To Share or Not To Share: A Comprehensive Appraisal of Weight-Sharing

Aloïs Pourchot1 2 Alexis Ducarouge1 Olivier Sigaud2

Abstract

Weight-sharing (WS) has recently emerged as a paradigm to accelerate the automated search for efficient neural architectures, a process dubbed Neural Architecture Search (NAS). Although very appealing, this framework is not without drawbacks and several works have started to question its capabilities on small hand-crafted benchmarks. In this paper, we take advantage of the NASBENCH-101 dataset to challenge the efficiency of WS on a representative search space. By comparing a SOTA WS approach to a plain random search we show that, despite decent correlations between evaluations using weight-sharing and standalone ones, WS is only rarely helpful to NAS. We highlight in particular the reliance of the benefits on the search space itself.

1. Introduction

Using deep neural networks (DNNs) has led to numerous breakthroughs on many hard machine learning tasks, such as object detection and recognition or natural language processing (LeCun et al., 2015). In the last years, a paradigm shift was observed, from hand-designing features that can be fed to a machine learning algorithm, to hand-designing neural architectures that can extract those features automatically. However, arranging DNNs is itself time-consuming, requires a lot of expertise and remains very domain-dependent. A promising approach is to automatically design them, a process referred to as Neural Architecture Search (NAS) (Elsken et al., 2018; Wistuba et al., 2019).

Regrettably, because of expensive training requirements, evaluating a single DNN architecture can take days to weeks. In turn, original NAS approaches (Real et al., 2018; Zoph et al., 2017; Zoph & Le, 2016) required thousands of GPU days worth of computing, only to find conformations slightly better than expert-designed ones. In light of this concern, many methods have been explored that could drastically cut the resources required to perform NAS, and today’s literature is blooming with approaches requiring less than a day of computations (Pham et al., 2018; Liu et al., 2018; Xie et al., 2019; Casale et al., 2019)

Most of these efficient methods rely on a computational trick called weight-sharing (WS). First introduced by Pham et al. (2018), WS proposes to reuse sets of weights from previously trained networks, rather than training each newly chosen architecture from scratch. The authors push this idea further by noticing that, in their search space, each network can be seen as a sub-graph of a larger graph: the "super-net". Using WS therefore allows the training of the whole search space at once, by using a single set of weights (represented by the super-net), from which each possible model can then extract its parameters.

Despite a growing literature, the effects of WS on the performances of NAS are still poorly understood. A particular concern is the quality of the scores obtained with the super-net. Employing WS implies substituting metrics obtained after standalone trainings with metrics derived from the shared set of parameters. The two quantities thus need to be correlated to some extent: if networks with excellent standalone performances were under-evaluated with the super-net or vice-versa, the process could be pointless, or even detrimental.

Studying this matter requires training many architectures and is itself remarkably costly. As described in Section 2, several works mitigate this issue by assessing the correlations between evaluations of the super-net and true evaluations in a reduced setting, either evaluating few architectures or studying a drastically reduced search space. In this paper we leverage the computing resources put into the creation of a dataset of architecture evaluations, NASBENCH-101 (Ying et al., 2019) to investigate whether WS can truly improve NAS. Our experimental results show that, with the correct methodology, one can get decent correlations between super-net proxy evaluations and real evaluations on several search spaces containing hundred thousands of architectures. Yet, we present evidence that those correlations: (i) might not be enough to benefit NAS, as exploiting WS is not systematically faster than using random search; (ii) might not be the limiting factor of WS, as search spaces offering better correlations do not always offer better WS performances. Finally, we reveal that the performances of WS strongly depend on the search space itself, and on how they bias the super-net evaluations.

---

1Gleamer 96bis Boulevard Raspail, 75006 Paris, France
2Sorbonne Université, CNRS UMR 7222, Institut des Systèmes Intelligents et de Robotique, F-75005 Paris, France. Correspondence to: Aloïs Pourchot <alois.pourchot@gleamer.ai>.

Under review at ICML 2020.
2. Related Work

In this section we describe several works trying to measure the efficiency of weight-sharing.

Bender et al. (2018) train a super-net on a search space of their own. Path dropout is applied during training to randomly zero-out some portions of the super-net. A simple random search is then used to find a good architecture. To validate the use of the super-net as a proxy to standalone accuracy, 20,000 architectures are sampled from the chosen search space and evaluated with the resulting super-net. Then this set is partitioned into several bins based on the obtained proxy scores. For each bin, 4 architectures are sampled and trained from scratch for a small number of epochs (around 10% of the length of a baseline training) before being evaluated. The authors note visually satisfying correlations between the two proxies, but do not report any numerical metrics. Correlations with full budget standalone accuracy are not reported, most likely for computational reasons. Moreover, because the few models evaluated are evenly spread across the range of possible proxy accuracies, the produced appealing correlations plots might not be representative of the whole search space.

Sciuto et al. (2019) quantify the impact of WS on a small language modeling task. They find that a simple random search baseline is competitive with and sometimes outperforms several NAS algorithms exploiting WS such as DARTS (Liu et al., 2018), ENAS (Pham et al., 2018) and NAO (Luo et al., 2018), which furthermore suffer from high variance. They report over a search space of 32 architectures a poor correlation between the ranks obtained using a super-net and with standalone evaluations. Their study is however limited by the small size of the search space and the used algorithms, which were not specifically designed to produce good correlations at the end of the training, but rather exploit them to rapidly converge to seemingly good architectures.

Zhang et al. (2020) explore another small search space of 64 architectures dedicated to computer vision. They train several super-nets using different seeds, and report high variance in the relative rankings of the architectures obtained with WS. They notice that during super-net training, strong interactions exist between architectures, as updates in some models can either improve or deteriorate the performance of others. They reach correct correlations with standalone rankings, albeit the important variance seems to hinder the practical implications of the super-net. They propose several approaches to reduce the amount of WS between architectures, such as fine-tuning parts of the super-net before evaluation or grouping architectures into different sets according to different strategies.

Chu et al. (2019) argue that random WS is limited by the fact that individual operations might be sampled unevenly throughout the learning process. Although they are seen equally often on average, some might locally be over-represented due to random chance, effectively biasing the weights of the super-net. To prevent this, they propose to average the gradient updates of the shared parameters over \( n \) samples, chosen such that each of the \( n \) elementary operations of the super-net appears exactly once in the resulting computational graphs. They combine their super-net trained with the aforementioned strategy with a multi-objective genetic algorithm, to search for a Pareto front of accurate architectures with adequate numbers of parameters and multiply-adds. They also report that, when sampling 13 models equally distant from the found Pareto front and comparing the accuracy of WS vs standalone, the rankings are very well preserved. However, details regarding this experiment are lacking and it is likely that the result do not hold for the whole search space.

Luo et al. (2019) also note a strong variance in the results of a few NAS algorithms exploiting WS. Using 50 random models sampled from their search space, they evaluate the correlations between the scores given by the super-net and the scores obtained by training from scratch. They report poor correlations, which they deem responsible for the impaired results of WS. They impute the meager correlations to several factors such as short training times and bias towards simple architectures and propose for each of them simple solutions that improve correlations.

Closest to us, Zela et al. (2020) use NASBENCH-101 to evaluate NAS algorithms exploiting WS. However, because the authors choose to study DARTS variants, they create their own search spaces to perform evaluations, whereas we can directly use the whole NASBENCH-101 search space. In one of their experiments, they report the evolution during training of the correlations between evaluations obtained using the super-net, and evaluations queried from the NASBENCH-101 dataset. They report poor or nonexistent correlations for most algorithms, which seems to contradict our findings.

3. Background

In this Section, we present the NASBENCH-101 dataset and the foundations of the WS approach. We furthermore introduce some improvements to the standard super-net training that have been suggested in the literature.

3.1. NASBench-101

Assessing in practice the quality of any NAS approach is costly since it requires to evaluate all the architectures of a realistic search space. Fortunately, catalogues of such evaluations are starting to appear. In our experiments, we use the NASBENCH-101 dataset (Ying et al., 2019) which matches 423,000 unique architectures trained on CIFAR-10.
to their training time, training accuracy, validation accuracy and test accuracy. This allows us to query the value of an architecture under constant time.

The search-space spanned by NASBENCH-101 is inspired by the search space described in (Zoph et al., 2017), which is a standard reference in the one-shot NAS literature (Liu et al., 2018; Xie et al., 2019; Casale et al., 2019). A global architecture consists of the successive iterations of a computational cell which optimal local architecture is to be found. Cells are represented by directed acyclic graphs where the first and last nodes correspond to the input and output. Other nodes represent applied operations. The flow of data itself is represented by the directed edges. At each active node, an operation is chosen among $3 \times 3$ max-pooling, $1 \times 1$ convolution and $3 \times 3$ convolution. We refer the reader to (Ying et al., 2019) for more details on the search space, their training and evaluation procedures, and how they split the CIFAR-10 dataset.

### 3.2. Weight-Sharing

Weight-sharing refers to the process of combining the weights of all the architectures of a search space into a single super-net. To access a model and its weights, one only needs to activate the corresponding computational subgraph. The shared parameters are learned by successively activating different parts of the super-net and performing the standard forward-backward propagation algorithm on minibatches of data. The optimization problem solved when performing NAS using WS can be written as successive iterations of two steps:

$$\text{Find } W \in \arg \min_{W \in \mathbb{R}^N} \Phi(A, f, W), \quad (1)$$

$$\text{Find } A \in \arg \min_{A \in A} F(A, W), \quad (2)$$

where $A$ is the set of possible architectures, $W$ the weights of the super-net, $F$ is the outer objective (usually a validation loss), $f$ is the inner objective (usually a training loss) and $\Phi$ is a function of the inner objective and the search space that dictates how to optimize $W$. $\Phi$ is usually expressed as an expectation of the inner objective over a distribution $P_\theta$ of architecture:

$$\Phi(A, f, W) = \mathbb{E}_{A \sim P_\theta(A)} \{ f(A, W) \}. \quad (3)$$

The way in which both phases are combined changes with each approach. Most alternate between the two, with minibatches of data respectively coming from the training and validation sets (Liu et al., 2018; Xie et al., 2019; Casale et al., 2019; Pham et al., 2018). In this work, we first train a super-net until convergence, and then use it to select possibly good architectures. We treat as our baseline the work of (Guo et al., 2019), where models are sampled uniformly from the set of all possible architectures and the super-net is updated in accordance. This paradigm facilitates the analysis of the correlations, as methods that train both weights and architectures together induce a bias towards architectures with good early evaluations. Besides, (Guo et al., 2019) report great performances when exploiting their trained super-net to perform NAS.

### 3.3. Enhancing Weight-Sharing Correlations

Several tricks and weight-sharing training variants have been introduced in the literature with the objective of improving the correlations offered by the super-net. We list several unrelated approaches here, which we explore in our experiments in Section 4.1 and 5.1.

During evaluation, it is possible to directly exploit the whole super-net and perform a standard forward pass on the impending data whilst activating the graph corresponding to the evaluated net. However, several works report the benefit of adapting the statistics of the inherited batch normalization layers (Bender et al., 2018; Guo et al., 2019).

The weights of the super-net $W$ are updated through gradient descent with respect to the objective in (1), using the formulation of $\Phi$ described in (3). The resulting gradient takes the form of an expectation over the distribution $P_\theta$:

$$\nabla_W \Phi(A, f, W) = \mathbb{E}_{A \sim P_\theta(A)} \{ \nabla_W f(A, W) \}. \quad (4)$$

This expectation is approximated by an empirical average, using random architecture sampled following $P_\theta(A)$. However, in practice (Guo et al., 2019) only use a single architecture to estimate the expectation. Although this process is unbiased, it results in high variance updates of $W$. Decreasing this variance by sampling more models could improve the super-net optimization, at a higher computational cost.

In (Stamoulis et al., 2019), the authors propose to not only share the weights of the basic operations between architectures, but to further merge the weights of all the basic operations at a given node into a single set of parameters. For instance, if two of the basic operations were a $x \times x$ convolution and a $y \times y$ convolution were $x > y$, then instead of representing both operations with two different sets of kernels, one could use a single set of kernels of size $x \times x$, and apply the $y \times y$ convolution by extracting the sub-kernels of size $y \times y$ from the bigger ones.

(Luo et al., 2019) identify in their work a bias towards architectures with fewer parameters to update, as they are easier to train than more complex ones. They propose to correct this bias by sampling architecture pro-rata to their number of parameters, resulting in more complex architectures being sampled more often.
4. Experimental Study

In this section, we describe the protocols used to address the following questions: Do validation accuracies of architectures obtained with WS correlate with standalone ones? Do we get the same correlations under various training and evaluation regimes? Can WS consistently outperform random search? Do the results vary between search spaces? The outcomes are described in Section 5.

4.1. Ranking Architectures with Weight-Sharing

We want to establish the achievable correlations between the accuracies obtained with a super-net, and the accuracies obtained after standalone trainings. We train several supernets on the CIFAR-10 dataset. Following Guo et al. (2019), for each mini-batch of data seen during training, a single architecture is uniformly sampled from the search space. The weights of the super-net are then updated according to the computational graph generated by the activation of this architecture. We reuse the hyper-parameters of Ying et al. (2019), which we detail in the supplementary document. The only notable differences are that we reduce the initial number of filters from 128 to 16, and train networks for four times longer to let the super-nets converge.

Our choice to reduce the number of initial filters to 16 is motivated by computational reasons. According to earlier iterations of our experiments, using the default 128 value requires fairly longer training times for the super-nets, without significantly improving the resulting correlations. Although fixing it to 16 is somewhat arbitrary, (Zela et al., 2020) also noted in their work that accuracies obtained after training architectures with 16 initial filters are greatly correlated with those obtained using the baseline 128 filters. We thus consider that the number of filters is not the limiting factor of our different WS experiments. Furthermore, this setup mimicks one-shot NAS approaches such as, (Liu et al., 2018; Casale et al., 2019; Pham et al., 2018), where the model found by the search is often up-scaled, to further improve accuracies.

We train 5 different super-nets on each search space. After the training of each super-net, we randomly sample 1,000 architectures from the search space and compute their proxy accuracies on a held-out validation dataset. We then match those accuracies with the average validation accuracies returned by NASBench-101. To quantify the quality of the correlations between the two, we make use of Spearman’s rank correlation coefficient.

We estimate accuracies with the super-nets using different setups: (i) performing no fine-tuning at all (NO-FT); (ii) fine-tuning the batch-norm statistics (BNS-FT). When performing no fine-tuning, we directly use all the parameters and batch-norm statistics of the super-net. When adapting the batch-norm statistics of the super-net, we average the batch-norm statistics over 4 mini-batches of data.

Additionally, we study the effect on correlations of the different training variations mentioned in Section 3.3. We follow the same protocol and always fine-tune the batch-norm statistics (BNS-FT). We consider four variants: averaging the gradients in Equation (3) over 3 architectures (AVG-3), sampling architecture pro-rata to their number of parameters (PRO-RATA), following the single-kernel approach of (Stamoulis et al., 2019) (SINGLE-K), and combining the pro-rata and single-kernel approaches (S-K + P-R).

4.2. Impact of Weight-Sharing on NAS Performances

Quantifying the correlations obtained with WS on realistically sized search-spaces is interesting as such, but it is not enough to conclude on the efficiency of WS itself. Indeed, it is not clear above what correlation level WS becomes useful. Here we aim at characterizing the interest of substituting super-net evaluations to the standalone evaluations when performing NAS. To investigate this, we evaluate two methods: a control random search, and a random search guided by a super-net. For random search we sample 1,000 architectures and report the evolution of the test regret as a function of time. The test regret is computed after each evaluation by comparing the mean test accuracy of the current most accurate architecture and the best mean test accuracy of the considered search space. When exploiting WS, we consider the same 1,000 architectures. However, instead of evaluating them in a random order, we assess their performances using a trained super-net and query NASBench-101 in decreasing order of proxy accuracy, under the batch-norm statistics fine-tuning (BNS-FT) setting. If correlations are sufficient to substitute real accuracies with proxy ones, the average test regret of the WS guided strategy should consistently lowerbound the average test regret of random search. We repeat this process 5 times per search space.

We focus in particular on the one-shot NAS paradigm, where the super-net is used to select a few good models, which are then re-trained from scratch. We consider selecting 1, 10 and 100 architectures, and respectively refer to those strategies as TOP-1, TOP-10, and TOP-100. For each strategy and each search guided by the super-net, we note the time necessary to train the best models, report the achieved test regrets, and measure the test regret of the corresponding random searches, when constrained to the same time budget. We average the test regret with and without WS and test for the significance of the difference. Given that the distributions of the regrets cannot be assumed Gaussian, and that samples of WS and random search regret come in correlated pairs (because they partly see the same archi-
To Share or Not To Share: A Comprehensive Appraisal of Weight-Sharing

...test (Wilcoxon, 1945), a non-parametric equivalent of the paired T-test. In practice, when the random search time budget is exceeded, we choose to ignore the training that would still be running, effectively putting WS to a very slight advantage.

For a fair time-wise comparison of the regrets, we account for the training times of the super-nets. However, the duration of a training depends on the hardware and we only had access to NVidia K80 GPUs, which are less efficient than the hardware used by (Ying et al., 2019). To compensate, we assume that most of the training time of the super-net comes from the different forward and backward propagation passes. Since for each sampled architecture the super-net only activates the necessary parts of the network, we approximate the training time of the super-net by the average training time of the considered architectures. We compute this quantity using NASBENCH-101, and multiply it by four given that we train the super-net four times longer.

4.3. Impact of Search Spaces on NAS Performances

To measure the impact of the search space on WS, we introduce several sub-sets of the NASBENCH-101 search space.

In NASBENCH-101, feature maps going to the output of a cell are concatenated. However, since the size of the output is fixed across all possible cells, the input size, output size, and number of filters of nodes may vary over architectures, possibly hindering the use of WS. A reasonable splitting strategy is to consider the sets \((A_i)_{i=1,...,4}\), in which architectures contain exactly \(i\) nodes connected to the output (beside the input node, which is always added), and therefore share as many parameters and feature maps2. We also consider the full NASBENCH-101 set, which we call \(A_0\): we solve the aforementioned problem by dynamically adapting the number of feature maps used for each node depending on the architecture: each node, as seen by the super-net, contains the maximum number of filters for the given layer, but sampled architectures only inherit the first \(n\) filters of the filter-bank, where \(n\) is determined so as to satisfy the constraints on the output size the architecture’s cell.

Early results additionally compelled us to study the influence of residual connections on WS. It is well known in the computer vision literature that edges connecting the input node to the output node, known as residual connections (He et al., 2015), significantly influence the quality of the optimization of individual architectures. Suspecting that this is also true when training super-nets, we consider two additional search spaces: \(A_0\) and \(A_0^\emptyset\), respectively containing all architectures of NASBENCH-101 with and without residual connections. Figure 1 sketches the structural properties of the different search spaces.

5. Results

We now describe the results of the above studies. We then discuss the influence of the search space on WS.

5.1. Ranking Capabilities of Weight-Sharing

For each search space, we report in the left part of Table 1 the average over 5 super-nets of the rank correlation reached between the average standalone accuracies returned by NASBENCH-101 and the proxy accuracies obtained after applying the two evaluation protocols described in Section 4.1. NO-FT refers to performing no fine-tuning, and BNS-FT to fine-tuning the batch-norm statistics. Performing no fine-tuning at all results in the worst correlations across all search-spaces, with substantial variance. Activating batch-norm statistics fine-tuning allows for an average 270% correlation increase from the no fine-tuning scheme, effectively leading to 3 times better results.

\[ \text{Figure 1. Structural properties of the different search-spaces. On each graph, "IN" and "OUT" denote the input and output of the cell, and "+" and "&" the sum and the concatenation of incoming features maps. We only display the edges discriminating at least one search space and represent the rest of the graph with the node } G. \text{ If an edge does not discriminate a specific search space, we represent it with a dotted line. } A_1 \text{ to } A_4 \text{ are characterized by the number of edges concatenated after } G, \text{ and } A_0 \text{ and } A_0^\emptyset \text{ by the presence or absence of a residual connection.} \]

In the right part of Table 1, we present the rank correlations obtained from training with the different variants of WS described in Section 3.3 and fine-tuning batch-norm statistics during evaluations. SINGLE-K refers to applying the single kernel variant, PRO-RATA to sampling architectures pro-rata to their number of parameters, and S-K + P-R the combination of the two. AVG-3 refers to averaging gradients over three architectures. We notice that all approaches lead to a small improvement of the correlations, as well as a slight variance reduction.
Table 1. Spearman’s rank correlation coefficient between WS evaluations and standalone evaluations for various search-spaces, WS variants, and evaluation schemes. We report the average and standard deviation over 5 independent runs. On the left, we use the baseline WS approach and the three evaluation schemes described in Section 3.2. On the right, we test some variants of WS described in 3.3 and always fine-tune batch-norm statistics during evaluations. Results marked with an asterisk * indicate that one of the super-net failed to converge, and that the reported statistics are computed using only the four others.

|                  | NO-FT | BNS-FT | SINGLE-K | PRO-RATA | AVG-3 | S-K + P-R |
|------------------|-------|--------|----------|----------|-------|-----------|
| \( A_4 \)        | 0.08 ± 0.17 | 0.64 ± 0.03 | 0.66 ± 0.01 | 0.68 ± 0.03 | * 0.67 ± 0.02 | 0.69 ± 0.02 |
| \( A_3 \)        | 0.12 ± 0.15 | 0.59 ± 0.03 | 0.62 ± 0.02 | 0.63 ± 0.02 | 0.61 ± 0.03 | 0.66 ± 0.02 |
| \( A_2 \)        | 0.24 ± 0.03 | 0.60 ± 0.04 | 0.64 ± 0.02 | 0.64 ± 0.02 | 0.61 ± 0.01 | 0.65 ± 0.02 |
| \( A_1 \)        | 0.32 ± 0.05 | 0.68 ± 0.02 | 0.72 ± 0.02 | 0.75 ± 0.02 | 0.73 ± 0.01 | 0.67 ± 0.01 |
| \( A_0 \)        | 0.24 ± 0.05 | 0.56 ± 0.04 | * 0.63 ± 0.02 | 0.59 ± 0.03 | 0.61 ± 0.02 | 0.58 ± 0.02 |
| \( A_0' \)       | * 0.11 ± 0.05 | * 0.46 ± 0.06 | * 0.58 ± 0.02 | * 0.56 ± 0.03 | 0.52 ± 0.02 | 0.49 ± 0.02 |
| \( A_0' \)       | 0.34 ± 0.10 | 0.71 ± 0.02 | 0.68 ± 0.01 | 0.72 ± 0.02 | 0.69 ± 0.02 | 0.66 ± 0.01 |

These simple results show that, as long as batch-norm statistics are adapted to the evaluated architectures, it is possible to get correlations between proxy evaluations performed with WS, and full-budget evaluations. We notice that all the works mentioning poor correlations in Section 2 do not detail their evaluation setup, and we suspect that they do not adapt batch-norm statistics. Additionally, it is possible to further improve the resulting correlations by slightly modifying the super-net training in different ways.

5.2. Can Weight-Sharing Accelerate NAS?

The results of the NAS experiments described in Section 4.2 are reported in Figure 2. We additionally detail in smaller “zoomed-in” plots the regret reached by the random and WS-guided searches in the TOP-1, TOP-10 and TOP-100 settings. As can be visually observed from the curves and the corresponding zoom-ins, the regret of the WS-guided strategy is rarely significantly below the random search one.

For each one-shot paradigm, we report in Table 2 the different \( p \)-values for the test of the statistical difference between the mean regret reached by the WS-guided strategy and by the random-search strategy. Coincidentally with the visual results of Figure 2, statistical significance is reached in only a few rare occasions. In the TOP-1 scenario, using WS results in a larger regret on \( A_4 \), and a smaller regret on \( A_2 \) and \( A_1 \). In the TOP-10 paradigm, WS is better than random search on \( A_3 \), \( A_1 \) and \( A_0' \), and worse on \( A_4 \) and \( A_0 \). Finally, when evaluating the TOP-100 architectures, WS is statistically indistinguishable from random search.

The results of Table 2 additionally suggest that, there is no clear link between the level of correlation reached by WS on a search space, and its ability to outperform random search: On \( A_0' \), where the correlations in Table 1 are the lowest, WS significantly outperforms random search under the TOP-10 paradigm. On \( A_2 \) and \( A_3 \), where the correlations reach similar values, WS is respectively worse and better than random search under the TOP-1 paradigm, but better and worse under the TOP-10 approach. Finally, on \( A_0' \), where WS offers the best correlations, random search and WS-guided search seem to be equivalent.

Using WS in a one-shot NAS paradigm is thus rarely significantly better than performing a random search, and can even be worse. Besides, there is no consistency in the number of architectures to select for WS to perform well, as the number required to be efficient or inefficient varies between search-spaces. To top it off, the level of correlation between proxy and standalone evaluations does not explain these significance results.

5.3. Variations between Search Spaces

The results of Section 5.2 suggest that the efficiency of the WS-guided NAS is often on par with random search, but can vary greatly with the considered search space. Coincidentally, we notice from the results introduced in Section 5.1, that simply changing the number of nodes connected to the output makes the average correlation vary between 0.59 on \( A_3 \) and 0.68 on \( A_1 \). Additionally, restricting the search space to architectures presenting a residual connection has a noticeable positive effect on the correlations, as they in-
To Share or Not To Share: A Comprehensive Appraisal of Weight-Sharing

Figure 2. Evolution of the test regret as a function of time. Colored lines correspond to the search guided by WS evaluations, whereas gray lines specify the results of random search. Bold lines display the average over 5 runs, whilst the variances are represented by the areas in lighter colors. Both axes are exponentially scaled. The search space is indicated at the top right of each graph. Additionally we detail the portions of the curves related to the \(\text{TOP-1}\), \(\text{TOP-10}\), and \(\text{TOP-100}\) one-shot paradigms. Following the same color code, individual results of all the searches are marked with a "\(+\)" and we report variances in the resulting test regrets with error-bars.

crease from 0.46 to 0.71 between \(A^0_3\) and \(A^5_3\). The search space itself has an important impact on the correlations, even more so than using the training enhancements described in Section 3.3.

The size of the datasets may explain the varying correlations. It has often been asserted in the literature that the more architectures there are in the search space, the harder it is to train the super-net. The Spearman rank’s correlation between the average correlation obtained with batch-size fine-tuning (BNS-FT) reported in Table 1 and the sizes of the dataset reaches \(-0.71\) (with a \(p\)-value of 0.07). This indicates that, on average, larger search-spaces lead to smaller correlations between proxy and standalone evaluations. However, results in Section 5.1 suggest that it is not the only aspect of the search space that matters. For instance, on \(A_3\) and \(A_2\) WS offers roughly the same level of correlation, despite \(A_2\) being twice larger than \(A_3\). The correlation achieved is 25% smaller in \(A^0_3\) than in \(A_0\), with 23% less architectures. Besides, few architectures are actually seen during training: given 432 training epochs of 157 mini-batches of data, less than 67,824 unique architectures are used to update the super-net. This might be enough to cover \(A_4\) or \(A^5_0\), but represents only a fraction of larger datasets. Although the size of the dataset has a non-negligible impact on the correlation capabilities of WS, it cannot entirely explain the discrepancies between the different search spaces.

We display in Figure 3 three scatter plots of the true validation accuracies and those obtained from 3 different super-nets, for 3 different search spaces. There is a noticeable variance in the visual appearance of the figures, which is corroborated with the variance in the correlation coefficients reported in Table 1. The numerical variance has been observed in several other studies of the literature (Sciuto et al., 2019; Luo et al., 2019; Zela et al., 2020; Zhang et al., 2020) and is due to the reliance of super-training on the sampled architectures. Several visible clusters seem also linked to proxy evaluations. For each scatter-plot, we report the distributions of the true validation and proxy accuracies over sampled architectures. Coincidently with the different visible architecture clumps, distributions of proxy evaluations are much less regular than their true validation counterparts, often presenting several modes. The clusters of architectures in the scatter-plots visually transcribe existing biases in proxy evaluations.

There is no trivial relation between different biases and particular structural properties of the architectures. Fortunately, some biases are easier to highlight than others. We focus on two such biases in Figure 4. On \(A_4\), architectures with a residual connection tend to get better evaluations than those without. On \(A_4\), the presence of a \(3 \times 3\) convolution on the first node triggers over-evaluation. Such clusters can be seen...
To Share or Not To Share: A Comprehensive Appraisal of Weight-Sharing

Figure 3. For $A_4$, $A_1$, $A_0$ and for three super-nets trained on these search spaces, we report the proxy accuracy computed after fine-tuning the batch-norm statistics (y-axis), and the average validation accuracy returned by NASBENCH-101 (x-axis) for 1,000 architectures. Furthermore, each scatter-plot reports the distributions of the proxy and standalone accuracies of the sampled architectures. More scatter-plots can be found in the supplementary document.

Figure 4. We report for a super-net trained on $A_1$ (left) and $A_4$ (right) the proxy accuracy computed after fine-tuning the batch-norm statistics (y-axis), and the average validation accuracy returned by NASBENCH-101 (x-axis) for 1,000 architectures. We highlight in a darker tone the points corresponding to architectures with residual connections (left) and architectures with a $3 \times 3$ convolution on the first node (right). Both examples reveal a clear bias in super-net evaluations. We also report the distributions of the proxy and standalone accuracies of the sampled architectures.

The patterns appearing in the scatter-plots may explain the search results in Section 5.2 better than the correlations level reached by WS. On $A_4$, the over-evaluation bias visible in Figure 4 creates a cluster of architectures with excellent proxy accuracies. As a result, WS neglects a large number of architectures with equal or better capabilities that random search does not miss. Although the cluster contains a few of the best architectures, the average standalone accuracy is particularly poor. This impedes WS from selecting good top-models, and makes the early WS-guided search worse than random search. On $A_1$, the over-evaluation bias towards residual connections benefits to the search, as architectures with residual connections are better on average and constitute most of the best architectures of the search space. The WS-guided search is in turn quite efficient. On $A_0^3$, where the results reveal no clear evaluation bias, WS is statistically identical to random search despite excellent correlations. The patterns of over/under-evaluations dictate the search behavior when exploiting WS. If WS is biased towards interesting patterns in the considered search space, then it is likely to outperform random search. Otherwise, the difference may not be significant. In the worst scenario, the bias can even be strong enough to undermine the performance of WS.

6. Conclusion

In this paper we have leveraged the NASBENCH-101 dataset to investigate the impact of weight-sharing on neural architecture search. Our results lead to the following conclusions. First, super-nets trained with WS can offer significant correlations between proxy evaluations and standalone evaluations, but fine-tuning the batch-norm statistics of the models is mandatory for the process to be successful. The results can be further improved by tweaking the WS training process, but the search space itself has a more significant influence over the quality of correlations.

More importantly, WS is not consistently faster than a random search baseline, the improvement being mostly search-space dependent. Super-nets resulting from optimization with WS can be biased towards specific structural patterns in the architectures, which also vary depending on the search-space. Those patterns, rather than the level of correlations, dictate the efficiency of NAS when exploiting WS. Given that each search space has its own specific biases, it is hard to foresee how well WS is going to perform. Understanding in what ways the search-space can bias the training of the super-nets emerges as a central question for the WS paradigm and as a promising lead for future work.

3See the supplementary document for results on all search spaces.
References

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pp. 549–558, 2018.

Casale, F. P., Gordon, J., and Fusi, N. Probabilistic Neural Architecture Search. *arXiv e-prints*, art. arXiv:1902.05116, Feb 2019.

Chu, X., Zhang, B., Xu, R., and Li, J. FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. *arXiv e-prints*, art. arXiv:1907.01845, Jul 2019.

Elsken, T., Hendrik Metzen, J., and Hutter, F. Neural Architecture Search: A Survey. *ArXiv e-prints*, August 2018.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single Path One-Shot Neural Architecture Search with Uniform Sampling. *arXiv e-prints*, art. arXiv:1904.00420, Mar 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436–444, 2015.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable Architecture Search. *ArXiv e-prints*, June 2018.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in neural information processing systems*, pp. 7816–7827, 2018.

Luo, R., Qin, T., and Chen, E. Understanding and Improving One-shot Neural Architecture Optimization. *arXiv e-prints*, art. arXiv:1909.10815, Sep 2019.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. URL http://arxiv.org/abs/1802.03268.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized Evolution for Image Classifier Architecture Search. *ArXiv e-prints*, February 2018.

Sciuto, C., Yu, K., Jaggi, M., Musat, C., and Salzmann, M. Evaluating the search phase of neural architecture search. *CoRR*, abs/1902.08142, 2019. URL http://arxiv.org/abs/1902.08142.

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., and Marculescu, D. Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours. *arXiv e-prints*, art. arXiv:1904.02877, Apr 2019.

Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. ISSN 00994987. URL http://www.jstor.org/stable/3001968.

Wistuba, M., Rawat, A., and Pedapati, T. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rylqooRqK7.

Zela, A., Siems, J., and Hutter, F. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *Submitted to International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJx9ngStPH.

Zoph, B. and Le, Q. V. Neural Architecture Search with Reinforcement Learning. *ArXiv e-prints*, November 2016.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning Transferable Architectures for Scalable Image Recognition. *ArXiv e-prints*, July 2017.
To Share or Not To Share: A Comprehensive Appraisal of Weight-Sharing;
Appendix

Aloïs Pourchot 1 2  Alexis Ducarouge 1  Olivier Sigaud 2

0.1. Super-Net Training

The super-net is trained with the RMSPROPT optimizer. The initial learning rate is set to 0.2 and decayed to 0 using a cosine annealing schedule over 432 training epochs, which is four times the original time budget used in NASBENCH-101. This longer training is required for the super-net to converge, as already noted by ?. The momentum is set to 0.9, weight decay to $10^{-4}$, and $\epsilon$ to 1. The batch size is kept to 256 and the momentum and $\epsilon$ of the batch normalization layers are respectively set to 0.997 and $10^{-5}$. We reduce the initial number of filters to 16 compared to the original 128 to accelerate training and evaluations.

0.2. Spearman’s Correlation Coefficient

Spearman’s rank correlation coefficient is used to determine how well the relationship between two variables can be described using a monotonic function. It is computed as

$$\rho_s = 1 - \frac{6\sum (r_{nb}^i - r_{ws}^i)^2}{n(n^2-1)}$$

where $r_{nb}^i$ and $r_{ws}^i$ respectively represent the ranks of the $i$-th architecture according to the dataset, and according to the super-net trained with WS.

0.3. Additional Figures

We report in Figure 1 the scatter-plots obtained on all search spaces and with all the super-nets trained. They can also be observed, in full resolution in the notebooks accompanying the code available with the paper.

1Gleamer 96bis Boulevard Raspail, 75006 Paris, France
2Sorbonne Université, CNRS UMR 7222, Institut des Systèmes Intelligents et de Robotique, F-75005 Paris, France. Correspondence to: Aloïs Pourchot <alois.pourchot@gleamer.ai>.

Under review at ICML 2020.
Figure 1. For each search space, and for three super-nets trained on the considered search space, we report the proxy accuracy computed after fine-tuning the batch-norm statistics (y-axis), and the average validation accuracy returned by NASBENCH-101 (x-axis) for 1,000 architectures. We furthermore report on each scatter-plot the distributions of the proxy and standalone accuracies of the sampled architectures. Different colors correspond to different search spaces: from top to bottom, $A_4$, $A_3$, $A_2$, $A_1$, $A_0$, $A_0^0$ and $A_0^r$. 