True Rank Guided Efficient Neural Architecture Search for End to End Low-Complexity Network Discovery*

Shahid Siddiqui$^{1[0000-0002-6176-2952]}$ and Christos Kyrkou$^{2[0000-0002-7926-7642]}$ and Theocharis Theocharides$^{1,2[0000-0001-7222-9152]}$

^{1} University of Cyprus
^{2} KIOS Research and Innovation Center of Excellence
https://www.kios.ucy.ac.cy/
{msiddi01,kyrkou.christos,ttheocharides}@ucy.ac.cy

Abstract. Neural architecture search (NAS) aims to automate neural network design process and has shown promising results for image classification tasks. Owing to combinatorially huge neural network design spaces coupled with training cost of candidates, NAS is computationally demanding. Hence, many NAS works focus on efficiency by constraining the search to only network building blocks (modular search) instead of searching for the entire architectures (global search), and by approximating candidates' performance instead of expensive training. Modular search, however, offers only partial network discovery and final architecture configuration such as network's depth or width requires manual trial and error. Further, approximating candidates' performance incur misleading search directions due to their inaccurate relative rankings. In this work, we revisit NAS for end to end network discovery and guide the search using true rankings of candidates by training each from scratch. However, it is computationally infeasible for existing search strategies to navigate huge search spaces and determine accurate rankings at the same time. Therefore, we propose a novel search space and an efficient search algorithm that offers high accuracy low complexity network discovery with competitive search cost. Our proposed approach is evaluated on the CIFAR-10, yielding an error rate of 4% while the search cost is just 4.5 GPU days. Moreover, our model is 7.3×, 3.7× and 5.5× smaller than the smallest models discovered by RL, evolutionary and gradient-based NAS methods respectively. ‡

Keywords: Neural Architecture Search · AutoML · Deep Learning · Image Classification · Convolutional Neural Networks.

*This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 739551 (KIOS CoE) and from the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development.

‡Code and results available at: https://github.com/siddikui/TRG-NAS

1 Introduction

Neural Architecture Search (NAS) is the task of automating the otherwise tedious and manual neural network design process and has the potential to truly democratize the use of deep learning. Early NAS works based on reinforcement learning (RL) and evolution [4,5,6,7] achieve impressive results for image classification tasks, but hundreds of days of search cost makes practical adaptation of these methods infeasible. Therefore, follow up research has mostly focused on accelerating NAS by transitioning from global to modular search spaces [9,14], replacing discrete optimization algorithms with continuous search strategies [14,15], and approximating candidate networks' performance instead of expensive training [10,8,14,22,21]. We refer the reader to [20] for more details on how NAS research has evolved.

The techniques adopted to speed up NAS, collectively, have greatly improved the search efficiency i.e., from 22400 GPU-days of RL [4] to 4 or less GPU-days of gradient descent [14,22], but lead to various trade-offs. For instance, NAS-Net [9] proposes a cell-based (modular) search space instead of searching for the entire architecture (global search) and many subsequent works have followed this approach [14,22]. However, once a cell is discovered, the decision of total number of cells to be stacked up (depth) or channels (width) needs manual trial and error. This contradicts the original idea of NAS i.e., automatic network discovery for a given dataset with minimal expert intervention. Moreover, [17] shows that cell-based search spaces have narrow accuracy ranges such that even a randomly sampled architecture performs quite well. Such restricted spaces, although guaranteeing good and quick results, do not possess performance-wise architectural diversity and hence do not allow application adaptive network design. Similarly, continuous search strategies [14,22] are coupled with parameter sharing techniques, which leads to inaccurate search directions as discussed in [20]. Therefore, we retreat to discrete search strategies for end to end network discovery using global search spaces.

Searching an architecture for maximizing a given objective, for e.g., accuracy on test dataset, requires evaluating relative rankings of candidate networks. Relative ranking is defined as how well or worst an architecture performs as compared to others on the test dataset. But to determine rankings, networks need to go through expensive training. Hence, various performance approximations are proposed instead. For example, [10,8] suggest weight reusing, but it is unclear whether accuracy improvement is because of better discovered network or because of inheriting pre-trained weights. Further, various performance predictors have been proposed to speed up NAS [21], but we argue that guaranteed true rankings of candidates can only be revealed by training each from scratch and till convergence. However, it is computationally infeasible for existing search strategies to navigate combinatorially huge search spaces and determine accurate rankings at the same time. Therefore, to guide the search by true rankings, we propose an efficient search framework. Our contribution can be summarized as follows:

- A minimal yet powerful search space allowing both macro i.e., depth and width and fine grain micro i.e., operation and kernel search.
- A true rank guided search strategy for end to end high accuracy low complexity network discovery with competitive search cost.

On CIFAR-10, our true rank guided NAS (TRG-NAS) discovers a 0.45M parameter model with an error rate of 4% and the search cost is just 4.5 days. Our model is 7.3×, 3.7× and 5.5× smaller than the smallest models discovered by RL [9], evolutionary [7] and gradient-based [15] NAS methods, respectively. The remaining paper is organized as follows. Section 2 presents related work, followed by the proposed methodology in Section 3. Experimental results are discussed in Section 4 and conclusion in Section 5.

2 Related Work

Contrary to trending NAS approaches focused on efficiency, our work is more closely related to early works focused on automating the design process as much as possible. Naturally, these works use global search spaces and treat NAS as a black box combinatorial optimization problem. We too have opted to revisit global search spaces and discrete optimization problem. Therefore, closely related works in terms of search space and search strategy are those of RL [4,5,10], evolution [6,7,8,23], gradient [16] and SMBO [11,18]. Soon after the proposal of searching in modular search spaces [9], the first work that revisits global (macro) neural architecture search is that of [13]. More recent work is that of [22] as it focuses on both micro and macro architecture search but it classifies micro search as optimal operation within a modular block and macro search as different choice of blocks, however it still manually stacks up the blocks to decide the final architecture. The most closely related work that emphasizes end to end network discovery with minimal human intervention and unconstrained search is that of [23]. This work proposes automatically generated search spaces from existing architectures, an evolutionary search algorithm and uses performance approximations to speed up search. From candidates' performance evaluation perspective, NAS is lacking research on the effect of training candidates from scratch as reported in [20], therefore only very early NAS works [4,5] are known to have used complete training to evaluate candidates.

3 Methodology

Our approach addresses the main components of NAS; 1) Search Space, 2) Search Strategy, and 3) Performance Estimation. In this section, we discuss our contribution to each of these NAS components.

3.1 Search Space Design

A **search space**, or just **space** from here within, is defined as a set of network variables from which various network configurations can be sampled. Table 1 shows that majority of the existing spaces [4,5,10,11] are influenced by

Table 1. Search Space Comparison: Our search space has lesser yet the most influential network variables to choose from.

NAS Method	Global Search Space Architectural Variables							
	Depth (Layers)	Width (Channels)	Operations per Layer	Convolutional Kernel	Strides	Pooling Layers	Fully Connected Layers	Skip Connections
NAS-RL[4]	✓	✓		✓	✓	✓		✓
Meta-QNN[5]	✓	✓		✓	✓	✓	✓	
Large-scale Evolution[6]	✓	✓		✓	✓			✓
EAS[10]	✓	✓		✓	✓	✓	✓	
Genetic Programming CNN[7]		✓		✓				✓
NASH-Net[8]	✓	✓		✓				✓
NASBOT[11]	✓	✓		✓	✓	✓	✓	✓
TRG-NAS(Ours)	✓	✓	✓	✓				

the early Conv-Pool-FC like architecture paradigm [1] and/or residual networks [3]. Moreover, network depth, width and convolutional kernel size are the most common network variables followed by convolution stride (Strides), skip connections, pooling layers and fully connected layers. Since, the number of possible network configurations grow exponentially with the number of search variables and their value ranges, we aim to setup the variables such that the resulting space, when coupled with our search strategy, is combinatorially feasible to explore. However, just to make the search efficient, we cannot simply drop most of the search variables. Otherwise, the resulting space cannot posses architecturally diverse networks in terms of network performance and complexity. Therefore, we aim to strike a balance between end to end network discovery, search space explorability, and wide ranged performance/complexity trade-off. Such a space can better adapt to varying complexity tasks, by offering smaller networks for easier tasks and relatively complex networks for harder ones, hence a step closer to the original idea of NAS. Next, we discuss the optimisations done to create one such search space.

Trimming Search Variables To start with, we can drop variables arising from early Conv-Pool-FC like architectures [1] by leveraging FCN like networks [2]. Therefore, fully connected (FC) layers can be replaced by a global pooling layer, and pooling layers can be replaced by convolutions with stride 2 for reducing spatial dimensions of an image. Additionally, we can drop skip connections since we are not explicitly seeking very deep networks. Hence, we trim down **Fully connected**, **Pooling layers**, and **Skip connections** from Table 1.

Channels Search Reduction Table 1 shows that all methods search for width (number of channels). This is done for each layer as in [4]. However, we limit the search to only the initial layer and use a fixed rate of doubling the channels whenever the spatial dimensions are halved as in [1,14]. This technique further reduces the search complexity (discussed in the next section), but still allows variable width architectural diversity. Therefore, we fix stride values of convolution layers to 1 for normal layers and 2 for when spatial dimensions are halved. Hence, we further drop **Strides** from the search space.

Performance/Complexity Trade-off We notice that existing global spaces do not allow operation search whereas operation type such as separable, dilated or plain convolution can allow significant architectural diversity and expressiveness. Although, we need a compact space but it should still maintain the original idea of previously unseen architectures. Hence, we allow searching for **Operation** type as either separable or plain convolution. Operation choice coupled with kernel choice of 3, 5 or 7 creates architectural variation for suitable wide ranged accuracy/parameter-efficiency trade-off.

To this end, we propose a novel search space with ***depth***, ***width***, ***operations***, and ***kernels*** variables, as shown in Table 1. This space is diverse in terms of performance and network complexity. On CIFAR-10, out of 10 randomly sampled networks from our space, the worst network achieves an accuracy of 88.7% and the best 95.8%, as compared to the most widely adopted DARTS' space, with worst network achieving 96.18% and the best 97.56% from within 214 sampled architectures by [17]. This shows that our global search has high variance in terms of performance as compared to modular search space, hence, better discovered architectures can be attributed to the superiority of the search strategy and not to expertly crafted space.

Search Space Complexity The complexity of the space may vary significantly depending on search bounds and increases exponentially with depth. For a depth range of D, width range of W, number of operations O and number of kernels K, and final discovered depth D_f, the maximum possible number of architectures N_{arch} is given in Eq. 1.

$$N_{arch} = (O \times K)^{D_f} \times D \times W \tag{1}$$

If we limit the search depth from 4 to 15 layers, the number of channels from 16 to 64 with steps of 16, i.e., $D = 12$ and $W = 4$, $O = 2$ and $K = 3$, then assuming $D_f = 15$, the space as described above has approximately 2.25×10^{13} candidate architectures. Alternatively, if we search for channels of each layer, W will be also be raised to the power of D_{max} and the resulting space will have 6.05×10^{21} architectures. The proposed trimmed and enhanced space is still combinatorially huge but we set up the search variables such that our algorithm can efficiently navigate it and discover good architectures. With search variables explained, we can now formally define the search problem.

Search Problem Let $\mathcal{L}_{train}$ and $\mathcal{L}_{test}$ denote the training and test loss, respectively. These losses are determined by the network architecture x as well as its weights θ. The goal for architecture search is to find x^* that minimizes the test loss $\mathcal{L}_{test}(\theta^*, x^*)$, where the weights θ^* associated with the architecture are obtained by minimizing the training loss $\theta^* = \arg\min_\theta \mathcal{L}_{train}(\theta, x^*)$. This is a bi-level optimization problem with x as outer-level while θ as inner-level optimization variable:

$$\min_{x \in \mathcal{X}} \mathcal{L}_{test}(\theta^*(x), x) \tag{2}$$

$$\textbf{s.t. } \theta^*(x) = \arg\min_{\theta} \mathcal{L}_{train}(\theta, x) \tag{3}$$

where $\mathcal{X} = (D, W, O, K) \mid D \in [D_{\min}, D_{\max}], W \in W_{\min} + ne \mid n \in \mathbb{N}_0, e \in E$, $O \in o_1, o_2, K \in k_1, k_2, k_3$. D, W, O and K determine network depth, width, operation type and kernel size, respectively.

Performance Estimation Solving Equation 3 is the most expensive component of NAS, hence many works have used some form of approximation [10,8,14,21]. However, the true $\mathcal{L}_{train}$ of an architecture can only be revealed by training from scratch and till convergence, hence we train each candidate to accurately reflect its $\mathcal{L}_{train}$ and use its $\mathcal{L}_{test}$ to confidently guide the search.

3.2 Search Algorithm

We introduce our search algorithm 1 specifically tailored to efficiently navigate the search space. Details of algorithm 1 are presented below:

Macro Architecture Search Since the search complexity increases exponentially with the number of layers, we first search for network depth. With numbers of channels set to maximum, we let candidate models ***Grow*** layers in an attempt to overfit the training data. Layers are added till they keep increasing accuracy by $L^+_{acc^+}$(accuracy gain by adding layer). By increasing a layer, if the accuracy does not drop below $L^+_{acc^-}$(accuracy drop by adding layer), we continue adding layers. The depth search is terminated if either D_{max} (upper bound of layers) is reached or the accuracy drops below $L^+_{acc^-}$. Once the depth is found, we ***Prune*** the number of channels until the accuracy drops below $C^-_{acc^-}$(accuracy drop by decreasing channels). We empirically determine the threshold values for $L^+_{acc^+}$, $L^+_{acc^-}$ and $C^-_{acc^-}$ to be 0.25, 0.15 and 0.5, respectively. This strategy gives the algorithm enough flexibility to adjust to target dataset at a macro level, i.e., network depth and width. Moreover, splitting the search this way effectively reduces the right term of complexity in Eq. 1 to $D'+W'$, where D' is the number of architectures evaluated when searching for depth and W' for width. At this point, we have an architecture with D_f and W_f which are the final number of layers and channels respectively to be used in further search.

Micro Architecture Search Macro search adapts the architecture to a good performance point. We subsequently try to fine-tune it with micro search for fine grain architectural details i.e., operation type and kernel size at each layer. We simply search operations and kernel sizes for each layer. The idea is to increase learnable parameters only if it improves accuracy. To achieve this, we ***Replace*** separable convolutions with plain ones and ***Update*** kernel sizes. Therefore, we search for operations by evaluating D_f architectures and learn O_f i.e., operation type at each layer, and for kernels by evaluating $2 \times D_f$ architectures and learn

Algorithm 1: TRG-NAS Search Algorithm

```
Input: Search bounds: $D_{min}$, $D_{max}$, $W_{min}$, $W_{max}$, $W_{res}$
Initialization:
$L = D_{min}$, $C = W_{max}$, $O = Sep$, $K = 3 \times 3$
1. Grow network $L \leftarrow L + 1$ while $Acc^{test}$ improves by $L^{+}_{acc^{+}}$
2. Prune network $C \leftarrow C - W_{res}$ while $Acc^{test}$ drops no more than $C^{-}_{acc^{-}}$
3. Replace operations $O_i \leftarrow Conv$ if improves $Acc^{test}$
4. Update kernels $Ki \leftarrow [5 \times 5, 7 \times 7]$ if improves $Acc^{test}$
Return architecture $\boldsymbol{x}$ and its weights $\theta$
```

K_f i.e., kernel sizes per layer. At this point we have adapted an architecture for the target dataset by evaluating only $N_{evaluated} = 3 \times D_f + D' + W'$ architectures instead of the number shown in Eq.1.

Parameter Efficient Networks As shown in Algorithm 1, we initialize search with minimum depth, maximum width, and all layers of separable convolutions with kernel sizes of 3×3. This decision is reached by empirically evaluating alternative initialization strategies where layers can initially be convolutions or kernel sizes be 7×7, as shown in Table 2. For example, for parameter efficiency, when kernel size is initialized to 7×7, we decrease it to 5×5 and 3×3 if the accuracy is retained. Similarly, since plain convolution is less parameter efficient than separable, we replace it with separable if the accuracy is retained. To single out the contribution of each strategy and for faster evaluation, we sample 10 binary sub-datasets from CIFAR-10 instead of using the entire dataset and record averaged accuracy and number of parameters. In Table 2, we show that the best strategy is to start with smaller networks and add parameters only if there is accuracy gain. This strategy significantly beats others in terms of accuracy/parameter efficiency trade-off.

Table 2. Effect of different initialization strategies on search.

Initialization Strategy	Conv-64-3x3	Conv-64-7x7	Sep-64-3x3	Sep-64-7x7
Accuracy (%)	97.85	97.35	**97.96**	97.73
Parameters (M)	0.65	0.64	**0.23**	0.90

4 Experiments

4.1 Dataset and Search Details

We use CIFAR-10 for our experiments. It contains 10 classes with 5000 training and 1000 test images, respectively, for each class. We run search with D_{min}=10,

D_{max}=20, W_{min}=16, W_{max}=72 and W_{res}=4. We use standard training settings as in [14,23] and do not take advantage of well-engineered training protocols that hide the contributions of the search strategy or search space [17]. In order to show the true contributions of the proposed method, we follow NAS best practices as suggested by [19,17]. During search, we train all candidate models for 600 epochs using SGD with momentum of 0.9 and weight decay of 3e-4. We use an initial learning rate of 0.025 annealed down to 0 using a cosine scheduler, batch size of 64 and cutout. The search experiments are carried on a single Nvidia Quadro RTX 8000 GPU and the search cost is 4.5 GPU-days.

4.2 Results

Random Search and Relative Improvement: To show the effectiveness of our search strategy, we first compare it with 10 ***Randomly Sampled*** architectures. In Table 3, we show that our approach achieves 2.95% less error with 0.25% fewer parameters on average. This clearly singles out the contribution of our algorithm. Further, we use the ***Relative Improvement*** metric (RI) introduced by [17], which is $RI = 100 \times (Acc_m - Acc_r)/Acc_r$, where Acc_m and Acc_r represent the accuracy of search method and average accuracy of randomly sampled architecture, respectively. According to [17], a good search strategy should achieve an $RI > 0$ across different runs. Our method consistently achieves an $RI > 2$ across 5 different search runs.

Comparison with state-of-the-art: Although our work is more closely related to discrete and global NAS methods, for the sake of completeness, we compare against continuous and modular strategies too, as shown in Table 3. Our approach achieves a 4% error rate with a small, 0.45M parameters model in just 4.5 GPU-days. Given that the network discovery is end to end, and the discovered architecture does not need further human intervention, the error rate is competitive with both global and modular search methods. Further, our model size is equal to that of DPP-Net [12] (0.45M), which is the smallest NAS discovered model for CIFAR-10, but we achieve 1.84% better accuracy. Overall, our approach offers a balanced trade-off of automatic network design, high accuracy, low model complexity and practical search cost.

Ablation Studies: To study the effect of operations in search space, we run search with and without operation variable. We use 10 different seeds for each scenario and train candidates for 20 epochs for faster search. Searching with operations, on average, yields 0.79% higher mean accuracy than searching without operations i.e. 89.92% and 89.13%, respectively. This behaviour is expected, since plain convolution increases learnable parameters as compared to separable. When searched for 600 epochs, the resulting best model without operations achieves an accuracy of 95.82% as compared to 96% with operations.

Table 3. Comparison with state-of-the-art NAS discovered architectures for CIFAR-10 dataset.

NAS Method	Test Err. (%)	Params (M)	Search Cost (GPU-days)	Search Space	Search Algorithm
NAS-RL[4]	3.65	37.4	22400	Global	RL
Meta-QNN[5]	6.92	11.2	100	Global	RL
EAS[10]	4.23	23.4	10	Global	RL
Large-scale Evolution[6]	5.40	5.4	2600	Global	EA
Genetic Programming CNN[7]	5.98	1.7	14.9	Global	EA
NASH-Net[8]	5.20	19.7	1	Global	EA
Macro-NAS[23]	4.23	6.7	1.03	Global	EA
RandGrow[13]	3.38	3.1	6	Global	RS
Petridish[16]	2.83	2.2	5	Global	Gradient
NASBOT[11]	8.69	N/A	1.7	Global	SMBO
NSGA-NET[18]	3.85	3.3	8	Global	SMBO
NASNet-A[9]	2.65	3.3	2000	Modular	RL
pEvoNAS-C10A[24]	2.48	3.6	1.20	Modular	EA
DPP-Net[12]	5.84	0.45	2	Modular	SMBO
DARTS[14]	2.76	3.3	4	Modular	Gradient
GDAS[15]	2.82	2.5	0.17	Modular	Gradient
AGNAS[22]	2.46	3.6	0.4	Modular	Gradient
Random (Ours)	6.95±2.18	0.77±0.70	-	Global	-
TRG-NAS (Ours)	**4.00**	**0.45**	**4.5**	Global	Greedy

5 Conclusion

In contrast to the prevailing trend of modular search, which provides only partial network discovery, we revisit global NAS and demonstrate that achieving end-to-end network discovery with an affordable search cost is not only feasible but can also lead to low-complexity networks. Moreover, instead of attaining performance gains using expertly crafted modules, our search space offers a wide range of network architectures with varying performance capabilities. This not only helps singling out the contribution of the search strategy in unveiling good architectures but also allows it to potentially adapt to datasets of varying difficulty. Hence, one promising avenue for future research lies in dataset adaptive neural architecture search.

References

1. Simonyan, Karen, and Andrew Zisserman. ”Very deep convolutional networks for large-scale image recognition.” arXiv preprint arXiv:1409.1556 (2014).
2. Long, Jonathan, Evan Shelhamer, and Trevor Darrell. ”Fully convolutional networks for semantic segmentation.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
3. He, Kaiming, et al. ”Deep residual learning for image recognition.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
4. Zoph, Barret, and Quoc V. Le. ”Neural architecture search with reinforcement learning.” arXiv preprint arXiv:1611.01578 (2016).

5. Baker, Bowen, et al. "Designing neural network architectures using reinforcement learning." arXiv preprint arXiv:1611.02167 (2016).
6. Real, Esteban, et al. "Large-scale evolution of image classifiers." International Conference on Machine Learning. PMLR, 2017.
7. Suganuma, Masanori, Shinichi Shirakawa, and Tomoharu Nagao. "A genetic programming approach to designing convolutional neural network architectures." Proceedings of the genetic and evolutionary computation conference. 2017.
8. Elsken, Thomas, Jan-Hendrik Metzen, and Frank Hutter. "Simple and efficient architecture search for convolutional neural networks." arXiv preprint arXiv:1711.04528 (2017).
9. Zoph, Barret, et al. "Learning transferable architectures for scalable image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
10. Cai, Han, et al. "Efficient architecture search by network transformation." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. No. 1. 2018.
11. Kandasamy, Kirthevasan, et al. "Neural architecture search with bayesian optimisation and optimal transport." Advances in neural information processing systems 31 (2018).
12. Dong, Jin-Dong, et al. "Dpp-net: Device-aware progressive search for pareto-optimal neural architectures." Proceedings of the European Conference on Computer Vision (ECCV). 2018.
13. Hu, Hanzhang, et al. "Macro neural architecture search revisited." 2nd Workshop on Meta-Learning at NeurIPS. 2018.
14. Liu, Hanxiao, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search." arXiv preprint arXiv:1806.09055 (2018).
15. Dong, Xuanyi, and Yi Yang. "Searching for a robust neural architecture in four gpu hours." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.
16. Hu, Hanzhang, et al. "Efficient forward architecture search." Advances in Neural Information Processing Systems 32 (2019).
17. Yang, Antoine, Pedro M. Esperança, and Fabio M. Carlucci. "NAS evaluation is frustratingly hard." arXiv preprint arXiv:1912.12522 (2019).
18. Lu, Zhichao, et al. "Nsga-net: neural architecture search using multi-objective genetic algorithm." Proceedings of the genetic and evolutionary computation conference. 2019.
19. Lindauer, Marius, and Frank Hutter. "Best practices for scientific research on neural architecture search." The Journal of Machine Learning Research 21.1 (2020): 9820-9837.
20. Ren, Pengzhen, et al. "A comprehensive survey of neural architecture search: Challenges and solutions." ACM Computing Surveys (CSUR) 54.4 (2021): 1-34.
21. White, Colin, et al. "How powerful are performance predictors in neural architecture search?." Advances in Neural Information Processing Systems 34 (2021): 28454-28469.
22. Sun, Zihao, et al. "AGNAS: Attention-Guided Micro and Macro-Architecture Search." International Conference on Machine Learning. PMLR, 2022.
23. Lopes, Vasco, and Luís A. Alexandre. "Towards Less Constrained Macro-Neural Architecture Search." arXiv preprint arXiv:2203.05508 (2022).
24. Sinha, Nilotpal, and Kuan-Wen Chen. "Neural architecture search using progressive evolution." Proceedings of the Genetic and Evolutionary Computation Conference. 2022.