SOTA Report: Neural Architecture Search

November 24, 2024

Abstract

Over the last ten years, deep learning has made great improvements in areas such as computer vision, natural language understanding, speech recognition, and reinforcement learning. The success of deep learning in these fields is greatly dependent on the use of specialized and powerful neural network designs. Neural architecture search (NAS) is a way to automate the process of designing these networks for specific tasks. This has already led to better results than what humans have designed in some cases. Since 2020, NAS research has grown a lot, with more than 1,000 papers published. In this survey, we give an overview of the NAS field. We explain the different types of search spaces, methods, and ways to speed up the process. We also discuss useful resources such as benchmarks, best practices, other surveys, and open-source tools.

Keywords: neural architecture search, automated machine learning, deep learning

1 Introduction

Deep learning has grown to become a key approach in machine learning over the last ten years, contributing to breakthroughs in several areas. These include computer vision, natural language understanding, speech recognition, and reinforcement learning; it is also becoming a very powerful approach for the analysis of tabular data. Deep learning became popular due to its ability to automate feature extraction, as well as improvements in data availability and computational power. However, a key part of its success has been the creation of effective neural network architectures.

Lately, just as deep learning replaced manual feature engineering with automated feature learning, the process of designing neural architectures is now being automated through neural architecture search (NAS). NAS involves automatically creating neural network designs for specific tasks. It has already produced architectures that outperform those designed by humans in many areas. This includes tasks like ImageNet classification, as well as less common datasets and situations with limited memory or processing power.

2 Problem Formulation

Neural Architecture Search (NAS) aims to automate the design of neural network architectures by identifying an optimal architecture $a \in \mathcal{A}$ from a predefined search space $\mathcal{A}$, such that the architecture achieves the best performance for a given task.

The NAS problem consists of several key components. The search space $\mathcal{A}$ defines the set of possible architectures that can be explored. These architectures are typically represented as directed acyclic graphs (DAGs), repeatable cells, or sequential layers, depending on the design. The optimization objective is often the validation performance of the architecture, although other metrics, such as latency, memory usage, or energy efficiency, can also be considered. The performance estimation strategy determines how the quality of a candidate architecture is evaluated. This evaluation may involve fully training the architecture, approximations like weight sharing, or using proxies to reduce computational cost. Additionally, NAS often operates under practical constraints, such as computational budget, inference latency, or model size, which further shape the search process.

NAS faces several challenges that make it a complex optimization problem. The search space is often vast, encompassing billions or trillions of potential architectures, which makes exhaustive search infeasible. Evaluating each candidate architecture is computationally expensive, especially when full training is required. Furthermore, balancing exploration and exploitation during the search introduces trade-offs between discovering novel architectures and refining known promising ones.

Variations of the NAS problem have emerged to address specific application needs. Multi-objective NAS seeks to optimize multiple objectives simultaneously, such as maximizing accuracy while minimizing latency. Constraint-aware NAS restricts the search space to architectures that meet hardware-specific requirements, making it suitable for resource-constrained environments like mobile devices. Transferable NAS focuses on discovering architectures that generalize well across different datasets or tasks. These variations highlight the versatility and adaptability of NAS methodologies to meet diverse real-world demands.

3 Theoretical Background

NAS surveys described NAS using three main parts: search space, search strategy, and performance evaluation strategy (see Figure 1). These terms are important to understand most NAS methods, and we explain them below.

Search Space: This is the set of all possible neural architectures that the NAS algorithm can choose from. Search spaces can range from a few thousand to more than 10^{20} options. While search spaces can be very broad, using domain knowledge can make the search easier. However, relying too much on domain knowledge might introduce human bias and limit the chance of discovering new architectures.

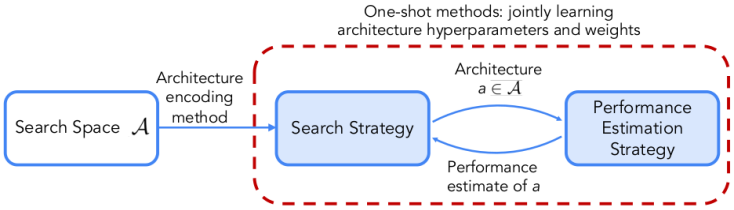

Figure 1: Overview of neural architecture search. A search strategy iteratively selects architectures (typically by using an architecture encoding method from a predefined search space $\mathcal{A}$. The architectures are passed to a performance estimation strategy, which returns the performance estimate to the search strategy. For one-shot methods, the search strategy and performance estimation strategy are inherently coupled.

Search Strategy: This refers to the method used to explore the search space and find a good architecture. There are two main types: Black-box optimization: Techniques like reinforcement learning, Bayesian optimization, and evolutionary algorithms. One-shot methods: These include supernet-based and hypernet-based approaches. Some methods don't fit neatly into either category.

Performance Estimation Strategy: This is a way to predict how well an architecture will perform without fully training it. For example, methods like learning curve extrapolation can make the search much faster. Without these strategies, we would have to train and evaluate every selected architecture fully, which is very time-consuming.

At its simplest, NAS can be defined as follows: Given a search space $\mathcal{A}$, a dataset $\mathcal{D}$, a training pipeline $\mathcal{P}$, and a time or computation budget t, the goal is to find an architecture $a \in \mathcal{A}$ within the budget t that performs best on the validation data when trained using dataset $\mathcal{D}$ and training pipeline $\mathcal{P}$. This is often expressed as:

$$\min_{a \in \mathcal{A}} \mathcal{L}_{\text{val}}(w^*(a), a) \quad \text{s.t.} \quad w^*(a) = \arg\min_{w} \mathcal{L}_{\text{train}}(w, a).$$

Here, $\mathcal{L}_{\text{val}}$ and $\mathcal{L}_{\text{train}}$ represent validation and training losses, respectively. While this is the basic definition, there are variations. For instance, some NAS methods include constraints like limiting the number of parameters or use meta-learning to improve results.

3.1 Search Space

The search space is one of the most important parts of NAS. Unlike other areas of AutoML, where similar optimization methods might be used, the architectural search space is specific to NAS. It's also usually the first thing to set up when starting NAS. Most well-known search spaces are designed for specific tasks and are inspired by the best manually designed architectures in their

fields. For instance, NAS-Bench-101, a commonly used search space for image classification, was influenced by ResNet and Inception.

The design of the search space involves a trade-off between human influence and search efficiency. A small search space with many preselected options makes it easier for NAS algorithms to find a good architecture quickly. However, a larger search space with more basic building blocks might take longer to explore but has the potential to uncover completely new architectures.

3.1.1 Terminology

The terminology used to describe search spaces can vary across the literature, depending on the type of search space. To avoid confusion, here we define the main terms:

- Operation/Primitive: This refers to the smallest unit in the search space. For most popular search spaces, it's a combination of a fixed activation function, an operation, and a fixed normalization method. For example, "ReLU-conv 1x1-batchnorm" consists of a fixed ReLU activation, a 1x1 convolution operation, and batch normalization.
- Layer: In chain-structured or macro search spaces, this term often means the same as an operation or primitive. However, it can also describe common combinations of operations, such as the inverted bottleneck residual.
- Block/Module: This usually refers to a sequential stack of layers, following the conventions used in most chain-structured and macro search spaces.
- Cell: In cell-based search spaces, this represents a directed acyclic graph (DAG) of operations. The number of operations within a cell is typically fixed.
- Motif: A motif refers to a sub-pattern formed by multiple operations in an architecture. In some papers, a cell is considered a higher-level motif, while a smaller group of operations is seen as a base-level motif.

3.1.2 Macro Search Spaces

Macro search spaces allow NAS to explore entire architectures represented as directed acyclic graphs (DAGs), where nodes are operations like convolution or pooling layers. These spaces can focus on designing full networks or optimizing macro-level hyperparameters, such as depth and width, as seen in EfficientNet. While macro search spaces are flexible and capable of discovering novel architectures, their large size makes them computationally demanding to explore.

3.1.3 Chain-Structured Search Spaces

Chain-structured search spaces organize operations sequentially, inspired by architectures like ResNet and MobileNetV2. These spaces allow NAS to optimize

parameters such as kernel sizes and expansion ratios, as done in ProxylessNAS. They are computationally efficient and simple to implement, but their fixed topology limits the potential for discovering entirely new designs.

3.1.4 Cell-Based Search Spaces

Cell-based search spaces focus on optimizing small, repeatable units (cells) instead of entire architectures. For example, NASNet uses normal and reduction cells, which are then stacked to build the network. This approach simplifies the search while maintaining flexibility. However, its reliance on fixed cell structures can restrict the discovery of more diverse designs.

3.1.5 Hierarchical Search Spaces

Hierarchical search spaces enable multi-level designs where motifs at lower levels are combined into more complex structures. Examples include MnasNet, which integrates macro-level parameters with cell designs. These spaces balance complexity and flexibility, allowing exploration of diverse architectures, though they can be challenging to navigate.

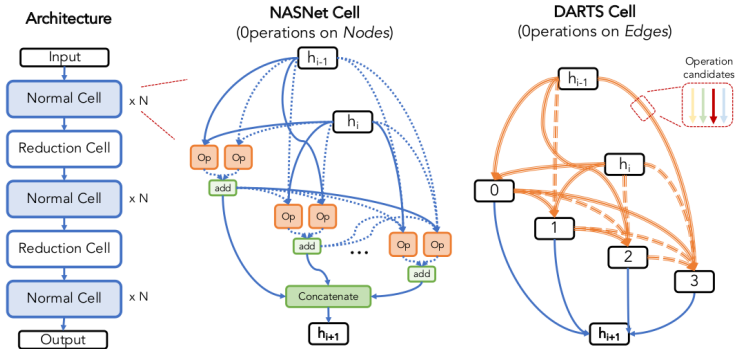

Figure 2: Illustration of cell-based search spaces. The outer skeleton across cells (left) is fixed, while the cells are searchable. NASNet assigns operations to nodes (middle) while DARTS assigns operations to edges (right).

3.2 Optimization Techniques

Now that we have discussed search spaces, we turn to one of the most studied parts of NAS: the search strategy. This is the method used to find the best architecture within the search space. Search strategies are usually divided into two main types: black-box optimization methods and one-shot methods. However, some approaches combine features of both or don't fit neatly into either category.

3.2.1 Baselines

A simple baseline for NAS is random search, where architectures are randomly selected from the search space, fully trained, and the one with the best validation accuracy is chosen. Surprisingly, random search often performs well, as shown in several studies. This is particularly true in well-designed search spaces with a high proportion of strong architectures. For instance, with a budget of k evaluations, random search is expected to find architectures within the top $(100/k)\%$ of the search space. However, on larger and more diverse search spaces, random search tends to underperform. Despite this limitation, random search is still recommended as a baseline for comparing new NAS algorithms.

Local search has also emerged as a strong baseline for NAS in both small and large search spaces. The simplest form of local search involves iteratively training and evaluating all neighbors of the best architecture found so far, where neighbors are defined as architectures that differ by one operation or edge. This approach can be significantly accelerated using network morphisms, which warm-start the optimization of neighboring architectures.

3.2.2 Black-Box Optimization Techniques

Black-box optimization treats the architecture evaluation process as a black box, focusing on exploring and optimizing the search space without assuming any specific structure or gradient information. Common methods include reinforcement learning (RL), evolutionary algorithms, Bayesian optimization (BO), and Monte Carlo tree search (MCTS). In these methods, architectures are trained independently, and their performance is used to guide the search. RL-based NAS uses a controller, often a recurrent neural network, to generate architectures and updates its policy based on rewards from performance feedback. Evolutionary algorithms mimic natural selection by iteratively mutating and recombining architectures to discover better-performing designs. BO leverages a probabilistic surrogate model, such as Gaussian processes, to predict the performance of architectures and balances exploration of new designs with exploitation of promising ones. MCTS builds a decision tree to navigate structured search spaces efficiently, using rollouts to estimate architecture performance. Black-box methods are robust, conceptually simple, and highly adaptable to new tasks and datasets. However, they can be computationally expensive, as they require fully training and evaluating numerous architectures

- *Reinforcement learning (RL)* approaches NAS as a sequential decision-making problem. A controller, often implemented as an RNN, generates candidate architectures, and their performance is used as a reward signal to update the controller's policy. Over iterations, the controller learns to prioritize generating higher-performing architectures. RL has been successfully applied to NAS, achieving competitive results across benchmarks. However, it is computationally expensive due to the need to evaluate many architectures. Techniques like weight sharing and proxy tasks have been proposed to mitigate this cost and improve efficiency

- *Evolutionary algorithms (EAs)* emulate natural evolution by iteratively improving a population of architectures. Starting with a randomly initialized population, EAs apply genetic operators such as mutation (small random changes to an architecture) and crossover (combining parts of two architectures) to generate new candidates. The performance of each candidate is evaluated, and the top-performing architectures are selected for the next generation. This process continues until the search budget is exhausted or satisfactory performance is achieved. EAs are particularly effective for exploring large, diverse search spaces and are naturally suited for parallelization. However, their reliance on full training for performance evaluation can make them resource-intensive

- *Bayesian optimization (BO)* builds a probabilistic model (surrogate model) of the objective function to approximate architecture performance based on past evaluations. This model is updated iteratively as new architectures are tested, allowing BO to predict performance and prioritize promising candidates. Techniques like acquisition functions are used to balance exploration (searching for diverse architectures) and exploitation (refining high-performing designs). BO is particularly useful for expensive-to-evaluate search spaces, as it minimizes the number of evaluations needed to find good architectures. However, BO's scalability can be limited in high-dimensional or highly structured search spaces

- *Monte Carlo tree search (MCTS)* is a decision tree-based technique originally designed for games like Go and Chess, but it has been adapted for NAS to explore structured search spaces. MCTS represents architectures as nodes in a tree and uses simulations (rollouts) to estimate the performance of unexplored nodes. Based on these estimates, MCTS dynamically grows the tree by focusing on promising regions of the search space while maintaining some degree of exploration. MCTS is particularly effective for hierarchical or graph-based search spaces and offers a structured way to balance exploration and exploitation during the search process

3.2.3 One-Shot Techniques

One-shot techniques have emerged as a powerful solution to address the computational inefficiencies of traditional NAS methods. These techniques rely on training a single over-parameterized model, referred to as a "supernetwork", that encompasses all possible architectures within the search space. This approach eliminates the need to train each candidate architecture independently, significantly reducing computational costs while enabling rapid evaluation.

- *Non-Differentiable Supernet-Based Methods* methods train the supernetwork first and then use non-differentiable optimization strategies, such as random search or evolutionary algorithms, to identify high-performing architectures. For example, the ENAS framework trains a supernetwork while simultaneously optimizing the architecture selection process using

reinforcement learning. The shared weights in ENAS allow for rapid evaluation of architectures, greatly improving efficiency compared to traditional NAS approaches.

- *Differentiable Supernet-Based Methods* , such as DARTS, relax the discrete search space into a continuous one, enabling gradient-based optimization. In these methods, the architecture weights and supernetwork parameters are optimized jointly, allowing for a highly efficient search process. By leveraging gradient descent, differentiable techniques scale well to large search spaces and provide competitive performance.

- *Hypernetwork-Based Methods* , such as DARTS, relax the discrete search space into a continuous one, enabling gradient-based optimization. In these methods, the architecture weights and supernetwork parameters are optimized jointly, allowing for a highly efficient search process. By leveraging gradient descent, differentiable techniques scale well to large search spaces and provide competitive performance.

4 Methodology

To begin the search in the filed we started by analysing a recent survey of it [7]. It described the general terminology, formal framework and the techniques used, and also pointed to papers and articles containing in-depth information about each. We proceeded further by analysing papers in the field introducing the techniques mentioned in the survey:

[8] introduced a method to automate neural network design using a controller RNN trained via reinforcement learning. This approach achieved state-of-the-art results on datasets like CIFAR-10 and Penn Treebank, surpassing manually crafted architectures. As of November 2024, this influential work has been cited over 5,000 times, highlighting its significant impact on the field.

[3] introduces a differentiable approach, enabling efficient gradient-based optimization instead of costly traditional methods. Using bilevel optimization, it achieves state-of-the-art results on CIFAR-10 and competitive performance on ImageNet in just a few GPU days. Despite concerns about generalization from proxy tasks and high memory cost, DARTS has set a new benchmark in automated architecture design.

[4] introduces ENAS, a method that reduces NAS computational costs by sharing parameters among architectures within a supernet. Using a reinforcement learning-based controller, ENAS efficiently identifies high-performing architectures, achieving competitive results on CIFAR-10 and Penn Treebank while drastically cutting search costs. This approach has significantly advanced scalable and practical NAS methods.

[6] introduces a graph-based neural predictor within a Bayesian Optimization framework to efficiently model and search neural architectures. BANANAS achieves state-of-the-art results on benchmarks like CIFAR-10 and ImageNet,

with reduced computational cost, making it a significant contribution to BO-based NAS approaches.

[5] introduces the NEAT algorithm, a groundbreaking method that evolves both the structure and weights of neural networks using genetic algorithms. NEAT starts with simple network topologies and incrementally complexifies them over generations, balancing innovation and efficiency. This approach proved effective in solving tasks like pole balancing and robot control, showcasing its ability to adapt network architectures to task requirements. As a foundational work in neuroevolution, NEAT has profoundly influenced subsequent research in neural network optimization and automated architecture design.

[2] introduces hypernetworks that generate weights for a target neural network, reducing trainable parameters and improving efficiency. Widely cited, this work has significantly influenced neural network optimization and NAS research.

Finally [1] introduces NAS-Bench-201, a benchmark dataset designed to standardize and extend reproducibility in Neural Architecture Search (NAS). It provides a fixed search space with 15,625 architectures and their corresponding performance on CIFAR-10, CIFAR-100, and ImageNet16-120, enabling fair comparisons across NAS methods. By offering precomputed results, the benchmark eliminates computational overhead during experimentation, fostering consistent evaluation and accelerating NAS research. This work has become a cornerstone for reproducible NAS studies.

5 Implementation

This section aims to present an recreate an implementation of a NAS techniques from a paper. We choose [5] to reproduce as it was the simplest of them. We applied evolutionary algorithm trying to find the best architecture for the CIFAR10 dataset. The evolutionary algorithms used $insert-details-about-the-algorithm-such-as-the-way-it-generates-the-population, -how-it-evolves-and-mutates$. After letting it train for x generation, the best accuracy of y was obtained by the network

$$-insert-figure-of-network-$$

The code that generated this network can be found on here

6 Conclusion

Neural Architecture Search (NAS) has made huge progress over the past few years. NAS algorithms have become way more efficient, even improving by orders of magnitude. There are now tools that let us compare NAS algorithms without needing GPUs, and researchers have developed lots of new techniques and different search spaces. NAS-discovered architectures are now state-of-

the-art for many tasks. But there are still plenty of challenges and exciting opportunities for future research.

References

[1] Dong, X., Yang, Y.: NAS-Bench-201: Extending the scope of reproducible neural architecture search. In: International Conference on Learning Representations (ICLR) (2020), https://openreview.net/forum?id=HJxyZkBKDr

[2] Ha, D., Dai, A., Le, Q.V.: Hypernetworks. arXiv preprint arXiv:1609.09106 (2016)

[3] Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)

[4] Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: International conference on machine learning. pp. 4095–4104. PMLR (2018)

[5] Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)

[6] White, C., Neiswanger, W., Savani, Y.: Bananas: Bayesian optimization with neural architectures for neural architecture search. In: Proceedings of the AAAI conference on artificial intelligence. vol. 35, pp. 10293–10301 (2021)

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

[8] Zoph, B.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)