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

The training-free neural architecture search (NAS) typically uses one or two score functions to evaluate how an neural network architecture performs. However, an architecture shall be treated as a complex system who has distinct characteristics to measure. To achieve the goal, we propose a rank-based training-free NAS algorithm, which combines three training-free score functions which is "complementary" to each others to evaluate an architecture by ranking. In this work, noise immunity (NI) considers the image generalization ability of the architecture, the correlation between binary activation paterns, named as NASWOT, considers image distinction ability, and LogSynFlow takes trainability into account. With that, a modified version of simulated annealing (SA) algorithm, SA-rank, 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 indicates that SA-rankoutperforms most of training-free score functions and shows the robustness between different search spaces.

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, 8 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 (NLP). [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.

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, abbreviated as NASWOT. On the other hand, Lee et al. [9] proposed purning parameters based on a saliency matric, which then extended by Tanaka et al. [10]. In [10], 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. [11] extended SynFlow to score a network architecture by summing synaptic saliency score over all parameters in the model. Cavagnero et al. [12] 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. [13] proposed to compute the condition number of neural tangent kernel (NTK) [14] [15], 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 characteristic of an architecture. To address the problem, we propose cooperating three complementary score functions by ranking, which allows every score functions evaluate an architecture without losing fairness. The first score function is noise immunity (NI) [17], as a measurement of the ability of image generalization. The second one is NASWOT, as score to estimate the ability of distinction of an architecture and also a complement of NI. The last one is LogSynFlow used as the measurement of trainability

Table 1: THE KENDALL CORRELATION BETWEEN TRAINING-FREE SCORE AND TEST ACCURACY, EVALUATED ON THE THREE DATASETS OF NATS-BENCH [16].

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

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

and also as a complement of NI and NASWOT. With these three complementary 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 complementary 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 abbreviated as 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. [17] 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 NI 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. 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 image-related aspects, there is another aspect to evaluate a network. *The Lottery Ticket Hypothesis* [18], 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 w_j the j-th element of the weight of the netwrok, 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. [19] 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

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

where $\mathbf{H}$ is the Hessian matrix, θ the parameters, and $\odot$ is Hadamard product. In [10], 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

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

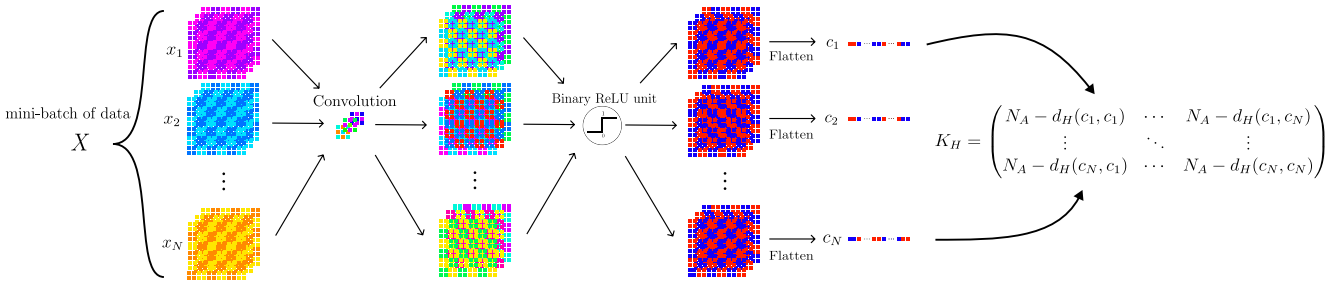

Figure 1: A simple example to illustrate the procedure of NASWOT.

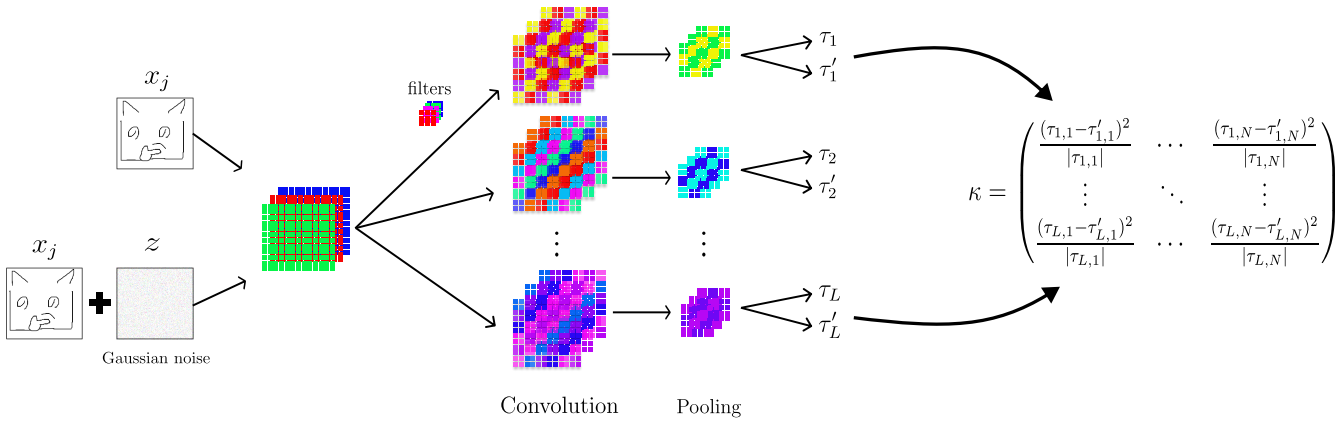

Figure 2: A simple example to illustrate the procedure of NI.

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. [11] 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. [12] 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

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

Except using the synaptic saliency score to measure the trainability, [13] uses the condition number of neural tangent kernel (NTK) [20] to measure it, which the score can be defined as

$$\frac{\lambda_m}{\lambda_0}, \tag{10}$$

where λ_0 is the biggest eigenvalue and λ_m is the smallest eigenvalue of the NTK. The rationale behind is the converge rate of the network is condition by the rate $\frac{\lambda_m}{\lambda_0}$. The higher the condition is, the faster the converge.

2.2 Search Algorithms

First assume that input data D are separated into two subset, D_r and D_t. Furthermore, 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:

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

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

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

and for training-free NAS

$$\mathbb{N}^b = \mathcal{A}_S(\mathcal{F}_S(\mathbb{N}^s, D_r), \mathbb{N}) \tag{13}$$

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] [21], 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 [22], the performance is similar to other search algorithms. But when it comes to a relatively larger search space, e.g., nasbench101 [23] and natsbenchsss [16], random search can no longer standout in other search algorithms.

In [24], 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. The goal is to maximize its expected reward, defined by

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R] \tag{14}$$

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

$$\nabla_{\theta_c} J(\theta_c) = \sum_t^T = 1E_{P(a_{1:T};\theta_c)}[\nabla_{\theta_c} \log P(a_t \mid a_{(t-1):1}; \theta_c)R] \tag{15}$$

As for an example of metaheuristic algorithm, in [25], 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 [17], Wu et al. used search economic (SE) [26] 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

The motivation origins from these three questions:

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

The answer is yes. If the chosen score functions is complementary to each others, the new correlation can eventually beyond the original ones. [25] 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 problem can be solved. Thus, if mix NASWOT with NI, the correlation and performance increases. An example is shown in Figure 3.

2. How to choose the score functions?

As mentioned, the key is to choose the complement of a score function. For example, score functions that evaluate the expressivity/distinction ability shall have other score functions to evalute the generalization ability. For these two graphic recognition score functions, the complement of them can be a measurement of trainability.

3. 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 make the proposed score function agnostic to the search space, it leverages multiple score functions by ranking, which also provides a solution to have equal contribution from each score function.

3.1 Simulated Annealing with Ranking Algorithm

The modified version of the simulated annealing (SA) algorithm with ranking algorithm (SA-rank) is outlined in Algorithm 1. The first step is to sample an architecture as current solution, denoted s, from search space. Then use s as a seed to generate the neighbourhood of it, denoted U, and the size of it is decided by a presetting parameter, the number of generated neighbours, N. After ranking with the seed, the neighbourhood and the historical best architectures set, H, it is maintained by adding the architecture (including s) which has highest rank, and the architecture (excluding s) which has highest rank is denoted by s^*. If the rank of s^* is higher than s, the current solution will be replaced by s^*, or be replaced by probability, defined by

$$e^{-\alpha \frac{\Delta E}{T}}, \tag{16}$$

where T is the current temperature, ΔE the difference of the rank between s and s^*, and α is a constant number adjusts the probability. The bigger the difference, the smaller the probability to accept the new solution. After the I iterations, the temperature scales down by r_t. The algorithm is done when $T \leq \tau$, and the current architecture will be retruned. Notbly, the size of H is based on the maximum searching number of SA, defined by

$$\beta(I \cdot N \cdot \frac{\ln \tau / T}{\ln r_t}), \tag{17}$$

where β is a constant number between 0 and 1.

Algorithm 1 The Simulated Annealing with Ranking Algorithm

Input: search space $\mathbb{N}^*$

Parameter: Initial temperature T, ending temperature τ, a real number between 0 and 1, r_t, the number of iteration, I, the number of generated neighbours, N, a real number, α

```
Initialize historical best architectures set, H
s = Randomly sample a architecture from N*
while T > τ do
    for each i ∈ [1 ... I] do
        U = neighbour(s, N)
        Ranking({s} ∪ U ∪ H)
        Maintain historical best architectures set, H
        s' = Best architecture excluding s
        ΔE = The difference of rank between s' and s
        Sample x from uniform distribution between 0 and 1
        if ΔE < 0 then
            s ← s'
        else
            if x ≤ e^{-α ΔE/T} then
                s ← s'
            end if
        end if
    end for
    T ← T × r_t
end while
Return s                          ▷ Return the current architecture
```

3.2 Encoding Schema

The encoding schema determines the efficiency of searching. The naïve encoding schema is to use a binary/integer vector as a representation of an architecture. For example, NAS-Bench-201 provides 5 operations and 6 nodes to select an cell and cells form an architecture, which implies the whole search space can be encoded into a 6 integers vector. However, a broken architecture appears since one of the operations is zeroize, which means the measurement of the broken architecture is meaningless. To exclude the broken architectures, the encoding schema in [17] is used in this paper. The detail of the encoding schema differs from each search spaces, but they all share the common core idea, preserving the critical path in an architecture. The encoding schema guarantees that there exists at least one path from input to output in a cell, and no zeroize operation is in it. Based on the encoding schema, the efficiency of searching can be improved, since it excludes the broken architectures.

3.3 Ranking Algorithm

The general proposed ranking algorithm is outlined in Algorithm 2. 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

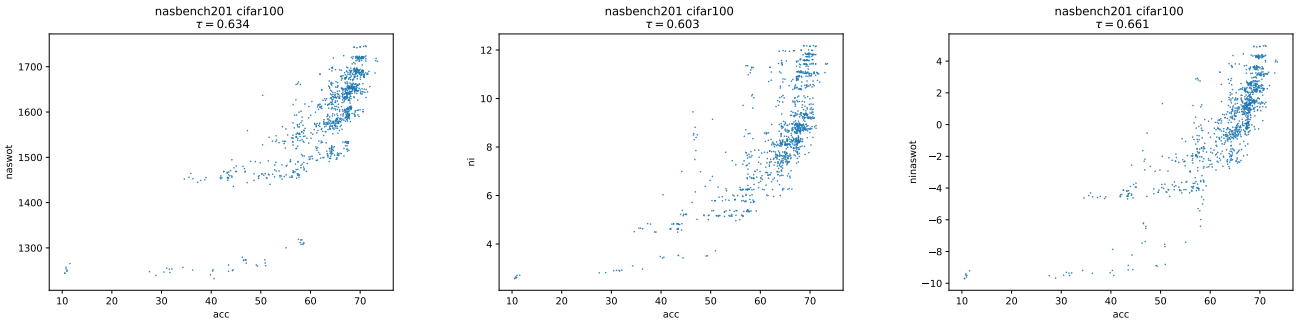

Figure 3: (a) NASWOT score for 1,000 randomly chosen architectures from NAS-Bench-201 in the CIFAR-10 dataset (b) NI score for 1,000 identical architectures from NAS-Bench-201 in the CIFAR-10 dataset. (c) NI score + NASWOT score for 1,000 identical architectures from NAS-Bench-201 in the CIFAR-10 dataset.

makes the contribution of each score function even, and therefore evaluate an architecture from different aspects.

Algorithm 2 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

With the proposed ranking algorithm as framework, the next question is to choose score functions that are complementary. The recent training-free score functions can be roughly classified into different kinds. NASWOT, linear regions distribution [13] [27], maximum entropy of an architecture [28] are measurements to the ability of distinction and expressivity of an architecture; NI is a measurement to the ability of image generalization of an architecture; the condition number of NTK, modified version of SNIP, GraSP, SynFlow, and LogSynFlow are measurements to the ability of trainability of an architecture. As mentioned, the chosen score function shall be complementary to each other. In this paper, NI and NASWOT are used as a complementary set, which take the ability of images generalization and distinction into account. These complementary set can be generalized as the ability of the graphic recognition. Therefore, the last missing piece is trainability of an architecture, which, LogSynFlow is chosen to measure it.

4 EXPERIMENT RESULT

We evalute the performance of the proposed algorithm, SA-rank, for searching the architectures of high accuracy. For comparison, several NAS algorithms are included, e.g., random search (RS), regularized evolution algorithm (REA) [29], reinforcement learning (RL) [30], Bayesian optimization with hyper band (BOHB) [31], RS with weight-sharing (RSWS) [32], differentiable architecture search with first order (DARTS-V1) [33], DARTS with second order (DARTS-V2) [33], gradient-based search using differentiable architecture sampler (GDAS) [34], self-evaluated template network (SETN) [35], efficient NAS (ENAS) [36], assisted REA (AREA) [3], zero-cost NAS [37], TE-NAS [13], gradient kernel-based NAS (KNAS) [38], RS with NASWOT [3], RS with noise immunity (NI) [17], and RS with LogSynFlow [12].

The three search spaces, NATS-bench-SSS, NATS-bench-TSS [16] and NAS-Bench-101 [23], are used here to evalute the performance of the NAS algorithms. The architectures of NATS-bench-TSS is same as NAS-Bench-201 [22]. The architectures in NATS-bench-SSS and NATS-bench-TSS are trained by the three datasets: CIFAR10, CIFAR100, ImageNet-16-120, while NAS-Bench-101 is trained by one: CIFAR10.

4.1 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. The simulations of all nonweight sharing NASs are carried out for 500 runs on all the search spaces except AREA which is 50 runs. For weight sharing NAS algorithms, each is carried out for 3 runs. And for training-free algorithms, each is carried out for 100 runs and are given an evaluation budget of 1000 for all the search space. The parameter settings of SA-rankare as follows: initial temperature, ending temperature, rate of decrease of temperature, the number of iteration, the constant number adjusting acceptance probability, the number of generated neighbours are set to 1, 0.0008, 0.745, 4, 0.25, 10, so that the maximum number of searching candidates is 1000.

4.2 NATS-bench-SSS

In Table 2, the proposed NAS algorithm and ranking function is compared with the other five nonweight sharing algorithm and NI, NASWOT, LogSynFlow. For fairness, the searching candidates for

the training-free score functions, i.e., NI, NASWOT, LogSynFlow, Rank, are identical. The performance of ranking function is similar to NI and lower than LogSynFlow. However, the proposed NAS algorithm imporved the results significantly and outperforms almost all the compared algorithms.

4.3 NATS-Bench-TSS

The result of NATS-Bench-TSS is shown in Table 3.

4.4 NAS-Bench-101

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

5 CONCLUSION

In this paper, a novel concept of "complementary" is proposed and used as a basis to choose traning-free score functions. The new ranking algorithm is then applied on simulated annealing algorithm. The simulation result shows that the proposed algorithm, namely SA-rk, not only outperforms the most state-of-art traning-free score functions, but also some of the nontraning-free methods. The proposed algorithm also shows the robustness, which performs well between different search spaces vary with neural network structures and size. The limitation of SA-rk is the lack of versatility, since the score functions are mainly chosen for common image recognition. That means SA-rk may not be applied on the search of architectures designed for other usages, i.e, speech recognition, NLP. However, the problem is able to be solved by choosing the corresponding complementary score functions. Some further questions are that how many score functions are required for precisely estimating the performance of architectures and what are they?

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

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

[12] 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

[13] 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

[14] 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

[15] 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]. Available: https://arxiv.org/abs/2109.00817

[16] 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

[17] 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.

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

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

[20] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," 2020.

[21] 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

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

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

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

[25] 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

[26] 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.

[27] M. Lin, P. Wang, Z. Sun, H. Chen, X. Sun, Q. Qian, H. Li, and R. Jin, "Zen-nas: A zero-shot nas for high-performance deep image recognition," 2021.

[28] 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," 2022.

[29] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," 2019.

[30] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[31] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," 2018.

[32] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," 2019.

[33] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2019.

[34] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," 2019.

[35] ——, "One-shot neural architecture search via self-evaluated template network," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, oct 2019. [Online]. Available: https://doi.org/10.1109%2Ficcv.2019.00378

[36] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018.

[37] M. S. Abdelfattah, A. Mehrotra, L. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight nas," 2021. [Online]. Available: https://arxiv.org/abs/2101.08134

[38] J. Xu, L. Zhao, J. Lin, R. Gao, X. Sun, and H. Yang, "Knas: Green neural architecture search," 2021.

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	90.37 ± 0.20	93.22 ± 0.16	70.23 ± 0.50	70.11 ± 0.61	45.30 ± 0.69	45.94 ± 0.92
RS	12,000	90.10 ± 0.26	93.03 ± 0.25	69.57 ± 0.57	69.72 ± 0.61	45.01 ± 0.74	45.42 ± 0.86
RL	12,000	90.25 ± 0.23	93.16 ± 0.21	69.84 ± 0.59	69.96 ± 0.57	45.06 ± 0.77	45.71 ± 0.93
BOHB	12,000	90.07 ± 0.28	93.01 ± 0.24	69.75 ± 0.60	69.90 ± 0.60	45.11 ± 0.69	45.56 ± 0.81
Training-free							
NI (N=1,000)	293	89.55 ± 0.16	92.56 ± 0.21	67.67 ± 0.91	67.61 ± 1.06	43.19 ± 0.73	43.22 ± 0.53
NASWOT (N=1,000)	208	89.25 ± 0.44	92.21 ± 0.31	65.09 ± 2.89	64.61 ± 3.16	41.29 ± 2.32	41.35 ± 2.31
LogSynFlow (N=1,000)	123	89.60 ± 0.11	92.61 ± 0.18	68.23 ± 0.49	68.30 ± 0.54	43.58 ± 0.58	43.48 ± 0.40
rank (N=1000)	612	89.59 ± 0.14	92.51 ± 0.17	67.56 ± 0.75	67.55 ± 0.82	43.36 ± 0.62	43.27 ± 0.45
SA-rank	712	90.38 ± 0.20	93.30 ± 0.19	69.94 ± 0.46	70.24 ± 0.42	45.41 ± 0.49	46.39 ± 0.69
SA-rankver2	X	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	$X \pm X$	46.43 ± 0.75
Optimal		90.71	93.65	70.92	71.34	46.73	47.40

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

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	91.25 ± 0.31	94.02 ± 0.31	72.28 ± 0.95	72.23 ± 0.84	45.71 ± 0.77	45.77 ± 0.80
RS	12,000	91.07 ± 0.26	93.86 ± 0.23	71.46 ± 0.97	71.55 ± 0.97	45.03 ± 0.91	45.28 ± 0.97
RL	12,000	91.12 ± 0.25	93.90 ± 0.26	71.80 ± 0.94	71.86 ± 0.89	45.37 ± 0.74	45.64 ± 0.78
BOHB	12,000	91.17 ± 0.27	93.94 ± 0.28	72.04 ± 0.93	72.00 ± 0.86	45.55 ± 0.79	45.70 ± 0.86
Weight sharing							
RSWS	4,154	87.60 ± 0.61	91.05 ± 0.66	68.27 ± 0.72	68.26 ± 0.96	39.73 ± 0.34	40.69 ± 0.36
DARTS-V1	5,475	49.27 ± 13.44	59.84 ± 7.84	61.08 ± 4.37	61.26 ± 4.43	38.07 ± 2.90	37.88 ± 2.91
DARTS-V2	16,114	58.78 ± 13.44	65.38 ± 7.84	59.48 ± 5.13	60.49 ± 4.95	37.56 ± 7.10	36.79 ± 7.59
GDAS	11,183	89.68 ± 0.72	93.23 ± 0.13	68.35 ± 2.71	68.17 ± 2.50	39.55 ± 0.00	39.40 ± 0.00
SETN	16,787	90.00 ± 0.97	92.72 ± 4.52	69.19 ± 1.42	69.36 ± 1.72	39.77 ± 0.33	39.51 ± 0.33
ENAS	7,061	90.20 ± 0.00	93.76 ± 0.00	70.21 ± 0.71	70.67 ± 0.62	40.78 ± 0.00	41.44 ± 0.00
Training-free							
NI (N=1,000)	X	$X \pm X$	93.21 ± 0.63	$X \pm X$	70.40 ± 0.84	$X \pm X$	43.34 ± 2.43
NASWOT (N=1,000)	X	$X \pm X$	92.80 ± 1.07	$X \pm X$	69.66 ± 1.23	$X \pm X$	43.84 ± 2.93
LogSynFlow (N=1,000)	X	$X \pm X$	92.92 ± 0.70	$X \pm X$	68.98 ± 1.97	$X \pm X$	40.23 ± 6.45
rank (N=1000)	X	$X \pm X$	93.32 ± 0.42	$X \pm X$	70.10 ± 1.45	$X \pm X$	44.07 ± 2.19
SA-rank	X	$X \pm X$	$\approx 93 \pm 2$	$X \pm X$	70.30 ± 1.26	$X \pm X$	$\approx 43 \pm 4$
SA-rankver2	X	$X \pm X$	$X \pm X$	$X \pm X$	70.63 ± 0.66	$X \pm X$	44.63 ± 1.64
Optimal		91.61	94.37	73.49	73.51	46.73	47.31

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	92.89 ± 2.10	92.55 ± 1.33
NASWOT (N=1,000)	162	92.27 ± 6.60	92.10 ± 4.44
LogSynFlow (N=1,000)	105	91.81 ± 1.52	91.21 ± 1.71
rank (N=1000)	516	92.44 ± 1.42	91.94 ± 2.01
SA-rank	374	92.89 ± 3.24	92.77 ± 1.01
SA-rankver2	374	93.16 ± 0.97	92.77 ± 1.03
Optimal		*X*	94.32