Best Practices for Scientific Research on Neural Architecture Search

Marius Lindauer[1] **& Frank Hutter**[1,2]
[1] Department of Computer Science, University of Freiburg
[2] Bosch Center for Artificial Intelligence
{lindauer,fh}@cs.uni-freiburg.de

Abstract

We describe a set of best practices for the young field of neural architecture search (NAS), which lead to the best practices checklist for NAS available at `http://automl.org/nas_checklist.pdf`.

1 Introduction

Neural architecture search (NAS) is currently one of the hottest topics in automated machine learning (AutoML), see the book by Hutter et al. (2019) for an overview), with a seemingly exponential increase in the number of papers written on the subject (see the figure on the right). While many NAS methods are fascinating (see the survey article by Elsken et al. (2019) 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.

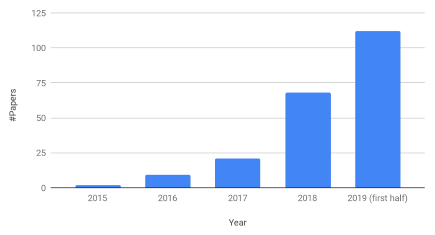

Figure 1: NAS papers per year based on the literature list on `automl.org`. The number for 2019 only considers the first half of 2019.

Although NAS methods steadily improve, the quality of empirical evaluation in this field is still lagging behind compared to other areas in machine learning, AI and optimization. We would therefore like to share some best practices for empirical evaluations of NAS methods, which we believe would 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[1] showed how to improve empirical evaluations of reinforcement learning algorithms, and several of her points carry over to NAS. For the NAS domain itself, Li & Talwalkar (2018) also recently released a paper discussing 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 papers accepted at recent conferences, and you will very likely find examples for most of the pitfalls we list.

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 & 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."

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

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. 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. The training pipeline includes the specifics of the optimization and regularization methods used. For example, for image datasets, next to the choice of optimizer and number of training epochs, important choices include activation functions (e.g., Swish (Elfwing et al., 2018)), learning rate schedules (e.g., cosine annealing (Loshchilov & Hutter, 2017)), data augmentation (e.g. by CutOut (Devries & Taylor, 2017), MixUp (Zhang et al., 2017) or Auto-Augment (Cubuk et al., 2019)), and regularization (e.g., by Dropout (Srivastava et al., 2014), Shake-Shake (Gastaldi, 2017), ScheduledDropPath (Zoph et al., 2018), or decoupled weight decay (Loshchilov & Hutter, 2019)). For example on CIFAR-10, each of these may improve the validation error rates a bit, whereas choosing the best neural architecture often has smaller effects (Li & Talwalkar, 2018; Liu et al., 2018; Cubuk et al., 2019).

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 or Pytorch 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, availability of the main source code strongly facilitates reproducing results on one's own setup.

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 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 the NeurIPS 2019 submission guidelines pointing to Joelle Pineau's reproducibility checklist[2]), the trend at top machine learning

[2] `https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf`

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).

Corrolary 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 (Pham et al., 2018; Liu et al., 2019b; Ying et al., 2019) that demonstrate that sharing NAS code is possible for industrial players if they want to.

3 Best Practices for Comparing NAS Methods

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*:

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

Two examples of such benchmarks are as follows:

Example 4. *A good example for a 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).*[5]

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

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 Practice 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 result (on a dataset like CIFAR-10). While NAS papers often get accepted based on these performance numbers (see also our note to reviewers in Section 5), 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. Therefore, we recommend to run ablation analyses to study the importance of individual algorithm components.

[3]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).

[4]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.

[5]The only thing that is unfortunately not available in their repo are the hyperparameter settings they used for their 100-epochs evaluation.

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: Compare Performance over Time

While knowing the overall runtime a NAS method required to obtain a result is very important, it would be even more informative to report performance as a function of the required time. This is possible since most NAS methods are anytime algorithms, and at each time point t one can report the performance of the architecture that *would* be returned if the search was terminated at t. This would also take into account that for some search spaces, it is trivial to obtain nearly the same performance as the optimal architecture, whereas for others this is quite hard.

Best Practice 8: Compare Against Random Search

As in other fields of machine learning, it is important for NAS research to compare against baselines. The simplest baseline is random search (i.e., sampling architectures uniformly at random and evaluating them), but somehow, many NAS papers avoid a comparison against this baseline. In our own experiments we observed that random search can be quite strong in a well-designed search space, and the evidence recently published by Sciuto et al. (2019) and Li & Talwalkar (2018) corroborates that finding. Therefore, we recommend to compare against random search, 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: Validate The Results Several Times

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 & Talwalkar, 2018). We acknowledge that running experiments for NAS can be very expensive; however, most recent methods often only need a few GPU days or even less. On the other hand, we experienced that 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 method to reproduce the results, indicating that the authors might have been lucky with the results reported in the paper. Therefore, we recommend that all methods should be repeated several times with different seeds and the authors report mean and standard deviation (or median and quartiles if the noise is not symmetric) across the repetitions. Besides improving the reproducibility of the results, this will also provide new insights on the stochasticity of NAS methods in practice. For exact replicability, following Li & Talwalkar (2018), we also 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 in such cases, we advocate running extensive evaluations on tabular benchmarks, such as NAS-Bench-101 (Ying et al., 2019) and the NAS-HPO benchmarks (Klein & Hutter, 2019), or on surrogate benchmarks as proposed by Eggensperger et al. (2015; 2018) and used by Falkner et al. (2018). These benchmarks allow even researchers without any GPU resources to perform systematic, comprehensive and reproducible NAS experiments by querying a table / a performance 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 non-tabular 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 versions of DL libraries, and different times to run the various methods. All these details can substantially impact the results, and we therefore recommend that such confounding factors should be controlled as much as possible (which often implies that X and Y have to be assessed by using the same hardware, DL libraries and so on). We encourage authors of individual papers to make a best effort to minimize these confounding factors.

We note that in the long run a better solution to allow unbiased apples-to-apples comparisons would be to develop an open-source library of NAS methods; see also our note in Section 5 concerning such a library.

4 Best Practices for Reporting Important Details

Best Practice 12: Report the Use of Hyperparameter Optimization

A particularly important detail is the hyperparameter optimization approach used. While the hyperparmaeters 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 emphasis in reporting results, every NAs method also has its own hyperparameters. It is well known that these hyperparameters can influence results substantially. Therefore, first of all (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). Last but not least, 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 a big alarm bell for everyone: AutoML, by its very definition, needs to work out of the box; 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. Also, 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.

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

Related to Best Practice 7, we note that the time for a NAS method has to be measured in an end-to-end fashion, i.e., the time between starting the NAS method and it returning the final architecture. 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 times for the individual phases, the time required for the validation phase has to be counted as part of the overall time used for the NAS method.

Example 6. *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, then the time requirement of the NAS method should be reported as $k \cdot (T_{search} + T_{valid})$.*

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

5 Ways Forward For the Community

The Need for Proper NAS Benchmarks

The seminal paper by Zoph & Le (2017) used the CIFAR-10 and PBT datasets for its empirical evaluation, and more than 200 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. Development and evaluation of NAS methods is done on the same datasets, so from a meta learning point of view, we are testing on our training set of two samples – obviously not a good idea when we want to know which methods would generalize.

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 PBT, we do have access to proper NAS benchmarks, based on the search spaces and source code from the DARTS paper (Liu et al., 2019b).

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 makes available a clean new NAS benchmark (as defined above) to complement CIFAR-10 and PTB.

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.

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. Such libraries of methods have had a very positive impact on other fields (e.g., RLlib (Liang et al., 2018)), and we expect a similarly positive impact for the field of NAS. While Kamath et al. (2018) already proposed a framework along these lines in the past, we are not aware of any established and maintained library of NAS methods. Optimally, such a library would also allow researchers to implement their algorithms easily and follow the best practices given here without much overhead.

Besides facilitating NAS research, such a library would also have a great impact for applying the best current NAS approaches to new datasets. We have therefore started developing such a library internally, but we would see much value in this becoming a community effort.

[6]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.

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 scientific 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`.

Acknowledgement

We thank Thomas Elsken, Arber Zela, and Matthias Feurer for comments on an earlier draft of this note.

References

J. Buckheit and D. Donoho. Wavelab and reproducible research. In *Wavelets and statistics*, pp. 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 (eds.), *Proceedings of the international conference on Advances in Neural Information Processing Systems 31 (NeurIPS)*, pp. 8713–8724, 2018.

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.

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

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*, 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. Neural architecture search. In Hutter et al. (2019), pp. 69–86. Available at http://automl.org/book.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Dy & Krause (2018), pp. 1436–1445.

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.

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

P. Kamath, A. Singh, and D. Dutta. AMLA: an AutoML framework for neural network design. In *International workshop on Automated Machine Learning*, 2018.

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

L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. In A. Globerson and R. Silva (eds.), *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 129. AUAI Press, 2018.

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 & Krause (2018), pp. 3059–3068.

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, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

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.

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.

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

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.

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 & Krause (2018), pp. 4771–4780.

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 (eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7105–7114. PMLR, 2019.

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

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)*, pp. 8697–8710. IEEE Computer Society, 2018.