# Training Free Neural Architecture Search

Lorenzo Melchionna
*Politecnico di Torino*
Torino, Italia
s304651@studenti.polito.it

Niccolò Revel Garrone
*Politecnico di Torino*
Torino, Italia
s302302@studenti.polito.it

Giuseppe Suriano
*Politecnico di Torino*
Torino, Italia
s296605@studenti.polito.it

***Abstract*—One of the main challenges regarding Neural Networks is designing their structure. Hand-design needs much human resources, so NAS algorithms have been developed in order to automate this process. These algorithms are very slow, expecting a complete training for every observed model. To speed up this procedure, there are some fast-to-compute metrics which can replace training to evaluate networks. In this work we analyze 2 of this metrics (NASWOT and Synflow) and we test different search approaches in order to find in some minutes the best network configuration for image classification of 3 different dataset throw the Nats-Bench TSS. Our final solution is very good to find the best Neural Network configuration within our search space.**

***Index Terms*—Neural Architectures Search Without Training, Deep Learning, Benchmark, Search Algorithm**

## I. INTRODUCTION

The discovery and use of machine learning and deep learning represents a crucial turning point in our development as a society and people. Standard approaches to the use of deep neural networks involve trial-and-error procedures in order to design the model structure. However, these processes are time-consuming and above all do not guarantee the optimality of the solution.
One approach to solve this problem is to implement a Neural Architectures Search (NAS) algorithm that identifies the best architecture in terms of performance within a search space of available model structure.
NAS algorithms are broadly based on the work of (Zoph and Le, 2016). An LSTM network, used as a controller and trained via Reinforcement Learning, samples from the policy distribution a model structure, then the sampled model is trained until convergence and its accuracy is used as reward function to train the controller's parameters. Training a network every time the controller is updated is time-consuming, in fact in the work of (Zoph and Le, 2016), indeed it needs 800 GPUs for 28 days.
To overcome this problem, a few solutions were applied: learning stackable cells instead of whole networks, and sharing weights between all networks within the search space Pham *et al.* (2018). These solutions, even though they have considerably decreased research times, reaching the use of a single gpu for half a day, can still be too time-consuming and expensive for some problems.

A solution that can drastically reduce computing time is to use NAS without any network training (Mellor *et al.*, 2021). The aim of this solution is to develop a metric that is able to evaluate a network without applying a training process. This metric is developed starting from a simple observation: given an untrained neural network and 2 different inputs, the difference between the binary codification of the ReLU's output of each input is very representative of how the network could be able to distinguish them.

### A. Search space

In our experiment we explore NATS-bench (Dong *et al.*, 2021).
NATS-Bench comprises two search spaces: a topology search space (NATS-Bench TSS) which contains 15,625 networks, and a size search space (NATS-Bench SSS) which contains 32,768 networks.
Both search spaces in the first layers have a 3x3 convolution with 16 output channels and a batch normalization layer. The central body of the skeleton includes 3 blocks connected by residual blocks. All cells in an architecture have the same topology. The operations in the reduction cells are a 2x2 average pooling with a stride of 2 in order to reduce the input size and a 1x1 convolution. The skeleton concludes with a global average pooling.
In the topology search space, each block consists of 5 cells. This skeleton is the same in the size search space with the difference that instead of changing the cell topology, the channel layers size is changed, considering 8 different candidates.
An important difference between the two search spaces are isomorphisms. Isomorphisms are not present in the size search space since the topology of the cell is fixed, whereas in the topology search space by changing the topology of the cell we obtain networks with different structure but with equal input we obtain equal output. In the topology search space the unique networks are 6466 while the remaining 9158 represent isomorphisms. Figure 1 represents an isomorphism where the common edges between different cells are highlighted.

Since all cells in an architecture have the same topology, an architecture candidate in the Topology Search Space corresponds to a different cell, which is represented as a densely connected DAG of 4 ordered nodes (A,B,C,D) where node A is the input node and node D is the output. In this graph, there is an edge connecting each node to all subsequent

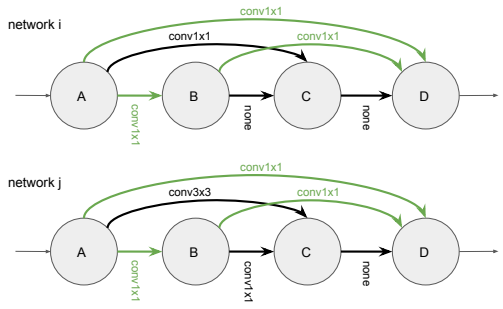


Fig. 1. An example of an isomorphism: network i identifies network 579, network j identifies network 1599

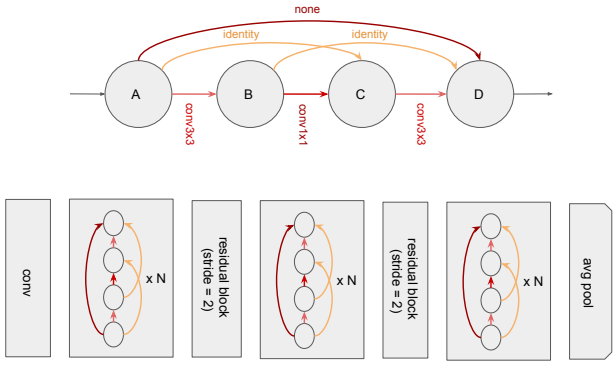


Fig. 2. **Top:** example of a NATS-Bench TSS cell. **Bottom :** the skeleton for NATS-Bench TSS where N = 5

nodes for a total of 6 edges, and each node can perform one of 5 possible operation (none, identity, 3x3 convolution, 1x1 convolution and 3x3 average pool).
Figure 2 shows an example of a cell and the NATS-Bench skeleton.

### *B. Datasets*

In this project we use CIFAR-10, CIFAR-100 (Krizhevsky *et al.*, 2009) and ImageNet-16-120 (Chrabaszcz *et al.*, 2017).

- **CIFAR-10** It is a standard image classification dataset and consists of 60K 32×32 colour images equally divided in 10 classes.
- **CIFAR-100** This dataset is just like CIFAR-10. It has the same images as CIFAR-10 but categorizes each image into 100 fine-grained classes.
- **ImageNet-16-120** is built from the down-sampled variant of ImageNet (ImageNet16×16). ImageNet-16-120 contains 151.7K training images, 3K validation images, and 3K test images with 120 classes.

## II. NASWOT

### *A. NASWOT score*

Our implementation : here
We are going now to explain in details how the NASWOT score is computed. The main idea is to quantify how a neural network is able to distinguish elements over the dataset.
In order to compute the score over a certain dataset, a random batch of elements is selected. Each element of the batch is mapped to a vector containing the binary codification of ReLU's output obtained performing a forward. In this way we obtain a $(batch\_size$ x $col\_dim)$ tensor, where col_dim depends on the block (between 2 blocks there is a reduction cell that reduces the input size) in which we are doing the computation and on the dimension of input image (cifar10 and cifar100 are composed by 32x32 images whereas Imagenet16-120 by 16x16 ones).
For each couple of binary codes, we consider the Hamming distance $d_H(E_i, E_j)$ to measure how dissimilar the 2 batch elements are for the network. In order to have a global view on our batch of correspondence between binary codes, we compute the kernel matrix $K_H(batch\_size$ x $batch\_size)$ where each element $(i, j)$ is $col\_dim - d_H(E_i, E_j)$ , which represents how similar the elements are for the network. In each ReLU instance of the model the process now explained is performed via forward hook.
The kernel matrix of the whole network is calculated by summing every kernel matrix generated in each forward hook.
The metric of interest is calculated as the determinant's logarithm of the kernel matrix.

We implemented the calculation of the K matrix, in order to minimize computation time, using Numba library. Numba supports the execution of a resticted subset of python functions (among which xor) into CUDA GPUs.
More specifcally, the improvements of using CUDA is that we are able to parallelize the construction of the k matrix through divinding the computation over different threads. On each thread a function is executed to calculate the $d_H$ between a couple of vectors $(v_i, v_j)$, so each thread compute the element $(i, j)$ of the Hamming distances matrix.
In this way is very fast to construct the Hammming distances matrix, that is the most computationally expensive part in computing NASWOT score.
Some considerations must be done about the way the score is computed. The batch size is not important, in fact as studied and reported by NASWOT (Mellor *et al.*, 2021), the metric grow proportionally as the batch size increases. Dividing the score by the minimum score obtained in the same batch size, we can observe that the metrics distinguish networks in the same way.
For the calculation time with respect to each batch size, Table I shows the average times and variance. In our project, we chose a batch size of 128 as it makes the calculation times relatively low.

The number of ReLUs within the search space can vary as shown in Figure 3. The number of ReLUs within an architecture changes depending on the network operations:

TABLE I
CALCULATION TIME FOR BATCH SIZE OVER 10 RUNS

| Size | Mean | Std |
|---|---|---|
| 8 | 0.156775 | 0.024288 |
| 16 | 0.151437 | 0.002760 |
| 128 | 0.181524 | 0.004438 |
| 256 | 0.258984 | 0.007100 |
| 512 | 0.462246 | 0.005050 |

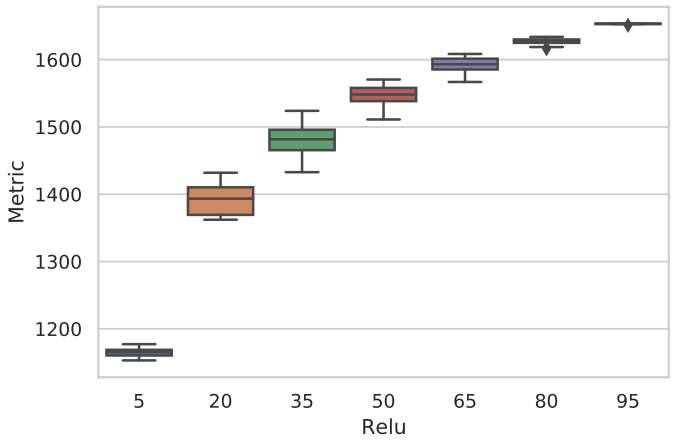


Fig. 3. Boxplot on the number of relu and metric value

skip and none operations contain no ReLUs, convolution and average pooling contain 15 ReLUs. The minimum number of ReLUs in a network is 5, which is the number of ReLU of the basic skeleton of NATS-Bench.
Figure 3 shows that the largest metrics are associated with the largest number of ReLUs.

### B. NASWOT algorithm (Random Search)

Our implementation : here
We are going to explain in details how the NASWOT algorithm works.
NASWOT randomly proposes a candidate within the search space for which the metric explained in the previous step is calculated. This operation is repeated N times and finally the network with the highest associated metric is chosen. At this point, the chosen model is trained to 200 epochs to observe the final accuracy.
Alternatively, we want to focus on another possible approach to find the best network within our search space, in order to compare performance and computation time with NASWOT algorithm. This approach consists in randomly sampling N network from the search space, and perform a 'short' training (12 epochs instead of 200) for each of them. After that the network with the highest test accuracy is chosen and trained to 200 epochs in order to observe the final accuracy .

Table II and Table III show the results of these two algorithms. For each N, the experiments are repeated 10 times to get better and more complete statistics. The results of the NASWOT algorithm show that by increasing N, the metric becomes larger.
However, it can be seen that there is no full correlation between the value of the metric and the final value of accuracy. In fact, by setting N=300, the metric becomes larger but the accuracy is smaller than the one calculated for N=200.
Table III shows the result of the alternative algorithm, in which we can see that there is a linear correlation between the accuracy results at epoch 12 and the final accuracy result at 200 epochs. In this case, by increasing the value of N, both the accuracy at 12 epochs and the accuracy at 200 epochs increase.
The calculation time of the accuracy at epoch 12 is high and unsustainable; therefore NASWOT algorithm is a better solution to achieve good performance and a low calculation time.

TABLE II
RANDOM SEARCH WITH NASWOT METRIC

| Dataset | N | Metric | | Accuracy 200 | | Time | |
|---|---|---|---|---|---|---|---|
| | | mean | std | mean | std | mean | std |
| *Cifar-10* | 50 | 1638.06 | 19.57 | 92.99 | 0.57 | 12.33 | 1.85 |
| | 100 | 1634.43 | 13.64 | 92.55 | 0.62 | 24.32 | 1.38 |
| | 200 | 1640.82 | 12.15 | 92.61 | 1.27 | 48.93 | 1.68 |
| | 300 | 1642.04 | 10.65 | 92.44 | 1.31 | 76.12 | 9.64 |
| *Cifar-100* | 50 | 1625.38 | 9.30 | 68.63 | 2.04 | 11.93 | 0.79 |
| | 100 | 1635.94 | 14.21 | 69.76 | 3.22 | 23.67 | 0.76 |
| | 200 | 1647.21 | 10.29 | 70.13 | 0.63 | 48.73 | 1.52 |
| | 300 | 1643.15 | 11.93 | 69.73 | 1.32 | 87.46 | 44.32 |
| *Imagenet-16-120* | 50 | 1448.23 | 18.60 | 41.28 | 5.75 | 15.79 | 1.70 |
| | 100 | 1455.41 | 14.04 | 42.32 | 5.45 | 30.12 | 0.71 |
| | 200 | 1466.28 | 8.47 | 44.64 | 1.42 | 64.18 | 7.96 |
| | 300 | 1465.67 | 9.93 | 43.63 | 2.31 | 88.80 | 1.41 |

TABLE III
RANDOM SEARCH WITH SHORT TRAINING

| Dataset | N | Accuracy 12 | | Accuracy 200 | | Time | |
|---|---|---|---|---|---|---|---|
| | | mean | std | mean | std | mean | std |
| *Cifar-10* | 50 | 87.65 | 0.91 | 93.47 | 0.54 | 10958.02 | 381.69 |
| | 100 | 88.22 | 0.54 | 93.81 | 0.31 | 21749.58 | 802.14 |
| | 200 | 88.51 | 0.32 | 93.91 | 0.17 | 44022.88 | 816.23 |
| | 300 | 88.70 | 0.27 | 94.02 | 0.18 | 65997.06 | 1125.67 |
| *Cifar-100* | 50 | 58.37 | 1.85 | 71.11 | 1.26 | 11113.52 | 462.33 |
| | 100 | 58.97 | 1.14 | 71.19 | 1.38 | 21849.75 | 623.26 |
| | 200 | 60.06 | 0.65 | 72.04 | 0.96 | 43893.63 | 511.68 |
| | 300 | 60.06 | 0.59 | 72.10 | 1.01 | 66105.22 | 946.97 |
| *Imagenet-16-120* | 50 | 34.42 | 1.58 | 44.11 | 1.34 | 33508.05 | 1060.23 |
| | 100 | 35.69 | 0.99 | 45.05 | 0.63 | 67756.18 | 1517.93 |
| | 200 | 35.89 | 1.25 | 45.11 | 1.03 | 133124.47 | 1697.19 |
| | 300 | 36.21 | 0.69 | 45.71 | 0.92 | 200368.13 | 2528.65 |

## III. METHODS

Our implementation : here

### A. Synflow

In this section we are going to explain another metric that we implemented and the search algorithm we adopted.
As reported by Gebhart *et al.* (2021), a neural network can be approximated to a linear function that map the inputs to the outputs. The Neural Tangent Kernel (NKT) is defined as the

Jacobian of the linear function respect to the parameters, and it can be decomposed, as we can see in Eq. 1, in 2 parts: the first one is the derivative of the function respect to the path values and the second one is the derivative of the path values respect to the parameters. The path values are the product of all weights within each path, as shown in Eq 2.
We can observe that the first part of the matrix of Eq 1 depends on the input, while the second one does not.

$$\nabla_\theta f(x,\theta) = \frac{df(x,\theta)}{d\theta} = \frac{df(x,\theta)}{dv(\theta)} \cdot \frac{dv(\theta)}{d\theta} = J_v^f(x) J_\theta^\nu \quad (1)$$

$$v_p(\theta) = \prod_{j=1}^{m} \theta_j^{p_j} \quad (2)$$

The Path Kernel is defined as follow:

$$\Pi_\theta = J_\theta^v J_\theta^{vT} \quad (3)$$

The authors explained that the trace of the Path Kernel could be approximated considering the derivative of a particular loss function. Synflow Gebhart *et al.* (2021), $l_1$ approximates the loss function, already mentioned, with $R_{SF}$ as the sum of the output vector when a all 1 data point is fed through the network with all absolute weights. Eq 4:

$$R_{SF}\left(\vec{1},\theta\right) = \vec{1}^T \left(\prod_{l=1}^{L} |\theta_l|\right) \vec{1} \quad (4)$$

Consequentially the trace of the path kernel is approximated summing the product of the derivative of $R_{SF}$ respect to the parameters weights and the value of the weights. Eq 5:

$$S_{SF}(\theta) = \frac{dR_{SF}\left(\vec{1},\theta\right)}{d\theta} \odot \theta \quad (5)$$

In our solution we calculate Synflow using the implementation of Abdelfattah *et al.* (2021) available at: https://github.com/SamsungLabs/zero-cost-nas.

### *B. REA algorithm*

Having to manage an exploration on a search space, a typical approach to this kind of problem is to consider searching algorithms.
In particular we focused on evolutionary algorithms that seems to fit well on our scope due to the common structure of the networks composing the search space.
We are going to introduce Regularized Evolution for Image Classifier Architecture Search (REA) Real *et al.* (2019).
The algorithm is based on two sets: the history containing all the architectures observed, and the population which contains the architectures that are candidates for the generation of a child at each cycle. In the algorithm, we have three fixed parameters:

- P representing the initial number of the population
- C representing the number of cycles and the number of architectures in the history at the end of the generation
- S representing the size of the random sample of the population from which the candidate to become the father is chosen on a particular scoring criterion.

The algorithm initialises the population with P architectures taken randomly from the search space.
For every architecture in P the score is calculated in order to evaluate it.
Subsequently, at each cycle, the algorithm selects a sample of size S randomly from the population and selects the model with the highest metric as a parent.
A new architecture is generated by the parent through mutation and it is called child.
In general, a mutation is defined as a random variation of an operation within the architecture.
The generated child is evaluated and added to the population and history. The oldest model (the one that has been in the population for several cycles) is discarded from the population.
This process is repeated until history reaches the size C and then the model highest metric in the history is selected as the best architecture.
This approach in which the model evaluated further back in time within the population is discarded is called *aging evolution*. This approach allows more exploration of the search space instead of focusing too early on the models with better metric in the population.

In addition, we explored different ways of generating children with the aim of exploring better the search space and have better performance.
In child generation via mutation, a randomly operation is chosen between two nodes and replaced with an operation randomly chosen from the 5 available.
Another generation method we have tried is to generate a child by vote. Instead of choosing only the network with the highest metric as the parent, we use all the networks within the sample. The new architecture is generated edge by edge, considering the 5 different operations available, picking the operation that has the maximum average metric in that particular edge taking into account the sampled networks.
Another approach to generating children is the inversion approach: taking the network with the highest metric in the sample as parent, the operations of the network are mapped into a vector and a subset, picked by two random indexes, is inverted.
The last approach we analysed for the generation of children is crossover: in this case, the two networks with the largest metric are taken from the sample as parents, and the child is generated by combining the operations in the parents.

After this work we thought about typical structure of the networks for this task and because of skip connections allow a better convergence and there are a lot of empirical proves that the phenomenon of vanish gradient is attenuated, in our experiment at each child generation we discard the

networks without skip connection. This consideration allows us to reduce the size of the search space and focus more on the architectures that perform better.

## IV. RESULTS

### Comparison between the 2 metrics:

The two proposed metrics are NASWOT and Synflow. NASWOT metric is a good starting point, being very fast to calculate but the correlation between the metrics obtained and the final accuracy is not very good. The network with the highest metrics is not always the one with the highest final accuracy.
As we showed earlier NASWOT is very dependent on the number of ReLU characterising the network while synflow is more related to the number of parameters of a network. In this search space, a high accuracy is characterised more by a large number of parameters than a large number of ReLU.

Table II, shown in section 2, and table IV show the results obtained using the random search algorithm and the two different metrics.
By setting N, the number of networks in the history, equal to 200, it can be seen that on average, the results of synflow are better than those of NASWOT metric.
Comparing the results obtained in the different datasets, in Cifar-10 and Cifar-100 Synflow gives higher results while in Imagenet-16-120 NASWOT gives slightly better results. Considering the randomness of the algorithm used, we can conclude that on average Synflow performs better than NASWOT metric.
Table V and table VI shows that when comparing the two metrics and using REA as the search algorithm, Synflow provides significantly better results than NASWOT metric.

TABLE IV
RANDOM SEARCH WITH SYNFLOW OVER 10 RUNS

| Dataset | Synflow | | Accuracy | | Time | |
|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std |
| *Cifar-10* | 120.16 | 10.44 | 93.39 | 0.39 | 53.50 | 11.89 |
| *Cifar-100* | 127.79 | 9.28 | 70.19 | 1.38 | 90.44 | 10.12 |
| *Imagenet-16-120* | 125.27 | 6.31 | 44.26 | 2.09 | 68.99 | 10.62 |

### REA instead of random search algoritm:

The two search algorithms we have explained are random search and REA.
Table IV and Table VI compare the two search algorithms using the same metrics.

TABLE V
REA WITH NASWOT METRIC OVER 10 RUNS

| Dataset | Metric | | Accuracy | | Time | |
|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std |
| *Cifar-10* | 1633.33 | 0.85 | 93.02 | 0.45 | 89.65 | 0.93 |
| *Cifar-100* | 1632.24 | 2.48 | 69.40 | 1.98 | 88.21 | 2.28 |
| *Imagenet-16-120* | 1451.99 | 1.21 | 43.20 | 3.11 | 78.06 | 0.92 |

It can be seen that the results obtained by REA over-perform the results produced by Random Search by achieving a higher average accuracy on all three datasets.
REA represents an evolutionary algorithm and allows the search space to be explored by directing the search towards architectures with higher metrics.
REA therefore represents a good solution in terms of performance, achieving networks with good final metrics, and with good time performance, remaining under 100 seconds evaluating 200 networks with the GPU available on standard Google Colab.
Random search chooses networks within the search space randomly and is therefore less precise than REA that through a focused search towards networks with a higher metric reaches higher accuracy.

TABLE VI
REA WITH SYNFLOW OVER 10 RUNS

| Dataset | Synflow | | Accuracy | | Time | |
|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std |
| *Cifar-10* | 133.89 | 0.34 | 94.19 | 0.35 | 74.72 | 8.71 |
| *Cifar-100* | 135.80 | 0.31 | 73.43 | 0.16 | 76.88 | 9.96 |
| *Imagenet-16-120* | 131.79 | 0.40 | 46.34 | 0.01 | 93.24 | 8.63 |

### Comparison between child generation methods:

In order to understand which method of generating children is best, we created for each dataset a history in which we produce children with all the types explained above.
To visualise which method has the most impact on the final value of the synflow, we use a boxplot. In the graph, only the last 80 models generated in a history of 300 for each type are shown because initially the search space is explored randomly and towards the end a more specific search is made, generating models that are oriented for an higher synflow. The latest generated models usually representing the most significant ones, it can be seen that using Cifar-100 as an example, figure 4 shows how children generated by mutation

obtain an higher average synflow than children obtained by other types.

Within the proposed child generation methods, mutation is the best solution for generating architectures. The other proposed generation methods produce networks that are never selected as optimal solutions at the end of the search, obtaining networks with lower Synflow than those obtained by mutation.

Mutation is therefore the best way to generate children and so we used this approach in our final solution

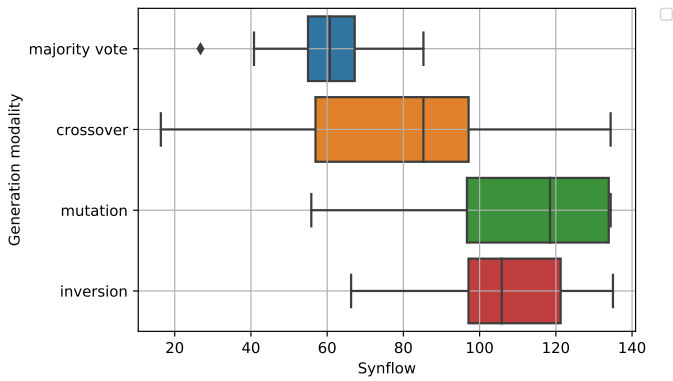


Fig. 4. The figure shows how the synflow varies with respect to different types of child generation

## V. Conclusions

Given the previous observation on the results of search algorithms and metrics, we conclude that the best solution for our search space is performing a REA search algorithm, where the children are generated only by mutation, and Synflow as score for evaluation.

Table 4 on Dong *et al.* (2021) shows the results obtained by different algorithms and the maximum accuracy for each Dataset within the search space.

On average, the accuracy obtained on table VI is lower then the maximum accuracy available in search space for all three DataSet.

We can observe that for cifar-10 and cifar-100 among the 10 runs used to compute the average accuracy, we find 8 times the maximum accuracy. For Imagenet-16-120 we obtain an higher accuracy then the other search algorithms. Observing that the network with maximum accuracy is never picked, we think that this is due to lower definition (16x16) and higher number of classes (120) that characterises Imagenet-16-120.

In conclusion we performed test only on NATS-Bench that is a small Search Space (due to the size of the cells), considering metrics like NASWOT and Synflow that are linearly dependent to the complexity of the networks (respectively to the number of ReLU and number of parameters).

In this Search Space the most complex networks perform better then the other ones but this could be not true in bigger Search Spaces where high networks complexity might not be the solution for the task.

## References


B. Zoph and Q. V. Le, *Neural architecture search with reinforcement learning*, 2016.

H. Pham, M. Guan, B. Zoph, Q. Le and J. Dean, Efficient neural architecture search via parameters sharing, 2018.

J. Mellor, J. Turner, A. Storkey and E. J. Crowley, Neural architecture search without training, 2021.

X. Dong, L. Liu, K. Musial and B. Gabrys, *Nats-bench: Benchmarking nas algorithms for architecture topology and size*, 2021.

A. Krizhevsky, G. Hinton *et al.*, *Learning multiple layers of features from tiny images*, 2009.

P. Chrabaszcz, I. Loshchilov and F. Hutter, *A downsampled variant of imagenet as an alternative to the cifar datasets*, 2017.

T. Gebhart, U. Saxena and P. Schrater, *A unified paths perspective for pruning at initialization*, 2021.

M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak and N. D. Lane, *Zero-cost proxies for lightweight NAS*, 2021.

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