Hidden Design Principles in Zero-Cost Performance Predictors for Neural Architecture Search

André Ramos Fernandes da Silva[†], Lucas Marcondes Pavelski[†] and Luiz Alberto Queiroz Cordovil Júnior[†,*]
[†]*Sidia R&D Institute, Retail Management System, Av. Darcy Vargas, 654, 69055-035, Manaus, AM, Brazil*
E-mail: {andre.fernandes, lucas.pavelski, luiz.cordovil}@sidia.com
[*]Corresponding Author. All authors have contributed equally.

2023 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-8867-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/IJCNN54540.2023.10191474

***Abstract*—Neural Architecture Search is an active research field that aims to design neural networks automatically. Nevertheless, this is usually an expensive process since the search algorithm must evaluate the performance of many candidate solutions from a vast search space. Because of that, different strategies have been proposed to perform efficient Neural Architecture Search. The recent development of zero-cost performance predictors has shown a lot of promise due to the possibility of predicting a network's performance without training. On the other hand, a predictor's correlation with a model's performance may depend on the network search space and on the benchmark dataset. Each performance predictor might lead the search processes to favor very different network patterns. A design principle is defined as a restriction in a hyperparameter distribution that is expected to yield optimum network performance. In this work, we propose an automatic iterative approach to uncover the design principles of deep neural networks optimized by zero-cost performance predictors, and we discuss insightful information obtained by its application.**

***Index Terms*—Deep Learning, Neural Architecture Search, Performance Predictors**

I. Introduction

Deep Neural Networks (DNNs) have become very popular because they excel at many machine learning tasks. On the other hand, designing DNNs is challenging since there are almost unlimited possibilities: architecture design choices, hyperparameter tuning, training schedules, and others. Nonetheless, many Neural Architecture Search (NAS) strategies have been proposed to help machine learning engineers develop DNNs automatically [1], [2]. Reinforcement Learning NAS can learn to design DNNs based on a reward signal that maximizes the network performance [3]–[5]. Evolutionary algorithms have also had great success in NAS [6]–[8], and gradient-based strategies have also been proposed [9]–[11].

However, NAS is a costly process since it usually demands a lot of computing power to evaluate the performance of many DNNs. It has motivated the development of different strategies for efficient NAS, such as weight inheritance, early stopping policy, reduced training set, reduced population, population memory, and others [2], [12]. Mellor et al. [13], [14] proposed the first zero-cost performance predictors for NAS. They argue that if different images have distinct activation patterns, the neural network will easily learn to distinguish and classify them. Activation patterns can be computed with a single forward pass at initialization without training. Furthermore, their score requires just a mini-batch of data. It has motivated researchers to develop new performance predictors and assess their usefulness for NAS in different scenarios [15]–[17]. It has been reported that most predictors are not robust, i.e., their correlation with the final accuracy and ranking varies significantly across different datasets and learning tasks. Understanding what makes a robust performance predictor is still an open question.

Radosavovic et al. [18], [19] proposed a different perspective for NAS called *designing network design spaces* (DNDS). Instead of optimizing networks individually, they analyze design spaces by observing trends in populations of DNNs. Their methodology consists of the following steps: generate random networks from a design space, compute statistics, analyze the trends in hyperparameters' distributions for the best networks, discover (or hypothesize) a new design principle, define a new design space, and repeat the process. In this context, a design principle is defined as a restriction in a hyperparameter distribution that is expected to yield optimum network performance. This process results in an optimized design space, which they call RegNet. The authors performed experiments on the AnyNet design space, which is a generalization of ResNet [20] and ResNeXt [21] design spaces. They argue that this method is useful for understanding the design principles of top-performing DNNs. However, their process involves manual attempts to improve the overall distribution of AnyNet subspaces.

In this work, we propose an automatic DNDS method to uncover the design principles of neural networks optimized by zero-cost performance predictors, and we discuss insightful information obtained by its application. Our method is based on iterated sampling, empirical bootstrap, and hyperparameters intervals refinement. The hidden design principles are then uncovered insofar as the trends are perceived and took into account to optimize the search space. Starting with the AnyNet design space, we analyze zero-cost performance predictors proposed by Mellor et al. [13], [14] and by Abdelfattah et al. [15]. Experiments are conducted on CIFAR-10 and CIFAR-100 benchmark datasets [22].

The remainder of this work is divided as follows. Section II discusses some related works. Section III introduces and defines the zero-cost performance predictors analyzed in this work. Section IV describes the AnyNet design space. Section V describes the methodology and experimental settings. Section VI presents the results. Section VII is the conclusion.

II. Related Works

Abdelfattah et al. [15] made the first study comparing different zero-cost performance predictors. Their work is particularly interesting because they observed that zero-cost pruning scores could also be adapted for NAS. They reported the performance of adapted pruning-at-initialization metrics — `snip` [23], `grasp` [24], `synflow` [25], and `fisher` [26], [27] — comparing their Spearman's rank correlation with common proxy tasks studied by Zhou et al. on EcoNAS [12].

Ning et al. [16] analyzed one-shot estimators — those that rely on sharing the parameters of a supernet between all candidate architectures — as well as zero-cost estimators that require no training. It was reported that different metrics might favor different architecture patterns. The authors also discussed the properties and weaknesses of performance predictors and suggested improvements.

White et al. [17] made a large-scale study of 31 performance predictors from different paradigms: curve extrapolation, weight sharing, supervised learning, and zero-cost proxies. They evaluated their ability to predict the models' performance and speed up the NAS process in different settings.

III. Zero-cost performance predictors

A. Gradient Norm

The `grad_norm` performance predictor was proposed by Abdelfattah et al. [15]. It is defined as the Euclidean norm of the gradients on a mini-batch of data.

B. SNIP

Lee et al. [23] proposed a network pruning algorithm called Single-shot Network Pruning based on Connection Sensitivity (SNIP). Their connection sensitivity score detects relevant connections at initialization, thus eliminating the need for expensive prune-retrain iterations. Abdelfattah et al. [15] adapted this and other zero-shot pruning scores to study their effectiveness as performance predictors for NAS. Let $\theta_1, \ldots, \theta_M$ be the network parameters and let $\mathcal{L}$ be the loss function on a mini-batch of data. The `snip` score for NAS is defined as:

$$\mathcal{S} = \sum_{i=1}^{M} \left| \theta_i \frac{\partial \mathcal{L}}{\partial \theta_i} \right|$$

C. GRASP

As `snip`, Abdelfattah et al. [15] also adapted the Gradient Signal Preservation (`grasp`) zero-cost pruning score proposed by Wang et al. [24]. Their criterion is also posed in terms of synaptic saliency. However, instead of preserving the loss at initialization, `grasp` differs from `snip` in that it aims at preserving the gradient flow. They argue that this yields better final performance than the preservation of the initial random loss because `snip` does not consider blocking gradient flow when a given connection (or parameter) is pruned. The adapted `grasp` score for NAS is defined as:

$$\mathcal{S} = \|-\boldsymbol{\theta} \odot \mathcal{H}_{\mathcal{L}} \nabla \mathcal{L}\|_1 = \sum_{i=1}^{M} -\theta_i \sum_{j=1}^{M} \frac{\partial^2 \mathcal{L}}{\partial \theta_i \partial \theta_j} \frac{\partial \mathcal{L}}{\partial \theta_j}$$

where $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_M)$ are the network's parameters, $\mathcal{H}_{\mathcal{L}}$ is the Hessian of the loss w.r.t. the parameters, $\nabla \mathcal{L}$ is the gradient of the loss, $\odot$ is the Hadamard product, and $\| \cdot \|_1$ is the ℓ_1-norm, that is, the sum of the elements of the vector.

D. Synaptic Flow

Iterative Synaptic Flow Pruning (`synflow`) is a pruning-based criterion to score neural networks supporting NAS [25]. In this context, the intersection between NAS and pruning is established in terms of the synaptic saliency score, which assesses the effect on loss when a single parameter or a set of parameters θ are removed from a given network.

Let $\mathcal{L}$ be a loss function comprising the product of all network parameters, the per-parameter synaptic saliency score $\mathcal{S}_p$ is defined as:

$$\mathcal{S}_p(\theta) = \frac{\partial \mathcal{L}}{\partial \theta} \odot \theta,$$

where $\odot$ denotes the Hadamard product, i.e., an element-wise matrix product. The total `synflow` score of a network as:

$$\mathcal{S} = \sum_{i}^{M} \mathcal{S}_p(\theta_i).$$

E. FISHER

This zero-cost predictor is based on pruning activation channels on the network. In practical terms, the channels (features maps) and their respective parameters, which have little effect on the model performance, are removed [26], [27].

Let be a neural network with a set of parameters named Θ trained to minimize a cross-entropy loss function $\mathcal{L}$, such that

$$\mathcal{L}(\Theta) = \mathbb{E}_P \left[-\log Q_\Theta(\mathbf{z} \mid \mathbf{I}) \right]$$

being $\mathbf{I}$, $\mathbf{z}$, $\mathbb{E}_P$ and, Q_Θ the inputs, outputs, expectation w.r.t. a given data distribution P and, Q_Θ, respectively. One can evaluate the effect of addition or removal termed as $\Delta\Theta$ of a given parameter in the loss function by applying a 2nd order Taylor expansion [26], as detailed as follows

$$\mathcal{L}(\Theta + \Delta\Theta) - \mathcal{L}(\Theta) \approx \mathbf{g}^\top \Delta\Theta + \frac{1}{2} \Delta\Theta^\top \mathbf{H} \Delta\Theta$$

where $\mathbf{g} = \nabla \mathcal{L}(\Theta)$ and $\mathbf{H} = \nabla^2 \mathcal{L}(\Theta)$. Let $\theta_i \in \Theta$ be a parameter to be dropped, i.e, set to 0, the 2nd order Taylor expansion can be rewritten as

$$\mathcal{L}(\Theta - \theta_i \mathbf{e}_i) - \mathcal{L}(\Theta) \approx -g_i \theta_i + \frac{1}{2} H_{ii} \theta_i^2$$

where $\mathbf{e}_i$ is a vector with just the i^{th} component set to 1 (the remainder is 0), and H_{ii} is the diagonal of Hessian matrix $\mathbf{H}$ defined

$$H_{ii} \approx \mathbb{E}_P \left[\left(\frac{\partial}{\partial \theta_i} \log Q_\Theta(\mathbf{z}|\mathbf{I}) \right)^2 \right] \approx \frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial \mathcal{L}_n}{\partial \theta_i} \right)^2.$$

where $n = 1, \ldots, N$ denotes the nth input of $\mathbf{I}$. The aforementioned approximation is carried out considering Fisher Information which transforms a second-order derivative into the square of first-order derivative [28]. As pointed out by Hassib and Stork [29], one can assume Θ to be at a local optimum which leads to $gi = \frac{\partial \mathcal{L}}{\partial \theta_i} \approx \frac{1}{M} \sum_i^M \frac{\partial \mathcal{L}_n}{\partial \theta_i} \approx 0$ insofar as the model converges. Therefore, the per parameter saliency score can be computed as

$$\mathcal{S}(\theta_i) = \frac{1}{2N}\theta_i^2 \sum_{n=1}^{N} g_{ni}^2 \;\propto\; \theta_i^2 \sum_{n=1}^{N} g_{ni}^2,$$

and considering the entire feature map one has

$$\mathcal{S} = \sum_{i}^{M} \theta_i^2 \sum_{n=1}^{N} g_{ni}^2 = \sum_{i=1}^{M}\sum_{n=1}^{N} (g_{ni}\theta_i)^2.$$

F. Jacobian Covariance

The Jacobian Covariance (`jacov`) was proposed by Mellor et al. in their first version of *Neural Architecture Search without Training* [13]. The score is computed on a mini-batch of data $X = \{x_1, ..., x_N\}$ with size N. Let $x_i \in \mathbb{R}^D$ be and input signal $x_i = (x_{i,1}, \ \ldots, \ x_{i,j}, \ \ldots, \ x_{i,D})$ that will be mapped through the network into the output $y_i \in \mathbb{R}^C$, where D is the dimension of the input signal and C is the dimension of the output signal, i.e., the number of classes. First of all, the score transforms the network output y_i to a scalar value $f_i \in \mathbb{R}$ that is the number of the output class. Let $\mathbf{J}$ be a matrix defined as:

$$\mathbf{J} = \begin{bmatrix} \nabla f(x_1) \\ \vdots \\ \nabla f(x_N) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_{1,1}} & \cdots & \frac{\partial f_1}{\partial x_{1,D}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_{N,1}} & \cdots & \frac{\partial f_N}{\partial x_{N,D}} \end{bmatrix},$$

where, for simplicity of notation:

$$\frac{\partial f_i}{\partial x_{i,j}} = \frac{\partial f(x_i)}{\partial x_{i,j}}$$

for $1 \le i \le N$ and $1 \le j \le D$, where N is the size of the mini-batch of data X, and D is the dimension of the input signal.

Let $\mathbf{C}_J$ be a covariance matrix $\mathbf{C}_J = (\mathbf{J} - \mathbf{M}_J)(\mathbf{J} - \mathbf{M}_J)^T$, where $\mathbf{M}_J$ is a matrix with elements:

$$(\mathbf{M}_J)_{i,j} = \frac{1}{N}\sum_{j=1}^{D} \mathbf{J}_{i,j}$$

Let the $\mathbf{\Sigma}_J$ be correlation matrix whose elements are:

$$(\mathbf{\Sigma}_J)_{i,j} = \frac{(\mathbf{C}_J)_{i,j}}{\sqrt{(\mathbf{C}_J)_{i,i}(\mathbf{C}_J)_{j,j}}}$$

Let $\sigma_{J,i} \le \cdots \le \sigma_{J,N}$ be the N eigenvalues of $\mathbf{\Sigma}_J$. Then, the `jacov` score is defined as:

$$S = -\sum_{i=1}^{N} \left[\log(\sigma_{J,i} + k) + (\sigma_{J,i} + k)^{-1}\right]$$

Where $k = 10^{-5}$ is a constant for numerical stability.

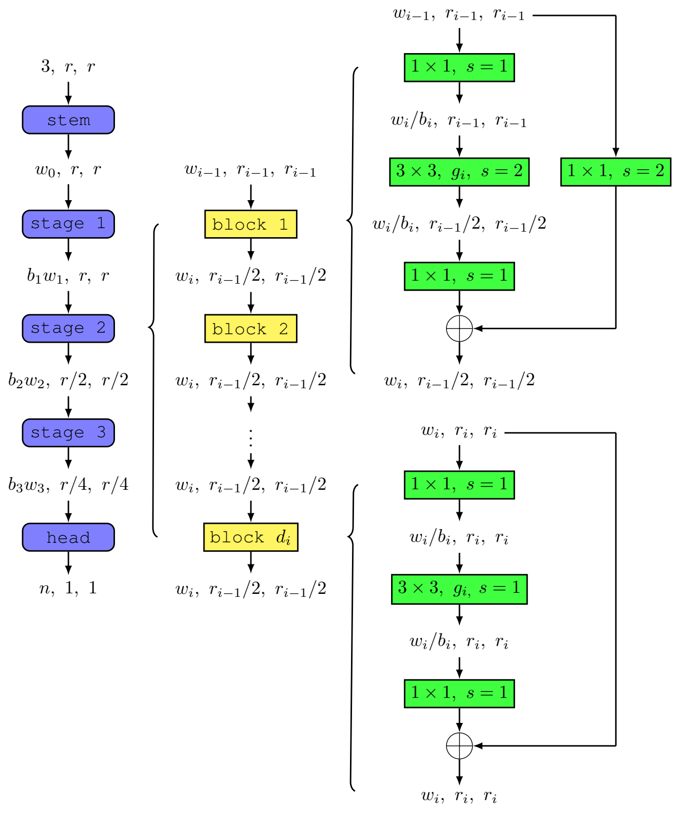

Fig. 1. An AnyNet starts with a stem, followed by a sequence of 3 stages, and then by a head that outputs the classification. The configuration of each stage is controlled by the following hyperparameters: d_i is the number of blocks of the ith stage, w_i is the number of filters, b_i is the bottleneck ratio, g_i is the number of parallel group convolutions and, n is the number of outputs (classes).

G. Logarithm of Determinant of Activation Patterns

This performance predictor was also proposed by Mellor et al. [14] in their final version of *Neural Architecture Search without Training* [13]. When an input signal flows through a neural network, a rectified linear unit (ReLU) may be activated (indicated as 1) or not (indicated as 0). Let c_i be the binary code of activation indicators for input x_i from a mini-batch of data $1 \le i \le N$. Let $d_H(c_i, c_j)$ be the Hamming distance between two codes, and let N_A be the total number of ReLUs in a given network. A kernel matrix $\mathbf{N}_H$ of activation patterns is constructed as follows:

$$\mathbf{N}_H = \begin{bmatrix} N_A - d_H(c_1, c_1) & \ldots & N_A - d_H(c_1, c_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(c_N, c_1) & \ldots & N_A - d_H(c_N, c_N) \end{bmatrix}$$

The `logdet` score is defined as the logarithm of the determinant of the kernel matrix, i.e., $S = \log|\mathbf{N}_H|$. It is worth noticing that the greater the difference in activation patterns, the greater the score.

IV. AnyNet design space

The AnyNet design space [19] can generate many different architectures based on ResNet [20] and ResNeXt [21] building blocks. Those architectures are very flexible and used in many computer vision tasks with good performance [30]. Figure 1 depicts a general AnyNet architecture for experiments on CIFAR datasets. An AnyNet consists of a linear sequence of convolutions followed by Batch Normalization (BN) and Rectified Linear Units (ReLU), as most deep convolutional networks.

Let the input be an image with $r \times r$ resolution and three color channels. It starts with an AnyNet stem: 3×3 convolution with w_0 filters and stride 1. Then, the AnyNet body performs the bulk of computation in a sequence of 3 stages. The ith stage has a sequence of d_i AnyNet blocks based on the residual bottleneck block with group convolutions [21]. They start with a 1×1 convolution with w_i/b_i filters, where b_i is the bottleneck ratio. It is followed by g_i parallel 3×3 convolutions, followed by a 1×1 convolution with w_i filters that are added to the block's input signal by the shortcut connection. Downsampling with stride 2 is performed in the first block of the second and third stages. Finally, the AnyNet head performs average pooling followed by a fully connected layer that predicts one of the n classes.

V. Methodology

A. Benchmark Dataset

The experiments are performed using the CIFAR-10 and CIFAR-100 datasets as baselines. Both contain $60,000$ images split into $50,000$ for training and $10,000$ for validation. The difference is in the number of classes since CIFAR-10 contains 10 classes and CIFAR-100 contains 100 classes [22]. These datasets were chosen because they remain challenging despite low compute requirements, and they are used by many NAS proposals [31]–[33].

B. AnyNet sampling and analysis

The AnyNet samples are generated as follows. All networks start with a fixed stem with $w_0 = 64$, followed by three stages of residual blocks, then a fixed head with global average pooling and a fully connected layer that performs the classification. The hyperparameters are sampled for each stage $1 \leq i \leq 3$ from uniform distributions with the following ranges:

- Depths: $1 \leq d_i \leq 16$;
- Group widths: $1 \leq g_i \leq 32$;
- Widths: $1 \leq w_i \leq 512$ divisible by g_i;
- Bottleneck ratios: $b_i \in \{1, 2, 4\}$;

where d_i is log-uniform, g_i and w_i are power-2-uniform. For a configuration with 3 stages, there are 12 hyperparameters and $(16.45.3)^3 \approx 10^{10}$ possible network models. It is also the initial cardinality of the subspace Ω for each predictor.

Following Radosavovic et al. [19], we only compare networks with similar complexities. Using the baseline complexity of ResNet-18 (40 MFLOPs), the random networks' complexities are limited between 10 and 50 MFLOPs regime. Suppose a random network is generated with lower (or higher) FLOPs. In that case, it is discarded, and a new sample is created until the desired number of random networks is obtained. After that, all networks are assessed with the aforementioned performance predictors (`grad_norm`, `snip`, `grasp`, `synflow`, `fisher`, `jacov`, and `logdet`). We observe the correlations between the predictors (Fig. 2), and we explore the relations between hyperparameters and predictors' scores using Decision Trees [34] to learn hyperparameters' rules that define top-scoring networks for each predictor (Fig. 3). This initial sample generates $5,000$ AnyNets with different architectures.

We used the implementation of zero-cost predictors provided by Mellor et al.[1] and by Abdelfattah et al.[2], and the AnyNet implementation provided by Radosavovic et al.[3]. The experiments were performed on a single NVIDIA A100 GPU, taking about two weeks for all results.

C. Automatic Evolution of AnyNet Subspaces

Instead of refining the AnyNet subspace manually as in [19], we automate the process by updating the sampling distribution iteratively using the hyperparameters' 95% confidence intervals (CIs) of the top-scoring networks. The CIs are computed according to [19] as follows.

Let (h_i, s_i) be a pair representing a hyperparameter h_i and a performance predictor score s_i of the ith network, for $1 \leq i \leq N$, where N is the sample size. The CIs are estimated using the empirical bootstrap: (1) resample with replacement 25% of the networks in the original sample, (2) select the pair (h_i, s_i) with the best score, (3) repeat this process $d = 10,000$ times, (4) compute the h_i's 2.5% and 97.5% percentiles. Those percentiles give the confidence interval limits, and the median is the most likely optimum value of the hyperparameter.

After computing the 95% CIs for all hyperparameters, we generate a new set of N random networks, updating the ranges of the uniform distributions using the CIs. This process is repeated until the CIs converge to a single value or the new sampling yields the same CIs as the previous one.

Therefore, starting with a general design space Ω_0, each iteration produces a subset $\Omega_j \subset \Omega_{j-1} \subset \cdots \subset \Omega_0$. As the number of iterations increases, that procedure should not be expected to produce subspaces that contain the global optimum networks since the probability p_j of the jth CI containing the optimum value is given by $p_j = 0.95^j$. Nonetheless, it will be empirically shown that this process improves each subset's overall score distribution (see Fig. 4). Furthermore, those subspaces reveal each predictor's preferences for different distributions of hyperparameters. Also, by using many samples ($N = 500$) and a conservative CI (95%), we intend to eliminate biases that might be introduced by NAS algorithms and focus on the direction induced by each performance predictor.

[1]https://github.com/BayesWatch/nas-without-training
[2]https://github.com/SamsungLabs/zero-cost-nas
[3]https://github.com/facebookresearch/pycls

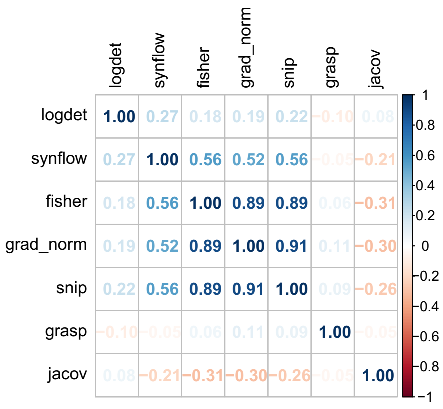

Fig. 2. Kendall's correlation between performance predictors.

D. Comparison Between AnyNet Subspaces

To evaluate the quality of the architectures generated by each performance predictor, we sample their final subspaces and perform full training. We generate 25 random networks for each final subspace and pick the top-scoring one. The models are fully trained using 300 epochs, batch size of 32, a learning rate of 0.025 decaying by 0.1 on epochs 150 and 225, weight-decay of 0.00005, standard gradient descent optimizer with Nesterov momentum of 0.9, label smoothing of 0.1, and standard augmentations for CIFAR-10 (normalization, 4 pixels padding, a random crop of 32x32 and, random horizontal flipping) [21]. The results are evaluated in terms of the error rate on the test dataset. We also compare these results with standards ResNet-18 [20] and ResNeXt-29 [21] of similar complexity.

VI. Results and Analysis

A. Trends in the initial AnyNet design space

In this section, we present the *hidden* design principle harvested by exploring the networks' scores of random sampling on the AnyNet space for the CIFAR-10 dataset. The predictor values distribution is the first observation about the $5,000$ AnyNet samples. Using Kolmogorov-Smirnov test [35], we found that all predictor values have exponential distributions with 95% significance. Therefore, the remaining analysis considers the natural logarithm for each predictor.

The relation between predictors' scores can determine if they lead to similar architecture parameters or not. Figure 2 shows Kendall's correlation between the performance predictor values. We see that `grad_norm`, `snip`, and `fisher` have similar preferences on the sampled space. Also, `synflow` has a small correlation with these metrics, `logdet` and `grasp` are not correlated, and `jacov` slightly disagrees with other metrics.

To understand what hyperparameters influence each predictor, we train decision trees considering depths d, widths w, group widths g, and bottleneck ratios b as inputs, and the predictor's scores (ranked as low, medium, or high) as outputs. The decision trees are trained with the 5,000 random networks at the initial search space. Figure 3 shows the decision trees obtained from each performance predictor. `grad_norm`, `snip`, `synflow`, and `fisher` assign higher scores to networks with higher depth on the last stage, i.e., their best 30% networks have $d_3 \geq 6$, although `synflow` differ regarding the decision rule for the medium and lower ranks. 48% of the best-evaluated networks by `grasp` have higher width on the last stage with $w_3 \geq 24$. 44% of `jacov`'s best networks have width $w_1 \geq 6$ on the first stage and depth $d_3 < 5$ on the last stage. At last, `logdet` high score is dependent mainly on a high number of residual blocks in the first stage, i.e., $d_1 \geq 3$.

B. Evolution of AnyNet subspaces

Figure 4 shows the evolution of the empirical distribution functions (EDF) caused by the automatic refinement of subspaces described in sec. V-C. Let S be a random variable representing a predictor's score. An EDF is a function such that $f(x) = P(S < x)$ estimated from the sample. As the area under the curve decreases, the probability of generating better networks increases significantly. The iterative processes also reduces the search space size $|\Omega|$ exponentially on both CIFAR-10 and CIFAR-100, as depicted in Figure 5.

The design principles (i.e., an optimum restriction on hyperparameters' values) obtained by each predictor are detailed in Table I as confidence intervals. Some hyperparameters converged to single value assignments. For instance, `synflow` assigned strict values for depth (d), group width (g), and bottleneck ratio (b) in all stages on both datasets. This provides empirical evidence that performance predictors favor very different design principles.

Figures 6, 7, 8 and 9 show the detailed evolution of CIs for each class of hyperparameters on CIFAR-10. The predictors `grad_norm`, `snip`, `grasp`, `synflow`, and `fisher` tend to maximize depths on all stages, whereas `jacov` rapidly converges to small depths. `logdet` diverges from the others by maximizing depths in the first stages and minimizing it in the last one. The predictors that prefer deeper networks choose

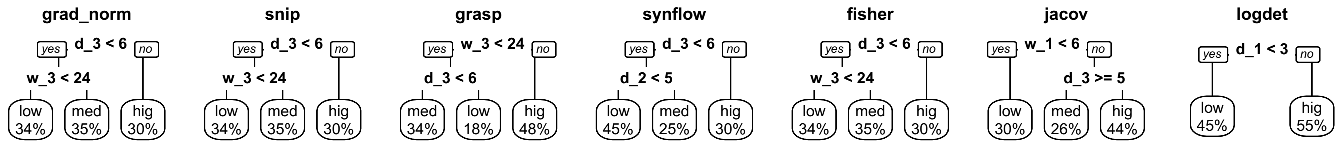

Fig. 3. Decision Trees trained to predict the score output using the hyperparameters as input. The levels low, medium and high are defined by ranking the scores. Leafs also show the percentage of networks that fall into each case.

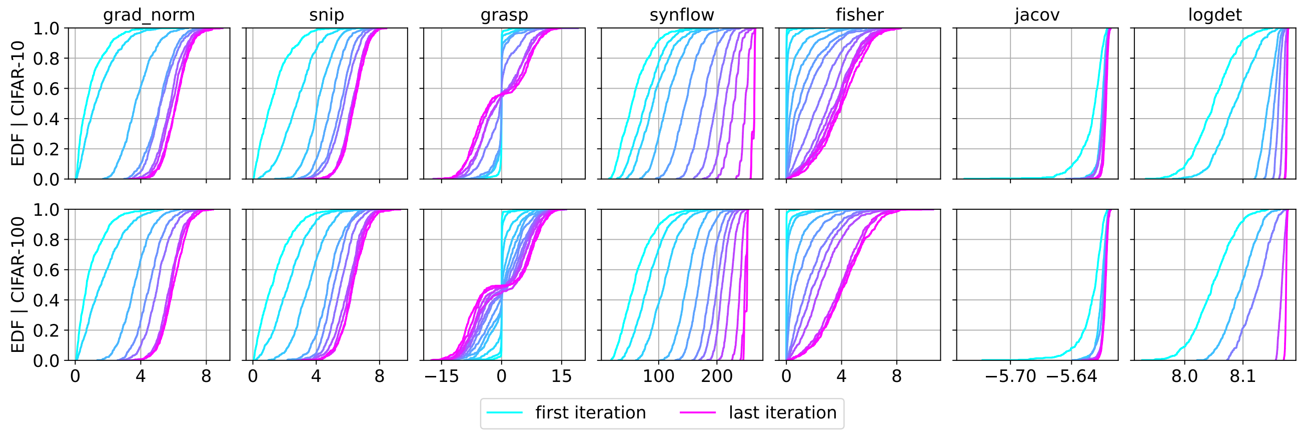

Fig. 4. Evolution of the EDFs using the iterated sampling method on CIFAR-10 and CIFAR-100. Each iteration results in a better score distribution. The x-axis scales the value of the scores.

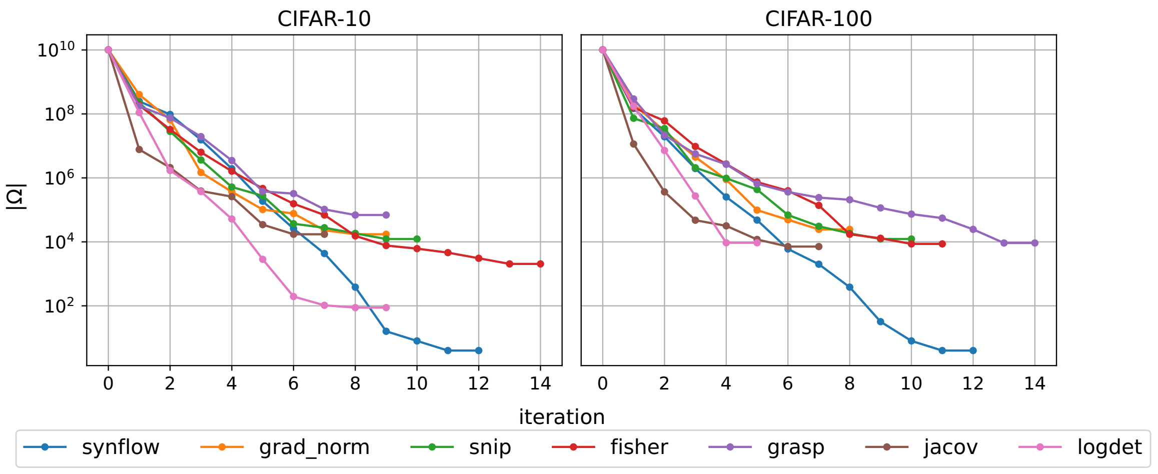

Fig. 5. Evolution of the subspace size Ω considering each zero-cost predictor along iterations on CIFAR-10 and CIFAR-100. These measurements point out all possibles networks considering a recursive refinement of 95% C.I. of best the hyperparameters estimated with empirical bootstrap.

TABLE I
UNCOVERED DESIGN PRINCIPLES FROM ANYNETS OPTIMIZED BY ZERO-COST PERFORMANCE PREDICTORS ON CIFAR-10 AND CIFAR-100.

Predictor	# iterations	d_1	d_2	d_3	w_1	w_2	w_3	g_1	g_2	g_3	b_1	b_2	b_3
CIFAR-10													
`grad_norm`	9	[15, 16]	[13, 16]	[11, 14]	[2, 8]	[2, 4]	[64, 128]	[1, 2]	[1, 2]	[1, 16]	[2, 4]	[1, 4]	[1, 2]
`snip`	10	[12, 16]	[13, 15]	[12, 14]	[4, 8]	2	[64, 128]	[1, 4]	[1, 2]	[1, 8]	[1, 4]	[1, 4]	[1, 4]
`grasp`	9	[9, 13]	[8, 11]	[10, 16]	[2, 4]	[2, 4]	[64, 128]	[1, 2]	[1, 4]	[1, 4]	[1, 4]	[1, 4]	[1, 4]
`synflow`	12	15	16	16	8	[16, 32]	[32, 64]	8	16	16	1	1	1
`fisher`	14	[12, 14]	[14, 16]	[13, 15]	2	2	[64, 128]	[1, 2]	[1, 2]	[1, 16]	[1, 4]	[1, 4]	[1, 4]
`jacov`	7	[2, 3]	[2, 3]	1	[32, 64]	[8, 64]	[16, 128]	[2, 8]	[2, 8]	[1, 16]	[1, 4]	[1, 4]	[1, 4]
`logdet`	9	13	12	[1, 5]	16	16	[2, 16]	1	1	[1, 4]	4	4	[1, 4]
CIFAR-100													
`grad_norm`	8	[12, 15]	[11, 15]	[12, 16]	[2, 4]	[2, 4]	[64, 128]	[1, 2]	[1, 2]	[1, 8]	[1, 4]	[1, 4]	1
`snip`	10	[12, 16]	[13, 16]	[10, 12]	[2, 4]	[2, 2]	[64, 128]	[1, 2]	[1, 2]	[2, 16]	[1, 4]	[1, 4]	[1, 2]
`grasp`	14	[9, 12]	[10, 13]	[13, 15]	[2, 4]	[2, 4]	[64, 128]	[1, 2]	[1, 2]	[2, 4]	[1, 4]	[1, 4]	[1, 4]
`synflow`	12	11	16	16	8	[16, 32]	[32, 64]	4	16	32	1	1	1
`fisher`	11	[13, 16]	[13, 15]	[12, 15]	2	2	[64, 128]	[1, 2]	[1, 2]	[1, 16]	[1, 4]	[1, 4]	[1, 4]
`jacov`	7	[2, 3]	[1, 2]	1	[16, 64]	[8, 32]	[2, 16]	[2, 8]	[2, 8]	[1, 4]	[1, 4]	[1, 4]	[1, 4]
`logdet`	5	16	[1, 7]	[1, 6]	16	[4, 16]	[2, 16]	1	[1, 2]	[1, 8]	4	[1, 4]	[1, 4]

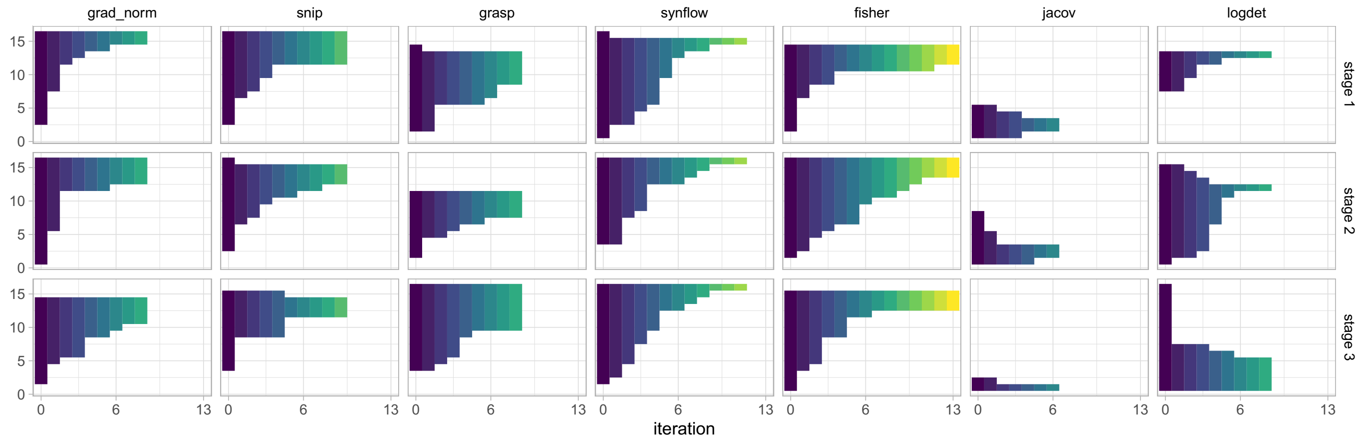

Fig. 6. Evolution of depths' design principles uncovered from networks optimized by zero-cost performance predictors on CIFAR-10.

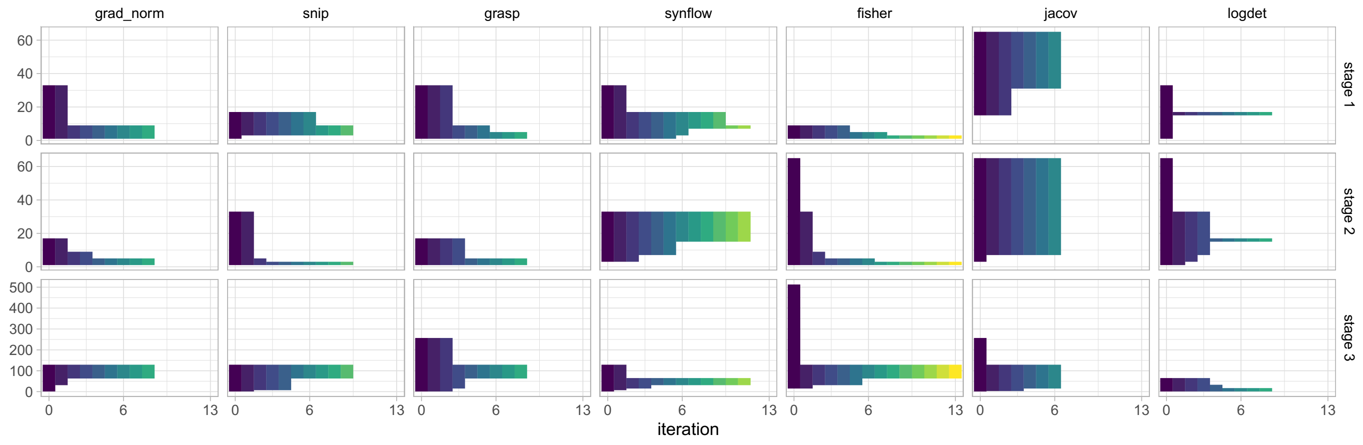

Fig. 7. Evolution of widths' design principles uncovered from networks optimized by zero-cost performance predictors on CIFAR-10.

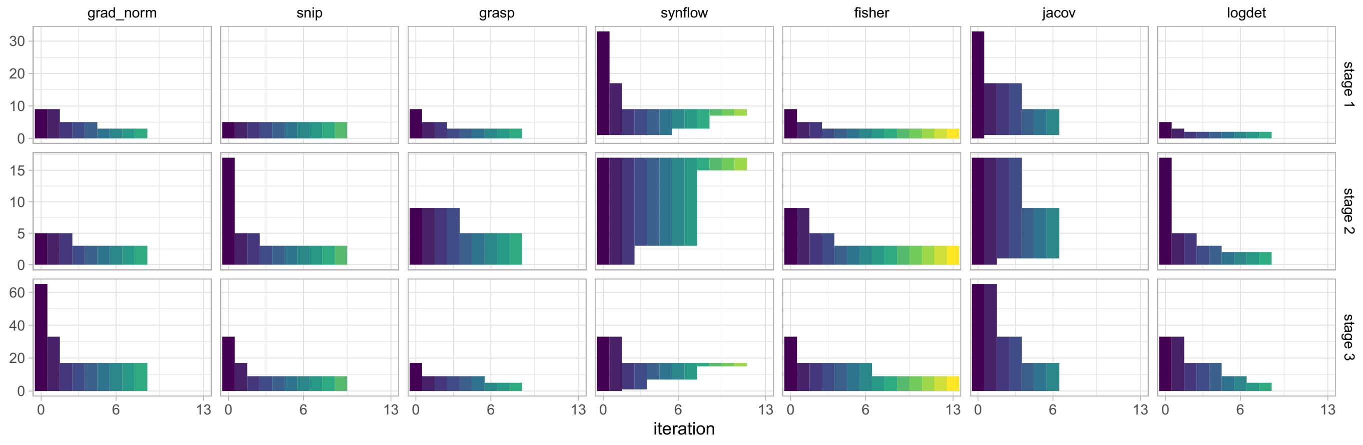

Fig. 8. Evolution of group widths' design principles uncovered from networks optimized by zero-cost performance predictors on CIFAR-10.

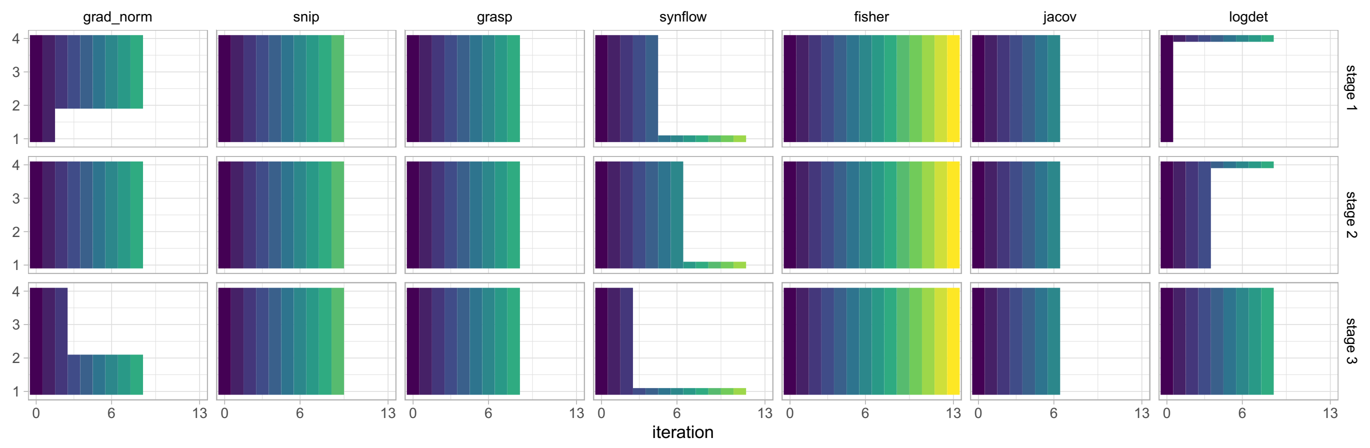

Fig. 9. Evolution of bottleneck ratios' design principles uncovered from networks optimized by zero-cost performance predictors on CIFAR-10.

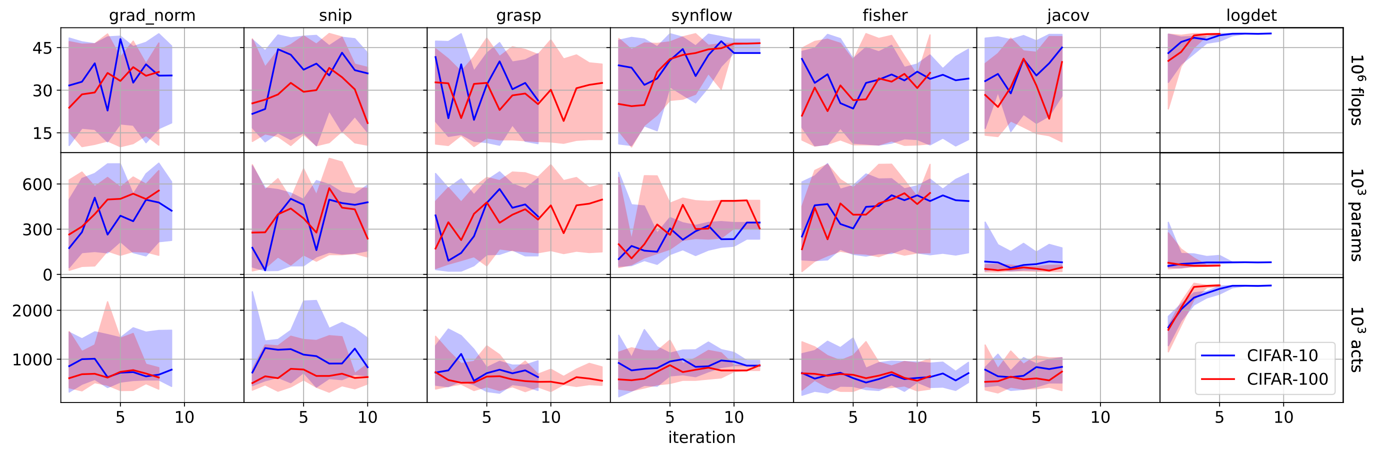

Fig. 10. Evolution of network complexities (i.e., FLOPs, number of parameters, and number of activations) for iterated sampling on CIFAR-10 and CIFAR-100. The solid line indicates the most likely optimum value and the shaded area indicates the 95% CI estimated with empirical bootstrap.

small width values in the first and second stages to compensate for complexity (Figure 7). Widths in the range $w_3 \in [64, 128]$ are most often perceived in the third stage for `grad_norm`, `snip`, `grasp` and `fisher`. Only `synflow` tends to double the width on each stage, as is usually done in ResNet architectures, since $w_1 = 8$, $w_2 \in [16,\ 32]$ and, $w_3 \in [32,\ 64]$. Contrary to the others, `jacov` prefers wider networks in the first stages with $w_1 \in [32,\ 64]$, and $w_2 \in [8,\ 64]$, and it does not converge to tight intervals. Regarding group widths, Figure 8 shows a noticeable difference for `jacov` converging to wider intervals, and for `synflow` defining $g_2 = 16$, which is a high value for the second stage compared to the others. In Figure 9, we see that most zero-cost predictors have no preferences for bottleneck ratios, except for `grad_norm` ($b_1 \in [2, 4]$ and $b_3 \in [1, 2]$), `synflow` ($b_1 = b_2 = b_3 = 1$), and `logdet` ($b_1 = b_2 = 4$).

Figure 10 shows the evolution of network complexities. Overall, most predictors have networks distributed across the FLOPs' budget. The exceptions are `synflow` and `logdet`, with `logdet` showing a strong tendency to increase the number of FLOPs to the budget's limit. `logdet` also showed a strong tendency to minimize the number of parameters and maximize the number of activations.

C. Performance evaluation of best subspaces

In the sequel, we perform a random search, then fully train the selected network according to the methodology in section V-D. Figure 11 shows the validation errors throughout the epochs for the sampled architectures from the final subspaces, as well as for ResNet and ResNeXt baselines of similar complexity. There is a great variation in the expected *performance* of the design principles obtained by each performance predictor. `synflow` yielded the best results comparable to the baselines. The other predictors yielded poor results. This result is consistent with Abdelfattah et al. [15], who reported that `synflow` was the most robust predictor across all datasets.

Table II shows the error rates, FLOPs, number of parameters, and number of activations for each architecture. ResNeXt-29 16x1 has the best trade-off between classification accuracy and network complexities, whereas `synflow` has the worst

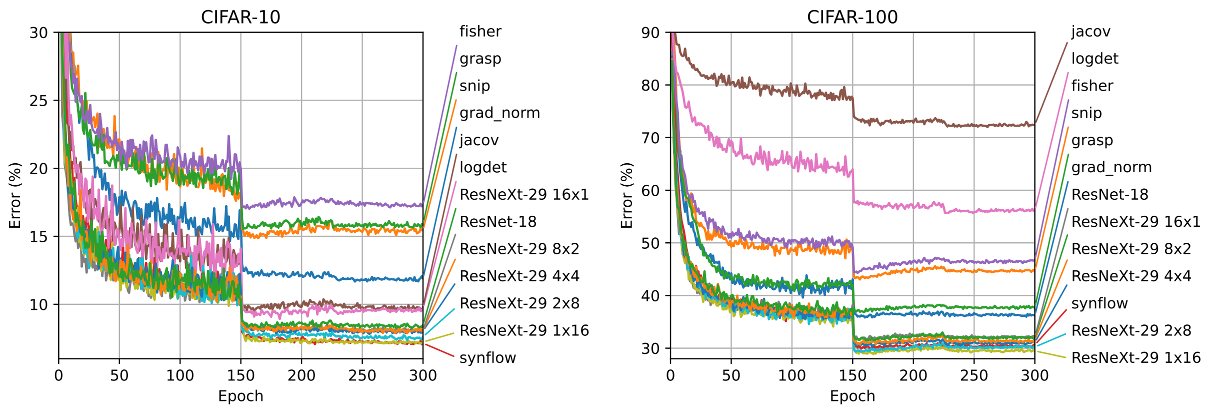

Fig. 11. Validation error on CIFAR-10 and CIFAR-100 of architectures created with the uncovered design principles for each performance predictor, and standard baseline architectures from the literature (ResNet and ResNeXt).

TABLE II
PERFORMANCE COMPARISON BETWEEN RESNET/RESNEXT BASELINES AND ARCHITECTURES CREATED WITH UNCOVERED DESIGN PRINCIPLES.

Architecture	CIFAR-10				CIFAR-100			
	Error (%)	FLOPs (10^6)	Parameters	Activations	Error (%)	FLOPs (10^6)	Parameters	Activations
`grad_norm`	11.9	45.60	667,214	636,938	36.21	31.42	468,596	562,276
`snip`	15.4	24.24	144,274	1,393,162	44.59	40.02	575,020	635,492
`grasp`	15.9	33.50	486,550	588,298	37.68	30.93	467,600	560,228
`synflow`	7.14	43.09	344,138	868,362	30.64	46.41	491,268	770,148
`fisher`	17.3	31.05	456,486	586,250	46.71	35.79	517,000	761,444
`jacov`	9.79	21.96	65,738	622,602	72.43	34.71	36,692	708,196
`logdet`	9.68	50.00	80,794	2,505,738	56.15	49.70	57,588	2,498,148
ResNet-18	8.11	40.81	272,474	200,714	32.07	40.82	278,324	200,804
ResNeXt-29 1x16	7.23	57.61	360,906	598,026	29.48	57.61	366,756	598,116
ResNeXt-29 2x8	7.62	36.37	215,754	598,026	30.23	36.38	221,604	598,116
ResNeXt-29 4x4	8.04	25.76	143,178	598,026	31.09	25.76	149,028	598,116
ResNeXt-29 8x2	8.06	20.45	106,890	598,026	31.55	20.45	112,740	598,116
ResNeXt-29 16x1	8.48	17.79	88,746	598,026	31.97	17.80	94,596	598,116

since it generates more costly models regarding FLOPs, number of parameters, and activations. This result is also consistent with `synflow`'s preference for large architectures reported by Ning et al. [16].

VII. Conclusion

This paper presented an approach to analyzing the relationship between zero-cost performance predictors and their hyperparameters' distribution, considering an iterative search space refinement for NAS. Our analysis aimed at getting insightful information about hyperparameter trends in the AnyNet design space, covering different ResNeXt-like models. We conducted experiments to evaluate the correlation between the performances of the zero-cost predictors to get an initial intuition about their scores and network properties.

We also analyzed the AnyNet subspaces evolution, considering an iterative sampling of network models using empirical bootstrap for the search space refinement. By observing the outcomes presented in Section VI, one could note the improvement of the predictors' scores as the subspace refinement provides convergent confidence intervals for the hyperparameters. Along all analyses performed, `synflow` converges for a specific model configuration as summarized in Table I and presented the best performance among predictors with competitive results when compared to the considered baselines, outperforming all of them on CIFAR-10 and outperforming ResNet-18, ResNeXt-29 16x1, ResNeXt-29 8x2, and ResNeXt-29 4x4 on CIFAR-100.

From all findings obtained with the developed research, we consider a deep study on the FLOPS and parameters regime an interesting issue for future works. The extension of these analyses with the inclusion of other zero-cost predictors — such as Neural Tangent Kernel [36] —, datasets — such as ImageNet — and search spaces — such as NATS-Bench [37] — are also targets for future works to evaluate the robustness and generalization ability of those predictors for different tasks and design principles, respectively.

The best zero-cost performance predictor was `synflow` both in terms of accuracy and robustness across the two datasets, although it yielded more complex networks. As Abdelfattah et al. [15] suggested, *"The most immediate open question for future investigation is why the `synflow` proxy works well — analytical insights will enable further research in zero-cost NAS proxies."* Based on our empirical results, there is a significant difference between `synflow`'s design principles compared to the other predictors, which we summarize as: bottleneck ratios $b_i = 1$, higher group widths, and a proportional progression in widths' intervals on each stage. However, these design principles are restricted to the AnyNet search space. Further research could uncover more general design principles when comparing different search spaces. We expect these analyses to help the understanding of zero-cost performance predictors and the development of novel strategies for NAS.

Acknowledgment

This work was partially supported by Samsung Eletrônica da Amazônia Ltda. under the Brazilian Informatics Law #8.387/91.

References

[1] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: a survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[2] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[3] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[4] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2423–2432.

[5] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

[6] Y. Xue, Y. Wang, J. Liang, and A. Slowik, "A self-adaptive mutation neural architecture search algorithm based on blocks," *IEEE Computational Intelligence Magazine*, vol. 16, no. 3, pp. 67–78, 2021.

[7] M. Javaheripi, M. Samragh, T. Javidi, and F. Koushanfar, "GeneCAI: genetic evolution for acquiring compact ai," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 350–358.

[8] A. Silva, L. Pavelski, L. Cordovil, P. Gomes, L. Azevedo, and F. Junior, "An evolutionary search algorithm for efficient ResNet-based architectures: a case study on gender recognition," in *IEEE Congress on Evolutionary Computation (CEC)*, 2022.

[9] N. Mitschke, M. Heizmann, K.-H. Noffz, and R. Wittmann, "Gradient based evolution to optimize the structure of convolutional neural networks," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 3438–3442.

[10] B. P. Evans, H. Al-Sahaf, B. Xue, and M. Zhang, "Genetic programming and gradient descent: A memetic approach to binary image classification," *arXiv preprint arXiv:1909.13030*, 2019.

[11] S. Yang, Y. Tian, C. He, X. Zhang, K. C. Tan, and Y. Jin, "A gradient-guided evolutionary approach to training deep neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[12] D. Zhou, X. Zhou, W. Zhang, C. C. Loy, S. Yi, X. Zhang, and W. Ouyang, "Econas: Finding proxies for economical neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 396–11 404.

[13] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," *arXiv preprint arXiv:2006.04647v1*, 2020. [Online]. Available: https://arxiv.org/abs/2006.04647v1

[14] ——, "Neural architecture search without training," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 7588–7598.

[15] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight NAS," in *International Conference on Learning Representations*, 2021.

[16] X. Ning, C. Tang, W. Li, Z. Zhou, S. Liang, H. Yang, and Y. Wang, "Evaluating efficient performance estimators of neural architectures," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 265–12 277, 2021.

[17] C. White, A. Zela, R. Ru, Y. Liu, and F. Hutter, "How powerful are performance predictors in neural architecture search?" *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 454–28 469, 2021.

[18] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollár, "On network design spaces for visual recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1882–1890.

[19] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 428–10 436.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[21] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1492–1500.

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

[23] N. Lee, T. Ajanthan, and P. Torr, "Snip: Single-shot network pruning based on connection sensitivity," in *International Conference on Learning Representations*, 2019.

[24] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," in *International Conference on Learning Representations*, 2020.

[25] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 6377–6389.

[26] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, "Faster gaze prediction with dense networks and fisher pruning," *arXiv preprint arXiv:1801.05787*, 2018.

[27] J. Turner, E. Crowley, M. O'Boyle, A. Storkey, and G. Gray, "Block-Swap: Fisher-guided block substitution for network compression on a budget," in *International Conference on Learning Representations*, 2020.

[28] L. Liu, S. Zhang, Z. Kuang, A. Zhou, J. Xue, X. Wang, Y. Chen, W. Yang, Q. Liao, and W. Zhang, "Group Fisher pruning for practical network compression," in *ICML*, 2021.

[29] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proceedings of the 5th International Conference on Neural Information Processing Systems*, ser. NIPS'92. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 164–171.

[30] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu, "Residual networks of residual networks: Multilevel residual networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 6, pp. 1303–1314, 2018.

[31] H. Tan, R. Cheng, S. Huang, C. He, C. Qiu, F. Yang, and P. Luo, "RelativeNAS: Relative neural architecture search via slow-fast learning," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021.

[32] Y. Weng, T. Zhou, L. Liu, and C. Xia, "Automatic convolutional neural architecture search for image classification under different scenes," *IEEE Access*, vol. 7, pp. 38 495–38 506, 2019.

[33] X. Zheng, R. Ji, Y. Chen, Q. Wang, B. Zhang, J. Chen, Q. Ye, F. Huang, and Y. Tian, "MIGO-NAS: Towards fast and generalizable neural architecture search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2936–2952, 2021.

[34] W.-Y. Loh, "Classification and regression trees," *WIREs Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.8

[35] H. W. Lilliefors, "On the Kolmogorov-Smirnov test for the exponential distribution with mean unknown," *Journal of the American Statistical Association*, vol. 64, no. 325, pp. 387–389, 1969.

[36] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[37] X. Dong, L. Liu, K. Musial, and B. Gabrys, "NATS-Bench: Benchmarking nas algorithms for architecture topology and size," *IEEE transactions on pattern analysis and machine intelligence*, 2021.