Long-term Reproducibility for Neural Architecture Search

David Towers, Matthew Forshaw, A. Stephen McGough
Newcastle University, UK
{d.towers2, matthew.forshaw, stephen.mcgough}
@newcastle.ac.uk

Amit Atapour-Abarghouei
Durham University, UK
amir.atapour-abarghouei
@durham.ac.uk

Abstract

It is a sad reflection of modern academia that code is often ignored after publication – there is no academic 'kudos' for bug fixes / maintenance. Code is often unavailable or, if available, contains bugs, is incomplete, or relies on out-of-date / unavailable libraries. This has a significant impact on reproducibility and general scientific progress. Neural Architecture Search (NAS) is no exception to this, with some prior work in reproducibility. However, we argue that these do not consider long-term reproducibility issues. We therefore propose a checklist for long-term NAS reproducibility. We evaluate our checklist against common NAS approaches along with proposing how we can retrospectively make these approaches more long-term reproducible.

1. Introduction

The popularity of Deep Learning (DL) has increased significantly over the last decade. It has enticed us with its ability to solve complex problems, such as image classification, without the need for manual feature identification. However, this has been at the cost of now not knowing the 'best' neural architecture to use for a given problem.

Neural Architecture Search (NAS) is a promising field which offers greater potential for finding the 'best' architecture for a given data set and problem space – without needing to train all possible architectures and/or the need of a deep learning expert. Historically, NAS has focused on high accuracy; however, more recently, a move to allow a combination of multiple metrics (e.g., accuracy, memory footprint, number of parameters) has emerged [1, 5, 13].

The potential for NAS was demonstrated by Zoph and Le [19] where they outperformed all but one of the human-designed networks they compared against. This potential was limited due to the number of GPU hours it took to find a solution – which has become the primary driver for future NAS research. New algorithms have reduced computation time – achieving orders of magnitude improvement [1, 2, 6, 7, 9, 12–15, 20]. However, as the authors move on, their code has been left unmaintained. This is problematic, as it makes it more difficult to use these codes for other data sets and problem spaces, and for comparison with new approaches.

However, there is little incentive to spend time updating old code. While an author could, every few years, spend time updating their previous projects to work with the updated libraries (a task with no academic reward and eventually all-time-consuming), they instead use this time to work on new research, which is more likely to lead to citations. Often driven by the infamous academic mantra of "Publish or Perish" [4]. This pressure can take away time academics would otherwise use for activities that do not result in publications, such as teaching students [17] or maintaining code. This neglect can leave codes which are bug-ridden, lacking necessary documentation, or simply out-of-date.

These issues lead to codes that cannot be used without being at least partially re-written. This problem hinders further research based on these codes, removes the ability for others to use the approach for real-world applications and prevents the purpose of the field, which is to replace manually-designed architectures.

Some prior work has looked at the problem of reproducibility [11, 16] by proposing a checklist to improve reproducibility of work at the time of publication. This has focused on standards, making sure that the code is complete and that all dependencies are resolvable. However, this only ensures that the work is reproducible at the time of publication. If we seek to make the work reproducible many years after publication, this brings in an additional set of constraints, which need to be considered. These include the long-term availability of dependencies for the code, how changes to libraries effect the running of the code and the availability of a containerised version of the software to encapsulate the code and environment to ensure a longer-term reference used for validation and comparison.

In an ideal world, all authors would adhere to our checklist – including a containerisation and ease of use. However, this is unlikely to happen for legacy NAS, nor can we expect 'usable' systems from new developments. As such, we propose the development of a Docker container approach for

legacy NAS approaches and a framework for NAS development separating the NAS approach from core tasks such as dataset input and comparison.

2. Related Work

Accessibility of Software Artifacts Many pieces of work are missing software artefacts, hosting dead links, or linking to bug-ridden code. Heumüller *et al.* show that of 604 reviewed papers that contain mention of software artefacts between 2007 and 2017, only 289 (47.8%) shared links to these artefacts, and only 163 (27%) were available. However, they do also show an increase in software artefact availability during this period [8]. This may suggest a more significant push for reproducibility in recent years.

This increase is due to several initiatives to encourage artefact sharing, such as ACM badging, including three categories of badges: Artifacts Evaluated, Artifacts Available, and Results Validated, awarded to papers based on a set of criteria. While there is not enough evidence to suggest a cause-and-effect relationship between these initiatives and the increase in software artefact availability, there is enough to suggest these initiatives have influenced this trend [3].

However, these initiatives do not include prolonged maintenance. Artefact Available badges are awarded for the artefact having persistent storage and an available link. The Artefact Evaluated is awarded for the quality of the artefact at the time of the award. There are no criteria for bug fixing, providing pseudocode, or ensuring usable code if the dependencies are no longer available.

Reproducibility in NAS Even when code is made available, there are still problems with reproducibility. We should be able to use virtualisation tools and settable (random) seeds to reproduce the exact experimental conditions. Li and Talwalker found that none of the twelve papers published between 2018-2019 that produced novel NAS works had included their experimental seeds [10], highlighting historical threats to reproducibility in NAS.

Reproducibility Checklists Lindauer and Hutter highlight difficulties in reproducing NAS algorithms. They have identified many pitfalls that authors can fall into when publishing their work. They offer a checklist of best practices authors can follow to reduce the chance of succumbing to these problems [11]. However, the points raised in their work are for reproducibility at the time of publication but does not encourage practices to prolong reproducibility.

The machine learning checklist[1] by Pineau *et al.* [16] is adopted by both ICML and NeurIPS. This checklist has several sections covering all aspects of reproducibility. One of these sections concerns related code and contains points regarding pre-trained models, code availability, and good documentation practices. While this work encourages reproducibility by asking for the code and a list of dependencies, it does not require the code to be bug-free, and if one of those dependencies becomes inaccessible, the code will no longer work.

While we agree with all points in both works, we believe they do not go far enough regarding software artefacts. While they include points about ensuring the code is released, there is little in making sure the code is usable and future-proofed. Following all their checkpoints could still result in code that does not work in a few years. Our checklist is designed to encourage prolonged reproducibility, and hopefully become the basis of a unifying NAS framework which will allow the works to be run into the future.

3. Reproducing Current Works

During our investigations into a range of common NAS algorithms [2, 12, 15, 18], we have found that it is not always a simple and easy process to get them working. Some algorithms still require infeasibly large amounts of computational power to complete such as the original NAS [19]. Others need specific versions of libraries or code updating to run. 75% of the algorithms we have attempted to get working have had some significant issues.

DARTS [12] for example, was one of the simplest to reproduce. However, they state[2] a dependency PyTorch 0.3.1. This is technically available, though only for MacOS[3]. In order to mitigate this, certain sections of DARTS were updated to PyTorch 1.0+ (specific changes on our GitLab page[4]). These are the sort of long-term issues which are not as obvious at the time of publication.

ENAS [15] is a more difficult case. On its GitHub page[5], they note the Language Model implementation is now hosted elsewhere. The Image Recognition implementation remains in the repository but lacks a requirements list in the documentation. It can be found, however, in an answered issue. There were also bugs in the code. Community members have fixed these bugs in the issues section. However, these fixes were not pushed to the main branch. Furthermore, ENAS needs a particular version of the `cudatoolkit` and `CuDNN` libraries. One of which is archived and needs an NVIDIA account to access, and then needs to be manually installed.

Containerisation tools, such as Docker, can alleviate some of these problems. A properly set-up Dockerfile will recreate the developer environment, so a dependency list would not be required as Docker would handle this automatically. Docker can also be used to save an image of

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

[2] https://github.com/quark0/darts

[3] https://pytorch.org/get-started/previous-versions

[4] https://gitlab.com/D-Towers/nas-reproducibility

[5] https://github.com/melodyguan/enas

Criteria	Descriptor	ENAS	DARTS	PC-DARTS	DrNAS
Code Stability					
Seed Reporting	The code should return the seed used in the logs/output.	✗	✓	✓	✓
Seed Setting	The seed should be able to be set.	✗	✓	✓	✓
Bug-Free	The code should be free of bugs which prevent sections of code from working.	✗	✓	✓	✗[i]
Documentation					
Examples	There should be examples of all the high-level commands.	✓	✓	✓	✓
Argument Details	The documentation should include details regarding additional arguments that can be applied to the high-level commands.	✗	✗	✗	✗
Dependency List	Environment specification indicating how to replicate the environment.	✗	✓	✓	✓
Pipeline Instructions	Instructions on how to search, train, and test a model from scratch.	✗[ii]	✗[iii]	✗[iii]	✗[iii]
Ease of Running					
Dependency Resilience	Dependencies should be installed with the code.	✗	✗	✗	✗
Executable	The code can be executed from an executable file, such as a bash script.	✓	✗	✗	✗
Intuitive Commands	The commands should be indicative to their function.	✓	✓	✓	✓
Standardisation					
Standard Phases	Search, Train, and Test.	✗	✓	✓	✗
Data Inclusive	Data is retrieved from a directory named `data` that accepts common data formats.	✗	✓	✓	✓
Standard Output	The model is saved in an output folder in a form that is easily imported into other scripts.	✗	✓	✓	✓
Future Proofing					
Accessible Pseudocode	Pseudocode should be made available to allow recreation of the code if all other methods are no longer available.	✗	✗	✗	✗
Containerised Environment	A compressed file of the containerised image (such as tar), which allows the entire environment to be saved, dependencies included.	✗	✗	✗	✗
Container Build File	A file containing the required info for a container manager to build the environment and run the code.	✗	✗	✗	✗
Source Code	Source code should be publicly accessible to allow others to inspect and change local versions.	✓	✓	✓	✓

Table 1. Evaluated NAS algorithms using our proposed checklist. [i]Code is missing a 'hp', which needs to be manually entered. [ii] Does not mention that you need to change the default architecture in the training file. [iii] Does not explicitly explain how to train a found architecture.

a container, providing a permanent working environment. While this increases the file size considerably, it has several advantages of providing a full set of dependencies, a known working environment and tool, along with removing the issue of dependencies no longer being available.

4. Proposed Checklist

We propose a checklist containing five categories which combines desired features we believe could help increase reproducibility in NAS. Some points in the checklist may become superseded with the inclusion of others. This redundancy should maintain reproducibility if some of the other aspects are not met.

In comparison to the already-existing checklists [11, 16], we have focused on aspects which we believe will help prolong reproducibility, with points such as dependency resilience, which ensures the code will remain usable even if a dependency is no loner available.

Table 1 shows our checklist applied to four common NAS approaches (ENAS [15], DARTS [12], PC-DARTS [18] and DrNAS [2]). The evaluated approaches have not been future-proofed well, as can be seen from the table. This is evidenced by DARTS requiring a no longer

accessible library. We are reproducing several NAS algorithms into a working state in Docker containers. This should allow others to build on our working containers with a working environment. Currently, we have reproduced ENAS into a working Docker Container. These containers will be available, along with any changes we made to the code, on our GitLab[4].

5. Conclusion

This paper has proposed steps which will increase the level of reproducibility in NAS. Some of our proposals are not fully maintainable. In the containers that we are creating for some of these older algorithms, while we have abstracted away most of the dependency installation, and can remove reliance on them using compressed environments, we still require nvidia-docker or one of its equivalents to be installed to allow access to the GPU. Docker is also a dependency which we are relying on a lot for these proposals. With these two dependencies, what happens if either of them are no longer hosted?

While this possibility exists, we imagine that new tools would replace them, along with converters that would allow companies which are heavily based on docker to migrate. Even now, there are several options: Docker, Podman, PyPi, and regular VMs.

Our proposals are not definitive, but we are hoping that they will at least prolong the life of code, and we have tried to include suitable redundancies so if some of the check points are not feasible, the code should still be reproducible.

We will extend this work by developing a framework removing the burden of reproducibility from the NAS developer thus allowing them to focus on their NAS.

References

[1] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *arXiv:1812.00332 [cs, stat]*, Feb. 2019. 1

[2] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. DrNAS: Dirichlet Neural Architecture Search. *arXiv:2006.10355 [cs, stat]*, Mar. 2021. arXiv: 2006.10355 version: 4. 1, 2, 3

[3] Bruce R. Childers and Panos K. Chrysanthis. Artifact Evaluation: FAD or Real News? In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1664–1665, Apr. 2018. ISSN: 2375-026X. 2

[4] Mark De Rond and Alan N. Miller. Publish or Perish: Bane or Boon of Academic Life? *Journal of Management Inquiry*, 14(4):321–329, Dec. 2005. 1

[5] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution. *arXiv:1804.09081 [cs, stat]*, Feb. 2019. 1

[6] Rob Geada and Andrew Stephen McGough. SpiderNet: Hybrid Differentiable-Evolutionary Architecture Search via Train-Free Metrics. Technical Report arXiv:2204.09320, arXiv, Apr. 2022. arXiv:2204.09320 [cs] type: article. 1

[7] Rob Geada, Dennis Prangle, and Andrew Stephen McGough. Bonsai-Net: One-Shot Neural Architecture Search via Differentiable Pruners. *arXiv:2006.09264*, June 2020. 1

[8] Robert Heumüller, Sebastian Nielebock, Jacob Krüger, and Frank Ortmeier. Publish or perish, but do not forget your software artifacts. *Empirical Software Engineering*, 25(6):4585–4616, Nov. 2020. 2

[9] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-Wisely Supervised Neural Architecture Search With Knowledge Distillation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1986–1995, Seattle, WA, USA, June 2020. IEEE. 1

[10] Liam Li and Ameet Talwalkar. Random Search and Reproducibility for Neural Architecture Search. *arXiv:1902.07638 [cs, stat]*, July 2019. arXiv: 1902.07638. 2

[11] Marius Lindauer and Frank Hutter. Best Practices for Scientific Research on Neural Architecture Search. Technical Report arXiv:1909.02453, arXiv, Nov. 2020. 1, 2, 3

[12] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. *arXiv:1806.09055 [cs, stat]*, Apr. 2019. arXiv: 1806.09055. 1, 2, 3

[13] Zhichao Lu, Gautam Sreekumar, Erik Goodman, Wolfgang Banzhaf, Kalyanmoy Deb, and Vishnu Naresh Boddeti. Neural Architecture Transfer. *IEEE Trans. PAMI*, 43(9):2971–2989, Sept. 2021. 1

[14] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural Architecture Search without Training. *arXiv:2006.04647 [cs, stat]*, June 2021. 1

[15] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient Neural Architecture Search via Parameters Sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, July 2018. 1, 2, 3

[16] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Hugo Larochelle. Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program). *arXiv: Learning*, Mar. 2020. 1, 2, 3

[17] Seema Rawat and Sanjay Meena. Publish or perish: Where are we heading? *Journal of Research in Medical Sciences : The Official Journal of Isfahan University of Medical Sciences*, 19(2):87–89, Feb. 2014. 1

[18] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. *arXiv:1907.05737 [cs]*, Apr. 2020. 2, 3

[19] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. *arXiv:1611.01578 [cs]*, Feb. 2017. arXiv: 1611.01578. 1, 2

[20] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, Salt Lake City, UT, June 2018. IEEE. 1