Automating Neural Network Design: Neural Architecture Search for Small and Efficient Models - Keynote

Aaron Klein[1,*]

[1]ELLIS Institute Tübingen

Abstract

Deep neural networks have achieved remarkable performance across a wide range of tasks; however, deploying them efficiently in practice often requires careful and task-specific architectural design. Neural Architecture Search (NAS) offers a data-driven framework to automate this process, enabling the systematic discovery of architectures that balance predictive performance and resource efficiency. In this paper, we present an accessible overview of the core concepts underlying NAS, including search spaces, optimization strategies, and multi-objective formulations. We further highlight several practical application scenarios, such as hardware-aware model design, model compression, and the development of efficient small-scale language models. Through these examples, we illustrate how NAS can serve as a key enabler for building practical, lightweight, and resource-efficient AI systems.

Keywords

Neural Architecture Search, Deep Neural Networks, Automated Machine Learning,

1. Introduction

Deep neural networks have achieved great success across various domains. However, training neural networks involves numerous architectural choices that typically require significant human expertise and extensive trial and error. This challenge becomes particularly pronounced when developing compact models for deployment in resource-constrained environments, where efficiency is as critical as accuracy. As the adoption of deep neural networks increases, inference costs, in terms of latency, energy consumption, and financial costs for hosting endpoints will continue to grow. Since they recur with every query, inference costs will eventually exceed the already substantial costs associated with training.

The goal of automated machine learning (AutoML) [1] is to progressively automate parts of the machine learning pipeline. AutoML has been quite successful in the past, for example, by outperforming human experts on Kaggle competitions [2] or configuring the Monte-Carlo Tree Search of Alpha-Go [3], and is deployed in several commercial services, such as Google Vizier, H2O or Amazon SageMaker. *Neural architecture search (NAS)* [4] has emerged as a subfield of AutoML, aiming to design better neural network architectures to improve performance and efficiency compared to manually designed counterparts.

In this paper, we discuss the potential of NAS to find more resource-efficient neural network architectures. Section 2 introduces NAS and explains the relevant concepts. Section 3 details three applications of NAS aimed at accelerating inference and training.

Proceedings of the First Workshop on Small and Efficient Large Language Models for Knowledge Extraction (SmaLLEXT) @ CIKM 2025

*Corresponding author.

kleiaaro@gmail.com (A. Klein)

https://aaronkl.github.io/ (A. Klein)

0009-0002-6761-7374 (A. Klein)

2. Neural Architecture Search

The goal of neural architecture search (NAS) [5, 6] is to find an architecture, encoded as $\theta \in \Theta$ that minimizes the prediction error: $\min_{\theta \in \Theta} L_{\text{valid}}(\boldsymbol{w}^\star, \theta)$ on some hold-out validation data. Here, L_{valid} denotes the validation error after training the neural network to obtain the optimal weights $\boldsymbol{w}^\star$ that minimize the training loss $\mathcal{L}_{train}$.

In practice, we are often interested not only in an architecture that minimizes the validation error but also in one that performs efficiently, for example, in terms of memory footprint, inference latency, or energy consumption. We can generalize the definition above to a multi-objective problem, $\min_{\theta \in \Theta}(f_{\text{error}}(\theta), f_{\text{efficiency}}(\theta))$, to simultaneously optimize the validation error (with $f_{\text{error}} = L_{\text{valid}}$) and the efficiency of the architecture. In the multi-objective setting, there is no single $\boldsymbol{\theta}_\star \in \Theta$ that simultaneously optimizes all $M = 2$ objectives. Let's define $\boldsymbol{\theta} \succ \boldsymbol{\theta}'$ iff $f_i(\boldsymbol{\theta}) \leq f_i(\boldsymbol{\theta}'), \forall i \in [M]$ and $\exists i \in [M] : f_i(\boldsymbol{\theta}) < f_i(\boldsymbol{\theta}')$. We aim to find the *Pareto Set*: $P_f = \{\boldsymbol{\theta} \in \Theta | \nexists \boldsymbol{\theta}' \in \Theta : \boldsymbol{\theta}' \succ \boldsymbol{\theta}\}$ of points that dominate all other points in the search space in at least one objective. W

Search Spaces: When applying NAS, the first step is to define the search space, Θ. Ideally, the search space should be expressive enough to yield new architectures that are both efficient and high-performing. However, since evaluating a single architecture θ often requires substantial computational resources, increasing the dimensionality of the search space quickly renders the search process computationally infeasible [7]. Typical search spaces could include for example the number of layers, the number of hidden units per layer or the activation function.

Previous work [8, 9, 10] also considers *cell search spaces*, which define a directed acyclic graph (cell) in which each node corresponds to a specific operation (e.g., a 3×3 convolution or max-pooling). The objective is to select the appropriate operation for each node as well as the connectivity (edges) between nodes. To construct the final architecture, these cells are stacked multiple times. While each cell maintains its own set of weights, the architectural structure is shared across all cells.

Search Methods: Given a defined search space, the next step is to identify the optimal architecture (in a single-objective setting) or a set of Pareto-optimal architectures (in a multi-objective setting). We can distinguish between two primary classes of search methods[7].

The first class comprises black-box or gradient-free optimization approaches, which fully train and evaluate an architecture θ each time the objective function $f_{\text{error}}(\theta)$ is evaluated. This process is inherently expensive and does not scale well to larger deep learning models, which may require hours or even days of training across multiple GPUs.

This process can be accelerated by employing early stopping for architectures that perform poorly in the early stages of training. So-called multi-fidelity approaches [11, 12, 13] extend the definition of the objective function to $f_{\text{error}}(\theta, r)$, where $r \in [r_{\min}, r_{\max}]$ specifies the amount of resources, such as the number of epochs or the size of the training dataset allocated to evaluating the neural network.

The second class of methods is often referred to as *weight-sharing* approaches [14, 15, 16]. In contrast to black-box methods that train multiple independent neural networks, weight-sharing methods train a single *super-network* that encapsulates all possible architectures ($\forall \theta \in \Theta$) within the search space. Once this super-network has been trained, we can evaluate $f_{\text{error}}(\theta)$ by selecting the subset of weights from the super-network corresponding to θ; this is known as a sub-network. Consequently, during the evaluation phase, we only need to validate the architectures rather than train them from scratch, making this approach orders of magnitude faster than black-box optimization.

However, when training the super-network, it is crucial to ensure that the sub-networks function effectively in isolation [17, 15]. Therefore, state-of-the-art [17, 15] approaches modify the training procedure so that, instead of updating all weights at every step, only the weights of specifically sampled sub-networks are updated.

3. Applications

In the following section, we discuss recent work on leveraging NAS to design efficient architectures for small-scale models, highlighting its potential to accelerate progress toward practical, lightweight, and resource-efficient AI systems.

3.1. Hardware-aware Neural Architecture Search

The first application we consider involves using NAS to search for architectures that are tailored to specific hardware devices. Especially for embedded systems, strict constraints on inference latency and memory footprint must be satisfied.

A popular approach in this setting is the once-for-all network proposed by Cai et al. [15], which trains a single super-network. To ensure strong performance of the sub-networks, the authors introduce a progressive shrinking strategy that enforces a training order from larger to smaller sub-networks. For deployment on a specific hardware device, they then apply an evolutionary algorithm to select the optimal sub-network that jointly optimizes latency on the target device and validation error. In a similar vein, Sukthanker et al. [18] propose a search space for super-network training of decoder-based [19] transformer architectures for language modeling tasks. Since directly measuring inference latency or energy consumption on the target device is often impractical, they introduce surrogate models trained on a limited number of hardware measurements to approximate these efficiency metrics.

3.2. Model Compression

Weight-sharing neural architecture search can be viewed as a form of structural pruning that, after training, removes structural components such as layers or neurons to obtain an optimal sub-network. Ideally, this sub-network achieves performance as close as possible to that of the original super-network.

For example, Klein et al. [20] use weight-sharing based NAS to compress encoder-based language models via multi-objective optimization, jointly reducing parameter count and validation error. Compared to classical structural pruning approaches, this strategy enables the identification of an entire Pareto front of models rather than a single compressed solution.

3.3. Warm-starting Pre-training of Small Language Models

Small language models [21] have recently attracted increased attention because, compared to large frontier models, they have substantially fewer parameters. Especially in academic settings or for specialized use cases, such models are appealing due to their relatively low inference costs while still achieving competitive performance on specific downstream tasks. However, pre-training these models remains computationally expensive and often exceeds the budgets of small academic labs or startups, amounting to multiple millions of GPU hours.

A recent line of work [22, 23] explores the use of neural architecture search to identify sub-networks within large pre-trained models for initializing smaller models. For instance, Muralidharan et al. [22] apply NAS-based pruning to a teacher model to extract compact sub-networks, which are subsequently refined through knowledge distillation.

Krishnakumar et al. [23] propose a constrained evolutionary search strategy to identify the sub-network that minimizes perplexity and serves as an effective initialization for a smaller model. They showed that, compared to the conventional approach of training from randomly initialized weights, these methods can significantly enhance both the efficiency and the overall effectiveness of the pre-training process.

4. Conclusions

We discussed Neural Architecture Search (NAS) as a principled framework for automatically identifying the optimal architecture of a deep neural network while jointly optimizing predictive performance and

efficiency. By formulating architecture design as a structured optimization problem, NAS enables a systematic and data-driven exploration of architectural choices that would otherwise require extensive manual engineering.

We outlined the core components of NAS, including the definition of the search space that captures all candidate architectures and the methods used to efficiently explore this space. In particular, we highlighted that NAS can be seen as a multi-objective optimization problem allowing practitioners to balance accuracy with computational and hardware constraints.

Finally, we presented three practical use cases that demonstrate the versatility of NAS: hardware-aware NAS for tailoring models to specific devices, model compression through structured sub-network selection, and the use of sub-networks from large pre-trained checkpoints to warm-start the training of small language models. Together, these examples illustrate the potential of NAS as a key enabler for developing efficient, scalable, and practically deployable AI systems.

Declaration on Generative AI

During the preparation of this work, the author used ChatGPT, Grammarly in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Automated Machine Learning: Methods, Systems, Challenges, Springer, 2019.

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

[3] Y. Chen, A. Huang, Z. Wang, I. Antonoglou, J. Schrittwieser, D. Silver, N. de Freitas, Bayesian optimization in AlphaGo, arXiv:1812.06855 [cs.LG] (2018).

[4] T. Elsken, J. H. Metzen, F. Hutter, Neural architecture search: A survey, Journal of Machine Learning Research (JMLR'19) (2019).

[5] B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, in: International Conference on Learning Representations (ICLR'17), 2017.

[6] E. Real, A. Aggarwal, Y. Huang, Q. V. Le, Regularized Evolution for Image Classifier Architecture Search, in: Proceedings of the Conference on Artificial Intelligence (AAAI'19), 2019.

[7] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, F. Hutter, Neural architecture search: Insights from 1000 papers, arXiv:2301.08727 [cs.LG] (2023).

[8] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le, Learning transferable architectures for scalable image recognition, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18), 2018.

[9] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, F. Hutter, NAS-Bench-101: Towards reproducible neural architecture search, in: Proceedings of the 36th International Conference on Machine Learning (ICML'19), 2019.

[10] X. Dong, Y. Yang, Nas-bench-201: Extending the scope of reproducible neural architecture search, in: International Conference on Learning Representations (ICLR'20), 2020.

[11] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: Bandit-based configuration evaluation for hyperparameter optimization, in: International Conference on Learning Representations (ICLR'17), 2017.

[12] S. Falkner, A. Klein, F. Hutter, BOHB: Robust and efficient hyperparameter optimization at scale, in: Proceedings of the 35th International Conference on Machine Learning (ICML 2018), 2018.

[13] A. Klein, S. Falkner, S. Bartels, P. Hennig, F. Hutter, Fast Bayesian optimization of machine learning hyperparameters on large datasets, in: Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS'17), 2017.

[14] H. Pham, M. Guan, B. Zoph, Q. Le, J. Dean, Efficient neural architecture search via parameters sharing, in: Proceedings of the 35th International Conference on Machine Learning (ICML'18), 2018.

[15] H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han, Once-for-all: Train one network and specialize it for efficient deployment, in: International Conference on Learning Representations (ICLR'20), 2020.

[16] G. Bender, P. Kindermans, B. Zoph, V. Vasudevan, Q. Le, Understanding and simplifying one-shot architecture search, in: Proceedings of the 35th International Conference on Machine Learning (ICML'18), 2018.

[17] J. Yu, P. Jin, H. Liu, G. Bender, P. J. Kindermans, M. Tan, T. Huang, X. Song, R. Pang, Q. Le, BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models, in: The European Conference on Computer Vision (ECCV'20), 2020.

[18] R. S. Sukthanker, A. Zela, B. Staffler, A. Klein, L. Purucker, J. K. H. Franke, F. Hutter, HW-GPT-bench: Hardware-aware architecture benchmark for language models, in: The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track (NeurIPS'24), 2024.

[19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language models are unsupervised multitask learners (2019).

[20] A. Klein, J. Golebiowski, X. Ma, V. Perrone, C. Archambeau, Structural pruning of pre-trained language models via neural architecture search, Transactions on Machine Learning Research (TMLR) (2024).

[21] L. B. Allal, A. Lozhkov, E. Bakouch, G. M. Blázquez, G. Penedo, L. Tunstall, A. Marafioti, H. Kydlíček, A. P. Lajarín, V. Srivastav, J. Lochner, C. Fahlgren, X. Nguyen, C. Fourrier, B. Burtenshaw, H. Larcher, H. Zhao, C. Zakka, M. Morlon, C. Raffel, L. von Werra, T. Wolf, Smollm2: When smol goes big – data-centric training of a small language model, arXiv:2502.02737 [cs.CL] (2025).

[22] S. Muralidharan, S. T. Sreenivas, R. Joshi, M. Chochowski, M. Patwary, M. Shoeybi, B. Catanzaro, J. Kautz, P. Molchanov, Compact language models via pruning and knowledge distillation, in: The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track (NeurIPS'24), 2024.

[23] A. Krishnakumar, R. S. Sukthanker, H. J. Mahadik, G. Kadlecová, V. Moroshan, T. Carstensen, F. Hutter, A. Klein, Where to begin: Efficient pretraining via subnetwork selection and distillation, arXiv:2510.07227 [cs.CL] (2025).