Lightweight NAS: A Comparative Empirical Study of Training-Free Metrics

Michele Gallina
Politecnico di Torino
S305936

Alberto Morcavallo
Politecnico di Torino
S290086

Nicola Orecchini
Politecnico di Torino
S303998

***Abstract*—The aim of NAS is to find optimal neural network architectures within a defined Search Space in an automated way. Search Spaces can be very large, and training every single architecture to find the best can be very expensive, both in terms of time and resources. This is what drives the research to find some proxy that could give an estimate of the performance of a network without training it, as well as to find some algorithm that could help explore the Search Space more efficiently. In our paper, we implement three training-free metrics (*ReLU*, *Synflow* and *Grad_Norm*) and two algorithms (*Random Search* and *Regularized Evolutionary Aging*), and perform an empirical study with every (metric, algorithm) combination on the NATS-Bench Topological Search Space, to understand whether these metrics can be used as surrogates of network's training and also to find possible synergies between a certain metric and a certain algorithm. Code for reproducing our experiments can be found: here.**

I. Introduction

The design of Artificial Neural Networks is a task that originally involved manual work. Neural Architecture Search (NAS) is a recent set of methodologies for automating this process. It constitutes an interesting topic for which research is being carried out. Indeed, at the time of this writing, multiple architectures designed with NAS are on par or outperformed hand-designed architectures [1] [2].

The NAS framework is made up of three elements:

1) a *Search Space* containing all possible architectures we want to examine;
2) a *Search Strategy*, used to explore the Search Space to find the optimal network architecture, i.e., the one that gives the highest validation accuracy. The baseline Search Strategy is the Random Search, which just samples architectures randomly;
3) a *Performance Estimation Strategy*, the criterion to estimate the performance of each network during the Search Space exploration. Indeed, we would like not to train every single architecture during the NAS (to save time and computational resources), but rather have an inferred estimation of their final performance, and finally only train the one best architecture found by NAS. This can be accomplished in various ways, for example training the networks only for a few epochs (e.g. 12 epochs, as in [3]), or training a smaller version of the network [4].

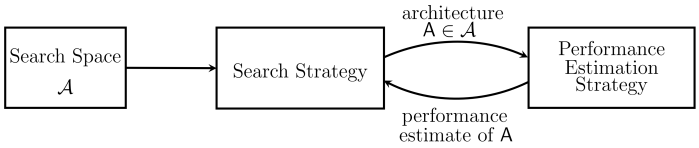

Fig. 1. The NAS framework (Image source: Elsken et al., 2019 [5]).

In our work, we would like firstly to do the exercise of implementing three performance estimation strategies:

- *ReLU Score* (this metric has not a specific name, so we refer to it in this way), from [3]
- *Synflow*, from [6]
- *Grad_Norm*, from [6]

and two search strategies:

- Random Search (the baseline algorithm)
- Regularized Evolutionary Aging Algorithm, from [3].

Then, we score the networks contained in the NATS-Bench Topological Search Space, which consists of 15.625 different architectures, each one trained on three different vision datasets: CIFAR10, CIFAR100 and, ImageNet16-120.

After that, we obtain three datasets (one for each Vision dataset) with all the three metrics (plus the validation accuracy after 12 epochs of training as a benchmark) for each network in the Search Space, storing the time required to compute each of the metrics too. At this point, we would like to perform some experiments on these datasets. In particular, questions we hope to answer, even if empirically, with our experiments, are the following.

- What are the limitations of a standard NAS pipeline (e.g. using validation accuracy as the metric and random search as the search strategy)?
- Are the best architectures found by a NAS algorithm that uses a training-free metric comparable to that of a NAS algorithm that trains every architecture in terms of final accuracy?
- Are there metrics that perform better than the others in all three vision datasets?

- Is there a specific (metric, algorithm) combination that performs better than others in all three vision datasets?

Our contribution is then represented by an empirical study on NATS-Bench, experimenting with every possible (metric, algorithm) combination obtainable from the three metrics and the two algorithms we chose to implement.

II. Related Work

In this section, after having given some background on NAS, we present the theory about metrics and algorithms we chose to implement. This is because these concepts are not our contributions, so we find it more correct to present them here while leaving the *Methods* section to explain our process.

First proposal of NAS Zoph & Le [1] made a pioneering work, in 2016, using a Recurrent Neural Network controller to generate candidate networks, which are trained and used to update the controller using Reinforcement Learning. This Search Strategy requires high computational cost, both in hardware resources and time: 800 GPUs and 28 days to search an architecture to classify CIFAR-10 dataset.

NasNET Search Space Zoph & al., in *Learning Transferable Architectures for Scalable Image Recognition* (2017) [2], proposed a new Search Space made of cells, based on the design of successful vision architectures: the *NasNET Search Space*, which describes every architecture as a series of cells.

NATS-Bench In *Benchmarking NAS Algorithms for Architecture Topology and Size* (2020) [7], the authors introduce a variation of the previous cell-based Search Space, the *NATS-Bench*, useful for unified benchmarking of different Search Strategies and Performance Estimation Strategies. Here, two possible Search Spaces are defined, based on the Search Strategy that one wants to adopt:

- the *Topological Search Space*, for searching architectures via the set of operations chosen to link different nodes (Zeroize, Skip Connection, Conv, Avg Pool);
- the *Size Search Space*, for searching the networks via the number of feature maps for each Convolutional Layer.

Evolutionary Aging NAS Real et al. [4], in 2018, introduced a new Search Strategy, namely an Evolutionary Algorithm with the introduction of an *age* component. This methodology can be summarized as taking a sample population of architectures, training all of them, and then the best performing architecture is selected as the parent. Now, the parent is mutated, via a specific set of mutations, to form a child architecture (the cell-based Search Space is useful for designing mutations). Then, the oldest architecture in the population is discarded, while the child is inserted into the population. This process, called *tournament selection*, continues until a certain number of generations is reached.
This method certainly has minor complexity and requires less computational power, compared to Reinforcement Learning. However, it still needs the training of the entire population.
Evolutionary Aging NAS lead to the finding of a neural network called AmoebaNet-A that had a validation accuracy comparable to state-of-the-art ImageNet models of the time.

NASWOT and AREA Some practitioners still considered NAS as a too slow process: Mellor et al., in *Neural Architecture Search without Training (2020)* [3], proposed a way to perform NAS without any network training at all. They accomplish this by defining a metric to score every architecture only from its initial design.
This metric is represented by a binary vector indicating which ReLU nodes each image did activate. The heuristic idea behind this metric is that if the network is well-designed, different inputs will activate different linear units, even before training the network. On the contrary, if different inputs activate a similar set of linear units, it will be more difficult for the network to learn to disentangle them. To measure dissimilarity between two different inputs, the Hamming distance between two binary codes is computed.
Indicating by $\boldsymbol{c}_i$ the characteristic binary code representing the activation of the ReLU units of the network by image i ($\boldsymbol{c}_i$ contains 1s where the unit has been activated and 0s anywhere else), it is possible to generate a similarity Matrix K_H computing the Hamming distances $d_H(\boldsymbol{c}_i, \boldsymbol{c}_j)$ between pair of images inside a mini-batch of images:

$$K_H = \begin{bmatrix} N_A - d_H(\boldsymbol{c}_1, \boldsymbol{c}_1) & \dots & N_A - d_H(\boldsymbol{c}_1, \boldsymbol{c}_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(\boldsymbol{c}_N, \boldsymbol{c}_1) & \dots & N_A - d_H(\boldsymbol{c}_N, \boldsymbol{c}_N) \end{bmatrix} \quad (1)$$

where N_A is a normalization addend equal to the total number of ReLU of the network (thus equal to the length of a binary code), and N is the size of the mini-batch.
The metric that the authors propose to use to score the network is the logarithm of the determinant of this matrix, so:

$$ReLU score : s = log|K_H| \quad (2)$$

If the network has good disentangling properties, K_H is closest to a diagonal matrix, and s is high.
A higher score s (computed before training) implies higher validation accuracy after training. So, this metric can be used as a Performance Evaluation Strategy, instead of training the network. The authors embody this metric in Random Search and Assisted Regularized Evolutionary Aging search strategies, and call the resulting algorithms respectively *NASWOT* and *AREA*, finding optimal architectures in a matter of seconds on a single GPU.

Zero-cost metrics The final piece we need before starting with our implementations is represented by *Zero-Cost Proxies for Lightweight NAS* (2021) [6] by Abdelfattah et al. Here, the authors introduce multiple metrics to score networks without training, and they perform experiments on NAS benchmarks. We chose to implement two of the metrics they introduce: *Synflow* and *Grad_Norm*, so a brief discussion of the theory behind these two scores follows.

Synflow, introduced by Tanaka et al. (2020) [8], is a so-called *synaptic saliency* score, that is a score specific for each unit of

the network expressing an approximation of the change in loss when a specific parameter is removed. These scores are used to prune a network by removing the less "significant" units, thus reducing its size and consequently its training time. A phenomenon that occur during parameter pruning is that a full layer can collapse, making the network untrainable. Synflow manages to get around this phenomenon, being also *"data agnostic"*: it just needs a vector of ones as the input.
Here is the expression of Synflow for a parameter p of a neural network:

$$Synflow : S_p(\theta) = \frac{\partial L}{\partial \theta} \odot \theta \tag{3}$$

where $\odot$ is the Hadamard product, and L the loss function of the neural network with parameters θ. To score the entire network, we simply sum over all parameters i in the model:

$$S_{net} = \sum_{i \in params} S_p(\theta)_i. \tag{4}$$

Finally, Grad_Norm, is simply the sum of the Euclidean norm of the gradients after a single minibatch of training data.

III. Methods

In this section, we illustrate the work we have done in preparation of the experiments, following a chronological order.

1) First, we implement the metrics.
2) Then, we used them to score the architectures in the NATS-Bench dataset, creating a dataset for our experiments.
3) At this point, we implemented the two search algorithms.

A. Metrics

Having already presented the theory behind the metrics in the *Related Work* section (II), and leaving the details of the implementation on the Github code, here we concentrate on giving some diagnostic of our metrics.

To assess the performance of the metrics, we report in Table I the Spearman's rank correlation coefficient between each metric and the test accuracy of the network after 200 epochs of training.

Dataset	val_acc	ReLU	synflow	grad norm
CIFAR-10	0.81	0.78	0.74	0.59
CIFAR-100	0.82	0.81	0.76	0.64
ImageNet16-120	0.82	0.79	0.75	0.58

TABLE I
SPEARMAN'S RANK CORRELATION COEFFICIENT BETWEEN THE METRIC (VALIDATION AFTER 12 EPOCHS OF TRAINING, RELU SCORE, SYNFLOW AND, GRAD_NORM) AND THE TEST ACCURACY OF THE NETWORK AFTER 200 EPOCHS OF TRAINING, FOR EACH DATASET.

Coherent values for *ReLU*, *synflow* and *grad norm* with ones obtained in [3] and [6] are good indicators of our metrics implementation, thus, we can further proceed in creating our dataset.

B. Creating our dataset

What we accomplish in this step is to create a data set with the scored architectures, both to guarantee the reproducibility of our experiments and to save time. The dataset has the following structure (only one row and three columns reported):

arch spec	metric value	time to compute metric
[1, 2, 3, 2, 0, 1]	1500	2.8s

TABLE II
STRUCTURE OF THE DATA SET WE CREATE FOR OUR EXPERIMENTS. THE SECOND AND THIRD COLUMNS WILL BE GENERATED FOR EACH METRIC WE INTEND TO USE.

Note that *arch spec* is the structure of the network expressed in terms of the 5 possible operations (*Zeroize*, *Skip Connection*, *1x1 Conv*, *3x3 Conv*, *3x3 Avg Pool*, labeled as numbers 0-4) in its layers, according to the NATS-Bench Topological Search Space. We need this column both to identify uniquely each architecture and to allow the evolutionary algorithm to make mutations (see later). We compose a table like this for each of the three Vision datasets (they can be found in our Github code).

C. Search Strategies

Let's move to the implementation of the Search Strategies.
Random search: Algorithm 1.

Algorithm 1 RandomSearch

```
generator ← RandomGenerator()
N ← Iterations
Best_net, Best_score ← None, 0
for i = 1 : N do
    net ← generator.generate()
    score ← net.score
    if score > Best_score then
        Best_net, Best_score ← net, score
    end if
end for
return Best_score
```

It consists in sampling an architecture randomly without replacement, and then evaluating it using a specific metric. If the value of the metric is greater than the previous, then that specific network becomes the best. We proceed with N iterations and then return the best score (we do not return the best network because we do not need it for our experiments).

Regularized Evolutionary Aging: Algorithm 2

The population is initialized by choosing P architectures randomly. Then, a subsample of size S is generated by random sampling with replacement (this is the additional "regularization" introduced, apart from the "aging" component, and it prevents the phenomenon of a certain architecture becoming a parent for a long time).
Here the *tournament selection* process starts. The architectures in the sub-sample are scored through a specific metric, and the one with the highest metric value becomes the parent.

Algorithm 2 Regularized Aging Evolution

$Population \leftarrow$ `Empty List`
$Max\_trained\_nets \leftarrow$ `Num of Mutations`
$Best\_net, Best\_score \leftarrow None, 0$
while $|Population| < P$ **do**
▷ Initialize the Population
$net \leftarrow RANDOMARCHITECTURE()$
`add` net `to` $Population$
end while
while $num\_scored\_nets < Max\_scored\_nets$ **do**
▷ Number of Evolutions
$Sample \leftarrow$ `Empty List`
while $|Sample| < S$ **do**
▷ Initialize the SubPopulation
$candidate \leftarrow$ `Random` net `from` $Population$
▷ With Replacement
`add` $candidate$ `to` $Sample$
end while
$parent.net \leftarrow$ `Highest score` net `in` $Sample$
$child.net \leftarrow Mutation(parent.net)$
$score \leftarrow child.score$
`add` $child$ `to` $Population$
`remove oldest` net `from` $Population$
▷ Regularized Aging Evolution
if $score > Best\_score$ **then**
$Best\_net, Best\_score \leftarrow child.net, score$
end if
$num\_scored\_nets += 1$
end while
return $Best\_score$

The parent network is then mutated by randomly changing an element in his *arch spec* vector, which corresponds to changing an operation of its layers. The child is appended to the population, while the oldest model present is removed.

The evolution happens by comparing the child's score to the best score, and, if greater, the best network and score are updated. After the number of scored models exceeds a threshold (*Max_scored_nets*), the algorithm returns the best score.

IV. Experiments, results, and discussion

The experiment we want to perform on each of the three datasets consists in embodying all three metrics we implemented (*ReLU Score*, *Synflow* and *Grad_Norm*), plus the validation accuracy after 12 epochs of training, first in the Random Search algorithm, then in the Regularized Aging Evolution. Our goal is to observe the behavior of the different metrics used in both algorithms and across the three different Vision datasets.

The number of experiments, n, is fixed at 30 as per project requirement. For the two algorithms, we tried the following hyperparameter configurations:

- RS: N in [100, 300, 1000, 3000]
- REA: (P,S) in [(100, 2), [100, 50), (20, 20), (20, 10), (100, 25), (64, 16)]

To propose a comparable length of the x axis for both algorithms (and to improve some evidence of stabilization after a certain number of steps), we decided to fix N and $Max\_trained\_nets$, to 1000.
At this point, all hyperparameters have been set for RS. Other than that, for REA we found $P = 20$, $S = 10$ as optimal hyperparameters.
In Fig. 2 and Tables III, IV and V, we have the results. We report some observations.

- The validation accuracy at 12 epochs of training performs the best in all situations, but requires a very expensive cost in terms of time. Therefore, a proxy training regime that takes advantage of the number of training epochs can still exceed training-free metrics, reducing computational time and resource requirements with respect to full training.
- The Regularized Evolutionary Aging algorithm stabilizes after a certain number of scored models, while the Random Search maintains a high standard deviation during all the searches. We expect this phenomenon, because REA moves toward population improvements, while the RS is more naive. We want to stress the fact that the number of models scored by REA could have been halved. In fact, for the most performant training-free metric inside our experiments, the evolution stabilizes after few mutations, typically 400/500, further reducing computational mean times.
- No specific combination (metric, algorithm) performs better across all three datasets. Following algorithms implementation, is evident how both algorithms tend to move toward the selection of architecture able to produce at each iteration an improved score, the RS following a random fashion, instead REA exploring the Search Space. In contrast, the resulted graphs show how these score improvements are not always followed by better test accuracy. We can suggest that the Expressivity (ReLU score) and Trainability (Synflow) Proxies used are good estimators for high accurate networks, but no indicator of the best architecture.
- For both algorithms Synflow metric has been the best training free score on CIFAR10 and CIFAR100 datasets, while on ImageNet, ReLU took the lead. To investigate this, we look at Table I: Spearman's correlation coefficient of the ReLU is always higher, for all datasets. However, ReLU only outperforms Synflow on ImageNet. To reason over this particular phenomenon, we note that in almost all experiments, the algorithm was able to find the best-scored network. Thus, we have taken a deeper look at the correlation parameter, by considering Spearman's rank correlation coefficient just for the top 10% networks ordered by metric scores, Table VI. This insight can explain the algorithms' behavior. We can see a lower correlation between the final test accuracy and

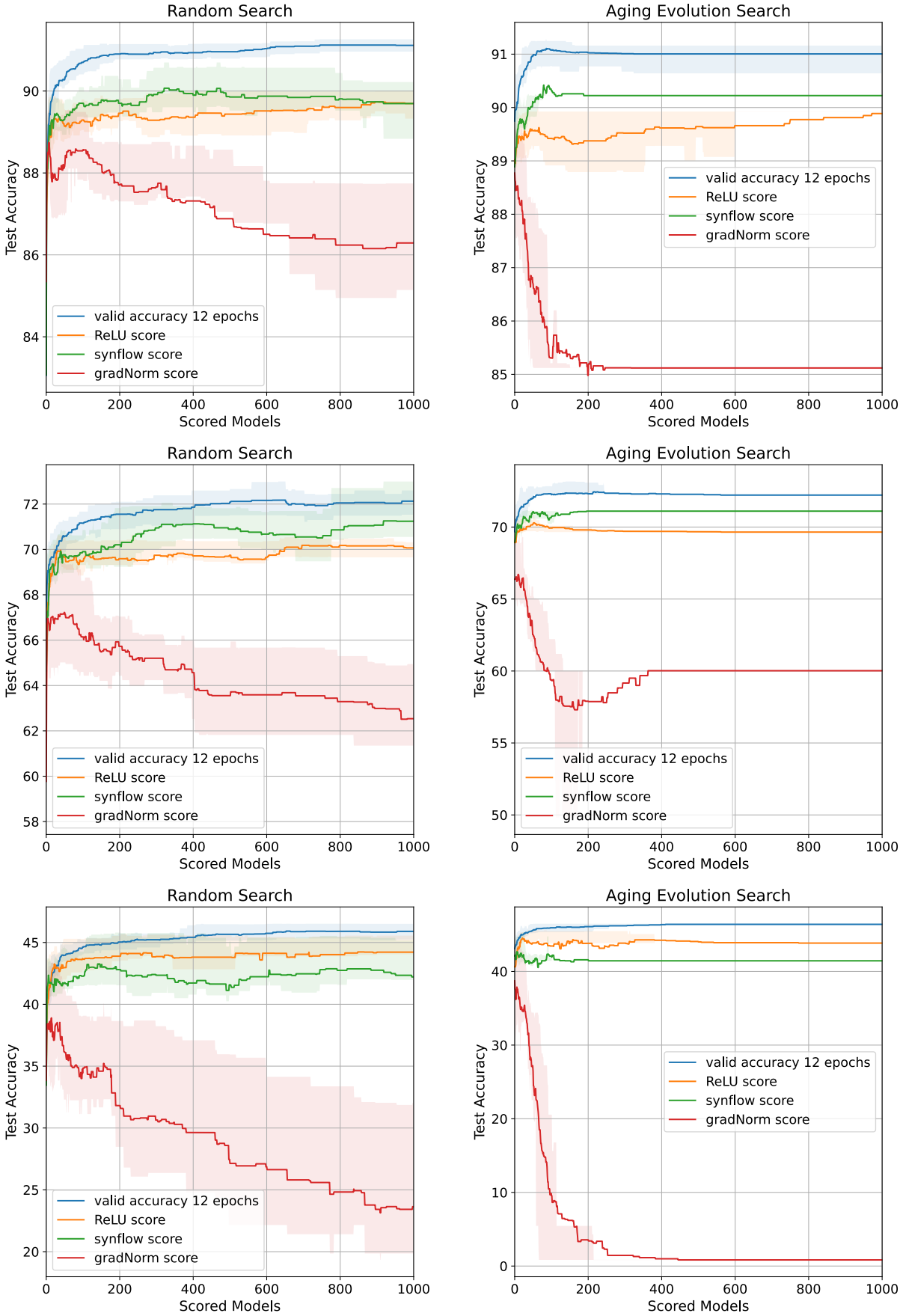

Fig. 2. Results of our experiments. We ran 30 simulations for each combination (metric, algorithm, data set). In the image, on every row, we have a dataset (first row: CIFAR10; second row: CIFAR100; third row: ImageNet16-120), and on the two columns, we have the different algorithms (RS and REA). Different colors represent the metrics. Each graph has on the x-axis the number of scored models, and on the y-axis the mean test accuracy of the network after 200 epochs of training (represented by the line) along with its upper and lower bounds found by adding/subtracting its standard deviation (represented by the shaded area).

	Random Search			Evolutionary		
	mean time(s)	**validation**	**test**	**mean time(s)**	**validation**	**test**
val_acc	147867.9	91.36±0.18	91.11±0.22	186277.0	91.32±0.26	91.00±0.24
ReLU	84.9	89.98±0.44	89.70±0.44	198.1	90.21±0.11	89.89±0.20
synflow	77.9	89.93±0.87	89.69±0.87	102.0	90.20±0.00	90.22±0.00
grad norm	87.2	86.73±2.52	86.29±2.57	98.9	85.51±0.00	85.12±0.00

TABLE III
CIFAR10.

	Random Search			Evolutionary		
	mean time(s)	**validation**	**test**	**mean time(s)**	**validation**	**test**
val_acc	227100.0	72.18±0.83	72.13±0.78	295347.6	72.83±0.55	72.22±0.36
ReLU	74.3	70.12±0.62	70.07±0.78	178.8	69.77±0.05	69.65±0.01
synflow	68.6	71.17±1.62	71.24±1.67	99.6	70.71±0.00	71.11±0.00
grad norm	74.9	62.35±3.99	62.54±3.89	98.3	59.88±0.00	60.02±0.00

TABLE IV
CIFAR100.

	Random Search			Evolutionary		
	mean time(s)	**validation**	**test**	**mean time(s)**	**validation**	**test**
val_acc	668721.4	45.75±0.38	45.90±0.66	881215.1	46.07±0.56	46.39±0.23
ReLU	37.3	43.79±2.57	44.22±2.60	74.1	43.52±0.00	43.85±0.00
synflow	63.9	42.03±3.53	42.21±3.60	111.0	40.78±0.00	41.44±0.00
grad norm	46.6	23.70±12.75	23.64±12.76	64.2	0.83±0.00	0.83±0.00

TABLE V
IMAGENET16-120.

IN THESE TABLES, WE REPORT THE MEAN ± ST.DEV. OF VALIDATION AND TEST ACCURACIES OF THE BEST NETWORK FOUND BY EACH ALGORITHM USING EACH METRIC, AVERAGED OVER 30 RUNS. MEAN SEARCH TIMES ARE ALSO INDICATED. THESE TIMES ARE RELATIVE TO A DEFAULT GOOGLE COLAB PRO CONFIGURATION EXCEPT FOR THOSE OF THE VALIDATION AT 12 EPOCHS THAT WE TOOK FROM THE NATS-BENCH API.

the metrics; 12 epochs validation accuracy still performs well, while different insights are shown over ReLU score and Synflow. The former reduces its ability to catch high accurate networks over the Cifar10 and Cifar100, keeping the lead only over the ImageNet16-120 database.

Dataset	**val_acc**	**ReLU**	**synflow**
CIFAR-10	0.77	0.19	0.27
CIFAR-100	0.72	0.23	0.31
ImageNet16-120	0.76	0.20	0.19

TABLE VI
LIKE TABLE I, BUT CONSIDERING JUST TOP 10% SCORES FOR VALIDATION AFTER 12 EPOCHS, RELU SCORE AND SYNFLOW.

- Grad_Norm is the only metric that deteriorates with the number of scored networks. This can be explained by looking at the values of Spearman's rank correlation coefficient in Table I: for Grad_Norm, it is always lower than the others.

V. CONCLUSIONS

After having discussed the results, we would like to give answers to the questions we posed at the beginning of the paper. We have seen that using the validation accuracy after 12 epochs of training as the Performance Estimation Strategy gives the best results across all three Vision datasets. The limitation of this approach is the computational time it requires, in the order of hundreds of thousands of seconds. On the other hand, the networks' accuracies exploited by training-free metrics are slightly lower, but with a ridiculous computation time.

Following the fact that we did not beat the benchmark, not even using the validation accuracy at 12 epochs, if we compare to state-of-the-art models found via Classic NAS (so with a complete training of the networks) [2], [4], our results are very far and certainly not comparable.

We found also that there are no training-free metrics that perform better in all three datasets, and neither there is any combination (metric, algorithm) that does so. The explanation of such a phenomenon does not stand in the incapacity of the algorithms of generating the best scoring networks but instead lies in the inaccuracy of high values for the training-free metrics in catching the best network.

As a final remark, we would like to point out that we did an actual training-free NAS experiment, in contrast, for example, to AREA [3], which only uses the score to initialize the population, while still training every network from that point onward. Indeed, all our algorithms use a training-free score to select the best network at each step. Thus, our results in terms of the accuracy of the best network found may not be comparable to that of other scholars, but still represent a new benchmark for *"Lightweight NAS"*.

REFERENCES

[1] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning, 2016; arXiv:1611.01578.

[2] Barret Zoph, Vijay Vasudevan, Jonathon Shlens and Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition, 2017; arXiv:1707.07012.

[3] Joseph Mellor, Jack Turner, Amos Storkey and Elliot J. Crowley. Neural Architecture Search without Training, 2020; arXiv:2006.04647.

[4] Esteban Real, Alok Aggarwal, Yanping Huang and Quoc V Le. Regularized Evolution for Image Classifier Architecture Search, 2018; arXiv:1802.01548.

[5] Thomas Elsken, Jan Hendrik Metzen and Frank Hutter. Neural Architecture Search: A Survey, 2018, Journal of Machine Learning Research 20 (2019) 1-21; arXiv:1808.05377.

[6] Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak and Nicholas D. Lane. Zero-Cost Proxies for Lightweight NAS, 2021; arXiv:2101.08134.

[7] Xuanyi Dong, Lu Liu, Katarzyna Musial and Bogdan Gabrys. NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size, 2020; arXiv:2009.00437.

[8] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow, 2020, Advances in Neural Information Processing Systems 2020; arXiv:2006.05467.