Random Search and Reproducibility for Neural Architecture Search

Liam Li
Carnegie Mellon University

Ameet Talwalkar
Carnegie Mellon University

Abstract

Neural architecture search (NAS) is a promising research direction that has the potential to replace expert-designed networks with learned, task-specific architectures. In order to help ground the empirical results in this field, we propose new NAS baselines that build off the following observations: (i) NAS is a specialized hyperparameter optimization problem; and (ii) random search is a competitive baseline for hyperparameter optimization. Leveraging these observations, we evaluate both random search with early-stopping and a novel random search with weight-sharing algorithm on two standard NAS benchmarks—PTB and CIFAR-10. Our results show that random search with early-stopping is a competitive NAS baseline, e.g., it performs at least as well as ENAS [39], a leading NAS method, on both benchmarks. Additionally, random search with weight-sharing outperforms random search with early-stopping, achieving a state-of-the-art NAS result on PTB and a highly competitive result on CIFAR-10. Finally, we explore the existing reproducibility issues of published NAS results.

1 INTRODUCTION

Deep learning offers the promise of bypassing the process of manual feature engineering by learning representations in conjunction with statistical models in an end-to-end fashion. However, neural network architectures themselves are typically designed by experts in a painstaking, ad-hoc fashion. Neural architecture search (NAS) presents a promising path for alleviating this pain by automatically identifying architectures that are superior to hand-designed ones. Since the work by Zoph and Le [49], there has been explosion of research activity on this problem [29, 30, 37, 11, 41, 1, 21, 5, 39, 48, 32, 45, 7]. Notably, there has been great industry interest in NAS, as evidenced by the vast computational [49, 50, 41] and marketing resources [16] committed to industry-driven NAS research. However, despite a steady stream of promising empirical results [49, 50, 41, 32, 33, 7], we see three fundamental issues with the current state of NAS research:

Inadequate Baselines. Leading NAS methods exploit many of the strategies that were initially explored in the context of traditional hyperparameter optimization tasks, e.g., evolutionary search [38, 19], Bayesian optimization [42, 4, 18], and gradient-based approaches [2, 34]. Moreover, the NAS problem is in fact a specialized instance of the broader hyperparameter optimization problem. However, in spite of the close relationship between these two problems, existing comparisons between novel NAS methods and standard hyperparameter optimization methods are inadequate. In particular, to the best of our knowledge, no state-of-the-art hyperparameter optimization methods have been evaluated on standard NAS benchmarks. *Without benchmarking against leading hyperparameter optimization baselines, it difficult to quantify the performance gains provided by specialized NAS methods.*

Complex Methods. We have witnessed a proliferation of novel NAS methods, with research progressing in many different directions. New approaches introduce a significant amount of algorithmic complexity in the search process, including complicated training routines [1, 39, 45, 7], architecture transformations [44, 41, 6, 30, 11], and modeling assumptions [21, 24, 48, 5, 29] (see Figure 1 and Appendix A.1 for more details). While many technically diverse NAS methods demonstrate good empirical performance, they often lack corresponding ablation studies [33, 48, 7], and as a result, *it is unclear what NAS component(s) are necessary to achieve a competitive empirical result*.

Lack of Reproducibility. Experimental reproducibility

is of paramount importance in the context of NAS research, given the empirical nature of the field, the complexity of new NAS methods, and the steep computational costs associated with empirical evaluation. In particular, there are (at least) two important notions of reproducibility to consider: (1) "exact" reproducibility i.e., whether it is possible to reproduce explicitly reported experimental results; and (2) "broad" reproducibility, i.e., the degree to which the reported experimental results are themselves robust and generalizable. Broad reproducibility is difficult to measure due to the computational burden of NAS methods and the high variance associated with extremal statistics. However, most of the published results in this field do not even satisfy exact reproducibility. *For example, of the 12 papers published since 2018 at NeurIPS, ICML, and ICLR that introduce novel NAS methods (see Table 5), none are exactly reproducible.* [1]

While addressing these challenges will require community-wide efforts, in this work we present results that aim to make some initial progress on each of these issues. In particular, our contributions are as follows:

1. We help ground existing NAS results by providing a new perspective on the gap between traditional hyperparameter optimization and leading NAS methods. Specifically, we evaluate a general hyperparameter optimization method combining random search with early-stopping [28] on two standard NAS benchmarks (CIFAR-10 and PTB). With approximately the same amount of compute as DARTS [32], a state-of-the-art (SOTA) NAS method, this simple method provides a much more competitive baseline for both benchmarks: (1) on PTB, random search with early-stopping reaches test perplexity of 56.4 compared to the published result for ENAS [39], a leading NAS method, of 56.3,[2] and (2) for CIFAR-10, random search with early-stopping achieves a test error of 2.85%, whereas the published result for ENAS is 2.89%. While SOTA NAS methods like DARTS still outperform this baseline, our results demonstrate that the gap is not nearly as large as that suggested by published random search baselines on these tasks [39, 32].

2. We identify a small subset of NAS components that are sufficient for achieving good empirical results. We construct a simple algorithm from the ground up starting from vanilla random search, and demonstrate that properly tuned random search with weight-sharing is competitive with much more complicated methods when using similar computational budgets. In particular, we identify the following meta-hyperparameters that impact the behavior of our algorithm: batch size, number of epochs, network size, and number of evaluated architectures. We evaluate our proposed method using the same search space and evaluation scheme as DARTS [32]. We explore a few modifications of the meta-hyperparameters to improve search quality and make full use of available GPU memory and computational resources, and observe SOTA performance on the PTB benchmark and comparable performance to DARTS on the CIFAR-10 benchmark. We emphasize that we do not perform additional hyperparameter tuning of the final architectures discovered at the end of the search process.

3. We open-source the code, random seeds, and documentation necessary to reproduce our experiments.[3] Our single machine results shown in Table 6 and Table 1 follow a deterministic experimental setup, given a fixed random seed, and satisfy exact reproducibility. For these experiments on the two standard benchmarks, we study the broad reproducibility of our random search with weight-sharing results by repeating our experiments multiple times. We observe non-trivial variance across independent runs and identify potential sources for these differences. Our results highlight the need for more careful reporting of experimental results, increased transparency of intermediate results, and more robust statistics to quantify the performance of NAS methods.

2 RELATED WORK

Figure 1 shows the three components of a general hyperparameter optimization problem, each of which can have NAS-specific approaches: (1) the search space defines a set of possible hyperparameter configurations, (2) the search method is used to select candidate configurations to evaluate, and (3) the evaluation method is used to obtain an estimate of the quality of different configurations. A more detailed discussion of each component is available in Appendix A.1. Given this background, we provide additional context for the three issues we identified regarding the current state of NAS research in Section 1.

[1] It is important to note that these works vary drastically in terms of what materials they provide, and some authors such as Liu et al. [32], provide a relatively complete codebase for their methods. However, even in the case of DARTS, the code for the CIFAR-10 benchmark is not deterministic. We were thus not able to reproduce the results in Liu et al. [32], but we were able to use the DARTS code repository (`https://github.com/quark0/darts`) as the launching point for our experimental setup.

[2] We could not reproduce this result using the final architecture and code provided by the authors.

[3] All material available at `https://github.com/liamcli/randomNAS_release`

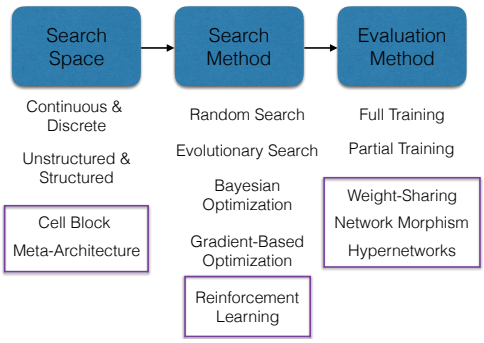

Figure 1: **Components of hyperparameter optimization.** Primarily NAS-specific methods are lined in purple.

2.1 INADEQUATE BASELINES

Existing works in NAS do not provide adequate comparison to random search and other hyperparameter optimization methods. Some works either compare to random search given a budget of just of few evaluations [39, 32] or Bayesian optimization methods without efficient architecture evaluation schemes [21]. While Real et al. [41] and Cai et al. [6] provide a thorough comparison to random search, they use random search with full training even though partial training methods have been shown to be orders-of-magnitude faster than standard random search [27, 28].

Although certain hyperparameter optimization methods [42, 34, 25] require non-trivial modification in order to work with NAS search spaces, others are easily applicable to NAS problems [18, 4, 10, 13, 27, 28]. Of these applicable methods, we choose to use a simple method combining random search with early-stopping called ASHA [28] to provide a competitive baseline for standard hyperparameter optimization. Li et al. [28] showed ASHA to be a state-of-the-art, theoretically principled, bandit-based partial training method that outperforms leading adaptive search strategies for hyperparameter optimization. We compare the empirical performance of ASHA with that of NAS methods in Section 4.

2.2 COMPLEX METHODS

Much of the complexity of NAS methods is introduced in the process of adapting search methods for NAS-specific search spaces, which usually involve discrete hyperparameters with a DAG representation where each node represents local computations and edges of the DAG represent the flow of data from one node to another [39, 32]. Evolutionary approaches need to define a set of possible mutations to apply to different architectures [40, 41]; Bayesian optimization approaches [21, 24] rely on specially designed kernels; gradient-based methods transform the discrete architecture search problem into a continuous optimization problem [33, 32, 45, 7]; and Zoph and Le [49], Zoph et al. [50], and Pham et al. [39] use reinforcement learning to train a recurrent neural network controller to generate good architectures. All of these search approaches add a significant amount of complexity with no clear winner, especially since methods some times use different search spaces and evaluation methods. To simplify the search process and help isolate important components of NAS, we use random search to sample architectures from the search space.

Additional complexity is also introduced by NAS-specific evaluation methods—like network morphisms; hypernetworks and performance prediction; and weight-sharing—that exploit the structure of NAS search spaces to speed up the evaluation of the quality of different architectures. Network morphisms require architecture transformations that satisfy certain criteria; hypernetworks and performance prediction methods encode information from previously seen architectures in an auxiliary network; and weight-sharing methods [39, 32, 1, 45, 7] use a single set of weights for all possible architectures and hence, can require careful training routines.

Despite their complexity, these more efficient NAS evaluation methods are 1 to 3 orders-of-magnitude cheaper than full training (see Table 1 and Table 6), at the expense of decreased fidelity to the true performance. Of these evaluation methods, network morphism still requires on the order of 100 GPU days [29, 11] and, while hypernetworks and prediction performance based methods can be cheaper, weight-sharing is less complex since it does not require training an auxiliary network. In addition to the computational efficiency of weight-sharing methods [32, 39, 7, 45], which only require computation on the order of fully training a single architecture, this approach has also achieved the best result on the two standard benchmarks [32, 7]. Hence, we use random search with weight-sharing as our starting point for a simple and efficient NAS method.

Our work is inspired by the result of Bender et al. [1], which showed that random search, combined with a well-trained set of shared weights can successfully differentiate good architectures from poor performing ones. However, their work required several modifications to stabilize training (e.g., a tunable path dropout schedule over edges of the search DAG and a specialized ghost batch normalization scheme [17]). Furthermore, they only report experimental results on the CIFAR-10 benchmark, on which they fell slightly short of the results for leading NAS methods. In contrast, our combination of random search with weight-sharing greatly simplifies the training routine and

we identify key variables needed to achieve competitive results on both CIFAR-10 and PTB benchmarks.

2.3 LACK OF REPRODUCIBILITY

The earliest NAS results lacked exact and broad reproducibility due to the tremendous amount of computation required to achieve the results [49, 50, 41]. Recently, it has become feasible to evaluate the exact and broad reproducibility of many SOTA methods due to their reduced computational cost. However, while many authors have released code for their work [e.g., 39, 32, 5, 6], others have not made their code publicly available [e.g., 45, 48], including the work most closely related to ours by Bender et al. [1].

Table 5 of Appendix A.4 summarizes the reproducibility of recent NAS publications at some of the major machine learning conferences according to the availability of components necessary for exact reproducibility: architecture search code, model evaluation code, random seeds used for search and evaluation, and documentation for hyperparameter tuning. None of the 12 papers shown in Table 5 satisfy exact reproducibility.

In terms of broad reproducibility, with the exception of NASBOT [24] and DARTS [32], the methods in Table 5 only report the performance of the best found architecture, presumably resulting from a single run of the search process. Although this is understandable in light of the computational costs for some of these methods [33, 6], the high variance of extremal statistics makes it difficult to isolate the impact of the novel contributions introduced in each work. DARTS is particularly commendable in acknowledging its dependence on random initialization, prompting the use multiple runs to select the best architecture. In our experiments in Section 4, we go one step further and evaluate the broad reproducibility of our results with multiple sets of random seeds.

3 METHODOLOGY

We now introduce our NAS algorithm that combines random search with weight-sharing. Our algorithm is designed for an arbitrary search space with a DAG representation.

For concreteness, consider the search space used by DARTS for designing a recurrent cell for the PTB benchmark: the DAG considered for the recurrent cell has $N = 8$ nodes and the operations considered include tanh, relu, sigmoid, and identity. Figure 2 shows an example of an architecture from this search space. To sample an architecture from this search space, we apply random search in the following manner:

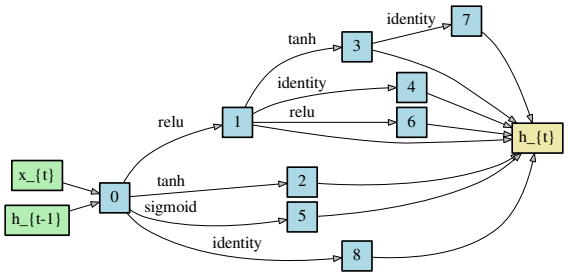

Figure 2: **Recurrent Cell on PTB Benchmark.** The best architecture found by random search with weight-sharing in Section A.3 is depicted. Each numbered square is a node of the DAG and each edge represents the flow of data from one node to another after applying the indicated operation along the edge. Nodes with multiple incoming edges (i.e., node 0 and output node `h_{t}` concatenate the inputs to form the output of the node).

1. For each node in the DAG, determine what decisions must be made. In the case of the PTB search space, we need to choose a node as input and a corresponding operation to apply to generate the output of the node.

2. For each decision, identify the possible choices for the given node. In the case of the PTB search space, if we number the nodes from 1 to N, node i can take the outputs of nodes 0 to node $i-1$ as input (the initial input to the cell is index 0 and is also a possible input). Additionally, we can choose an operation from $\{$tanh, relu, sigmoid, and identity$\}$ to apply to the output of node i.

3. Finally, moving from node to node, we sample uniformly from the set of possible choices for each decision that needs to be made.

In order to combine random search with weight-sharing, we simply use randomly sampled architectures to train the shared weights. In the case of the PTB benchmark, the same weights are applied to all possible inputs to a node. Shared weights are updated by selecting a single architecture for a given minibatch and updating the shared weights by back-propagating through the network with only the edges and operations as indicated by the architecture activated. Hence, the number of architectures used to update the shared weights is equivalent to the total number of minibatch training iterations.

After training the shared weights for a certain number of epochs, we use these trained shared weights to evaluate the performance of a number of randomly sampled architectures on a separate held out dataset. We select the best

performing one as the final architecture, i.e., as the output of our search algorithm.

3.1 RELEVANT META-HYPERPARAMETERS

There are a few key meta-hyperparameters that impact the behavior of our search algorithm. We describe each of them below, along with a description of how we expect them to impact the search algorithm, both in terms of search quality and computational costs.

Training epochs. Increasing the number of training epochs while keeping all other parameters the same increases the total number of minibatch updates and hence, the number of architectures used to update the shared weights. Intuitively, training with more architectures should help the shared weights generalize better to what are likely unseen architectures in the evaluation step. Unsurprisingly, more epochs increase the computational time required for architecture search.

Batch size. Decreasing the batch size while keeping all other parameters the same also increases the number of minibatch updates but at the cost of noisier gradient update. Hence, we expect reducing the batch size to have a similar effect as increasing the number of training epochs but may necessitate adjusting other meta-hyperparameters to account for the noisier gradient update. Intuitively, more minibatch updates increase the computational time required for architecture search.

Network size. Increasing the search network size increases the dimension of the shared weights. Intuitively, this should boost performance since a larger search network can store more information about different architectures. Unsurprisingly, larger networks require more GPU memory.

Number of evaluated architectures. Increasing the number of architectures that we evaluate using the shared weights allows for more exploration in the architecture search space. Intuitively, this should help assuming that there is a high correlation between the performance of an architecture evaluated using shared weights and the ground truth performance of that architecture when trained from scratch [1]. Unsurprisingly, evaluating more architectures increases the computational time required for architecture search.

Other learning meta-hyperparameters will likely need to be adjusted accordingly for different settings of the key relevant meta-hyperparameters listed above. In our experiments in Section 4, we tune *gradient clipping* as a fifth meta-hyperparameter, though there are other possible meta-hyperparameters that may benefit from additional tuning (e.g., learning rate, momentum).

In Section 4, following these intuitions, we incrementally explore the design space of our search method in order to improve search quality and make full use of the available GPU memory and computational resources.

3.2 MEMORY FOOTPRINT

Since we train the shared weights using a single architecture at a time, the memory footprint of our random search with weight-sharing can be reduced to that of a single model. In this sense, our approach is similar to ProxylessNAS [7] and allows us to perform architecture search with weight-sharing on the larger "proxyless" models that are usually used in the final architecture evaluation step instead of the smaller proxy models that are usually used in the search step. We take advantage of this in a subset of our experiments for the PTB benchmark in Section A.3; performing random search with weight-sharing on a proxyless network for the CIFAR-10 benchmark is a direction for future work.

In contrast, Bender et al. [1] trains the shared weights with a path dropout schedule that incrementally prunes edges within the DAG so that the sub-DAGs used to train the shared weights become sparser as training progresses. Under this training routine, since most of the edges in the search DAG are activated in the beginning, the memory footprint cannot be reduced to that of a single model to allow a proxyless network for the shared weights.

4 EXPERIMENTS

In line with prior work [49, 39, 32], we consider the two standard benchmarks for neural architecture search: (1) language modeling on the Penn Treebank (PTB) dataset [35] and (2) image classification on CIFAR-10 [26]. For each of these benchmarks, we consider the same search space and use the same experimental setups as DARTS [32], and by association SNAS [45], to facilitate a fair comparison of our results to existing work.

To evaluate the performance of random search with weight-sharing on these two benchmarks, we proceed in the same three stages as Liu et al. [32]:

Stage 1: Perform architecture search for a cell block on a cheaper search task.

Stage 2: Evaluate the best architecture from the first stage by retraining a larger network formed from multiple cell blocks from scratch. This stage is used to select the best architecture from multiple trials.

Stage 3: Perform a full evaluation of the best found architecture from the second stage by either training for more epochs (PTB) or training with more seeds (CIFAR-10).

We start with the same meta-hyperparameter settings used by DARTS to train the shared weights. Then, we incrementally modify the meta-hyperparameters identified in Section 3.1 to improve performance until we either reach state-of-the-art performance (for PTB) or match the performance of DARTS and SNAS (for CIFAR-10).

For our evaluation of random search with early-stopping (i.e., ASHA) on these two benchmarks, we apply partial training to the stage (2) evaluation network and then select the best architecture for stage (3) evaluation. For both benchmarks, we run ASHA with a starting resource per architecture of $r = 1$ epoch, a maximum resource of 300 epochs, and a promotion rate of $\eta = 4$, indicating the top $1/4$ of architectures will be promoted in each round and trained for $4\times$ more resource.

Due to space limitations, we present the results for the more commonly studied CIFAR-10 benchmark [45, 48] below and defer the results for the PTB benchmark, which mirror that for CIFAR-10, to Appendix A.3. For the PTB benchmark, our results in Table 6 of the Appendix show ASHA to be a competitive baseline for NAS, matching the published performance of the best architecture found by ENAS, and random search with weight-sharing to reach SOTA for NAS methods.

4.1 CIFAR-10 BENCHMARK

The DAG considered for the convolutional cell has $N = 4$ search nodes and the operations considered include 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, and 3×3 average pooling, and zero [32]. To sample an architecture from this search space, we have to choose, for each node, 2 input nodes from previous nodes and associated operations to perform on each input (there are two initial inputs to the cell that are also possible choices); we sample in this fashion twice, once for the normal convolution cell and one for the reduction cell (e.g., see Figure 3).

Due to higher memory requirements for weight-sharing, Liu et al. [32] uses a smaller network with 8 stacked cells and 16 initial channels to perform the convolutional cell search, followed by a larger network with 20 stacked cells and 36 initial channels to perform the evaluation. Again, we will refer to the network used in the first stage as the proxy network and the network in the second stage the proxyless network.

We will next present the final search results for the CIFAR-10 benchmark, and then dive deeper into these results to explore the impact of meta-hyperparameters on stage (2) intermediate results, and finally evaluate associated reproducibility ramifications.

4.1.1 Final Search Results

We now present our results after performing the final evaluation in stage (3). We use the same evaluation scheme used to produce the results in Table 1 of Liu et al. [32]. In particular, we train the proxyless network configured according to the best architectures found by different methods with 10 different seeds and report the average and standard deviation. We discuss these results in the context of the three issues—baselines, complex methods, reproducibility—introduced in Section 1.

First, we evaluate the ASHA baseline using 9 GPU days, which is comparable to the 10 GPU days we allotted to our independent run of DARTS. In contrast to the one random architecture evaluated by Pham et al. [39] and the 24 evaluated by Liu et al. [32] for their random search baselines, ASHA evaluated over 700 architectures in the allotted computation time. The best architecture found by ASHA achieves an average error of 3.03 ± 0.13, which is significantly better than the random search baseline provided by Liu et al. [32] and comparable to DARTS (first order). Additionally, the best performing seed reached a test error of 2.85, which is lower than the published result for ENAS. Similar to the PTB benchmark, these results suggest that the gap between SOTA NAS methods and standard hyperparameter optimization is much smaller than previously reported [39, 32].

Next, we evaluate random search with weight-sharing with tuned meta-hyperparameters (see Section 4.1.2 for details). This method finds an architecture that achieves an average test error of 2.85 ± 0.08, which is comparable to the reported results for SNAS and DARTS, the top 2 weight-sharing algorithms that use a comparable search space, as well as GHN [48]. Note that while the two manually tuned architectures we show in Table 1 outperform the best architecture discovered by random search with weight-sharing, they have over $7\times$ more parameters. Additionally, the best-performing efficient NAS method, ProxylessNAS, uses a larger proxyless network and a significantly different search space than the one we consider. As mentioned in Section 3, random search with weight-sharing can also directly search over larger proxyless networks since it trains using discrete architectures. We hypothesize that using a proxyless network and applying random search with weight-sharing to the same search space as ProxylessNAS would further improve our results; we leave this as a direction for future work.

Finally, we examine the reproducibility of the NAS methods using a comparable search space with available code for both architecture search and evaluation (i.e., DARTS and ENAS; to our knowledge, code is not currently available for SNAS). For DARTS, exact reproducibility was not feasible since the code is non-deterministic and Liu

Table 1: **CIFAR-10 Benchmark: Comparison with state-of-the-art NAS methods and manually designed networks.** The results are grouped by those for manually designed networks, published NAS methods, and the methods that we evaluated. Models for all methods are trained with cutout. Test error for our contributions are averaged over 10 random seeds. Table entries denoted by "-" indicate that the field does not apply, while entries denoted by "N/A" indicate unknown entries. The search cost is measured in GPU days. Note that the search cost is hardware dependent and the search cost shown for our results are calculated for Tesla P100 GPUs; all other numbers follow those reported by Liu et al. [32].
* We show results for the variants of these networks with comparable number of parameters. Larger versions of these networks achieve lower errors.
# Reported test error averaged over 5 seeds.
† The stage (1) cost shown is that for 1 trial as opposed to the cost for 4 trials shown for DARTS and Random search WS. It is unclear whether multiple trials followed by stage (2) evaluation are required in order to find a good architecture.
‡ Due to the longer evaluation we employ in stage (2) to account for unstable rankings, the cost for stage (2) is 1 GPU day for results reported by Liu et al. [32] and 6 GPU days for our results.

		Test Error		Params	Search Cost			Comparable	Search
Architecture	**Source**	Best	Average	(M)	Stage 1	Stage 2	Total	**Search Space?**	**Method**
Shake-Shake#	[9]	N/A	2.56	26.2	-	-	-	-	manual
PyramidNet	[46]	2.31	N/A	26	-	-	-	-	manual
NASNet-A#*	[50]	N/A	2.65	3.3	-	-	2000	N	RL
AmoebaNet-B*	[41]	N/A	2.55 ± 0.05	2.8	-	-	3150	N	evolution
ProxylessNAS†	[7]	2.08	N/A	5.7	4	N/A	N/A	N	gradient-based
GHN#†	[48]	N/A	2.84 ± 0.07	5.7	0.84	N/A	N/A	N	hypernetwork
SNAS†	[45]	N/A	2.85 ± 0.02	2.8	1.5	N/A	N/A	Y	gradient-based
ENAS†	[39]	2.89	N/A	4.6	0.5	N/A	N/A	Y	RL
ENAS	[32]	2.91	N/A	4.2	4	2	6	Y	RL
Random search baseline	[32]	N/A	3.29 ± 0.15	3.2	-	-	4	Y	random
DARTS (first order)	[32]	N/A	3.00 ± 0.14	3.3	1.5	1	2.5	Y	gradient-based
DARTS (second order)	[32]	N/A	2.76 ± 0.09	3.3	4	1	5	Y	gradient-based
DARTS (second order)‡	Ours	2.62	2.78 ± 0.12	3.3	4	6	10	Y	gradient-based
ASHA baseline	Ours	2.85	3.03 ± 0.13	2.2	-	-	9	Y	random
Random search WS‡	Ours	2.71	2.85 ± 0.08	4.3	2.7	6	9.7	Y	random

et al. [32] do not provide random seeds for the search process; hence, we focus on broad reproducibility of the results. In our independent run, DARTS reached an average test error of 2.78 ± 0.12 compared to the published result of 2.76 ± 0.09. Notably, we observed that the process of selecting the best architecture in stage (2) is unstable when training stage (2) models for only 100 epochs; see Section 4.1.3 for details. Hence, we use 600 epochs in all of our CIFAR experiments, including our independent DARTS run, which explains the discrepancy in stage (2) costs between original DARTS and our independent run.

For ENAS, the published results do not satisfy exact reproducibility due to the same issues as those for DARTS. We show in Table 1 the broad reproducibility experiment conducted by Liu et al. [32] for ENAS; here, ENAS found an architecture that achieved a comparable test error of 2.91 in $8\times$ the reported stage (1) search cost. We then investigated the reproducibility of random search with weight-sharing. We verified exact reproducibility and then examined broad reproducibility by evaluating 5 additional independent runs of our method. We observe performance below 2.90 test error in 2 of the 5 runs and an average of 2.92 across all 6 runs. We investigate various sources for these discrepancies in Section 4.1.3.

4.1.2 Impact of Meta-Hyperparameters

We next detail the meta-hyperparameter settings that we tried in order to reach competitive performance on the CIFAR-10 benchmark via random search with weight-sharing. Similar to DARTS, in these preliminary experiments we performed 4 separate trials of each version of random search with weight-sharing, where each trial consists of executing stage (1) followed by stage (2). In stage (1), we train the shared weights and use them to evaluate a given number of randomly sampled architectures on the test set. In stage (2), we select the best architecture, according to the shared weights, to train from scratch using the proxyless network for 600 epochs.

We incrementally tune random search with weight-sharing by adjusting the following meta-hyperparameters that impact both the training of shared weights and the evaluation of architectures using these trained weights: number of training epochs, gradient clipping, number of architectures evaluated using shared weights, and network size. The settings we consider for random search proceed as

Table 2: **CIFAR-10 Benchmark: Comparison of Stage (2) Intermediate Search Results for Weight-Sharing Methods.** In stage (1), random search is run with different settings to train the shared weights. The shared weights are then used to evaluate the indicated number of randomly sampled architectures. In stage (2), the best of these architectures for each trial is then trained from scratch for 600 epochs. We report the performance of the best architecture after stage (2) for each trial for each search method.

	Setting				Trial					
Method	Epochs	Gradient Clipping	Initial Channels	# Archs Evaluated	1	2	3	4	Best	Average
Reproduced DARTS†	50	5	16	-	2.92	2.77	3.00	3.05	**2.77**	**2.94**
Random (1)	50	5	16	1000	3.25	4.00	2.98	3.58	2.98	3.45
Random (2)	150	5	16	5000	2.93	3.80	3.19	2.96	2.93	3.22
Random (3)	150	1	16	5000	3.50	3.42	2.97	2.95	2.97	3.21
Random (4)	300	1	16	11000	3.04	2.90	3.14	3.09	2.90	3.04
Random (5) Run 1	150	1	24	5000	2.96	3.33	2.83	3.00	**2.83**	**3.03**

follows:

Random (1): We start by training the shared weights with the proxy network used by DARTS and default values for number of epochs, gradient clipping, and number of initial filters; all other meta-hyperparameters are the same.

Random (2): We increase the number of training epochs from 50 to 150, which concurrently increases the number of architectures used to update the shared weights.

Random (3): We reduce the maximum gradient norm from 5 to 1 to adjust for discrete architectures instead of the weighted combination used by DARTS.

Random (4): We further increase the number of epochs for training the proxy network with shared weights to 300 and increase the number of architectures evaluated using the shared weights to 11k.

Random (5): We separately increase the proxy network size to be as large as possible given the available memory on a Nvidia Tesla P100 GPU (i.e. by $\approx 50\%$ due to increasing the number of initial channels from 16 to 24).

The performance of the final architecture after retraining from scratch for each of these settings is shown in Table 2. The best setting for random search was Random (5), which has a larger network size. The best trial for this setting reached a test error of 2.83 when retraining from scratch; we show the normal and reduction cells found by this trial in Figure 3. In light of these stage (2) results, we focus in stage (3) on the best architecture found by Random (5) Run 1, and achieve an average test error of 2.85 ± 0.08 over 10 random seeds as shown in Table 1.

4.1.3 Investigating Reproducibility

Our results in this section show that although DARTS appears broadly reproducible, this result is surprising given the unstable ranking in architectures observed between 100 and 600 epochs for stage (2) evaluation. To begin, the first row of Table 2 shows our reproduced results for DARTS after training the best architecture for each trial from scratch for 600 epochs. In our reproduced run, DARTS reaches an average test error of 2.94 and a minimum of 2.77 across 4 trials (see Table 2). Note that this is not a direct comparison to the published result for DARTS since there, the stage (2) evaluation was performed after training for only 100 epochs.

Table 3: **CIFAR-10 Benchmark: Ranking of Intermediate Test Error for DARTS.** Architectures are retrained from scratch using the proxyless network and the error on the test set is reported after training for the indicated number of epochs. Rank is calculated across the 4 trials. We also show the average over 10 seeds for the best architecture from the top trial for reference.

		Epochs				Across 10 Seeds	
		100		600			
Search Method	Trial	Value	Rank	Value	Rank	Min	Avg
Reproduced Darts‡	1	7.63	2	2.92	2	2.62	2.78 ± 0.12
	2	7.67	3	2.77	1		
	3	8.38	4	3.00	3		
	4	7.51	1	3.05	4		

Delving into the intermediate results, we compare the performance of the best architectures across trials from our independent run of DARTS after training each from scratch for 100 epochs and 600 epochs (see Table 3). We see that the ranking is unstable between 100 epochs and 600 epochs, which motivated our strategy of training the final architectures across trials to 600 epochs in order to select the best architecture for final evaluation across 10 seeds. This suggests we should be cautious when using noisy signals for the performance of different architectures, especially since architecture search is conducted for DARTS and Random (5) for only 50 and 150 epochs

Table 4: **CIFAR-10 Benchmark: Broad Reproducibility of Random Search WS** We report the stage 3 performance of the final architecture from 6 independent runs of random search with weight-sharing.
† This run was performed using the DARTS code before we corrected for non-determinism (see Appendix A.4).

Test Error Across 10 Seeds						
Run 1	Run 2†	Run 3	Run 4	Run 5	Run 6	**Average**
2.85 ± 0.08	2.86 ± 0.09	2.88 ± 0.10	2.95 ± 0.09	2.98 ± 0.12	3.00 ± 0.19	2.92

respectively.

Finally, we investigate the variance of random search with weight-sharing with 5 additional runs as shown in Table 4. The stage (3) evaluation of the best architecture for these 5 additional runs reveal that 2 out of 5 achieve similar performance as Run 1, while the 3 remainder underperform but still reach a better test error than ASHA. These broad reproducibility results show that random search with weight-sharing has high variance between runs, which is not surprising given the change in intermediate rankings that we observed for DARTS.

5 CONCLUSION

We conclude by summarizing our results and proposing suggestions to push the field forward and foster broader adoption of NAS methods.

Better baselines that accurately quantify the performance gains of NAS methods. The performance of random search with early-stopping evaluated in Section 4 reveals a surprisingly small performance gap between leading general-purpose hyperparameter optimization methods and specialized methods tailored for NAS. In traditional hyperparameter optimization, an informative measure of the performance of a novel algorithm is its 'multiple of random search,' i.e., how much more compute would random search need to achieve similar performance [27]. An analogous baseline could be useful for NAS, where the impact of a novel NAS method can be quantified in terms of a multiplicative speedup relative to a standard hyperparameter optimization method such as random search with early-stopping.

Ablation studies that isolate the impact of individual NAS components. Our head-to-head experimental evaluation of two variants of random search (with early stopping and with weight-sharing) allows us to pinpoint the performance gains associated with the cheaper weight-sharing evaluation scheme. In contrast, the fact that random search with weight-sharing is comparable in performance to leading NAS methods calls into question the necessity of the complicated algorithmic components employed by ENAS, SNAS, and DARTS. Relatedly, while ProxylessNAS achieves better average test error on CIFAR-10 than random search with weight-sharing, it is unclear to what degree these performance gains are attributable to the search space, search method, and/or proxyless shared-weights evaluation method. To promote scientific progress, we believe that ablation studies should be conducted to answer these questions in isolation.

Reproducible results that engender confidence and foster scientific progress. Reproducibility is a core tenet of scientific progress and crucial to promoting wider adoption of NAS methods. In traditional hyperparameter optimization, it is standard for empirical results to be reported over 10 independent experimental runs [14, 25, 23, 27]. In contrast, as we discuss Section 2, results for NAS methods are often reported over a single experimental run [39, 7, 45, 48], without exact reproducibility. This is a consequence of the steep time and computational cost required to perform NAS experiments. However, in order to adequately differentiate between various methods, results need to be reported over several independent experimental runs, especially given the nature of the extremal statistics that are being reported. Consequently, we conclude that either significantly more computational resources need to be devoted to evaluating NAS methods and/or more computationally tractable benchmarks need to be developed to lower the barrier for performing adequate empirical evaluations.

Acknowledgments

We thank Maruan Al-Shedivat, Sebastian Caldas, Greg Ganger, Kevin Jamieson, Angela Jiang, Mikhail Khodak, Gregory Plumb, Afshin Rostamizadeh, Virginia Smith, and Daniel Wong for helpful comments and valuable discussion. Thanks also to Julien Siems and Frank Hutter's group for their efforts to reproduce our work, which led to insights on reproducibility and motivated additional experiments. Finally, this work was supported in part by DARPA FA875017C0141, the National Science Foundation grants IIS1705121 and IIS1838017, an Okawa Grant, a Google Faculty Award, an Amazon Web Services Award, and a Carnegie Bosch Institute Research Award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, the National Science Foundation, or any other funding agency.

References

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

[2] Y. Bengio. Gradient-based optimization of hyperparameters. In *Neural Computation*, 2000.

[3] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[4] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[5] A. Brock, T. Lim, J. Ritchie, and N. Weston. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018.

[6] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu. Path-level network transformation for efficient architecture search. In *International Conference on Machine Learning*, 2018.

[7] H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[8] S. Cao, X. Wang, and K. M. Kitani. Learnable embedding space for efficient neural architecture compression. In *International Conference on Learning Representations*, 2019.

[9] T. Devries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv:1708.04552*, 2017.

[10] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *International Joint Conferences on Artificial Intelligence*, 2015.

[11] T. Elsken, J. H. Metzen, and F. Hutter. Multi-objective Architecture Search for CNNs. *arXiv:1804.09081*, 2018.

[12] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[13] S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, 2018.

[14] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, 2015.

[15] D. Golovin, B. Sonik, S. Moitra, G. Kochanski, J. Karro, and D.Sculley. Google vizier: A service for black-box optimization. In *SIGKDD Conference on Knowledge Discovery and Data Mining*, 2017.

[16] Google. Google AutoML. `https://cloud.google.com/automl/`, 2018.

[17] E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, 2017.

[18] F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, 2011.

[19] M. Jaderberg, V. Dalibard, S. Osindero, W. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, et al. Population based training of neural networks. *arXiv:1711.09846*, 2017.

[20] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics*, 2015.

[21] H. Jin, Q. Song, and X. Hu. Auto-Keras: Efficient Neural Architecture Search with Network Morphism. *arXiv:1806.10282*, 2018.

[22] K. Kandasamy, G. Dasarathy, J. B. Oliva, J. Schneider, and B. Póczos. Gaussian process bandit optimisation with multi-fidelity evaluations. In *Advances in Neural Information Processing Systems*, 2016.

[23] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos. Multi-fidelity Bayesian optimisation with continuous approximations. In *International Conference on Machine Learning*, 2017.

[24] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. Xing. Neural Architecture Search with Bayesian Optimization and Optimal Transport. *Advances in Neural Information Processing Systems*, 2018.

[25] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. *International Conference on Artificial Intelligence and Statistics*, 2017.

[26] A. Krizhevsky. Learning multiple layers of features from tiny images. In *Technical report, Department of Computer Science, Univsersity of Toronto*, 2009.

[27] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization.

International Conference on Learning Representation, 17, 2017.

[28] L. Li, K. G. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar. Massively parallel hyperparameter tuning. *arXiv:1810.05934*, 2019.

[29] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive Neural Architecture Search. *European Conference on Computer Vision*, 2018.

[30] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018.

[31] H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. *arXiv:1806.09055*, 2018.

[32] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[33] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu. Neural Architecture Optimization. *Advances In Neural Information Processing Systems*, 2018.

[34] D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, 2015.

[35] M. Marcus, M. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2): 313–330, 1993.

[36] S. Merity, N. Keskar, and R. Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*, 2018.

[37] R. Negrinho and G. Gordon. DeepArchitect: Automatically Designing and Training Deep Architectures. *arXiv:1704.08792*, 2017.

[38] R. S. Olson and J. H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on Automatic Machine Learning*, 2016.

[39] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, 2018.

[40] E. Real, S. Moore, A. Selle, S. Saxena, Y. Leon Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

[41] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized Evolution for Image Classifier Architecture Search. *arXiv:1802.01548*, 2018.

[42] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

[43] K. Swersky, J. Snoek, and R. Adams. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems*, 2013.

[44] T. Wei, C. Wang, Y. Rui, and C. W. Chen. Network morphism. In *International Conference on Machine Learning*, 2016.

[45] S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

[46] Y. Yamada, M. Iwamura, and K. Kise. Shakedrop regularization. *arXiv:1802.02375*, 2018.

[47] Z. Yang, Z. Dai, R. Salakhutdinov, and W. W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*, 2018.

[48] C. Zhang, M. Ren, and R. Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*, 2019.

[49] B. Zoph and Q. V. Le. Neural Architecture Search with Reinforcement Learning. *International Conference on Learning Representation*, 2017.

[50] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2018.