org/10.1109%2Fwacv56688.2023.00154

[14] W. Chen, X. Gong, and Z. Wang, "Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective," 2021. [Online]. Available: https://arxiv.org/abs/2102.11535

[15] H. Wang, Y. Wang, R. Sun, and B. Li, "Global convergence of maml and theory-inspired neural architecture search for few-shot learning," 2022. [Online]. Available: https://arxiv.org/abs/2203.09137

[16] Y. Shu, S. Cai, Z. Dai, B. C. Ooi, and B. K. H. Low, "Nasi: Label-and data-agnostic neural architecture search at initialization," 2021. [Online].

Table 3: COMPARISON OF RANK-BASED NAS AND ALL THE OTHER NAS ALGORITHMS IN NAS-BENCH-201.

Method	Search (s)	CIFAR-10		CIFAR-100		ImageNet-16-120	
		validation	test	validation	test	validation	test
Non-weight sharing							
AREA	12,000	91.18 ± 0.43	93.95 ± 0.39	71.84 ± 1.21	71.92 ± 1.29	45.04 ± 1.03	45.40 ± 1.14
REA	12,000	91.08 ± 0.54	93.89 ± 0.50	71.69 ± 1.34	71.83 ± 1.33	44.96 ± 1.41	45.30 ± 1.51
RS	12,000	90.91 ± 0.33	93.67 ± 0.33	70.91 ± 1.04	70.99 ± 0.99	44.52 ± 0.99	44.56 ± 1.25
RL	12,000	90.87 ± 0.41	93.63 ± 0.36	70.62 ± 1.08	70.77 ± 1.05	44.20 ± 1.22	44.23 ± 1.37
BOHB	12,000	88.47 ± 1.19	91.79 ± 1.11	67.18 ± 2.05	67.50 ± 2.05	38.94 ± 3.58	39.00 ± 3.73
Weight sharing							
RSWS	4,154	76.95 ± 16.74	82.60 ± 12.10	52.51 ± 18.33	52.93 ± 18.32	29.76 ± 9.50	29.16 ± 9.61
DARTS-V1	5,475	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
DARTS-V2	16,114	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
GDAS	11,183	90.05 ± 0.23	93.46 ± 0.13	71.02 ± 0.31	70.56 ± 0.24	41.77 ± 1.24	41.96 ± 0.90
SETN	16,787	84.25 ± 5.05	88.01 ± 4.52	59.72 ± 7.30	59.91 ± 7.51	33.93 ± 3.85	33.48 ± 4.22
ENAS	7,061	40.11 ± 3.28	56.33 ± 3.70	14.09 ± 1.60	14.77 ± 1.45	16.20 ± 0.48	15.93 ± 0.67
Training-free							
NI (N=1,000)	X	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$
NASWOT (N=1,000)	X	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$
LogSynFlow (N=1,000)	X	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$
rk (N=1000)	X	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$
GA-rk	747.25	89.93 ± 0.196	93.41 ± 0.083	70.70 ± 0.417	70.76 ± 0.378	42.70 ± 1.315	43.10 ± 1.428
SA-rk	X	89.95 ± 0.194	93.37 ± 0.114	70.69 ± 0.391	70.75 ± 0.532	$X \pm X$	43.10 ± 1.506

Table 4: COMPARISON OF RANK-BASED NAS AND ALL THE OTHER NAS ALGORITHMS IN NAS-BENCH-101.

Method	Search (s)	CIFAR-10	
		validation	test
Non-weight sharing			
AREA	12,000	93.67 ± 0.48	93.02 ± 0.48
REA	12,000	93.63 ± 0.50	93.02 ± 0.52
Weight sharing			
DARTS-V1	20,483	83.50 ± 0.11	83.74 ± 0.03
ENAS	22,325	93.83 ± 0.28	93.34 ± 0.26
Training-free			
Zero-Cost NASWOT	18,940	91.72 ± 1.80	91.29 ± 1.82
NI (N=1,000)	280	93.15 ± 1.48	92.72 ± 1.06
NASWOT (N=1,000)	162	92.40 ± 4.07	91.97 ± 4.20
LogSynFlow (N=1,000)	105	90.70 ± 8.41	90.98 ± 2.56
rank (N=1000)	516	92.84 ± 1.19	92.51 ± 1.14
GA-rank	X	$X \pm X$	$X \pm X$
SA-rank	X	93.01 ± 1.19	$X \pm X$

Available: https://arxiv.org/abs/2109.00817

[17] X. Dong, L. Liu, K. Musial, and B. Gabrys, "NATS-bench: Benchmarking NAS algorithms for architecture topology and size," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. [Online]. Available: https://doi.org/10.1109%2Ftpami.2021.3054824

[18] M.-T. Wu, H.-I. Lin, and C.-W. Tsai, "A training-free neural architecture search algorithm based on search economics," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2023.

[19] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[20] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[21] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2019.

[22] V. Lopes, S. Alirezazadeh, and L. A. Alexandre, "EPE-NAS: Efficient performance estimation without training for neural architecture search," in *Lecture Notes in Computer Science*. Springer International Publishing, 2021, pp. 552–563. [Online]. Available: https://doi.org/10.1007%2F978-3-030-86383-8_44

[23] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," 2020.

[24] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," 2019.

[25] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017.

[26] M.-T. Wu, H.-I. Lin, and C.-W. Tsai, "A training-free genetic neural architecture search," in *Proceedings of the 2021 ACM International Conference on Intelligent Computing and Its Emerging Applications*, ser. ACM ICEA '21. New York, NY, USA: Association for Computing Machinery, 2022, p. 65–70. [Online]. Available: https://doi.org/10.1145/3491396.3506510

[27] C.-W. Tsai, "Search economics: A solution space and computing resource aware search method," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, 2015, pp. 2555–2560.