AZ-NAS: Assembling Zero-Cost Proxies for Network Architecture Search

Junghyup Lee[1] Bumsub Ham[1,2*]

[1]Yonsei University [2]Korea Institute of Science and Technology (KIST)

https://cvlab.yonsei.ac.kr/projects/AZNAS

Abstract

Training-free network architecture search (NAS) aims to discover high-performing networks with zero-cost proxies, capturing network characteristics related to the final performance. However, network rankings estimated by previous training-free NAS methods have shown weak correlations with the performance. To address this issue, we propose AZ-NAS, a novel approach that leverages the ensemble of various zero-cost proxies to enhance the correlation between a predicted ranking of networks and the ground truth substantially in terms of the performance. To achieve this, we introduce four novel zero-cost proxies that are complementary to each other, analyzing distinct traits of architectures in the views of expressivity, progressivity, trainability, and complexity. The proxy scores can be obtained simultaneously within a single forward and backward pass, making an overall NAS process highly efficient. In order to integrate the rankings predicted by our proxies effectively, we introduce a non-linear ranking aggregation method that highlights the networks highly-ranked consistently across all the proxies. Experimental results conclusively demonstrate the efficacy and efficiency of AZ-NAS, outperforming state-of-the-art methods on standard benchmarks, all while maintaining a reasonable runtime cost.

1. Introduction

Representative neural network architectures, including ResNets [23], MobileNets [24, 50], and Vision Transformers [17, 39], have been developed by experts through tedious trial-and-error processes, making it hard to design new architectures for various configurations. To overcome this problem, network architecture search (NAS) has emerged as a promising paradigm, capable of automatically seeking top-performing architectures under specified constraints. Recently, training-free NAS [1, 10, 35, 36, 41] has gained significant attention primarily due to its remarkable efficiency. It reduces computational and time costs of a NAS process drastically in comparison to earlier methods using an iterative training [3, 49] or training parameter-shared networks [6, 38, 47, 66].

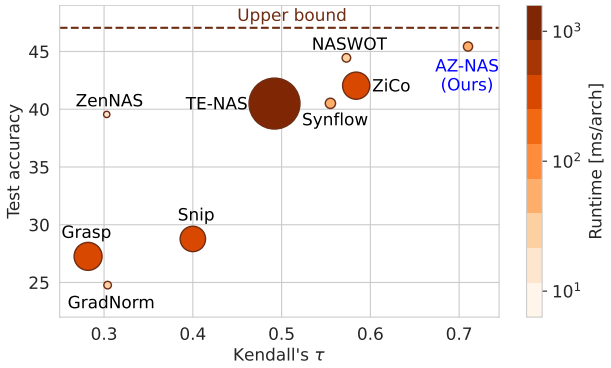


Figure 1. Comparison of training-free NAS methods on ImageNet16-120 of NAS-Bench-201 [15]. We compare correlation coefficients (Kendall's τ) between predicted rankings of networks and the ground truth in the x-axis, and test accuracies for the selected networks in the y-axis. The runtime costs are visualized by the circle size and color. By assembling the proposed zero-cost proxies, AZ-NAS achieves the best consistency between predicted rankings of the networks and the ground truth efficiently, which helps to find the network with the highest accuracy.

The training-free NAS methods predict the ranking of candidate networks in terms of performance via zero-cost proxies designed by empirical insights or theoretical evidences. These proxies analyze activations or gradients to capture *e.g.*, an ability to dissect an input space into linear regions [10, 41], saliency of weights/channels [1, 34, 56, 58], or training dynamics [10, 52]. However, they are often less practical than the training-based NAS methods mainly due to the weak correlations with the final performance of networks. For example, recent studies [35, 44] point out that the simple proxies based on the number of parameters (#Params) or floating-point operations (FLOPs) often provide better or competitive ranking consistency w.r.t the final performance of networks on the NAS benchmark [15], compared to many training-free NAS methods [1, 10, 34, 36, 41, 56, 58]. This limitation might arise from the fact that the current training-free NAS methods

*Corresponding author.

evaluate networks from narrow perspectives, typically relying on a single proxy only [1, 35, 36, 41, 65]. Considering a single network characteristic might not suffice to accurately predict the network ranking without training, since various factors can significantly influence the final performance. For instance, networks with a large number of parameters do not always achieve better performance due to *e.g.*, gradient vanishing or exploding problems, which could not be identified if we exploit #Params solely as a proxy.

In this paper, we introduce a novel training-free NAS method, AZ-NAS, that assembles multiple zero-cost proxies to evaluate networks from various perspectives. Naively assembling existing zero-cost proxies is, however, less effective due to the following reasons. First, it has been shown that gradient-based proxies [1, 34, 52, 58] are correlated with each other theoretically, providing similar NAS results [53]. This suggests that directly using an ensemble of them could prevent a comprehensive evaluation of networks, offering a limited improvement [44] while requiring additional computational costs. Second, several zero-cost proxies [10, 65] suffer from computational inefficiency, impeding scalability to large search spaces especially when they are combined with other proxies. To address these problems, we devise novel zero-cost proxies that can be obtained efficiently and capture unique network characteristics complementing each other, where the proxy scores show positive correlations with the network's performance. Each of them leverages activations, gradients, or architectural information (*i.e.*, FLOPs) to assess networks comprehensively in terms of expressivity, progressivity, trainability, and complexity. The proxy scores are computed simultaneously within a single forward and backward pass, making the entire NAS process efficient. We also present a non-linear ranking aggregation method to effectively merge the rankings provided by the proxies, preferring highly-ranked networks across all the proxies. We show that AZ-NAS achieves the best ranking consistency w.r.t the performance on NAS-Bench-201 [15], and it finds networks showing the state-of-the-art performance on the large-scale search spaces [9, 36, 50], with a reasonable runtime cost (Fig. 1), demonstrating its efficiency and effectiveness. We summarize the main contributions of our work as follows:

- We propose to assemble multiple zero-cost proxies for training-free NAS, assessing network architectures from various perspectives to predict a reliable ranking of the candidate architectures without training.
- We design novel zero-cost proxies tailored for AZ-NAS to capture distinct network characteristics, enabling a comprehensive evaluation of networks efficiently.
- We present a non-linear ranking aggregation method to combine rankings predicted from the proposed proxies, selecting a network highly-ranked across the proxies.
- We achieve the best NAS results on various search spaces, and provide extensive analyses that verify the efficacy and efficiency of AZ-NAS.

2. Related work

NAS automates designing novel network architectures by seeking for an optimal combination of operations [15, 38, 62] and/or the width and depth of networks [5, 36, 50] under limited FLOPs/#Params budgets. Early approaches adopt reinforcement learning [68, 69] or an evolutionary algorithm [49] that require training networks iteratively, making the NAS process computationally expensive.

To address this problem, one-/few-shot NAS methods [4–6, 25, 38, 66] use parameter-shared networks, called supernets, where each sub-path in the supernets corresponds a specific network architecture in a search space. They train a single [4–6, 38] or few [25, 66] supernets, and measure the accuracies of candidate networks by sampling corresponding sub-paths from the supernets, saving the computational costs associated with an iterative training process. Sharing parameters in supernets, however, often leads to a forgetting problem [63] or biased selections [12], requiring specialized training algorithms for supernets. Moreover, constructing supernets and storing the parameters cause significant memory requirements [4, 5].

Training-free NAS removes the network training process in the search phase. They use zero-cost proxies that can reflect the final performance, typically leveraging activations or gradients. Motivated by the finding that each ReLU function in a network divides an input space into two distinct pieces, called linear regions [20, 21], the works of [10, 41] examine activations to figure out the number of linear regions over the input space, which is useful for assessing the network's expressivity. These methods are however only applicable to networks employing ReLU non-linearities. ZenNAS [36] evaluates the expressivity of networks based on an expected Gaussian complexity [30], requiring additional forward passes with perturbed inputs. Inspired by network pruning [19], the work of [1] proposes to use the pruning-at-initialization metrics [34, 56, 58] for training-free NAS, estimating the importance of each weight parameter by analyzing its gradient. TE-NAS [10] exploits a neural tangent kernel (NTK) [29] to investigate training dynamics, and GradSign [65] attempts to search networks whose local optima obtained with different input samples are close to each other. They are relatively slower than other methods, since computing NTK [10] and sample-wise analysis [65] are computationally expensive, respectively. The works of [35, 54] analyze networks in terms of convergence and generalization by using gradient statistics computed from multiple forward and backward passes, favoring networks with gradients being large magnitudes and low variances. Despite the theoretical and empirical findings of all the aforementioned methods, the rankings of net-

works predicted by these methods often show weak correlations with the ground truth. For example, it has proven that most training-free NAS methods [1, 34, 36, 41, 56, 58] are not effective, compared to basic proxies using FLOPs and #Params [35, 44]. We conjecture that the reason for this limitation might be assessing networks with a single zero-cost proxy. We thus consider various proxies designed under distinct points of view, scoring networks comprehensively by leveraging activations, gradients, and FLOPs.

Similar to ours, the works of [1, 10, 44] combine zero-cost proxies to evaluate network architectures. TE-NAS [10] uses both the number of linear regions [20, 59] and the condition number of NTK [29, 32] to assess expressivity and trainability of networks, respectively. However, counting linear regions is infeasible for large networks [36], and calculating NTK is computationally demanding [45], making it difficult to apply TE-NAS to search spaces for large network architectures [5, 36, 50]. On the contrary, AZ-NAS remains computationally efficient while leveraging four zero-cost proxies jointly, retaining the advantage of training-free NAS. The works of [1, 44] propose to combine a few zero-cost proxies. However, simply combining the proxies is not sufficient for leveraging complementary features among the proxies [44]. For example, the work of [1] chooses multiple pruning-at-initialization metrics [34, 56] for the ensemble, but they capture similar network characteristics useful for pruning, *i.e.*, the saliency of weight parameters. Differently, we design zero-cost proxies to analyze networks from various perspectives, making them mutually complementary. We also integrate the proxies using a non-linear ranking aggregation method, assembling them effectively and boosting the NAS performance significantly.

3. Method

Our zero-cost proxies analyze activations and their gradients on *primary blocks* of a network, where each block consists of a set of layers with different types of operations and multiple paths. For example, we consider a cell structure of NAS-Bench-201 [15], a residual block [23], or an attention block of transformers [17, 57] as a primary block. In the following, we describe the zero-cost proxies of AZ-NAS (Sec. 3.1) and the non-linear ranking aggregation method (Sec. 3.2) in detail.

3.1. Zero-cost proxies of AZ-NAS

Expressivity. We evaluate the expressivity of a network by examining how uniformly features are distributed across the orientations in a feature space, given randomly initialized network weights and Gaussian random inputs. We refer to this as an isotropy of a feature space [7]. The stronger isotropy of the feature space at initialization implies that the features are less correlated to each other [13, 26, 27]. The network thus has more capacity to store various semantics during training, and it is likely to show high performance without suffering from *e.g.*, dead neurons [40] or a dimensional collapse problem [26]. Motivated by principal component analysis (PCA), we quantify the isotropy of a feature space based on the coefficients of principal components (PCs) for the space. Specifically, we denote by $f_l \in \mathbb{R}^{c\times n}$ c-dimensional output features of the l-th primary block, where n is the number of features. To obtain the coefficients of PCs, we first center the features as follows:

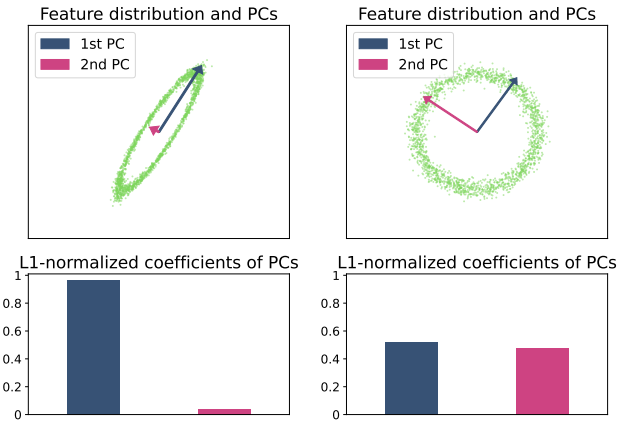
$$\bar{f}_l(p) = f_l(p) - \frac{1}{n}\sum_{q=1}^{n} f_l(q), \tag{1}$$

where we denote by $f_l(p) \in \mathbb{R}^{c\times 1}$ the p-th feature vector of f_l. We then compute a covariance matrix of the centered features as follows:

$$V_l = \frac{1}{n-1}\bar{f}_l\bar{f}_l^{\top}. \tag{2}$$

With the covariance matrix $V_l \in \mathbb{R}^{c\times c}$ at hand, we apply eigenvalue decomposition to obtain a set of coefficients for PCs, denoted by $\lambda_l \in \mathbb{R}^{c\times 1}$. The coefficients are proportional to the variances of corresponding PCs, implying the importance of each PC. If a feature space is dominated by few PCs, *e.g.*, due to the dimensional collapse [26], only corresponding coefficients are large, while the others become small (Fig. 2(a)). On the other hand, if features are distributed isotropically in a space, all PCs become equally important with similar coefficients (Fig. 2(b)). Based on this, we define the expressivity score of the l-th primary block, $s_l^{\mathcal{E}}$, as an entropy score, posing L1-normalized coefficients of PCs as probabilities:

$$s_l^{\mathcal{E}} = \sum_{i=1}^{c} -\tilde{\lambda}_l(i)\log\tilde{\lambda}_l(i), \tag{3}$$



(a) An example with $s^{\mathcal{E}} = 0.23$. (b) An example with $s^{\mathcal{E}} = 1.00$.

Figure 2. Toy examples for the expressivity score $s^{\mathcal{E}}$. In (a) and (b), we synthesize 2-dimensional features (green dots) using different covariances, and compare the L1-normalized coefficients of PCs. The features in (b) exhibit a higher expressivity score, forming an isotropic feature space.

where we denote by $\tilde{\lambda}_l$ a set of L1-normalized coefficients. We then obtain the expressivity score of a network $s^{\mathcal{E}}$ by summing up the block-level scores $s_l^{\mathcal{E}}$ over the primary blocks:

$$s^{\mathcal{E}} = \sum_{l=1}^{L} s_l^{\mathcal{E}}, \tag{4}$$

where L is the total number of primary blocks in a network. The expressivity proxy is particularly useful for detecting redundancy in a network, such as reducible channel dimensions caused by dead neurons.

Progressivity. The widths of modern network architectures [23, 50, 55] gradually increase for deeper blocks, making it possible to capture high-level semantics through deep features with large capacities. Building upon this, we propose a progressivity proxy that evaluates an ability to expand a feature space progressively according to the depth of primary blocks. We define the progressivity score $s^{\mathcal{P}}$ using the difference of the block-level expressivity scores in Eq. (3):

$$s^{\mathcal{P}} = \min_{l \in \{2,\cdots,L\}} s_l^{\mathcal{E}} - s_{l-1}^{\mathcal{E}}, \tag{5}$$

which computes the smallest difference in expressivity scores between neighboring primary blocks. A high progressivity score of a network indicates that its block-level expressivity scores consistently increase, at least by the value of the progressivity score, along the depth of the primary blocks. Our progressivity proxy automates the NAS process by considering how features evolve throughout a forward pass explicitly in terms of expressivity, without requiring strict and manual constraints on the width of a network *e.g.*, as in [51].

Trainability. The seminal work of [42] has shown that gradients can be backpropagated without diverging or vanishing, only when the spectral norm of a Jacobian matrix for each layer is close to 1. Motivated by this, we design a trainability score with the spectral norm of a Jacobian matrix. This evaluates the stable propagation of gradients at initialization, which has proven to be crucial for high-performing networks [18, 22, 48]. However, different from the previous work [42] dealing with a simple network architecture, we should consider complicated structures of primary blocks consisting of various operations and multiple paths across layers (*e.g.*, a cell structure [15]). In this case, obtaining a Jacobian matrix is not a trivial task. To circumvent this issue, we devise an efficient approximation of the Jacobian matrix of a primary block, and propose to use the approximation to compute the trainability score. We first simplify the forward pass ψ_l of the l-th primary block by approximating it as a linear system that inputs c'-dimensional features $f_{l-1} \in \mathbb{R}^{c' \times n}$ and outputs c-dimensional features $f_l \in \mathbb{R}^{c \times n}$:

$$f_l = \psi_l(f_{l-1}) \approx A_l f_{l-1}, \tag{6}$$

where $A_l \in \mathbb{R}^{c \times c'}$ is a linear transformation matrix. Under this approximation, the backward pass ϕ_l of the primary block can be represented as follows:

$$g_{l-1} = \phi_l(g_l) \approx A_l^\top g_l, \tag{7}$$

where $g_{l-1} \in \mathbb{R}^{c' \times n}$ and $g_l \in \mathbb{R}^{c \times n}$ are the gradients of f_{l-1} and f_l, respectively, and the transformation matrix A_l serves as a Jacobian matrix. Motivated by the Hutchinson's method [2] used in [16, 33, 61], we introduce a Rademacher random vector $v \in \mathbb{R}^{c \times 1}$, whose elements are randomly drawn from $\{-1, 1\}$ with a equal probability, to obtain the Jacobian matrix A_l in the approximation. Specifically, we rewrite Eq. (7) by substituting the output gradient with the Rademacher random vector v:

$$u = \phi_l(v) \approx A_l^\top v, \tag{8}$$

where $u \in \mathbb{R}^{c' \times 1}$ is considered as a gradient propagated from v. By using the property that $\mathbb{E}\left[vv^\top\right]$ is an identity matrix [33, 61], we have the following relationship:

$$\mathbb{E}\left[uv^\top\right] \approx \mathbb{E}\left[A_l^\top vv^\top\right] = A_l^\top \mathbb{E}\left[vv^\top\right] = A_l^\top. \tag{9}$$

That is, if we can compute u in Eq. (8) by feeding the Rademacher random vector v into the function ϕ_l, we can compute $A_l^\top$ using $\mathbb{E}\left[uv^\top\right]$. Based on this, we compute the Jacobian matrix A_l in the approximation of Eq. (7) as follows:

$$A_l^\top = \frac{1}{n} \sum_{p=1}^{n} g_{l-1}(p) g_l^\top(p), \tag{10}$$

where we synthesize the output gradients g_l with Rademacher random vectors and obtain the input gradients g_{l-1} by an automatic backpropagation tool [46]. Consequently, we define the trainability score $s^{\mathcal{T}}$ as follows:

$$s^{\mathcal{T}} = \frac{1}{L-1} \sum_{l=2}^{L} -\sigma_l - \frac{1}{\sigma_l} + 2, \tag{11}$$

which has a maximum value, when the spectral norm of A_l, denoted by σ_l, is equal to 1 for all l. In Eq. (11), both the spectral norm σ_l and its reciprocal value contribute equally to the score. This penalizes, for example, the spectral norms σ_l of 2 and $\frac{1}{2}$ to the same extent, equivalently treating the increasing and decreasing scale differences relative to 1. The proposed trainability proxy provides distinct advantages compared to previous zero-cost proxies using gradients. First, it is relatively faster than others [10, 34, 35, 56, 58, 65], and the score can be obtained simultaneously with other proxy scores of AZ-NAS (*i.e.*, expressivity and progressivity) with the same input sample. Second, our trainability proxy is not tied to specific operations, since it computes the score using input and output gradients of any types of primary blocks. In contrast,

existing gradient-based proxies [1, 10, 34, 35, 56, 58, 65] exploit gradients of trainable weight parameters, suggesting that they cannot evaluate the blocks consisting of non-parametric operations (*e.g.*, a cell with average pooling) directly.

Complexity. Recent studies [35, 44] have shown that architectural characteristics related to hardware resources, such as #Params or FLOPs, are correlated closely with the performance of networks. Following this observation, we propose to use FLOPs itself as a complexity score $s^{\mathcal{C}}$, preferring the networks that maximally use computational resources within a given budget.

3.2. Non-linear ranking aggregation

A straightforward way for integrating the proxies into the final scores would be a linear summation of rankings [10]. If we have both lowly- and highly-ranked proxies from a network, the linear aggregation makes it difficult to highlight an undesirable characteristic captured by the lowly-ranked proxy. This is because the low ranking of the proxy could be easily offset by the highly-ranked proxies. We use a non-linear ranking aggregation method to combine our proxies more effectively. Specifically, we denote by $\mathbb{S}^{\mathcal{M}} = [s^{\mathcal{M}}(1), \cdots, s^{\mathcal{M}}(m)]$ a set of scores for the proxy $\mathcal{M} \in \{\mathcal{E}, \mathcal{P}, \mathcal{T}, \mathcal{C}\}$, obtained with m candidate architectures. The non-linear ranking aggregation outputs the final AZ-NAS score for the i-th network $s^{\text{AZ}}(i)$ as follows:

$$s^{\text{AZ}}(i) = \sum_{\mathcal{M} \in \{\mathcal{E},\, \mathcal{P},\, \mathcal{T},\, \mathcal{C}\}} \log \frac{\text{Rank}\left(s^{\mathcal{M}}(i)\right)}{m}, \qquad (12)$$

where $\text{Rank}(\cdot)$ assigns the ranking of an input score $s^{\mathcal{M}}(i)$ over the scores in the set $\mathbb{S}^{\mathcal{M}}$ in ascending order. It penalizes the final score more severely when one of the proxies indicates subpar performance, allowing us to find a network highly-ranked across all the proxies evenly. As will be shown in the ablation study (Sec. 4.3), this approach substantially boosts the ranking consistency between the final AZ-NAS scores and the performance, compared to the linear aggregation of rankings. When we perform an evolutionary search, we apply the non-linear ranking aggregation method to the proxy scores obtained with all candidate architectures. For generating a new candidate during the search, we mutate one of the top-k network architectures in terms of the final AZ-NAS scores (Algorithm 1).

4. Experiment

In this section, we describe experimental settings (Sec. 4.1) and compare AZ-NAS with the state of the art (Sec. 4.2). We then present an ablation study and an analysis on integrating other zero-cost proxies (Sec. 4.3). More results can be found in the supplement.

Algorithm 1 Evolutionary search using AZ-NAS.

Input: search space Z; number of NAS iteration T; computational budget B; number of top-scoring architectures for mutation k.
Output: selected architecture F^*.

1: Initialize the first architecture F_1 for an evolutionary search.
2: Initialize empty history sets for storing architectures $\mathbb{F}$ and proxy scores $\mathbb{S}^{\mathcal{E}}$, $\mathbb{S}^{\mathcal{P}}$, $\mathbb{S}^{\mathcal{T}}$, and $\mathbb{S}^{\mathcal{C}}$.
3: **while** $i = 1$ **to** T **do**
4: Compute the proxy scores $s^{\mathcal{E}}$, $s^{\mathcal{P}}$, $s^{\mathcal{T}}$, and $s^{\mathcal{C}}$ of the architecture F_i.
5: Append the architecture F_i and the scores $s^{\mathcal{E}}$, $s^{\mathcal{P}}$, $s^{\mathcal{T}}$, $s^{\mathcal{C}}$ to the corresponding history sets $\mathbb{F}$, $\mathbb{S}^{\mathcal{E}}$, $\mathbb{S}^{\mathcal{P}}$, $\mathbb{S}^{\mathcal{T}}$, and $\mathbb{S}^{\mathcal{C}}$, respectively.
6: Compute the AZ-NAS scores of the architectures in $\mathbb{F}$ using the non-linear ranking aggregation with the proxy scores in $\mathbb{S}^{\mathcal{E}}$, $\mathbb{S}^{\mathcal{P}}$, $\mathbb{S}^{\mathcal{T}}$, and $\mathbb{S}^{\mathcal{C}}$.
7: Generate a new candidate architecture F_{i+1} belonging to the search space Z by mutating one of the top-k architectures based on the AZ-NAS scores, within the computational budget B.
8: **end while**
9: Select the architecture F^* showing the highest AZ-NAS score among the ones in the history set $\mathbb{F}$.

4.1. Experimental settings

We perform extensive experiments on the NAS-Bench-201 [15], MobileNetV2 [36, 50] and AutoFormer [9] search spaces. During the search phase, we measure the proxy scores of AZ-NAS using a single batch of Gaussian random inputs with a batch size of 64. In the following, we describe experimental settings for each search space in detail.

NAS-Bench-201. NAS-Bench-201 [15] consists of 15625 network architectures, each of which uses a unique cell structure. It provides the diagnostic information *e.g.*, test accuracies and training losses on CIFAR-10/100 [31] and ImageNet16-120 (IN16-120) [11]. We measure the correlation coefficients in terms of Kendall's τ and Spearman's ρ between predicted and ground-truth network rankings across all the candidate architectures. We also report the top-1 test accuracies of selected networks, averaged over 5 runs, together with standard deviations. For each run, we randomly sample 3000 architectures, and discover the best one based on the AZ-NAS score.

MobileNetV2. We follow the configuration of the MobileNetV2 [50] search space proposed in [36] for a fair comparison with previous works [35, 36]. This space includes candidate architectures built with inverted residual blocks [50], with varying depth, width, and expansion ratio of the block. We perform an evolutionary search in Al-

Table 1. Quantitative comparison of the training-free NAS methods on NAS-Bench-201 [15]. We categorize the types of zero-cost proxies into architectural (A), backward (B), and forward (F) ones depending on the inputs of the proxies. We report Kendall's τ (KT) and Spearman's ρ (SPR) computed with all candidate architectures, together with an average runtime. We also provide the average and standard deviation of test accuracies (Acc.) on each dataset, where they are obtained through 5 random runs. To this end, we randomly sample 3000 candidate architectures for each run and share the same architecture sets across all the methods. All results are reproduced with the official codes provided by the authors.

Method	Type	CIFAR-10			CIFAR-100			ImageNet16-120			Runtime (ms/arch)
		KT	SPR	Acc.	KT	SPR	Acc.	KT	SPR	Acc.	
#Params	A	0.578	0.753	93.50 ± 0.17	0.552	0.728	70.63 ± 0.29	0.520	0.691	41.91 ± 0.70	-
FLOPs	A	0.578	0.753	93.50 ± 0.17	0.551	0.727	70.63 ± 0.29	0.517	0.691	41.91 ± 0.70	-
GradNorm [1]	B	0.357	0.484	90.13 ± 1.00	0.350	0.475	62.95 ± 1.38	0.304	0.412	24.77 ± 4.67	28.8
Grasp [1, 58]	B	0.318	0.460	89.49 ± 1.63	0.315	0.453	61.72 ± 2.97	0.282	0.406	27.26 ± 4.92	395.9
Snip [1, 34]	B	0.454	0.615	90.32 ± 1.39	0.451	0.609	63.44 ± 2.52	0.400	0.539	28.77 ± 5.74	326.5
Synflow [1, 56]	B	0.571	0.769	93.06 ± 0.83	0.565	0.761	69.24 ± 2.07	0.555	0.747	40.51 ± 5.70	53.4
NASWOT [41]	F	0.557	0.743	92.67 ± 0.24	0.579	0.769	69.91 ± 0.27	0.573	0.760	44.45 ± 0.83	36.9
TE-NAS [10]	B+F	0.536	0.731	92.30 ± 0.33	0.535	0.728	67.87 ± 1.05	0.492	0.680	40.49 ± 2.16	1311.8
ZenNAS [36]	F	0.296	0.385	90.30 ± 0.36	0.283	0.361	67.70 ± 1.00	0.303	0.409	39.55 ± 1.27	19.9
GradSign [65]	B	0.618	0.809	93.52 ± 0.19	0.594	0.784	70.57 ± 0.31	0.575	0.765	41.89 ± 0.69	1823.9
ZiCo [35]	B	0.589	0.784	93.50 ± 0.18	0.590	0.785	70.62 ± **0.26**	0.584	0.778	42.04 ± 0.82	372.8
AZ-NAS (Ours)	A+B+F	**0.741**	**0.913**	**93.53 ± 0.15**	**0.723**	**0.900**	**70.75** ± 0.48	**0.710**	**0.886**	**45.43 ± 0.29**	42.7
Ground truth	-	-	-	94.29 ± 0.13	-	-	73.25 ± 0.26	-	-	47.05 ± 0.30	-

gorithm 1 to find the optimal networks, while setting the number of iterations T and k to 1e5 and 1024, respectively. Following [35], we search for networks under FLOPs constraints of 450M, 600M, and 1000M. We train the selected networks on ImageNet [14] using the same training setting as in [35, 36], and report the top-1 validation accuracy.

AutoFormer. The AutoFormer [9] search space is designed to evaluate the NAS methods specialized to Vision Transformers (ViTs)[1]. It contains ViT architectures with variable factors of depth, embedding dimension, number of attention heads, and expansion ratio for multi-layer perceptrons. The search space is further divided into the Tiny, Small, and Base subsets according to the model sizes. We select a ViT architecture with the best AZ-NAS score for each subset, among 10000 randomly chosen architectures, similar to [67]. We train and evaluate the selected ViTs on ImageNet following the configuration in [9], except for the one from the Base subset, for which we reduce the number of training epochs as in [67] to avoid overfitting.

4.2. Results

NAS-Bench-201. We show in Table 1 a quantitative comparison between AZ-NAS and the state-of-the-art training-free NAS methods on NAS-Bench-201 [15]. We report correlation coefficients between predicted and ground-truth rankings of networks, and top-1 test accuracies of selected networks. We can see that AZ-NAS achieves the best ranking consistency w.r.t the performance in terms of Kendall's τ and Spearman's ρ, outperforming others by significant margins across all the datasets. This enables discovering networks that consistently show better performance compared to the networks chosen from other methods. The training-free NAS methods in Table 1 focus on capturing a single network characteristic using either activations or gradients only, except for TE-NAS [10] requiring lots of computational costs. On the contrary, AZ-NAS examines networks from various perspectives based on activations, gradients, and FLOPs, providing a comprehensive evaluation of networks. Moreover, the proposed proxies tailored for AZ-NAS are computationally efficient and can be computed simultaneously within a single forward and backward pass. Note that several training-free NAS methods exploit specialized techniques for implementation, such as removing batch normalization [56], analyzing activations associated only with ReLU non-linearities [10, 41], or iterating multiple forward and/or backward passes [10, 35, 36, 65]. In contrast, AZ-NAS performs without additional complex operations or modifications to network architectures, assessing the networks efficiently without bells and whistles.

MobileNetV2. We compare in Table 2 the performance of various NAS methods in terms of the top-1 validation accuracy on ImageNet [14]. We can observe from the table that AZ-NAS provides the best NAS results, achieving the highest accuracies across all FLOPs constraints. It even outperforms the training-based NAS approaches (MS and OS) typically requiring expensive search costs, demonstrating its effectiveness and efficiency. AZ-NAS also pro-

[1]We do not apply the progressivity proxy for the AutoFormer search space, since we empirically find that attention modules in ViTs produce similar attention values across tokens with Gaussian random inputs at initialization. In this case, output features of the attention modules thus tend to become close in a feature space, which could not guarantee the feature space to be expanded along the depth. Developing zero-cost proxies suitable for ViTs could enhance the NAS performance of AZ-NAS for ViTs, and we leave this for a future work.

Table 2. Quantitative comparison of networks chosen by the NAS methods on ImageNet [14] in terms of the top-1 validation accuracy. We group the networks with similar FLOPs (*i.e.*, 450M, 600M, and 1000M) and categorize the NAS methods into multi-shot (MS), one-shot (OS), and zero-shot (ZS) ones, according to the number of networks to train during the search phase. For each FLOPs constraint, we report the results of AZ-NAS averaged over three random runs starting from the search phase. All numbers except ours are taken from [35].

Method	FLOPs	Top-1 acc.	Type	Search cost (GPU days)
450M				
NASNet-B [69]	488M	72.8	MS	1800
CARS-D [60]	496M	73.3	MS	0.4
BN-NAS [8]	470M	75.7	MS	0.8
OFA [6]	406M	77.7	OS	50
RLNAS [64]	473M	75.6	OS	-
DONNA [43]	501M	78.0	OS	405
# Params	451M	63.5	ZS	0.02
ZiCo [35]	448M	78.1	ZS	0.4
AZ-NAS (Ours)	462M $\pm$ 1.5M	**78.6** $\pm$ 0.2	ZS	0.4
600M				
PNAS [37]	588M	74.2	MS	224
CARS-I [60]	591M	75.2	MS	0.4
DARTS [38]	574M	73.3	OS	4
ProxylessNAS [5]	595M	76.0	OS	8.3
OFA [6]	662M	78.7	OS	50
RLNAS [64]	597M	75.9	OS	-
DONNA [43]	599M	78.4	OS	405
ZenNAS [36]	611M	79.1	ZS	0.5
ZiCo [35]	603M	79.4	ZS	0.4
AZ-NAS (Ours)	615M $\pm$ 2.2M	**79.9** $\pm$ 0.3	ZS	0.6
1000M				
sharpDARTS [28]	950M	76.0	OS	-
ZenNAS [36]	934M	80.8	ZS	0.5
ZiCo [35]	1005M	80.5	ZS	0.4
AZ-NAS (Ours)	1022M $\pm$ 5.1M	**81.1** $\pm$ 0.1	ZS	0.7

vides high-performing networks consistently over multiple random runs with different seed numbers. Note that the previous training-free NAS methods [35, 36] apply an evolutionary search using MobileNetV2-styled networks, while removing residual connections. They focus on analyzing vanilla convolutional neural networks [36] during the search, where multiple convolutional layers are simply stacked without additional connections. The residual connections are then restored to train selected networks, potentially leading to a discrepancy between the expected and final performance at the search and training phases, respectively. In contrast, AZ-NAS does not rely on such techniques, improving the practicality and avoiding the discrepancy issue. We additionally provide in the supplement a comprehensive comparison with the training-free NAS methods [35, 36] in terms of reproducibility and a search cost, under the fair settings for network search and training.

AutoFormer. To validate the generalization ability of AZ-NAS on ViT architectures, we perform experiments on the AutoFormer [9] search space. We provide in Table 3 a quantitative comparison to the state-of-the-art NAS methods [9, 67], specially designed for ViTs, in terms of the validation accuracy on ImageNet [14]. For each subset of the search space (*i.e.*, Tiny, Small, and Base), we find two ViTs with the #Params constraints similar to AutoFormer [9] and TF-TAS [67], separately, for a fair comparison. We can see that AZ-NAS discovers ViTs showing better performance compared to other methods in most cases, with significantly lower search costs. This suggests that our method is not limited to a specific type of network architecture, demonstrating the generalization ability of our proxies.

Table 3. Quantitative comparison for the AutoFormer [9] search space. We report top-1 validation accuracies of selected networks on ImageNet [14]. All numbers are taken from corresponding papers, except for #Params and FLOPs of TF-TAS [67]. We measure these numbers using the same source code as AutoFormer for a fair comparison.

Method	#Params	FLOPs	Top-1 acc.	Type	Search cost (GPU days)
Tiny					
AutoFormer [9]	5.70M	1.30G	74.7	OS	24
AZ-NAS (Ours)	5.92M	1.38G	**76.1**	ZS	0.03
TF-TAS [67]	6.20M	1.43G	75.3	ZS	0.5
AZ-NAS (Ours)	6.16M	1.43G	**76.4**	ZS	0.04
Small					
AutoFormer [9]	22.9M	5.10G	81.7	OS	24
AZ-NAS (Ours)	23.0M	4.94G	**82.0**	ZS	0.06
TF-TAS [67]	23.9M	5.16G	81.9	ZS	0.5
AZ-NAS (Ours)	23.8M	5.13G	**82.2**	ZS	0.07
Base					
AutoFormer [9]	54.0M	11.0G	**82.4**	OS	24
AZ-NAS (Ours)	53.7M	11.4G	82.1	ZS	0.11
TF-TAS [67]	56.5M	11.9G	82.2	ZS	0.5
AZ-NAS (Ours)	54.1M	11.5G	**82.3**	ZS	0.17

4.3. Discussion

Ablation study. We provide in Table 4 an ablation study on each component of AZ-NAS. We evaluate the NAS performance on NAS-Bench-201 [15] using various combinations of the proposed proxies and the ranking aggregation methods. We summarize our findings as follows: (1) We can see from ① to ④ in Table 4 that exploiting a single proxy does not provide satisfactory results. Aggregating more proxies improves the NAS performance substantially, outperforming most state-of-the-art NAS methods in Table 1, even when two of the proxies in AZ-NAS are selected in ⑤ to ⑧. These results demonstrate the effectiveness of assembling various zero-cost proxies, confirming the importance of a comprehensive evaluation from various perspectives for training-free NAS. (2) By comparing ⑪ and ⑫ (or ⑬ and ⑭), we can clearly see that the non-linear ranking aggregation successfully boosts the ranking consistency w.r.t the performance. The aggregation is particularly useful with more proxies, verifying its significance in AZ-

Table 4. Ablation study for the zero-cost proxies of AZ-NAS on NAS-Bench-201 [15]. When multiple proxies are used, we integrate them into the final scores using either the linear (L) or non-linear (NL) ranking aggregation (RA) methods. We compare the ranking consistency w.r.t the performance in terms of Kendall's τ.

	$s^{\mathcal{E}}$	$s^{\mathcal{P}}$	$s^{\mathcal{T}}$	$s^{\mathcal{C}}$	RA	CIFAR-10	CIFAR-100	IN16-120
①	✓				-	0.569	**0.563**	0.506
②		✓			-	0.521	0.508	0.489
③			✓		-	0.349	0.353	0.407
④				✓	-	**0.578**	0.551	**0.517**
⑤	✓	✓			NL	0.601	0.590	0.547
⑥	✓		✓		NL	0.635	0.631	0.639
⑦	✓			✓	NL	**0.674**	**0.653**	0.601
⑧			✓	✓	NL	0.629	0.615	**0.644**
⑨	✓	✓	✓		NL	0.679	0.673	0.669
⑩	✓	✓		✓	NL	0.683	0.661	0.616
⑪	✓		✓	✓	NL	**0.731**	**0.714**	**0.708**
⑫	✓		✓	✓	L	0.706	0.692	0.678
⑬	✓	✓	✓	✓	NL	**0.741**	**0.723**	**0.710**
⑭	✓	✓	✓	✓	L	0.697	0.681	0.663

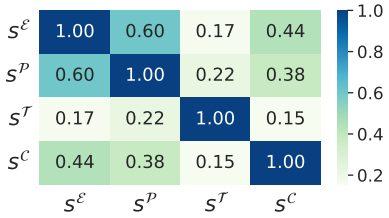


Figure 3. Correlation analysis on the zero-cost proxies of AZ-NAS. We report Kendall's τ between the estimated network rankings on ImageNet16-120 of NAS-Bench-201 [15].

NAS. (3) The trainability proxy alone might not be effective as shown in ③, since it mainly focuses on the stable gradient propagation, without considering *e.g.*, a network's capacity to learn. Nevertheless, we can see from ⑤ to ⑪ that the trainability proxy helps others to achieve strong ranking consistency, especially on ImageNet16-120. For an in-depth analysis, we show in Fig. 3 how the proposed proxies are correlated with each other on ImageNet16-120. We can see that the trainability proxy is less correlated with others. Based on these observations, we can conclude that coupling less correlated proxies improves the NAS performance more significantly (*e.g.*, the results on ImageNet16-120 in ⑥ and ⑧). This is because such proxies tend to capture unique network characteristics, making them mutually complementary and leading to a synergy effect. AZ-NAS fully leverages the complementary features among the proxies, boosting the NAS performance drastically.

Assembling other zero-cost proxies. AZ-NAS is built upon the idea that assembling various zero-cost proxies, similar to ensemble learning, could bring better performance for training-free NAS. To further verify this idea, we incorporate the proposed proxies into existing ones [1, 35, 56] and show in Table 5 the results on NAS-Bench-201 [15]. We can see that an ensemble of our zero-cost proxies and the previous one improves the ranking consistency w.r.t the performance consistently, suggesting that the basic idea of AZ-NAS is also applicable to other proxies. Similar to the observation in the ablation study, we can obtain better ranking consistency by exploiting more proxies. In particular, we can further boost the performance in terms of Kendall's τ by aggregating all of our proxies and the previous one, at the cost of the additional runtime.

Table 5. Quantitative comparison of incorporating our zero-cost proxies with existing ones. We adopt our non-linear ranking aggregation method to estimate the predicted network rankings. We report the correlation coefficients (Kendall's τ) between predicted and ground-truth network rankings on NAS-Bench-201 [15], together with average runtimes.

Aggregated zero-cost proxies	CIFAR-10	CIFAR-100	IN16-120	Runtime (ms/arch)
$s^{\mathcal{C}}$	0.578	0.551	0.517	-
$s^{\mathcal{C}} + s^{\mathcal{E}}$	0.674	0.653	0.601	22.3
$s^{\mathcal{C}} + s^{\mathcal{E}} + s^{\mathcal{T}}$	0.731	0.714	0.708	42.7
$s^{\mathcal{C}} + s^{\mathcal{E}} + s^{\mathcal{T}} + s^{\mathcal{P}}$	0.741	0.723	0.710	42.7
ZiCo [35]	0.589	0.590	0.584	372.8
ZiCo + $s^{\mathcal{C}}$	0.632	0.615	0.595	372.8
ZiCo + $s^{\mathcal{C}} + s^{\mathcal{E}}$	0.733	0.717	0.678	395.1
ZiCo + $s^{\mathcal{C}} + s^{\mathcal{E}} + s^{\mathcal{T}}$	0.761	0.749	0.743	415.5
ZiCo + $s^{\mathcal{C}} + s^{\mathcal{E}} + s^{\mathcal{T}} + s^{\mathcal{P}}$	0.773	0.757	0.747	415.5
Synflow [1, 56]	0.571	0.565	0.555	53.4
Synflow + $s^{\mathcal{C}}$	0.636	0.616	0.594	53.4
Synflow + $s^{\mathcal{C}} + s^{\mathcal{E}}$	0.741	0.721	0.681	75.7
Synflow + $s^{\mathcal{C}} + s^{\mathcal{E}} + s^{\mathcal{T}}$	0.768	0.753	0.746	96.1
Synflow + $s^{\mathcal{C}} + s^{\mathcal{E}} + s^{\mathcal{T}} + s^{\mathcal{P}}$	0.776	0.758	0.747	96.1

5. Conclusion

We have presented AZ-NAS, a training-free NAS approach that assembles various zero-cost proxies to substantially enhance the NAS performance. To this end, we have designed novel zero-cost proxies from distinct and complementary perspectives. We have also proposed to integrate them into a final score effectively with a non-linear ranking aggregation technique. Extensive experiments clearly demonstrate the efficiency and efficacy of AZ-NAS, surpassing previous training-free NAS methods with a reasonably fast runtime. We expect that AZ-NAS can provide a plug-and-play solution, allowing other NAS methods to maximize the performance by seamlessly incorporating our zero-cost proxies during the search, with a minimal increase in the computational cost.

Acknowledgments. This work was partly supported by IITP grants funded by the Korea government (MSIT) (No.RS-2022-00143524, Development of Fundamental Technology and Integrated Solution for Next-Generation Automatic Artificial Intelligence System, No.2022-0-00124, Development of Artificial Intelligence Technology for Self-Improving Competency-Aware Learning Capabilities) and the KIST Institutional Program (Project No.2E31051-21-203).

References

[1] Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. Zero-cost proxies for lightweight NAS. In *Int. Conf. Learn. Represent.*, 2021. 1, 2, 3, 5, 6, 8

[2] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 58(2):1–34, 2011. 4

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *Int. Conf. Learn. Represent.*, 2017. 1

[4] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *Int. Conf. Mach. Learn.*, pages 550–559, 2018. 2

[5] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *Int. Conf. Learn. Represent.*, 2019. 2, 3, 7

[6] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-All: Train one network and specialize it for efficient deployment. In *Int. Conf. Learn. Represent.*, 2020. 1, 2, 7

[7] Xingyu Cai, Jiaji Huang, Yuchen Bian, and Kenneth Church. Isotropy in the contextual embedding space: Clusters and manifolds. In *Int. Conf. Learn. Represent.*, 2021. 3

[8] Boyu Chen, Peixia Li, Baopu Li, Chen Lin, Chuming Li, Ming Sun, Junjie Yan, and Wanli Ouyang. BN-NAS: Neural architecture search with batch normalization. In *Int. Conf. Comput. Vis.*, pages 307–316, 2021. 7

[9] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. AutoFormer: Searching transformers for visual recognition. In *Int. Conf. Comput. Vis.*, pages 12270–12280, 2021. 2, 5, 6, 7

[10] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on ImageNet in four GPU hours: A theoretically inspired perspective. In *Int. Conf. Learn. Represent.*, 2021. 1, 2, 3, 4, 5, 6

[11] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of ImageNet as an alternative to the CIFAR datasets. *arXiv preprint arXiv:1707.08819*, 2017. 5

[12] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search. In *Int. Conf. Comput. Vis.*, pages 12239–12248, 2021. 2

[13] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. In *Int. Conf. Learn. Represent.*, 2016. 3

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 248–255, 2009. 6, 7

[15] Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Int. Conf. Learn. Represent.*, 2020. 1, 2, 3, 4, 5, 6, 7, 8

[16] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. In *Adv. Neural Inform. Process. Syst.*, pages 18518–18529, 2020. 4

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Int. Conf. Learn. Represent.*, 2021. 1, 3

[18] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010. 4

[19] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Adv. Neural Inform. Process. Syst.*, 2015. 2

[20] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *Int. Conf. Mach. Learn.*, pages 2596–2604, 2019. 2, 3

[21] Boris Hanin and David Rolnick. Deep ReLU networks have surprisingly few activation patterns. In *Adv. Neural Inform. Process. Syst.*, 2019. 2

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Int. Conf. Comput. Vis.*, pages 1026–1034, 2015. 4

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 770–778, 2016. 1, 3, 4

[24] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1

[25] Shoukang Hu, Ruochen Wang, Hong Lanqing, Zhenguo Li, Cho-Jui Hsieh, and Jiashi Feng. Generalizing few-shot NAS with gradient matching. In *Int. Conf. Learn. Represent.*, 2022. 2

[26] Tianyu Hua, Wenxiao Wang, Zihui Xue, Sucheng Ren, Yue Wang, and Hang Zhao. On feature decorrelation in self-supervised learning. In *Int. Conf. Comput. Vis.*, pages 9598–9608, 2021. 3

[27] Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 791–800, 2018. 3

[28] Andrew Hundt, Varun Jain, and Gregory D Hager. sharpDARTS: Faster and more accurate differentiable architecture search. *arXiv preprint arXiv:1903.09900*, 2019. 7

[29] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Adv. Neural Inform. Process. Syst.*, 2018. 2, 3

[30] Sham M Kakade, Karthik Sridharan, and Ambuj Tewari. On the complexity of linear prediction: Risk bounds, margin bounds, and regularization. In *Adv. Neural Inform. Process. Syst.*, 2008. 2

[31] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009. 5

[32] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Adv. Neural Inform. Process. Syst.*, 2019. 3

[33] Junghyup Lee, Dohyung Kim, and Bumsub Ham. Network quantization with element-wise gradient scaling. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 6448–6457, 2021. 4

[34] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *Int. Conf. Learn. Represent.*, 2019. 1, 2, 3, 4, 5, 6

[35] Guihong Li, Yuedong Yang, Kartikeya Bhardwaj, and Radu Marculescu. ZiCo: Zero-shot NAS via inverse coefficient of variation on gradients. In *Int. Conf. Learn. Represent.*, 2023. 1, 2, 3, 4, 5, 6, 7, 8

[36] Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-NAS: A zero-shot nas for high-performance image recognition. In *Int. Conf. Comput. Vis.*, pages 347–356, 2021. 1, 2, 3, 5, 6, 7

[37] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Eur. Conf. Comput. Vis.*, pages 19–34, 2018. 7

[38] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Int. Conf. Learn. Represent.*, 2019. 1, 2, 7

[39] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Int. Conf. Comput. Vis.*, pages 10012–10022, 2021. 1

[40] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying ReLU and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019. 3

[41] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *Int. Conf. Mach. Learn.*, pages 7588–7598, 2021. 1, 2, 3, 6

[42] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *Int. Conf. Learn. Represent.*, 2018. 4

[43] Bert Moons, Parham Noorzad, Andrii Skliar, Giovanni Mariani, Dushyant Mehta, Chris Lott, and Tijmen Blankevoort. Distilling optimal neural networks: Rapid search in diverse spaces. In *Int. Conf. Comput. Vis.*, pages 12229–12238, 2021. 7

[44] Xuefei Ning, Changcheng Tang, Wenshuo Li, Zixuan Zhou, Shuang Liang, Huazhong Yang, and Yu Wang. Evaluating efficient performance estimators of neural architectures. In *Adv. Neural Inform. Process. Syst.*, pages 12265–12277, 2021. 1, 2, 3, 5

[45] Roman Novak, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Fast finite width neural tangent kernel. In *Int. Conf. Mach. Learn.*, pages 17018–17044, 2022. 3

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Adv. Neural Inform. Process. Syst.*, 2019. 4

[47] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *Int. Conf. Mach. Learn.*, pages 4095–4104, 2018. 1

[48] Xianbiao Qi, Jianan Wang, Yihao Chen, Yukai Shi, and Lei Zhang. LipsFormer: Introducing lipschitz continuity to vision transformers. In *Int. Conf. Learn. Represent.*, 2023. 4

[49] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, pages 4780–4789, 2019. 1, 2

[50] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4510–4520, 2018. 1, 2, 3, 4, 5

[51] Xuan Shen, Yaohua Wang, Ming Lin, Yilun Huang, Hao Tang, Xiuyu Sun, and Yanzhi Wang. DeepMAD: Mathematical architecture design for deep convolutional neural network. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 6163–6173, 2023. 4

[52] Yao Shu, Shaofeng Cai, Zhongxiang Dai, Beng Chin Ooi, and Bryan Kian Hsiang Low. NASI: Label-and data-agnostic neural architecture search at initialization. In *Int. Conf. Learn. Represent.*, 2022. 1, 2

[53] Yao Shu, Zhongxiang Dai, Zhaoxuan Wu, and Bryan Kian Hsiang Low. Unifying and boosting gradient-based training-free neural architecture search. In *Adv. Neural Inform. Process. Syst.*, pages 33001–33015, 2022. 2

[54] Zihao Sun, Yu Sun, Longxing Yang, Shun Lu, Jilin Mei, Wenxiao Zhao, and Yu Hu. Unleashing the power of gradient signal-to-noise ratio for zero-shot NAS. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5763–5773, 2023. 2

[55] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Int. Conf. Mach. Learn.*, pages 6105–6114, 2019. 4

[56] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Adv. Neural Inform. Process. Syst.*, pages 6377–6389, 2020. 1, 2, 3, 4, 5, 6, 8

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Adv. Neural Inform. Process. Syst.*, 2017. 3

[58] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *Int. Conf. Learn. Represent.*, 2020. 1, 2, 3, 4, 5, 6

[59] Huan Xiong, Lei Huang, Mengyang Yu, Li Liu, Fan Zhu, and Ling Shao. On the number of linear regions of convolutional neural networks. In *Int. Conf. Mach. Learn.*, pages 10514–10523, 2020. 3

[60] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. CARS: Continuous evolution for efficient neural architecture search.

In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1829–1838, 2020. 7

[61] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. PyHessian: Neural networks through the lens of the Hessian. In *Int. Conf. Learn. Represent. Workshop*, 2020. 4

[62] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Int. Conf. Mach. Learn.*, pages 7105–7114, 2019. 2

[63] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, and Steven Su. Overcoming multi-model forgetting in one-shot NAS with diversity maximization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7809–7818, 2020. 2

[64] Xuanyang Zhang, Pengfei Hou, Xiangyu Zhang, and Jian Sun. Neural architecture search with random labels. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10907–10916, 2021. 7

[65] Zhihao Zhang and Zhihao Jia. GradSign: Model performance inference with theoretical insights. In *Int. Conf. Learn. Represent.*, 2022. 2, 4, 5, 6

[66] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *Int. Conf. Mach. Learn.*, pages 12707–12718, 2021. 1, 2

[67] Qinqin Zhou, Kekai Sheng, Xiawu Zheng, Ke Li, Xing Sun, Yonghong Tian, Jie Chen, and Rongrong Ji. Training-free transformer architecture search. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10894–10903, 2022. 6, 7

[68] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *Int. Conf. Learn. Represent.*, 2017. 2

[69] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8697–8710, 2018. 2, 7