How Powerful are Skip Connections in Neural Architecture Search

Sara Ferrua
Politecnico di Torino
Torino, Italia
s292946@studenti.polito.it

Jasmine Guglielmi
Politecnico di Torino
Torino, Italia
s303105@studenti.polito.it

***Abstract*—Neural Architecture Search is a technique that aspires to discover the best architecture for a neural network based on a specific need. Briefly, NAS uses a controller network that produces an architecture suggestion specified by a mutable-length string. The neural network is then trained by a child network on real data, generating a signal fed into the controller, which will subsequently give a higher probability of architectures reaching greater accuracies. In the last few years, researchers have employed their efforts to implement algorithms that bring improvements in the quality of the NAS. This research, which is based on this concept, aims to demonstrate how the application of certain variations on the search algorithm, might provide better outcomes. The first phase accomplished on NAS, consists of computing ahead of time the training-free metrics for all the possible architectures, scoring each of them at initialization. Subsequently, the NASWOT algorithm is implemented, a similar random search for picking randomly a candidate from the search space and iteratively finding the best network. To do a level up gaining higher scores, existing aging research is examined, finding the Regularized Evolutionary Algorithm, which can discriminate between the best architectures based on a genotype structure at each iteration. However REA needs a way to evaluate each architecture, so a zero-cost proxy is provided, the synaptic flow score, known as SynFlow, which was originally considered a pruner. Based on this research, the focus moved on the mutation of the REA's parent architecture, because following certain considerations, it was noted that skip connections have an importance during the aging evolution. The source code for running the experiments can be found at** `https://github.com/jjguglielmi/How-Powerful-are-Skip-Connections-in-Neural-Architecture-Search-without-Training`

I. Introduction

Due to its adaptability to a multitude of scenarios, NAS is a branch of machine learning that has been attracting an increasing amount of interest. The first experiments involved designing architectures using software. This approach proved to be rather time-consuming and frequently led to errors. For this reason, subsequent studies have gone on to implement algorithms that, starting from a set of pre-existing architectures, could independently search for the best one given a certain task. This step includes the training of all the architectures on the datasets, which is a very resource-intensive procedure and has an intense computational time. So, to ensure reproducibility and to save time during initialisation, a score was given to each untrained architecture in a way that reflects its reliability for the given purpose. NATS-Bench is a benchmark that was used to evaluate the efficacy of our technique; it consists of a set of 5 possible operations, and it comprises two different search spaces, the size one and the topology one. The latter is the search space that was used. Obtaining scores for all the 15625 architectures, the NASWOT algorithm [1] approach was followed, in which a candidate is picked randomly from the search space instead of having a network as a generator. After being picking, it is scored in its untrained state rather than training it. This process is iterated N times, constructing a sample of N architectures, on which the best network is chosen. The next step involved moving away from this more traditional approach to focus on improving efficiency. After a thorough search, we came across a group of optimization algorithms known as Evolutionary Algorithms which are based on mutations and reproduction. REA was specifically built which, drawing inspiration from the biological concept of age, removes older architectures throughout iteration cycles in favour of recently created ones.

II. NASWOT

The aim was to observe the overlap of linear maps composed by the activations of ReLU between inputs in a batch of data for an untrained network, to come up with a network scored at initialisation. As is described in previous work [1], a neural network with ReLUs was picked. For each layer in each unit a binary indicator was defined obtained by alternating linear maps at each layer with the ReLUs and then multiplying them. This technique is founded on the notion that it is more complex for the network to acquire some knowledge to differentiate these inputs if the binary codes related to the two inputs are more alike. This is because if two images have the same binary codes, they share the same linear region, thus the disentanglement is extremely hard.
To calculate the distance between the two binary codes, the *Simon H operator* known as the "@" operator was used. First the matrix multiplication was calculated between the binary indicator x and its own transposed. Next, all the zero values in the binary code x were taken and changed into ones and vice-versa. The target was to carry out the previously mentioned multiplication but with the newly obtained tensor. Lastly, these two new tensors were summed with a *global variable Ktemp* to achieve the Kernel Matrix. This method is an alternative of the Hamming Distance, which is a metric

for comparing two equal-length binary strings, that returns the number of positions where the two inputs differ. The scoring metric for each architecture in the case study is the logarithm of the determinant of the tensor mentioned above. After obtaining the architectures with the required scores, we avoided utilizing the GPU again for demanding tasks and instead concentrated on running the NASWOT algorithm. First of all some experiments were conducted generating a N different samples of architectures at random, run over a loop of 30 iterations, and outputs the architectures with the highest score. It was decided to train that network for 200 epochs to boost the chances that the best model had been chosen.

After observing the results the same experiments were conducted using test accuracies for a N different samples of networks trained for 12 epochs, for each group of sample we picked the architecture with the best test accuracy and finally considered its 200 epochs' test accuracy.

Table I illustrates how the metric for the NASWOT search algorithm kept expanding, apart from Cifar10 with *N=50* and *N=100*. In this case the search space's dispersion in relation to its mean is greater for a smaller sample with a higher metric, rather than a bigger sample with a lower metric.

Moreover, the trend of the accuracy as N grows is different for each dataset. A linear growth can be seen only for ImageNet, and not for Cifar10 and Cifar100.

The time listed in Table I, shows the time required to compute the Kernel Metrix and the time required to compute the test accuracy of 200 epochs. While Table II, illustrates the sum of the training of all time for 12 epochs taken from the search space's API and the time that occurs for performing the training and the accuracies. The test accuracy in Table II for 200 epochs is higher than that for 12 epochs for every N different sample, because if the accuracy begins to drop to the increase of the number of epochs, it could mean that the network is starting to overfit. This implementation can be found in the cited GitHub as "*step1_and_2*".

TABLE I
NASWOT SEARCH ALGORITHM BASED ON TRAINING FREE SCORE

Dataset	N	Metric		Accuracy 200epochs		Time	
		mean	std	mean	std	mean	std
Cifar10	50	1634.57	13.02	92.89	0.71	19.02	0.66
	100	1633.91	8.64	92.51	1.20	38.42	1.39
	200	1650.02	10.57	92.96	0.68	76.29	1.89
	400	1654.51	7.41	93.04	0.78	154.83	3.19
Cifar100	50	1626.40	12.34	69.51	1.07	19.39	1.28
	100	1637.78	12.94	69.93	1.48	38.30	1.81
	200	1650.18	10.36	69.99	1.20	78.48	2.47
	400	1652.21	8.86	69.77	1.36	154.38	5.21
ImageNet 16-120	50	1448.32	17.07	40.85	4.14	17.94	1.36
	100	1459.44	9.56	41.82	3.77	35.81	1.77
	200	1467.64	10.63	43.01	2.76	78.98	3.2
	400	1470.40	9.18	43.62	2.22	160.01	3.58

TABLE II
NASWOT SEARCH ALGORITHM USING VALIDATION ACCURACY AFTER 12 EPOCHS

Dataset	N	Accuracy 12epochs		Accuracy 200epochs		Time 12epochs	
		mean	std	mean	std	mean	std
Cifar10	50	87.84	0.59	93.54	0.46	10970.45	372.64
	100	88.22	0.42	93.79	0.33	22068.01	533.35
	200	88.36	0.29	93.84	0.26	43674.74	742.85
	400	88.67	0.28	93.95	0.24	87665.29	1103.61
Cifar100	50	58.18	1.66	70.61	1.36	10962.43	413.01
	100	59.01	1.20	71.06	1.08	21816.75	531.62
	200	59.73	0.75	71.65	0.89	43834.76	861.88
	400	60.20	0.58	72.20	0.61	87918.12	1133.13
ImageNet 16-120	50	34.51	1.78	44.47	1.46	33669.47	1178.09
	100	35.59	1.11	45.25	0.85	66573.62	1700.56
	200	36.03	0.76	45.55	0.97	133512.20	2302.47
	400	36.44	0.63	45.78	0.74	264813.89	4023.50

III. METHODS

A. Search Space

NATS-Bench is the architecture dataset that was used to perform our program. It is divided into two sections: the *Size Search Space*, which has 32768 total architectures, and the *Topology Search Space*, which contains 15625 neural candidates. Our analysis was based on the latter search space where all the architectures are made up of cells with the same topology. A DAG can be created to formally describe the structure; specifically, a group of 4 nodes connected by directional edges (6 overall) that starts from the i-th node and arrives to the j-th node, where ($i < j$). There are a total of 5 operations: zeroize, skip connection, 1-by-1 convolution, 3-by-3 convolution, and 3-by-3 average pooling layer. An operation is associated to a specific edge, but also each operation can be repeated over the edges. So, the set of these 6 operations defines an architecture. Mathematically every node reflects the total number of feature maps that have been altered by the corresponding operations of the edges leading to that node. The convolution in this operation set is an abbreviation of a sequence of the operation ReLU, convolution, and batch normalization. [2]

For what concerns the batch of data in which our set of architecture was evaluated, three datasets of images were used.

Cifar10 is composed of 60000 colour images, represented as a 32x32, divided into 10 classes each of which contains 6000 pictures. Due to the need of a validation set, a small subset of the 10,000 data for the batch test was used, and the remaining 50,000 for the batch training in which more images may belong to a class rather than another. [2]

Cifar100 is a dataset that is slightly different from the one described above as it is divided into 100 classes each of which has 600 images. Additionally, the split into training and test is different since 100 pictures for each class were used as test and the remaining images as training. As the amount of data was

greatly increased compared to Cifar10, the 100 classes were grouped into macrostructures defined as superclasses equal to 20. By analysing a specific image, it was possible to locate which class it belonged to, thanks to a “fine” label with the respective “coarse” label, so its superclass. [2]

ImageNet16-120 was the final dataset that was used; it is a down sample of ImageNet in which images characterized by 16x16 pixels can be found. In our case, in ImageNet16-120, were found 120 classes that contain altogether 151.7K training images and a small subset with only 6K pictures equally divided between testing and validation. [2] [3]

B. Regularized Evolution Algorithm

A search algorithm was implemented in order to improve efficiency, and that is REA, Regularized Evolutionary Algorithm, which is part of a macro group of optimization algorithms, based on the concept of age. This type of process prefers in the iterative development to remove architectures defined as ”old” in favour of the latest mutated, the ”new”. This concept is known as aging evolution.
Firstly it initializes the population with P random architectures, secondly the starting population is then improved via evolution in C cycles each of which introduces one model and removes another. At each iteration, it draws S random models from the population, each selected evenly at random with replacement. The model with the best validation accuracy within this sample is chosen as the parent. The application of a transformation known as a mutation creates a new architecture known as the child from the parent. A mutation alters the architecture in a basic and random way. Once the architecture of the child is built, trained and evaluated, it is incorporated into the population. This is known as tournament selection.
Skip connections are a module in which an algorithm can skip multiple layers of architecture in convolutional neural networks. These connections are used to connect features derived from previous layers to further layers. The method can also avoid the problem with a small element size, when in previous layers there is multiplication on very small elements. Skip connections are known for reducing the length of the information propagation path after capturing long-term dependencies. In fact, during the previous experiments described, we found that architectures with skip connections scored higher, as is evident in Fig.1, despite the fact that there are more of those architectures without skip connections. To plot the graph in Fig.1, 120 iterations conducted for Cifar10 were considered with 4 different sample sizes, each of these executed 30 times. The top 10 accuracy models were extracted from Fig.1.
As can be noted from Table III there are both skip connection containing and skip connection free architectures. Despite the fact that the best test accuracy comes from the model containing a skip connection, in order not to delete a consistent part of the dataset, a study comparing two different ways of executing REA, with and without the skip connection filter, is suggested. [4] [5].

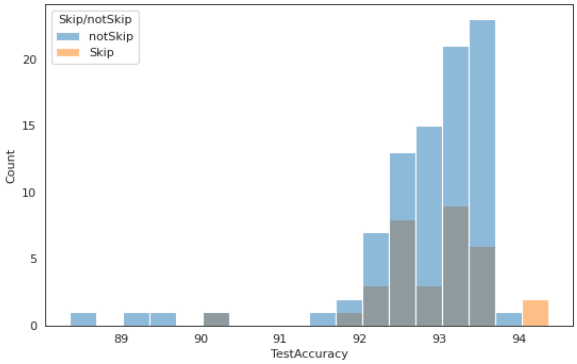

Fig. 1. Skip/noSkip for Cifar10

TABLE III
SKIP-POSITION AND TEST ACCURACY ON CIFAR10

Index	Test-Accuracy	CountSkip
34	94.37	1
15	94.05	1
102	93.89	0
76	93.67	0
33	93.67	0
65	93.65	0
97	93.62	0
113	93.61	0
82	93.60	0
17	93.58	0

C. SynFlow

In this section the alternative way to score architectures adopted in our research is explained in detail. SynFlow is a metric that was used to evaluate the search algorithms proposed. Originally it was born as a metric that prunes the network: it is used to minimize the number of parameters of the architectures identifying the less importants, with the purpose of reducing the cost of the search algorithm. SynFlow was developed with some features that differentiate it from other optimization algorithms: the calculation of the score is independent of the data in input, moreover, to compute it, a loss function R_{SF} defined as the product of all parameters in the network (1) was used.
In more detail, $\mathbb{1}$ is a vector composed of all 1 data point and $\theta^{[l]}$ is the parameter referred to the layer l^{th}.

$$R_{SF} = \mathbb{1}^T \left(\prod_{l=1}^{L} |\theta^{[l]}| \right) \mathbb{1} \tag{1}$$

This loss function is nothing more than the l_1 norm of the model while the scoring metric for a parameter is a part of the norm throughout the single parameter. Generally, the SynFlow

score is defined as $S_p(\theta)$ (2), where $\odot$ the Hadamard product is used to multiply the derivative of R_{SF} respect to the weights of parameters and the weights θ.

$$S_p(\theta) = \frac{\partial R_{SF}}{\partial \theta} \odot \theta \tag{2}$$

To obtain the score of an entire architecture, all *N* parameters of the neural network (3) were summed:

$$S_n = \sum_{i}^{N} S_p(\theta)_i \tag{3}$$

SynFlow was implemented following the algorithm proposed at:
`https://github.com/SamsungLabs/zero-cost-nas` [7] [8]

D. Experimental set-up

For the initial set up of the experiments and to shorten the execution time of the research, Pytorch provided a helpful machine learning open source that allows the tensors to be used more easily and quickly.
With Pytorch, it is possible to use the parallel computing platform's semantics of CUDA, which enables the software to employ GPUs as devices. It keeps records of the chosen GPU and by default all the tensors are generated on that device. These tensors allow operations to be carried out and save the results on the same device as the tensor.

Regarding the core produced work, the fixed parameters for REA are:

- population P sets 50,
- sample size S sets 10,
- cycles set to 200.

The experiments for each dataset were run 10 times, with four possible combinations with NASWOT and SynFlow metrics, skip connections filter and REA. So, the possible combinations are:

1) REA with NASWOT metric without any modifications.
2) REA with SynFlow metric also without modifications.
3) REA with NASWOT metric with modification where the constraint of one or zero skips in models is implemented.
4) REA with SynFlow metric with the same modification applied on antecedent combination.

All combinations of finding a satisfying architecture performed quite well, in fact some great results were achieved.

IV. Results

The two metrics presented in the previous sections were implemented in combination with the REA algorithm using 1 single GPU on the Google Colab notebook.
Comparing the NASWOT and SynFlow algorithms, it is evident that the first is more focused on considering the number of ReLUs inside the network, while the latter gives greater consideration to the number of parameters. It should be highlighted that, when these measures are combined with REA, high accuracy is influenced by a greater number of parameters than ReLUs.

The first two outcomes to be examined are those related to "**REA + NASWOT without modifications**" and "**REA + NASWOT with modifications**" executed setting the numbers of population and sample size respectively 50 and 10, performed for 10 runs. The results show how the mean metrics for the different datasets are extremely close, despite the fact that the standard deviation for Cifar10 and ImageNet increases in the second case. Although, focusing on the accuracy, for Cifar10 and Cifar100 there is an improvement in the algorithm with modifications, and there is a significant reduction of the standard deviation for the Cifar100's accuracy. One important aspect is less time needed to end the REA + NASWOT with modification respect to the previous one. The time in all tables corresponds to the mean to find the best accuracy model running the algorithm 10 times. Summing all these single run times, for each dataset in Table IV, a total time of 26,8 minutes is reached, compared to 24,85 minutes for Table V. Even though the differences run times are only a few minutes, the computational time is reduced.

TABLE IV
REA WITH NASWOT FOR 10 RUNS (BATCH SIZE = 128)

Dataset	Metric		Test-Accuracy		Time [s]	
	mean	std	mean	std	mean	std
Cifar10	1748.73	0.84	92.44	1.11	50.33	4.28
Cifar100	1747.27	1.18	69.69	1.57	50,49	1.48
ImageNet 16-120	1568.37	1.5	44.01	3.08	60,36	0.96

TABLE V
REA WITH NASWOT AND MODIFICATIONS FOR 10 RUNS (BATCH SIZE = 128)

Dataset	Metric		Test-Accuracy		Time [s]	
	mean	std	mean	std	mean	std
Cifar10	1748.92	1.18	92.55	1.16	46.43	0.63
Cifar100	1747.64	0.71	70.06	0.67	46.67	0.74
ImageNet 16-120	1568.61	1.87	42.68	3.91	56.02	1.18

Considering the SynFlow metric, in Table VI and Table VII, there respectively "**REA + SynFlow without modifications**" and "**REA + SynFlow with modifications**". The mean metrics in the different datasets varies from 1 to 4 points, nevertheless the standard deviation for Cifar10 and ImageNet is slightly reduced. Of note is the significant increasing standard deviation for Cifar100 with the application of the skip connection filter. Regarding accuracy, it is considerably better for the new variation for Cifar10 and Cifar100, together with the standard deviation for Cifar100 and ImageNet. The total run time for each dataset in Table VI is 47,38 minutes when all of these individual run times are summed, compared to 47 minutes for Table VII. Similarly, in this scenario, we discover a slight

improvement of the computational time after applying the modification in the algorithm.

TABLE VI
REA WITH SYNFLOW FOR 10 RUNS (BATCH SIZE = 128)

Dataset	SynFlow		Test-Accuracy		Time [s]	
	mean	std	mean	std	mean	std
Cifar10	104.85	3.99	93.49	0.49	89.51	3.45
Cifar100	108.54	0.78	69.70	2.42	87.80	0.88
ImageNet 16-120	101.97	7.40	42.82	3.16	107.01	1.26

TABLE VII
REA WITH SYNFLOW AND MODIFICATIONS FOR 10 RUNS (BATCH SIZE = 128)

Dataset	SynFlow		Test-Accuracy		Time [s]	
	mean	std	mean	std	mean	std
Cifar10	105.75	3.92	93.36	0.69	89.49	0.47
Cifar100	104.56	6.53	70.83	1.02	88.75	0.69
ImageNet 16-120	103.91	4.31	43.68	1.84	104.10	0.58

In Fig.2, it can be seen the directed acyclic graph of the best architecture found for the dataset Cifar10, exactly at index 13714.

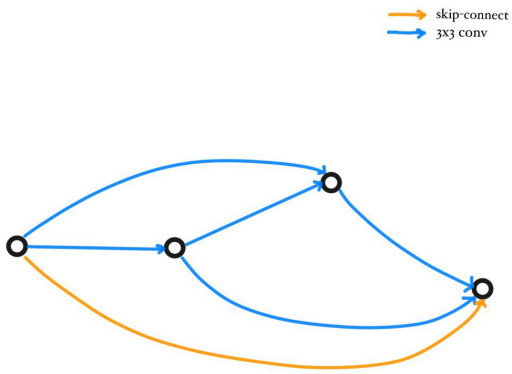

Fig. 2. Directed Acyclic Graph of best Cifar10 Architecture

V. CONCLUSION

Considering all the observations made above for search algorithms and metrics, the research concludes with the solution that the combination of REA and SynFlow with the condition of zero skip connections or just one, works very well for the given task, image classification. The averages achieved during our work are comparatively good, especially for the computational time point of view, but still optimal results were not reached. The reason for the discrepancy between accuracies could be the difference in image definition and the number of classes that changes from dataset to dataset. Tests were performed with a single search space, NATS-Bench, but it would be possible to expand the research to other spaces in order to understand what the behaviour of our project would be.

REFERENCES

[1] J. Mellor, J.Turner, A. Storkey and E. J. Crowley, Neural architecture search without training, 2021.
[2] X. Dong, L. Liu, K. Musial and B. Gabrys, Nats-bench: Benchmarking nas algorithms for architecture topology and size, 2021.
[3] P. Chrabaszcz, I. Loshchilov and F. Hutter, A downsampled variant of imagenet as an alternative to the cifar datasets, 2017.
[4] E. Real, A. Aggarwal, Y. Huang and Q. V. Le, Regularized evolution for image classifier architecture search, 2019.
[5] D. Wiczew, Evolve your neural net now! AutoML with regularized evolution from scratch, 17 July 2020.
[6] D. Shan, X. Zhang, W. Shi, L. Li, Neural Architecture Search for a Highly Efficient Network with Random Skip Connections, 27 May 2020.
[7] M. S. Abdelfattah, A. Mehrotra, L.Dudziak and N. D. Lane, Zero-cost proxies for lightweight NAS, 2021.
[8] H. Tanaka, D. Kunin, D. L. k. Yamins, S. Ganguli, Pruning neural network without any data by iteratively conserving synapting flow.