Understanding Few-Shot Neural Architecture Search with Zero-Cost Proxies

Timotée LY-MANSON[1] Mathieu LÉONARDON[1] Abdeldjalil AISSA EL BEY[1]
[1]IMT Atlantique, UMR CNRS 6285 Lab-STICC, 29238 Brest, France

Résumé – Dans cet article, nous développons le sujet de la recherche d'architectures neurales (NAS) avec poids partagés. Suivant les travaux sur « few-shot NAS » dans lesquels des supernets contenant toutes les architectures de l'espace de recherche sont partitionés de manière itérative, nous introduisons un environnement pour la division efficace des supernets en utilisant des métriques « sans coût ». Nous étudions le comportement de diverses métriques dans cet environnement pour mieux comprendre les principes généraux de la division des supernets.

Abstract – In this paper, we expand on Neural Architecture Search (NAS) in the weight-sharing setting. Following the few-shot NAS line of work, where supernets that contain every architecture in the search space are iteratively partitioned, we introduce a framework for efficient splitting of supernets using zero-cost metrics. We study the behavior of various metrics within this framework and gain insight on general principles of supernet splitting.

1 Introduction

Neural networks have recently seen great success in many tasks from various fields including computer vision and language. Finding the right neural network architecture for a given task typically requires careful tuning by human experts. Neural Architecture Search (NAS) is the field dedicated to automating neural network architecture design, whose aim is to alleviate the need for expert tuning. Early NAS methods leverage reinforcement learning [14, 17] or evolutionary algorithms [13] with unreasonable computation costs. Thus, recent efforts have been directed towards reducing the search costs of NAS. One-shot methods [12] are designed to tackle this issue.

While one-shot methods [4, 6, 10] have successfully reduced the time cost and complexity of NAS using the weight-sharing paradigm, they are criticized for being poor proxies to the performance of real architectures [2, 15]. Few-shot NAS [16] is proposed as an answer by partitioning the search space into smaller parts, and is further enhanced by learning these partitions [7].

In parallel, so-called zero-shot NAS methods [1, 3, 11] have had increasing success. These methods remove the need to train any network by guiding the search phase of NAS using zero-cost proxies that estimate the relevance of a candidate architecture using a single batch of data.

In the present paper, we introduce a framework to iteratively split supernets using any metric. We implement several zero-cost proxies as metrics within this framework and study the behavior of supernet splits. In particular, we introduce the split distance as way to measure the consistency of metrics across splitting iterations.

2 Related Works

2.1 One-shot NAS

Weight sharing [12] is introduced as a technique to compress the prohibitive training times of other NAS methods. The need to train many architectures in order to obtain a label indicative of their performance is reduced to the training of a single model, the supernet, which enables the joint training of every architecture in the search space. Typically, in a one-shot framework, architectures in the search space are built from a cell represented as a directed acyclic graph, where the nodes are intermediate features while the edges are candidate operations. The output at the k-th cell is given by :

$$\bar{o}_k(x) = \sum_{i=1}^{N} \sum_{j=1}^{i} o_{i,j}(x, w_{i,j})$$

where N is the number of nodes in the cell, $o_{i,j}$ is the operation situated on the edge between the i-th and j-th nodes, w are the parameters of the model, x is the input. The cell is then repeated and stacked K times to form a complete architecture. In contrast, the supernet's edges are a superposition of all possible operations in the operation space $\mathcal{O}$:

$$\bar{o}_k(x) = \sum_{i=1}^{N} \sum_{j=1}^{i} \sum_{o \in \mathcal{O}} o(x, w_{i,j})$$

Many classical methods such as SPOS [6] or FairNAS [4] treat the supernet as a proxy to evaluate architecture performance and guide the search. At each search iteration, an architecture is sampled using each method's specific strategy. Then the weights learned by the supernet are copied to the sampled architecture, which is evaluated without any further retraining. However, while one-shot NAS can divide search costs by up to 1000 compared to evaluating each candidate from scratch, the performance of the final selected architectures suffer a noticeable decrease. As showcased in [15], one-shot algorithms may only be marginally better than random search for a similar time budget. Furthermore, by evaluating and ranking every architecture in a given search space, they find that rankings made by the supernet proxy are poorly correlated with ground truth rankings.

Following these previous observations, *co-adaptation* is introduced in [2] as an intuitive reason for supernets' short-

comings. During supernet training, weights are optimized for joint use of the operations, as opposed to the standalone setup in candidate architectures.As a result, the most important operation on a given edge might become under-optimized and not be chosen in the subsequent search phase.

2.2 Few-shot NAS

Few-shot NAS [16] is proposed as a way to counteract co-adaptation and increase performance while still benefiting from the low complexity of the one-shot framework. After training the supernet, an edge is chosen and the supernet is split exhaustively into sub-supernets along this edge. For every sub-supernet, there is a single operation along the target edge, removing any co-adaptation effects.

These sub-supernets are fine-tuned for several epochs, then the top k sub-supernets are selected and the architecture search is conducted over these. Each sub-supernet can further be splitted along a new edge to increase performance, resulting in a supernet splitting tree with the full supernet at its root.

Expanding on few-shot NAS, GM-NAS [7] hypothesizes that the influence of co-adaptation is heavily dependant on which operations are grouped together. Thus, splits should be learned in such a way that operations which cooperate well together remain joined in the sub-supernet, as opposed to splitting exhaustively over operations. To this end, the *gradient matching* metric is introduced, which quantifies the similarity of the effects of two operations on the gradients of the sub-supernet :

$$GM_k(o_i, o_j) = S_C[\nabla_w(\mathcal{L}(m^k_{\Omega\setminus o_i}, w)), \nabla_w(\mathcal{L}(m^k_{\Omega\setminus o_j}, w))]$$

where S_C is the cosine similarity function, $\mathcal{L}$ is a loss function and $m^k_{\Omega\setminus o}$ is a supernet with operation o disabled on edge k.

In practice, temporary models with a single disabled operation over the target edge are created. The pairwise gradient matching values are stored in a matrix. Finally, operation groups are created by min cut optimization on the matrix.

Differently from few-shot NAS, GM-NAS does not fine-tune sub-supernets starting from the weights of the supernet, as co-adaptation effects might already exist at this point. Instead, sub-supernets are trained from scratch. Because gradient matching based splits lead to fewer sub-supernets than exhaustive splits, it is less costly to evaluate deeper levels of the splitting tree.

2.3 Zero-shot NAS

While weight sharing in one-shot frameworks can significantly reduce the cost of NAS, it cannot go below the cost of training the supernet. In contrast, zero-shot NAS methods that completely bypass training at any stage of the search have been proposed. These methods rely on zero-cost proxies to guide the search, which can be computed using a single batch of input data from a freshly initialized model.

In [1], the authors use a naive baseline proxy `gradnorm`, and saliency metrics inspired by the neural network pruning literature, namely `snip`, `grasp` and `synflow`. These zero-cost proxies can be used to warmup or guide a reinforcement learning or evolutionary-based search algorithm. In [11], it is proposed to form a matrix from the binary codes of activations and use it to estimate the best architecture in the space in a greedy fashion. We hereafter refer to this metric as `jacobcov`.

Another idea explored in [3] is to combine the strengths of two zero-cost proxies and iteratively prune a supernet into a single-path network. Intuitively, the condition number of the Neural Tangent Kernel (NTK) [8] is a proxy for the *trainability* of the model, while the number of linear regions estimates the *expressivity* of the model. The two metrics are normalized and summed to select which operations to prune at each step. These metrics are hereafter referred to as `ntk` and `lr`.

3 Methods

3.1 Supernet splitting framework

Algorithm 1: *Constant* policy

Let $\mathcal{S}$ a supernet, N a target number of leaf sub-supernets, b a branching factors, m a metric
$\mathbf{S} = \{\mathcal{S}\}$
$n \leftarrow 0$
while $n \times b \leq N$ **do**
  Split every element of $\mathbf{S}$ into b sub-supernets with metric m and obtain $\bar{\mathbf{S}} = \{\bar{\mathcal{S}}_i\}_{[1, card(\mathbf{S})\times b]}$
  $\mathbf{S} \leftarrow \bar{\mathbf{S}}$
  $n \leftarrow card(\mathbf{S})$
end

Few-shot NAS [16] and GM-NAS [7] introduce the supernet splitting problem. Given an initial supernet representing a search space, we seek to sequentially select edges and split operations along these edges in such a way to minimize co-adaptation. Following the foundations laid out by GM-NAS, we leverage metric evaluation of the sub-supernets as the main heuristic for deterministic splits. To this end, we employ various metrics introduced in the zero-shot NAS literature. We generalize GM-NAS as a framework with three distinct components :

Splitting policy. Given the supernet as input, the splitting policy constructs the complete tree of sub-supernets by defining at each stage the edge to split along and a branching factor - the number of operation groups to form. The groups are created by solving a min-cut optimization over a distance matrix formed using any metric.

Training. Following GM-NAS, every leaf sub-supernet from the splitting tree is trained from scratch. While some warmup epochs can occur between splits as part of the splitting policy, resetting weights ensures that leaves do not suffer from any co-adaptation effects inherited from the supernet.

Search. Search for an optimal architecture is conducted over leaf sub-supernets. The type of optimization to conduct (reinforcement learning, evolution, Bayesian optimization) is a core part of this component. A heuristic to choose which sub-supernet to explore should also be included.

We leave the design of search algorithms tailored to the few-shot framework to future work. In this work, we focus on establishing a splitting policy and evaluating the influence of various metrics on the splitting tree.

We present the naive *constant* policy (Algorithm 1), where sub-supernets are iteratively splitted following a constant branching factor at each step. Considering its simplicity and proximity with GM-NAS, this policy can serve as an effective baseline which we use going forward. However, the possibilities brought by splitting policies are not limited to this simple baseline policy. Policies can be driven by heuristics - e.g. increasing the number of splits at each level to progressively decrease the diversity of architectures in each sub-supernet. Further, the policy can be designed as a hyperparameter optimization problem over the number of splits at each branch of the tree.

We implement gradient matching `gm` [7] as well as `gradnorm`, `snip`, `grasp`, `synflow` [1], `jacobcov` [11], `ntk` and `lr` [3] for use within our framework.

3.2 Split distance

While benchmarking metrics is conveniently done in a supernet splitting framework, there are numerous ways in which a search phase can be implemented. When evaluated on search performance such as test accuracy, we can expect different metrics to perform well for different search phases.

Consequently, gaining deeper insight on the behavior of the metrics is valuable and can lead to choosing metrics more suited to the search phase *a priori*.

In order to compare the impact of metrics on the splitting tree, we introduce the split distance.

Definition 1 (Split distance) *Let $\mathcal{O}$ a closed set of operations, $\mathcal{A}$ and $\mathcal{B}$ sets of disjointed, complementary sets of $\mathcal{O}$:*

$$\mathcal{A} = \{A_1, A_2, \dots, A_n\} \quad s.t. \begin{cases} \bigcap_{i=1}^{n} A_i = \mathcal{O} \\ \forall i, j \in [1, n], A_i \bigcup A_j = \emptyset \end{cases} \tag{1}$$

$\forall o \in \mathcal{O}$, let us define $G_{o,\mathcal{A}}$, $G_{o,\mathcal{B}}$ the sets of elements of $\mathcal{O}$ that are in the same set of $\mathcal{A}$ and $\mathcal{B}$ as o, respectively.

$$G_{o,\mathcal{A}} = \{i \neq o | i \in A, o \in A, A \in \mathcal{A}\}$$

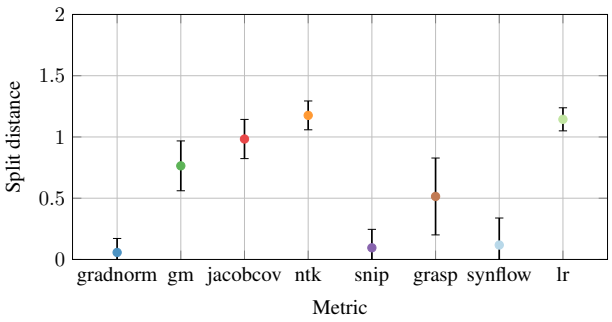
We define the split distance as :

$$S(\mathcal{A}, \mathcal{B}) = \frac{1}{2 \cdot card(\mathcal{O}) - 1} \sum_{o \in \Omega} [\sum_{i \in G_{o,\mathcal{A}}} \mathbb{1}_{i \notin G_{o,\mathcal{B}}} + \sum_{i \in G_{o,\mathcal{B}}} \mathbb{1}_{i \notin G_{o,\mathcal{A}}}]$$

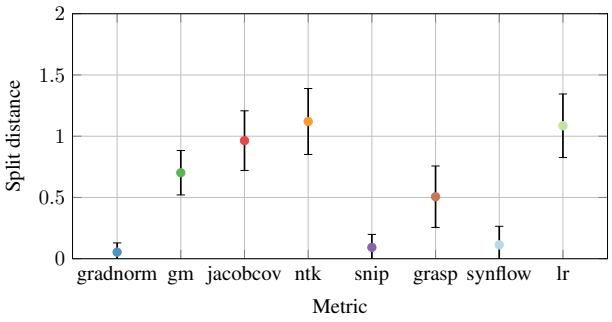
Intuitively, the split distance counts how many operations need to be moved in split $\mathcal{A}$ in order to obtain split $\mathcal{B}$. By reparameterizing $\mathcal{O}$ into natural space $\mathbb{N}$, we can verify that the split distance satisfies all properties of a distance function.

4 Experiments

We study the behavior of splitting trees created by various metrics under two settings : *internal* and *external*. Under the internal setting, a split created by a metric is compared to every split located in the same splitting tree, following the same seeding. Metrics which score low on internal split distance tend to group the same operations together and can be expected to provide meaningful knowledge on the intrinsic properties of operations. In contrast, metrics which score high tend to group different operations together at different stages of the splitting tree and can be expected to better take into account the importance of operations relative to their edge-wise location in the cell. Under the external setting, a split created by a metric is compared to splits located in other splitting trees, following different seedings. While mean values are expected to be closely related to internal split distance values, high variance in the external setting can indicate that the metric has poor robustness to random initialization.



(a) Internal split distance



(b) External split distance

Figure 1 – Fig 1a : internal split distance represents the average distance of splits to other splits from the same splitting tree. Fig 1b : external split distance represents the average distance of splits to other splits in splitting trees obtained with a different seed.

We compute internal and external split distance over 10 seeds for a splitting tree of depth 3 following the *constant* policy with branching factor 2 - i.e there are 8 leaf sub-supernets. The considered search space is NAS-Bench-201 [5] and the supernet is trained on CIFAR-10 [9]. The results for all considered metrics are reported in Fig 1.

We observe two major ways in which metrics behave. Metrics such as `gradnorm`, `snip` or `synflow` produce similar splits most of the time with high consistency. On the other hand, metrics such as `jacobcov`, `ntk` and `lr` produce widely different splits at various edges of the same splitting tree, displaying high expressivity. We deepen the analysis by computing split distances level by level. In this experiment, splits are only compared against other splits at the same depth level in the splitting tree. We report the results in Fig 2.

Results confirm the clear partition between metric types. We show that several metrics consistently generate the same

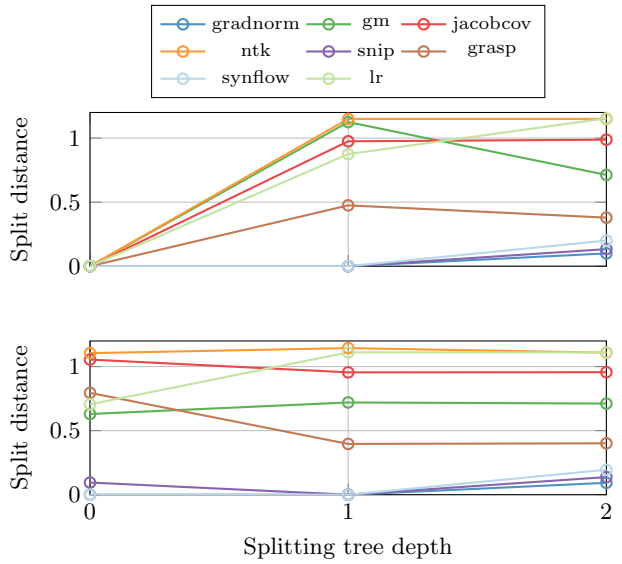


Figure 2 – Split distance grouped by depth in the splitting tree. **Top :** internal split distance. **Bottom :** external split distance.

splits until reaching the third split, while others generate a diverse array of splits starting from the second split. This stark contrast in behaviors could indicate that metrics have different roles from one another and might be more effective when used jointly to maximize the exploration versus exploitation tradeoff. We leave such analysis to future work.

5 Conclusion

We introduce a framework to efficiently split supernets using any metric and use it to gain deeper insight on various metrics from the zero-shot NAS literature. We introduce the split distance as a tool to compare and contrast splitting trees generated by these metrics and evidence the drastic difference in their behavior, indicating that the choice of such metric is an important hyperparameter engineering point for the few-shot NAS line of methods.

References

[1] Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas Donald Lane. Zero-cost proxies for lightweight {nas}. In *International Conference on Learning Representations*, 2021.

[2] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR, 2018.

[3] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four {gpu} hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2021.

[4] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *Proceedings of the IEEE/CVF International Conference on computer vision*, pages 12239–12248, 2021.

[5] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[6] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 544–560. Springer, 2020.

[7] Shoukang Hu, Ruochen Wang, Lanqing Hong, Zhenguo Li, Cho-Jui Hsieh, and Jiashi Feng. Generalizing few-shot nas with gradient matching. *arXiv preprint arXiv:2203.15207*, 2022.

[8] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[9] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[10] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[11] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, pages 7588–7598. PMLR, 2021.

[12] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[13] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[14] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828, 2019.

[15] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020.

[16] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *International Conference on Machine Learning*, pages 12707–12718. PMLR, 2021.

[17] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.