Neural Architecture Search and tinyML

Daniel Duclos-Cavalcanti

***Abstract*—There is no denying the increasing success that Deep Neural Networks (DNNs) have displayed across various tasks such as image classification, speech recognition, machine translation and many others. This progress can be attributed largely due to *architecture engineering*, which is a process that requires immense domain expertise, intuition and some amount of trial and error. This is not ideal as it is both time consuming and error-prone. Neural Architecture Search (NAS) rises as the next logical step aiming to both automate architecture discovery and further the understanding of the inner workings of DNNs. This field has grown remarkably within the last 5 years and among the many challenges of NAS, there are concerns regarding computational costs and time feasability. This however changes within the context of *tinyML*, which is an expanding field at the intersection of machine learning and embedded systems. Due to the resource constrained conditions involved in most embedded devices and tinyML applications, the stages concerning training and performance evaluation are substantially faster in comparison to the more complex models used in the research of NAS. Thus presenting an interesting opportunity to explore NAS applications within a different paradigm. Throughout this work, an overview of existing research in NAS, specifically concerned with the use of evolutionary algorithms and reinforcement learning methods will be presented, as well as highlighting relevant applicabilities to tinyML.**

***Index Terms*—Neural Architecture Search (NAS), evolutionary computation (EC), Deep Learning, tinyML.**

I. Introduction

The advancement of deep learning, although extremely beneficial, has also caused a continuous demand for architecture design. This coupled with a growing model complexity still requires ample time and expert knowledge to continue this progress.

Finally, after the work in [1] proposed by Google, it was shown for the first time that NAS algorithms have the potential to find models that rival the current state of the art. However, done so in an automated fashion, minimizing human participation. Since then, many different methods have appeared.

To better visualize NAS, one can categorize it within three dimensions [2]:

- **Search Space**: defines all the possible architectures that can in principle be considered.
- **Search Strategy**: defines how the *search space* is explored by the algorithm.
- **Performance Estimation Strategy**: defines how performance is evaluated at every architecture iteration.

A. Search Space

NAS is an optimization problem, whose search space is the defining factor to its complexity. The smaller the search space is, the faster the search may converge, as well as requiring less computational resources. This comes at the cost of less freedom to explore unseen architectures and also possibly limiting the complexity of the design.

The simplest approach is to define an architecture as a *chain-structured neural network*, which essentially consists of a sequence of layers whose inputs are the output of their preceding layer. In this case the space is parametrized by maximum number of layers, type of operations per layer, and hyperparameters conditioned by the chosen operation.

The next step consists then of including more modern design elements such as skip connections, which has already been seen in [1] and [3]. This allows to build more complex *multi-branch networks*, which cannot be described as simple sequential layer chaining, but as a structure where each layer's input is a function of previous layer outputs. Therefore, a chain-structured neural network can be seen as an edge-case or simplification of the multi-branch network. This increases significantly the degrees of freedom of architecture design, which leads to a much larger search space.

Another predominant trend includes the search within *cells* or *blocks*, initially considered by works such as [4] and [5]. What is proposed, is to break-off architectures in cells, such that the search space is then designated within a single cell per time. This drastically reduces the search space, as there are substantially less layers within a cell in comparison to an entire architecture. Additionally, subdivision of architectures into blocks is considered a good design practice, which also enables easy transferability of blocks to other data sets. The same modelling used on multi-branch networks can be used with cells, simply replacing layers with cell architectures.

This drives then the discussion between *micro-architectures* versus *macro-architectures*. The macro architecture attempts to determine how cells should be connected and how many are needed to build a model. On the other hand, the micro architecture aims to find the optimal structure for each cell. Ideally, both viewpoints should be optimized jointly, which of course leads to a complex search space. There have been efforts that aimed to minimize this endeavor by fixing macro-architectures with known working topographies such as in [6] with DenseNet [7]. This practice, dubbed as *human knowledge injection* attempts to reduce the search space through using domain expertise known to obtain effective results. This includes human bias in the model.

B. Search Strategy

As any space search problem, there is a *exploration-exploitation* trade-off to be considered [2]. Obtaining a high-performing architecture within a feasible amount of time is desired. However, converting too early to a suboptimal result is also not the goal.

Based on the current state of the art, NAS search algorithms can be classified mainly intro three different categories [8]:

*1) **Reinforcement Learning** (RL)-based NAS Algorithms:* If one considers the development of a neural architecture the agent's action, where the action space is the same as the search space, it is then possible to frame NAS as a reinforcement learning problem [2]. After evaluating the performance of the given trained architecture on fitness data, it is possible to determine the agent's reward. How this estimation is performed will depend on the given method, more to these approaches will be seen at I-C. Furthermore, how the agent's policy is represented and it's optimization will also vary. More on trade-offs and details to these approaches will be seen in II.

*2) **Gradient**-based NAS Algorithms:* Consists of transforming the search space from discrete to continuous and performing gradient descent with respect to the fitness data set.

*3) **Evolutionary Computation** (EC)-based NAS Algorithms:* By the application of well established EC methods, which are various techniques based on the evolution of species within nature, many different efforts were implemented to navigate their respective search space. These varied from genetic algorithms (GAs), genetic programming (GP) up until particle swarm optimization (PSO) techniques.

C. Performance Estimation Strategy

Independent of the search strategy, it is necessary to know how any given architecture performed in order to guide the next steps of the algorithm. There are many ways to estimate said performance, whereas the simplest form would be complete training and validation. Given the complexity and size of the search spaces within NAS, this requires GPU days in the order of thousands [2]. This is why extensive research has been employed to reduce time on performance estimation, since it is by far the greatest bottleneck. Some of the approaches such as lower fidelity techniques, learning curve extrapolation, weight inheritance and one-shot models were utilized to ensure achievable periods of search.

II. State of the Art

A. Overview

Currently RL-based algorithms are extremely costly in terms of computation, requiring thousands of graphics processing cards (GPUs) for days. This is already the case for median-scale data sets, as as data grows more complex, so does the processing time neede to search for a suitable architecture.

B. Evolutionary Computation-based Efforts

C. tinyML and NAS

III. Conclusion and Future Work

References

[1] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

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

[3] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 2902–2911. [Online]. Available: https://proceedings.mlr.press/v70/real17a.html

[4] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2423–2432.

[5] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[6] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 678–687. [Online]. Available: https://proceedings.mlr.press/v80/cai18a.html

[7] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[8] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE transactions on neural networks and learning systems*, 2021.