Journal of Machine Learning Research 21 (2020) 1-18 Submitted 1/20; Revised 11/20; Published 11/20

Best Practices for Scientific Research on Neural Architecture Search

Marius Lindauer LINDAUER@TNT.UNI-HANNOVER.DE
Leibniz University of Hannover
Hannover, 30167, Germany

Frank Hutter FH@CS.UNI-FREIBURG.DE
University of Freiburg & Bosch Center for Artificial Intelligence
Freiburg im Breisgau, 79110, Germany

Editor: Joelle Pineau

Abstract

Finding a well-performing architecture is often tedious for both deep learning practitioners and researchers, leading to tremendous interest in the automation of this task by means of neural architecture search (NAS). Although the community has made major strides in developing better NAS methods, the quality of scientific empirical evaluations in the young field of NAS is still lacking behind that of other areas of machine learning. To address this issue, we describe a set of possible issues and ways to avoid them, leading to the NAS best practices checklist available at `http://automl.org/nas_checklist.pdf`.

Keywords: Neural Architecture Search, Scientific Best Practices, Empirical Evaluation

1. Introduction

Neural architecture search (NAS), the task of finding a well-performing architecture of a neural network for a given dataset, is currently one of the hottest topics in automated machine learning (AutoML; see Hutter et al. (2019) for an overview), with a seemingly exponential increase in the number of papers written on the subject (see Figure 1)[1]. While many NAS methods are fascinating (see the survey article by Elsken et al. (2019b) for an overview of the main trends and a taxonomy of NAS methods), in this note we will not focus on these methods themselves, but on how to scientifically evaluate them and report one's findings.

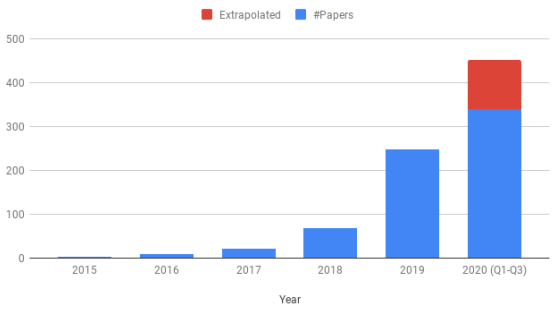

Figure 1: NAS papers per year based on the literature list on `automl.org`. Extrapolation for 2020 based on the first 9 months of the year.

Although NAS methods steadily improve, the quality of empirical evaluation

1. The literature list on NAS at `www.automl.org` that this figure is based on is manually curated and lists all NAS papers we are aware of. The list contains both published papers and unreviewed arXiv papers.

in this field is still lagging behind compared to other areas in machine learning, AI and optimization. Over the last few years, we observed many times that results could not be reproduced because of one of the pitfalls mentioned throughout this paper. Although some might try to justify bad practices by the faster experimentation they allow, research on scientific methods shows that they rather increase confirmation bias (Nickerson, 1998; Fanelli et al., 2017). This will in turn slow down the overall research progress rather than speeding it up. We therefore propose best practices for empirical evaluations of NAS methods, which we believe will facilitate sustained and measurable progress in the field.

We note that discussions about reproducibility and empirical evaluations are currently taking place in several fields of AI. For example, Joelle Pineau's keynote at NeurIPS 2018[2] showed how to improve empirical evaluations of reinforcement learning algorithms (Henderson et al., 2018), and several of her points carry over to NAS. For the NAS domain itself, Li and Talwalkar (2019) also discuss reproducibility and simple baselines.

We resist the temptation to point to papers with flawed experiments, as no paper is perfect, including our own. However, to see examples for the pitfalls we mention, please randomly open five recent NAS papers, and you will very likely find examples for most of the pitfalls we mention and try to avoid.

We note that there are several recent papers that cast doubts on much of the work in the field of NAS (Sciuto et al., 2019; Li and Talwalkar, 2019; Xie et al., 2019; Yang et al., 2020), and these have led to serious scepticism of outsiders concerning NAS. In this paper, we take into account all of the issues raised in those works, and several additional ones, in order to define a prescriptive set of best practices that will facilitate sustained progress in the field of NAS.

2. Best Practices for Releasing Code

Let's start with what is perhaps the most controversial set of best practices. This concerns reproducibility, a cornerstone of good science. As Buckheit and Donoho (1995) put it:

"An article about computational science in a scientific publication is not the scholarship itself, it is merely the advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures."

Availability of code facilitates progress. To facilitate fast progress in the field, it is important to be able to reproduce existing results. This helps studying and understanding existing methods, and to properly evaluate a new idea (see Section 3).

Reproducing someone else's NAS experiments is often next to impossible without code. The reproducibility crisis in machine learning has already shown how hard it is to reproduce each other's experiments without code in machine learning in general, but in NAS, this is further complicated by the fact that important settings are hidden both in the training pipeline (see Best Practice 1), and in the NAS method itself (see Best Practice 2). If the NAS-optimizer uses a neural network itself there is even more room for hidden choices.

2. https://videos.videoken.com/index.php/videos/neurips-2018-invited-talk-on-reproducible-reusable-and-robust-reinforcement-learning/

Therefore, we strongly advertise that each paper should come with a link to source code in order to facilitate reproducibility and sustained progress in the field.

Best Practice 1: Release Code for the Training Pipeline(s) you use

The training pipeline used is often far more important for achieving good performance than the precise neural architecture used (Yang et al., 2020). The training pipeline includes the specifics of the optimization and regularization methods used. For example, for image datasets, next to the choice of optimizer, number of training epochs, important choices include activation functions (e.g., Swish (Elfwing et al., 2018)), learning rate schedules (e.g., cosine annealing (Loshchilov and Hutter, 2017)), data augmentation (e.g. by CutOut (Devries and Taylor, 2017), MixUp (Zhang et al., 2017) or Auto-Augment (Cubuk et al., 2019; Lim et al., 2019)), auxiliary towers (Zoph et al., 2018), the depth and width of the network, and regularization (e.g., by Dropout (Srivastava et al., 2014), Shake-Shake (Gastaldi, 2017), ScheduledDropPath (Zoph et al., 2018), L1/L2 regularization, or decoupled weight decay (Loshchilov and Hutter, 2019)). For example, on CIFAR-10, using the search space from DARTS (Liu et al., 2019b), the combination of CutOut, ScheduledDropPath, auxiliary towers, Auto-Augment, increasing the number of channels and increasing the number of epochs for training yielded a combined improvement of 3% test accuracy, compared to less than 1% for choosing the best neural architecture (Yang et al., 2020).

Therefore, the final performance results of paper A and paper B are *incomparable* unless they use the same training pipeline. Releasing your training pipeline ensures that others can meaningfully compare against your results. Especially the training pipeline for a dataset like CIFAR-10 should be trivial to make available, since this routinely consists of a single file relying only on open-source Tensorflow, Pytorch or MXNet code. Complex parallel training pipelines for larger datasets should also be easy to make available; even if there are some special dependencies that cannot be made available (e.g., a specialized framework for parallel training across many GPUs), availability of the main source code strongly facilitates reproducing results.

Best Practice 2: Release Code for Your NAS Method

While releasing the training pipeline allows researchers to fairly compare against your stated results, releasing the code for your NAS method allows others to also use it on new datasets. As an additional motivation next to following good scientific practice: papers with available source code tend to have far more impact and receive more citations than those without, because other researchers can build upon your code base.

Best Practice 3: Don't Wait Until You've Cleaned up the Code; That Time May Never Come

We encourage anyone who can do so to simply put a copy of the code online as it was used, appropriately labelled as prototype research code, without using extra time to clean it up. This simply owes to the fact that, due to our busy lives as machine learners, the statement *"The code will be available once I find the time to clean it up"* in practice all too often de facto (and without ill intent) translates to *"The code will never be available"*. Of course, it

is even better if you can release cleaned code in addition to the code dump we encourage. However, to make sure that you do release at all, please consider doing the code dump first. Indeed, we are pleased to observe that code releases are becoming far more common, partly due to the following fact and corollary.

Fact 1 *Reproducibility is ever more in the limelight.*

With the growing emphasis on reproducibility (e.g., as evidenced by ICML and NeurIPS now requesting authors to fill out Joelle Pineau's reproducibility checklist[3] as part of the paper submission process), the trend at top machine learning venues is going towards authors having to justify cases in which code is not made available (and where the acceptance probability is reduced when no good reasons exist).

Corollary 2 *A progressive policy for sharing code presents a competitive advantage in hiring for industrial research labs.*

Since top researchers want to publish in the top venues, and since this may become easier when sharing code, labs with a progressive policy for publishing code may soon have a competitive advantage in publishing at the top venues and thus in the global hunt for talent. We acknowledge that it is not always easy in industrial research environments to publish code, e.g., due to dependencies on proprietary components. However, there are by now many positive examples (e.g., Pham et al., 2018; Liu et al., 2019b; Ying et al., 2019) that demonstrate that sharing NAS code is possible for industrial players if it is made a priority.

3. Best Practices for Comparing NAS Methods

To ensure a fair comparison between NAS methods, but also to shed some light on the reasons that are responsible for a NAS method to perform well, we propose the following best practices.

Best Practice 4: Use the Same NAS Benchmarks, not Just the Same Datasets

A very common way to compare NAS methods is a big table with the results different papers reported for a dataset such as CIFAR-10. However, we would like to emphasize that the numbers in these tables are often *incomparable* due to the use of different search spaces and different optimization or regularization techniques (see also Best Practice 1). Rather, we propose the use of consistent *NAS benchmarks*:

3. `https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf`

Definition 3 (NAS Benchmark) *A NAS benchmark consists of a dataset (with a pre-defined training-test split*[4]*), a search space*[5]*, and available runnable code with pre-defined hyperparameters for training the architectures.*

We now give some good examples of such NAS benchmarks:

Example 4 *A prominent NAS benchmark is the publicly available search space and training pipeline of DARTS (Liu et al., 2019b), evaluated on CIFAR-10 (with standard training/test split).*[6]

Zela et al. (2020a) also defined an additional 12 smaller publicly available NAS benchmarks to demonstrate failure modes of DARTS. These are based on the search space of DARTS, but with fewer operators on the edges.

Example 5 *Another prominent NAS benchmark is NAS-Bench-101 (Ying et al., 2019), the first* tabular *NAS benchmark. On top of a publicly available search space, training pipeline and dataset, it also provides pre-computed evaluations with that training pipeline for all possible cells in the search space.*

Tabular NAS benchmarks speed up experimentation dramatically compared to NAS benchmarks that involve training neural networks for every evaluation, as they replace this expensive step by a table lookup. Further tabular NAS benchmarks along similar lines are NAS-Bench-1Shot1 (Zela et al., 2020b), NAS-Bench-201 (Dong and Yang, 2020), and NAS-Bench-NLP (Klyuchnikov et al., 2020).[7] While all tabular NAS benchmarks are inherently limited to small search spaces due to their requirement of exhaustively evaluating every architecture in the search space, *surrogate* benchmarks scale to larger spaces:

Example 6 *NAS-Bench-301 (Siems et al., 2020) is the first* surrogate *NAS benchmark. On top of a publicly available large search space (the one of DARTS, with over* 10^{18} *architectures), training pipeline and dataset, it releases data for a subset of 60k evaluated architectures in the search space, as well surrogate models that can be used to predict the performance of any architecture in the space.*

We strongly believe that more such NAS benchmarks are needed for the community to make sustained and quantifiable progress (see also our note in Section 5 concerning the need for new NAS benchmarks).

4. If a validation set is required, this should be split off the training set; the test set should only be used for reporting the final performance. We do not request the validation set to be part of the definition of a NAS benchmark since different NAS methods may require validation sets of different sizes (e.g., only for hyperparameter optimization, or also for gradient-based architecture search).
5. We note that the representation of the search space is sometimes also quite different. For example, it matters whether operations are in the nodes or on the edges.
6. The only information that is unfortunately not available in their repository are the hyperparameter settings they used for their 100-epochs evaluation.
7. There is also a paper on NAS-Bench-ASR (ANOMYNOUS, 2020), but the table for this tabular benchmark has not yet been released, and the code for generating it has not yet been open-sourced.

Best Practice 5: Run Ablation Studies

NAS methods tend to have many moving pieces, some of which are more important than others. Also, unfortunately, some papers modify the NAS benchmark itself (e.g., the hyperparameters used for training the architecture; see Best Practices 1 and 4), another component of the experimental pipeline, or various components of a NAS method and leave it unclear which modification was most important to achieve their final results. While NAS papers often get accepted based on these performance numbers, to strive for scientific insight, we should understand *why* the final results are better than before. If a paper changes components other than the NAS method, then it is especially important to quantify the impact of these changes. Overall, as a community we also still lack thorough insights into what the most important aspects of NAS are, and only recently there is some progress on that front (Sciuto et al., 2019; Li and Talwalkar, 2019; Yang et al., 2020). Therefore, we recommend to run ablation analyses to study the importance of individual components that affect performance, incl. any changes of the NAS benchmark and the NAS method itself. We would like to highlight the work by Yang et al. (2020) as particularly valuable in this regards, by showing that well engineered training protocols, the search space and macro architecture design substantially impact the overall performance.

Best Practice 6: Use the Same Evaluation Protocol for the Methods Being Compared

So far, there is no single gold-standard on how to evaluate and compare NAS methods. In some cases, the outcome of a NAS run is only taken to be a single final architecture; in other cases, thousands of architectures are sampled and evaluated in order to select the best one. Of course, the latter is much less efficient, but it can lead to better performance. These different evaluation schemes are one of the reasons why results from different NAS papers are often incomparable. Selecting the architecture with best performance on the *test* set would of course lead to an optimistic estimate of performance, but selecting the best-performing architecture on a validation split is a perfectly reasonable building block of NAS algorithms; however, this step then becomes an integral part of the NAS method, and its runtime should be counted as part of the method (see also Best Practice 13).

Best Practice 7: Evaluate Performance as a Function of Compute Resources

While it is important to know the overall compute resources a NAS method required to obtain a result, it would be even more informative to report performance as a function of the required compute resources.[8] This is possible in most cases since most NAS methods are anytime algorithms, and at each time point t we can report the performance of the architecture that *would* be returned if the search was terminated at t (the so-called *incumbent* architecture). This would also take into account that for some search spaces, it is trivial to

8. Compute resources can be measured by different metrics, for example wallclock-time (Coleman et al., 2019), which will be hardware-dependent, or hardware-independent proxies, such as number of floating point operations (FLOPS), which is not always linearly correlated with the runtime (Justus et al., 2018), or in epochs (if these are roughly similarly expensive for different architectures and hyperparameter settings).

obtain nearly the same performance as the optimal architecture, whereas for others this is quite hard. We note that reporting the score of incumbents over compute resources comes at no extra cost when using tabular or surrogate NAS benchmarks.

We also define two possible variants of NAS that differ w.r.t. how performance over compute resources should be measured:

Definition 7 (Architecture identification variant of NAS) *In this variant, only the compute resources for identifying and returning the final architecture counts.*

In an offline evaluation using a private test set, in this variant, at each compute resource step t we would plot the performance of the current incumbent architecture when trained with the final evaluation pipeline.

Definition 8 (AutoML variant of NAS) *In this variant, next to the compute resources for identifying the final architecture, we also count the compute resources to train the final architecture and return a model.*

In an offline evaluation using a private test set, in this variant, at each step t, we would plot the performance of the current incumbent *model* (not architecture), *without retraining*. This reflects a true AutoML setting, where at the end of the compute budget one needs to be directly ready to make predictions.

Since NAS has not been used much in a full AutoML setting, the architecture identification variant is by far most widely used in the literature, but when all that we care about is a good model for a dataset at hand, the AutoML variant may be more suitable.

Best Practice 8: Compare Against Random Sampling and Random Search

As in other fields of machine learning, it is important for NAS research to compare against baselines. The simplest baselines are random sampling and random search (Bergstra and Bengio, 2012). Even though both of these rely on drawing uniform random samples from the architecture space, it is important to note that these are two different algorithms:

Definition 9 (Random sampling) *Random sampling draws a* single *random sample from the architecture space and returns it.*

The runtime of random sampling is therefore basically zero (wrt Definition 7). Its expected performance is the average performance across all architectures in the search space.

Definition 10 (Random search) *Random search draws random samples from the architecture space, evaluates them (with a criterion to be defined, such as a short training run or the full evaluation pipeline), and keeps track of the incumbent architecture with the best evaluation so far. At any given time, when stopped it returns this incumbent architecture.*

Random search is an anytime procedure that should be run for the same amount of time as other approaches being compared to. To minimize confounding factors, it is useful to use the same evaluation criterion as the NAS method being compared to, both for random sampling and search.

Name	Reference	# arch	supports one-shot?	comments
NAS-Bench-101	Ying et al. (2019)	423k	no	constrained space
NAS-Bench-1Shot1	Zela et al. (2020b)	6k – 364k	yes	3 subspaces of NB-101
NAS-Bench-201	Dong and Yang (2020)	6k	yes	3 datasets; learning curves
NAS-HPO	Klein and Hutter (2019)	62 208	no	4 datasets; NAS + HPO
NAS-Bench-NLP	Klyuchnikov et al. (2020)	15k	yes	NLP
NAS-Bench-301	Siems et al. (2020)	10^{18}	yes	surrogate benchmark

Table 1: Overview over tabular and surrogate NAS benchmarks available so far.

Random sampling and random search are extremely simple procedures, but nevertheless many NAS papers avoid a comparison against these baselines. As Sciuto et al. (2019) and Xie et al. (2019) show, random sampling can already yield strong performance in a well-designed search space, and Li and Talwalkar (2019) show that random search can be very competitive. Therefore, we recommend to compare against both of these baselines, to assess whether good performance is due to a well-designed search space (and training pipeline) or due to the NAS method.

Best Practice 9: Perform Multiple Runs with Different Seeds

NAS methods are almost always stochastic. Therefore, re-running the same method on the same dataset does not necessarily lead to the same result (Li and Talwalkar, 2019). Additionally, some results can be quite hard to reproduce even if the source code is available. Sometimes, we observed that we needed several runs of the same code to reproduce the published results, indicating that the authors might have been lucky with the results reported in the paper. Therefore, we recommend that, if possible in terms of compute budgets, all methods should be repeated several times with different seeds and the authors should report mean and standard deviation (or median and quartiles if the noise is not symmetric) across the repetitions. Ideally, one would control and report all used random seeds in a reproducible and simple sequence, e.g., random seeds from 1 to 10. Besides improving the reproducibility of results, this will also provide new insights on the stochasticity of NAS methods in practice. For exact replicability, following Li and Talwalkar (2019), in either case we encourage the release of the exact seeds used for the NAS methods and final evaluation pipelines.

Best Practice 10: Use Tabular or Surrogate Benchmarks If Possible

We note that on standard NAS benchmarks, for most researchers, due to limited computational resources it will be impossible to satisfy the best practices in this section (especially ablation studies and performing several repeated runs). Especially in such cases, we advocate running extensive evaluations on tabular NAS benchmarks, or on surrogate benchmarks as proposed by Siems et al. (2020) for NAS following the work of Eggensperger et al. (2015; 2018). We list available tabular and surrogate NAS benchmarks in Table 1. These benchmarks allow even researchers without any GPU resources to perform systematic, comprehensive and reproducible NAS experiments by querying a table or a perfor-

mance predictor instead of performing a costly optimization on special-purpose hardware. Importantly, by their very design, they also allow fair comparisons of different methods, without the many possible confounding factors of different training pipelines, hyperparameters, search spaces, and so on. We therefore advocate for running large-scale experiments on these tabular/surrogate benchmarks (studying the results of many repetitions, ablation studies, etc), and to complement these comprehensive experiments with additional small-scale experiments on real benchmarks.

We do note, however, that not all methods can be evaluated on tabular/surrogate benchmarks without retraining models (and therefore requiring substantial compute resources). While blackbox and multi-fidelity optimization methods *can* be directly evaluated on these benchmarks only based on efficient lookups of performance, for weight sharing methods (e.g., DARTS (Liu et al., 2019b)) and weight inheritance methods (e.g., LEMONADE (Elsken et al., 2019a)) we cannot speed up the search process. We *can*, however, still speed up the evaluation of the selected architectures (by just looking up their performance) and can likewise plot the performance of the incumbent architecture as a function of the compute resources, as proposed by Zela et al. (2020b).

Since the search phase of weight sharing and weight inheritance methods still needs to be run fully even for tabular/surrogate benchmarks, authors should not be expected to execute many runs of these methods (even when using tabular/surrogate benchmarks).

Best Practice 11: Control Confounding Factors

Even when different papers use the same NAS benchmark, the performance results they report are still often incomparable due to various other confounding factors, such as different hardware, different runtimes used, and even different versions of DL libraries. All these details can substantially impact the results, and we therefore recommend that such confounding factors should be controlled as much as possible. One convenient way of achieving unbiased apples-to-apples comparisons would be to perform experiments using an open-source library of NAS methods that allows running all methods without any confounding factors; see also our note in Section 5 concerning such a library.

4. Best Practices for Reporting Important Details

To ensure reproducibility, it is important to report all the little, but important details that were responsible for the performance achieved by a NAS method. In the following, we propose best practices for details that are often missing but crucial in our experience.

Best Practice 12: Report the Use of Hyperparameter Optimization

A particularly important detail is the hyperparameter optimization approach used. While the hyperparameters of the final evaluation pipeline are part of the NAS benchmark used (see Definition 3) and thus should not be changed without good reason and appropriate emphasis in the reporting of results, every NAS method also has its own hyperparameters. It is well known that these hyperparameters can influence results substantially – e.g., for DARTS (Liu et al., 2019b), they can make the difference between state-of-the-art performance and converging to degenerate architectures with very poor performance (Zela et al.,

2020a). Therefore, firstly (and connected to Best Practices 1, 2, 4 and 11), the used hyperparameter setting is an important experimental detail that should be reported. Secondly, how this setting was obtained is important for applying a NAS method to a new dataset (which may require a different setting). Finally, when facing a new dataset, the time required for hyperparameter optimization should be considered as part of a NAS method's runtime. More than once we have heard statements like *"Of course, NAS method X does not work out of the box for a new dataset, you first need to tune its hyperparameters"*, and we note that this should ring an alarm bell since this just replaces manual architecture engineering by manual hyperparameter optimization of the NAS method. Also, AutoML, by its very definition, needs to be robust; therefore, when viewed from an AutoML point of view, the hyperparameter optimization strategy in essence becomes part of the NAS method and ought to count as part of its runtime. Relatedly, we would also like to remark that statements like *"We only applied a limited amount of hyperparameter optimization"* or *"We slightly tuned the hyperparameters"* are too vague and not useful for reproducing results in a scientific way.

Best Practice 13: Report End-to-End Resources Required for the Entire NAS Method

Related to Best Practice 7, we note that the compute resources required for a NAS method should be measured in an end-to-end fashion. This is particularly important if different NAS methods run differently (see also Best Practice 6). In particular, some NAS methods propose multiple potential architectures after a first phase and then select the final architecture among these in a validation phase. In such a case, in addition to reporting the compute resources for the individual phases, the compute resources required for the validation phase have to be counted as part of the overall resources used for the NAS method.

Example 11 *If a NAS method performs k parallel search runs of time T_{search} and selects the best of the resulting k architectures in a validation phase that takes time T_{valid} for each of the k architectures, and the final architecture takes time T_{final} to train, then the time requirement of the NAS method should be reported as $k \cdot (T_{search} + T_{valid})$ in the architecture identification variant (Definition 7), and as $k \cdot (T_{search} + T_{valid}) + T_{final}$ in the AutoML variant (Definition 8).*

Best Practice 14: Report All the Details of Your Experimental Setup

These days, one of the main foci in NAS is to obtain good architectures faster. Therefore, results typically include the achieved accuracy (or similar metrics) and the time used to achieve these results. However, to assess and reproduce such results, it is important to know the hardware used (type of GPU/TPU, etc) and also the deep learning libraries and their versions.[9] If method A needed twice as much time as method B, but method A was evaluated on an old GPU and method B on a recent one, the difference in GPU may explain the entire difference in speed. Overall, we recommend to report all the details required to

9. Deep Learning libraries, such as tensorflow, pytorch and co are getting more efficient over time, but which version was actually used is unfortunately only reported rarely.

reproduce results—all top machine learning conferences allow for a long appendix, such that space is never a reason to omit these details.

For anyone wanting to publish code (also see Best Practice 2), in order to avoid forgetting to include any software dependencies that are publicly available, a possibility to consider would be to publish a container, e.g., based on Docker or Singularity.

5. Further Ways Forward For the Community

Besides striving for the best practices listed above, we identified two structural problems in the NAS community. We believe that addressing these will facilitate NAS research.

The Need for Proper NAS Benchmarks

The seminal paper by Zoph and Le (2017) used the CIFAR-10 and PTB datasets for its empirical evaluation, and more than 300 NAS papers later, these datasets still dominate in empirical evaluations. While this is nice in terms of comparing methods on standardized datasets, it also involves a big risk of overfitting NAS to them. From a meta learning point of view, we are testing on our training set of two samples – which is obviously not a good basis for developing methods that apply in general.

We do not argue for abandoning these datasets, but we do argue for the creation of a larger, standardized suite of well-defined *NAS benchmarks*. Recall from Definition 3 that such a NAS benchmark includes not only a dataset, but also a search space and a training pipeline with fully available source code and known hyperparameters. For CIFAR-10 and PTB, we do have access to proper NAS benchmarks, based on the search spaces and source code from the DARTS paper (Liu et al., 2019b), and it would be very helpful for the community to have more of these. Access to a set of interesting NAS benchmarks would allow the community to satisfy Best Practice 4 (comparing on the same benchmarks) by construction, and it would also make Best Practice 11 (controlling confounding factors) much easier to follow, since it would allow developers of new NAS methods to compare these under (nearly) the same conditions.

We note that application papers in NAS have already started tackling non-standard applications, such as image restoration (Suganuma et al., 2018), semantic segmentation (Chen et al., 2018; Nekrasov et al., 2018; Liu et al., 2019a), disparity estimation (Saikia et al., 2019), machine translation (So et al., 2019), reinforcement learning (Runge et al., 2019), and GANs (Gong et al., 2019). However, to the best of our knowledge, none of these papers make a clean new NAS benchmark available (as defined above) to complement CIFAR-10 and PTB.[10]

We therefore encourage researchers who work on exciting applications of NAS to create new NAS benchmarks based on their applications. In fact, we believe that at this point of time, a paper that simply evaluates *existing* NAS methods on a new exciting application and makes available a new fully reproducible NAS benchmark based on this would have a more lasting positive impact on the development of the NAS community than a paper introducing a slightly improved NAS method.

10. However, we do note that while this paper was under review, new works introduced tabular NAS benchmarks for natural language processing (NLP, Klyuchnikov et al. (2020)) and automatic speech recognition (ASR, ANOMYNOUS (2020)).

In the long run, we envision a library of NAS benchmarks that is (i) diverse with respect to (a) difficulty, (b) search space properties, (c) datasets and applications, (d) training pipelines; and (e) a mix of real, tabular, and surrogate benchmarks; and that has (ii) a standardized API which allows developers of NAS methods to easily benchmark their ideas on them. Although this might be an ambitious goal in view of the young field, other meta-domains showed that it is in fact feasible and worthwhile (Eggensperger et al., 2013; Hutter et al., 2014; Bischl et al., 2016). First steps towards such libraries are already taken by the series of NAS-Bench papers (Ying et al., 2019; Zela et al., 2020b; Dong and Yang, 2020; Siems et al., 2020; Klyuchnikov et al., 2020).

Properties of a NAS Benchmark

To obtain an interesting set of NAS benchmarks, we propose to aim for diversity in the following properties for NAS search spaces:

Difficulty How hard is it to achieve a good performance on the given benchmark? Many of the current NAS benchmarks are relatively easy in the sense that even random search performs very well (often within 0.5% of optimal), and we would encourage also including some benchmarks where the best architectures perform substantially better than the ones found by random search.

Expressive Power How many architectures can be designed based on the given search space? These days most efficient NAS methods search on fairly limited search spaces (like the typical cell search spaces), and it would be useful to also include much larger search spaces.

Complexity How complex is the search space? Possible examples for quantifying complexity include the number of allowed operators, the potential complexity of pathways through the network, the density of possible connections, etc.

Novelty Does the search space allow to find novel architectures, or does it only include known types of architectures? For new tasks, we are sometimes looking for completely novel architectures, since no well performing architectures are yet known for the task. But on other tasks, we might already know which architectures are likely to work well and are only interested in finding the best architecture in this limited space.

Achievable Performance Since practical applications of NAS aim for obtaining new state-of-the-art performance it is useful for NAS benchmarks to include networks that yield competitive performance (rather than only optimizing architectures that are clearly dominated). However, NAS benchmarks that have proven useful to the community should not be disregarded once other types of networks achieve better performance, but should still be preserved to warrant comparability of results over the years and also provide a potentially cheaper experimentation environment than the newest (and likely most expensive) benchmarks.

Likewise, we strongly encourage a library of NAS benchmarks to include a set of diverse applications, datasets and training pipelines.

Practical Recommendations for Tabular and Surrogate NAS Benchmarks

Since several new tabular NAS benchmarks are being introduced these days, we would also like to give some recommendations on how to create a good tabular NAS benchmark:

1. Release the table in an easy-to-access form. Papers reporting on tabular NAS benchmarks for which the table is not available are not very useful for the community.

2. Release code. While a tabular benchmark can be useful in itself, it is far more useful if the code for generating it is also released (again, along with hyperparameters and, optimally, seeds). If code is not available, the benchmark cannot be used to evaluate NAS methods based on weight sharing or weight inheritance.

3. If possible, keep track of, and release other metrics being computed, such as the number of weights, flops, etc.

Likewise, for building surrogate NAS benchmarks, as discussed by Siems et al. (2020, Appendix F), special consideration should be paid to data collection, construction of several surrogate models, verification and validation, version numbering, and release of training data & source code.

The Need for an Open-Source Library of NAS Methods

In addition to a well-defined benchmark suite, there is also a need for an open-source library of NAS methods that allows for (i) a common interface to NAS methods, (ii) the control of confounding factors, (iii) fair and easy-to-run comparisons of different NAS methods on several benchmarks and (iv) an assessment of how important each component of a NAS method is. Although several well-engineered and flexible AutoDL packages were recently proposed (Jin et al., 2019; O'Malley et al., 2019; Erickson et al., 2020; Zimmer et al., 2020), their main goal is ease of use, rather than allowing clean comparisons of several NAS methods. Libraries of methods have had a very positive impact on other fields (e.g., RLlib (Liang et al., 2018), Tensorforce (Kuhnle et al., 2017) or OpenAI (Dhariwal et al., 2017)), and we expect a similarly positive impact for the field of NAS. The DeepArchitect library (Negrinho and Gordon, 2017) already proposed a unified search space definition, and several recent projects are making further progress along these lines (Shah, 2020; Ning et al., 2020; Ruchte et al., 2020). Optimally, an open-source library would allow researchers to implement their algorithms easily and in a search-space-independent fashion to directly follow most of the best practices given here by minimizing confounding factors. If the library allows unified access to a broad collection of NAS benchmarks, and if it allows the comparison of different methods without confounding factors, work carried out on top of it will satisfy many of the best practices described here by construction. Besides facilitating NAS research, such a library could also have a great impact by giving interested users of NAS access to the best current NAS approaches.

6. Conclusion

We proposed 14 best practices for scientific research on neural architecture search (NAS) methods. We believe that gradually striving for them as guidelines will increase the scien-

tific rigor of NAS papers and help the community to make sustained progress on this key problem.

Similar to Joelle Pineau's reproducibility checklist, we have compiled the best practices for NAS research described here into a checklist for authors and reviewers alike. We hope that this checklist will help to easily assess the state of a paper. This checklist is available at the following URL: `http://automl.org/NAS_checklist.pdf`.

6. Acknowledgement

We thank Thomas Elsken, Arber Zela, Katharina Eggensperger, Matthias Feurer, as well as the anonymous reviewers, for comments on an earlier draft of this note.

6. References

ANOMYNOUS. NAS-Bench-ASR: reproducible neural architecture search for speech recognition. *OpenReview*, 2020. `https://openreview.net/forum?id=CU0APx9LMaL`.

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

B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

J. Buckheit and D. Donoho. Wavelab and reproducible research. In *Wavelets and statistics*, pages 55–81. Springer, 1995.

L. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. Searching for efficient multi-scale architectures for dense image prediction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the international conference on Advances in Neural Information Processing Systems 31 (NeurIPS)*, pages 8713–8724, 2018.

C. Coleman, D. Kang, D. Narayanan, L. Nardi, T. Zhao, J. Zhang, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia. Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark. *ACM SIGOPS Oper. Syst. Rev.*, 53(1):14–25, 2019.

E. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. Le. AutoAugment: Learning Augmentation Policies from Data. In *Proceedings of the international conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

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

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines, 2017. URL `https://github.com/openai/baselines`.

X. Dong and Y. Yang. NAS-Bench-201: extending the scope of reproducible neural architecture search. In *Proceedings of the International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HJxyZkBKDr`.

J. Dy and A. Krause, editors. *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *JMLR Workshop and Conference Proceedings*, 2018. JMLR.org.

K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NeurIPS workshop on Bayesian Optimization in Theory and Practice*, 2013.

K. Eggensperger, F. Hutter, H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1114–1120. AAAI Press, 2015.

K. Eggensperger, M. Lindauer, H. Hoos, F. Hutter, and K Leyton-Brown. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107: 15–41, 2018.

S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

T. Elsken, J. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *Proceedings of the International Conference on Learning Representations*, 2019a. URL `https://openreview.net/forum?id=ByME42AqK7`.

T. Elsken, J. Metzen, and F. Hutter. Neural architecture search. In Hutter et al. (2019), pages 69–86. Available at http://automl.org/book.

N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv:2003.06505 [stat.ML]*, 2020.

D. Fanelli, R. Costas, and J. Ioannidis. Meta-assessment of bias in science. *Proceedings of the National Academy of Sciences*, 114(14):3714–3719, 2017.

X. Gastaldi. Shake-shake regularization. In *Proceedings of the International Conference on Learning Representations Workshop (ICLR)*, 2017.

X. Gong, S. Chang, Y. Jiang, and Z. Wang. AutoGAN: Neural architecture search for generative adversarial networks. In *International Conference on Computer Vision (ICCV)*, 2019.

P. Henderson, R.t Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In S. McIlraith and K. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 3207–3214. AAAI Press, 2018.

F. Hutter, M. López-Ibáñez, C. Fawcett, M. Lindauer, H. Hoos, K. Leyton-Brown, and T. Stützle. Aclib: A benchmark library for algorithm configuration. In *Proceedings of the international conference on Learning and Intelligent Optimization*, volume 8426 of *Lecture Notes in Computer Science*, pages 36–40. Springer, 2014.

F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019. Available at http://automl.org/book.

H. Jin, Q. Song, and X. Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019.

D. Justus, J. Brennan, S. Bonner, and A. McGough. Predicting the computational cost of deep learning models. In *IEEE International Conference on Big Data*, pages 3873–3882. IEEE, 2018.

A. Klein and F. Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv:1905.04970 [cs.LG]*, 2019.

N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, and E. Burnaev. NAS-Bench-NLP: neural architecture search benchmark for natural language processing. *arXiv: 2006.07116 [cs.LG]*, 2020.

A. Kuhnle, M. Schaarschmidt, and Kai K. Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. Web page, 2017. URL `https://github.com/tensorforce/tensorforce`.

L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. In A. Globerson and R. Silva, editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, page 129. AUAI Press, 2019.

E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for distributed reinforcement learning. In Dy and Krause (2018), pages 3059–3068.

S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim. Fast autoaugment. *arXiv:1905.00397 [cs.LG]*, 2019.

C. Liu, L. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei. Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation. *arXiv:1901.02985 [cs.CV]*, 2019a.

H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019b.

I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

R. Negrinho and G. Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv:1704.08792 [stats.ML]*, 2017.

V. Nekrasov, H. Chen, C. Shen, and I. Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. *arXiv:1810.10804 [cs.CV]*, 2018.

R. Nickerson. Confirmation bias: A ubiquitous phenomenon in many guises. *Review of general psychology*, 2(2):175–220, 1998.

X. Ning, W. Li, Z. Zhou, T. Zhao, Y. Zheng, S. Liang, H. Yang, and Y. Wang. A surgery of the neural architecture evaluators. *arXiv:2008.03064 [cs.CV]*, 2020.

T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, et al. Keras Tuner. `https://github.com/keras-team/keras-tuner`, 2019.

H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In Dy and Krause (2018), pages 4092–4101.

M. Ruchte, A. Zela, J. Siems, J. Grabocka, and F. Hutter. NASLib: a modular and flexible neural architecture search library. `https://github.com/automl/NASLib`, 2020.

F. Runge, D. Stoll, S. Falkner, and F. Hutter. Learning to design RNA. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

T. Saikia, Y. Marrakchi, A. Zela, F. Hutter, and T. Brox. AutoDispNet: Improving disparity estimation with AutoML. *arXiv:1905.07443 [cs.CV]*, 2019.

C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann. Evaluating the search phase of neural architecture search. *arXiv:1902.08142 [cs.LG]*, 2019.

S. Shah. Archai, 2020. `https://github.com/microsoft/archai`.

J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv:2008.09777 [cs.LG]*, 2020.

D. So, C. Liang, and Q. Le. The evolved transformer. *arXiv:1901.11117 [cs.LG]*, 2019.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

M. Suganuma, M. Ozay, and T. Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In Dy and Krause (2018), pages 4771–4780.

S. Xie, A. Kirillov, R. Girshick, and K. He. Exploring randomly wired neural networks for image recognition. *arXiv:1904.01569 [cs.CV]*, 2019.

A. Yang, P. Esperança, and F. Carlucci. NAS evaluation is frustratingly hard. In *Proceedings of the International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HygrdpVKvr`.

C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114. PMLR, 2019.

A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. Understanding and robustifying differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2020a. URL `https://openreview.net/forum?id=H1gDNyrKDS`.

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

H. Zhang, M. Cissé, Y. Dauphin, and D. Lopez-Paz. Mixup: Beyond empirical risk minimization. *arXiv:1710.09412 [cs.LG]*, 2017.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *arXiv:2006.13799 [cs.LG]*, 2020.

B. Zoph and Q. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

B. Zoph, V. Vasudevan, J. Shlens, and Q. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710. IEEE Computer Society, 2018.