Systematic review on neural architecture search

Salmani Pour Avval, Sasan; Eskue, Nathan D.; Groves, Roger M.; Yaghoubi, Vahid

DOI
10.1007/s10462-024-11058-w

Publication date
2025

Document Version
Final published version

Published in
Artificial Intelligence Review

Artificial Intelligence Review (2025) 58:73
https://doi.org/10.1007/s10462-024-11058-w

Systematic review on neural architecture search

Sasan Salmani Pour Avval[1,2] · Nathan D. Eskue[1] · Roger M. Groves[1] · Vahid Yaghoubi[1,2]

Accepted: 29 November 2024

Abstract
Machine Learning (ML) has revolutionized various fields, enabling the development of intelligent systems capable of solving complex problems. However, the process of manually designing and optimizing ML models is often time-consuming, labor-intensive, and requires specialized expertise. To address these challenges, Automatic Machine Learning (AutoML) has emerged as a promising approach that automates the process of selecting and optimizing ML models. Within the realm of AutoML, Neural Architecture Search (NAS) has emerged as a powerful technique that automates the design of neural network architectures, the core components of ML models. It has recently gained significant attraction due to its capability to discover novel and efficient architectures that surpass human-designed counterparts. This manuscript aims to present a systematic review of the literature on this topic published between 2017 and 2023 to identify, analyze, and classify the different types of algorithms developed for NAS. The methodology follows the guidelines of Systematic Literature Review (SLR) methods. Consequently, this study identified 160 articles that provide a comprehensive overview of the field of NAS, encompassing discussion on current works, their purposes, conclusions, and predictions of the direction of this science branch in its main core pillars: Search Space (SSp), Search Strategy (SSt), and Validation Strategy (VSt). Subsequently, the key milestones and advancements that have shaped the field are highlighted. Moreover, we discuss the challenges and open issues that remain in the field. We envision that NAS will continue to play a pivotal role in the advancement of ML, enabling the development of more intelligent and efficient ML models for a wide range of applications.

Keywords Neural architecture search (NAS) · Search space (SSp) · Search strategy (SSt) · Validation strategy (VSt) · Systematic literature review (SLR)

✉ Sasan Salmani Pour Avval
S.Salmanipouravval@tudelft.nl

[1] Aerospace Structures and Materials, Faculty of Aerospace Engineering, Delft University of Technology, Delft, Netherlands

[2] Q-VAIbe group, Aerospace Structures and Materials, Faculty of Aerospace Engineering, Delft University of Technology, Delft, Netherlands

Published online: 06 January 2025

1 Introduction

Since the advent of Machine Learning (ML) and Artificial Intelligence (AI), several breakthroughs have been made in the software Targ et al. (2016); Zhu and Newsam (2017); Dillon et al. (2017) and the hardware Barnell et al. (2022); Ditty (2022); Cook (2012). Along with a wide range of application approaches in AI and optimization have been explored across various engineering fields, including bio-medical engineering Giveki and Karami (2020); Rastegar and Giveki (2023); Rastegar et al. (2024), mechanical engineering Hu et al. (2021); Yaghoubi and Kumru (2024); Yaghoubi et al. (2022), process engineering Salmanipour et al. (2023), material science Caglar et al. (2022), etc. All are pointed to improve the performance of models in the sense of accuracy, training time, energy consumption, etc., to allow the improvement of faster and cheaper AI models and hardware for different applications. However, since different datasets (e.g. 1-D vibration dataset Mey et al. (2020), 2-D MNIST LeCun et al. (2010), 3-D RGB images like CIFAR-10 Krizhevsky and Hinton (2009), or 4-D RGBD images Silberman and Fergus (2011)) have different dimensions and information types, neither method nor model can be used for all datasets; therefore, one should alter either the method or the (hyper)parameters to be able to train on a new dataset. Analyzing data, changing the architecture, and tuning the (hyper)parameters can make a huge difference in the performance of the AI models. Therefore, the model's performance highly depends on the experience of the developer in AI to design the architecture, and the background knowledge in the applied field to understand the data Yaghoubi et al. (2022). Furthermore, the developer needs to have a good knowledge of optimizing the model to generate the optimized model for a dataset. To solve this problem, Automatic Machine Learning (AutoML) was introduced as a new solution in ML to develop an optimized model automatically. For this purpose, the whole hyperparameters associated with AI models, e.g. learning, architecture, optimizer, operations in layers, data analysis, etc., during the learning process will be optimized Zhao et al. (2021a). For instance, Lange et al. Lange and Riedmiller (2010) used a reinforcement learning algorithm to change the numbers of units in a Multilayer Perceptron (MLP) model to increase the performance and decrease the validation error.

When convolutional neural networks became popular, researchers also tried to bring auto-tuning algorithms to these models. The convolutional neural fabric Saxena and Verbeek (2016) was one of the first attempts in this area in which a 3D trellis of layers, scales, and kernels was created as a search space for the algorithm to find the proper network. The algorithm searched for the best pipeline in this 3D Search Space that eventually defines the network. Fig. 1 demonstrates a simplified version of this method with a 2D search space of scales and layers. In this figure, models A and B have been created using different paths in the search space. The first layer of both models is appointed by *"Input"* during the optimization each of them followed different paths to the *"Output"* which is the output of the networks. The result of the procedure is two models shown in the right image.

As neural networks gained popularity for their ability to model complex patterns and make accurate predictions, the need for optimized architectures became apparent. This led to the development of Neural Architecture Search (NAS) Zoph and Le (2016), a specialized branch of AutoML, dedicated to automating the process of finding the most effective neural network architecture. It was initiated by the work of Zoph and Le Zoph and Le (2016) in which a reinforcement learning algorithm was developed to make an architecture with the

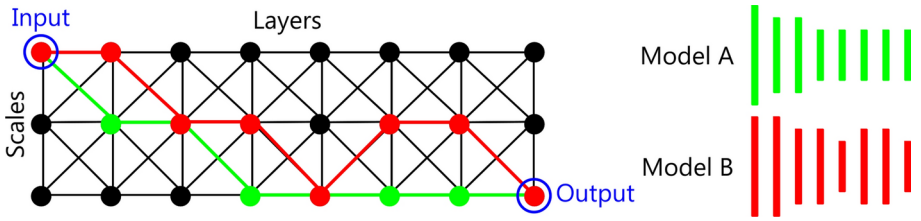

Fig. 1 Convolutional Neural Fabric Saxena and Verbeek (2016): on the left, there is the simplified search space and on the right, there are two different models with different colors that are generated by the paths which are illustrated in the search space

highest validation accuracy but with the expense of 800 Graphics Processing Unit (GPU)s for an image classification dataset. This means the models designed by the NAS algorithm can outperform models designed manually Zoph et al. (2018). In addition, NAS can be adjusted to generate the most accurate, lightest, and/or fastest models for image classification Real et al. (2019b), object detection Zoph et al. (2018), or semantic segmentation Liu et al. (2019). Furthermore, the NAS algorithms can optimize models for Internet of Things (IoT) devices as well as Microcontroller Unit (MCU)s Lin et al. (2020); Saha et al. (2022). However, NAS requires a computing power that is not accessible to all industries and people. Therefore, the main goal of NAS was pushed toward making it computationally cheaper in the sense of time and memory usage.

The rest of the paper is outlined as follows: In section 2, the materials and methods used in this study will be presented. In section 3, basic knowledge of the NAS algorithms will be introduced. In Sect. 4, the improvements achieved by different researchers will be elaborated on. In Sect. 5 a discussion and in Sect. 6 a conclusion is provided.

2 Materials and methods for systematic literature review

This paper aims to introduce a systematic review of NAS algorithms as well as a discussion on current works, their purposes, conclusions, and predictions of the direction of this science branch. In this study, we used the methodologies of Systematic Literature Review (SLR) Keele (2007); García-Holgado et al. (2020). A SLR is a comprehensive and unbiased review of the literature on a particular topic; it involves identifying, evaluating, and synthesizing all relevant research on the topic. Within this section, you can find sources, tools, review plans, research questions, and review processes that are used in this article.

2.1 Tools

For preparing a systematic literature review, we used Parsifal (https://parsif.al/) as a systematic review manager and Mendeley (https://www.mendeley.com/) as a reference manager. Also, we used Overleaf (https://overleaf.com/) as a text editor, Inkscape (https://inkscape.org/) for drawing figures, and RStudio (https://posit.co/products/open-source/rstudio/) for plotting.

2.2 Sources

The review process is illustrated in Fig. 2 using a PRISMA diagram Moher et al. (2009) to show the searching process clearly. Both journal and conference papers are considered due to the publishing culture in computer science. First of all, some digital libraries were selected due to their popularity on NAS. As shown in the PRISMA diagram in Fig. 2, between 2017 and mid-2023, 3508 papers were found in the following digital libraries, 620 published in Institute of Electrical and Electronics Engineers (IEEE), 551 published in Association for Computing Machinery (ACM), 1011 published in Web of Science (WOS), 669 published in ScienceDirect (SD), and 657 published in Scopus. The query string used for this search was "Neural Architecture Search (NAS)," "architecture searching algorithm," and "predict performance." After eliminating duplicated publications, pruning non-related papers, and adding our pre-studied papers to this collection, 160 papers were accumulated.

Figure 3, shows how fast NAS algorithms are getting popular among researchers, with more than 3000 papers between 2017 and mid-2023 in mentioned digital libraries. Furthermore, the geographical distribution of the NAS researchers worldwide, is shown in Fig. 4. It indicates that the US and China are leading the field by contributing to more than half of the relevant papers.

2.3 Research questions

The main research questions that we aimed to answer in this paper are (reasons that these questions are being elaborated exist in section 3):

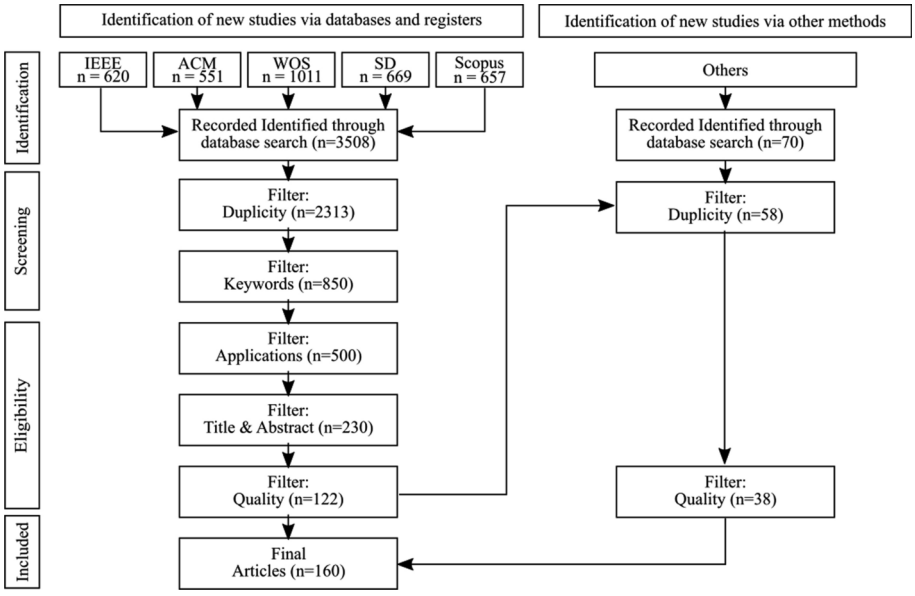

Fig. 2 PRISMA Diagram Moher et al. (2009) on NAS. The 'n' indicates the number of papers for each step. At the beginning of the process, various papers have been gathered from different digital libraries. then, the gathered papers were filtered using different properties of them that were defined in every box. In the end, 160 papers were selected and studied to satisfy the needs of this systematic review paper

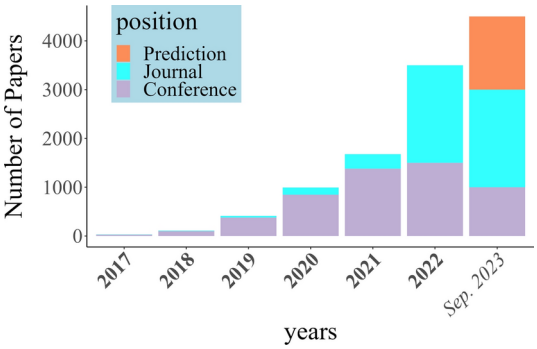

Fig. 3 Conference and journal papers on NAS till September of 2023

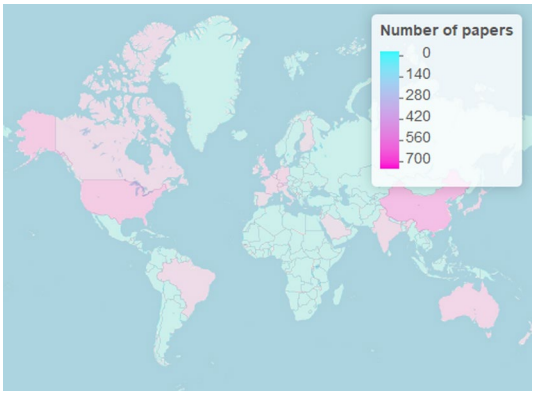

Fig. 4 Geographical distribution of NAS researchers around the world

1. What specific improvements did each paper make in the NAS field to achieve better performance?
2. How long does the developed NAS algorithm take to reach the optimized model?
3. How good is the NAS algorithm in finding the optimized architecture?

2.4 Criteria

After defining the research questions, we need to define the inclusion criteria as clearly as possible and exclude otherwise. They are listed as follows,

(i) The papers advance the main NAS pillars, i.e. Search Space, Search Strategy, and Validation Strategy (elaborated in section 3), AND
(ii) The papers are available to download or review in full under the license from Delft University of Technology (TU Delft) AND
(iii) The papers are written in English

3 Basic knowledge

Neural Architecture Search (NAS), as a subset of AutoML, is a framework for optimizing hyperparameters related to the architecture of Neural Network (NN) models Del Valle et al. (2023). It has three interconnected pillars as shown in Fig. 5:

Search Space (SSp): It is the space of all possible architectures, layers, hyperparameters, etc. for making a model.

Search Strategy (SSt): It is an optimization problem determining how to explore the SSp to make an architecture or model with the best performance. The choice of SSt can have a significant impact on the efficiency and effectiveness of the search process.

Validation Strategy (VSt): It determines how to assess the performance of each architecture or model during the SSt. For this purpose, the dataset is divided into training and validation sets to be used for training a model and evaluating its performance, respectively. The results of the VSt can be directly fed into SSp and SSt for their modifications Zhang et al. (2021a); Elsken et al. (2019).

To understand the NAS algorithm more in detail, the following subsections describe the three main pillars and the methods that have been developed in each one. Figure 6 gives an overview of these different methodologies.

3.1 Search space

Search Space (SSp) is a space containing all hyperparameters of a model, such as layer types, layer connections, activation functions, kernel sizes, kernel numbers, etc. Such an SSp ensures the presence of the optimized model, but can simply lead to an infinite space that is impossible to make and explore. In practice, some constraints should be imposed on the SSp to limit its size to reduce the searching time and push the SSt toward the near-optimal architecture. In this regard, one commonly used approach is that instead of having a single layer as a selection unit from SSp, a set of connected layers, called a cell, will be chosen. This means the SSp will affect the architecture of the near-optimal model as well as its performance. For instance, the type of numbers of operations, connections, and layers can dictate the latency and accuracy of the optimized model Mao et al. (2021).

Layer-based Search Space (*figure 7a*):

In the *layer-based* architectures each layer, as the selection unit, can only be connected to its immediate layers. In other words, the optimizer has to look for in-series architecture in the SSp, as shown in (figure 7a). For instance, in FBNet Wu et al. (2019) the depth of the model has been fixed to 22 layers and each layer type can be chosen from the nine specific

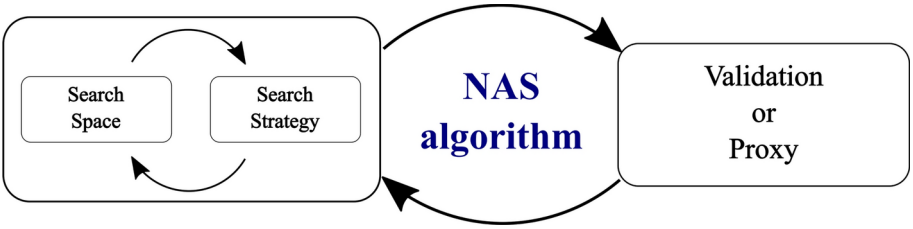

Fig. 5 Neural architecture search

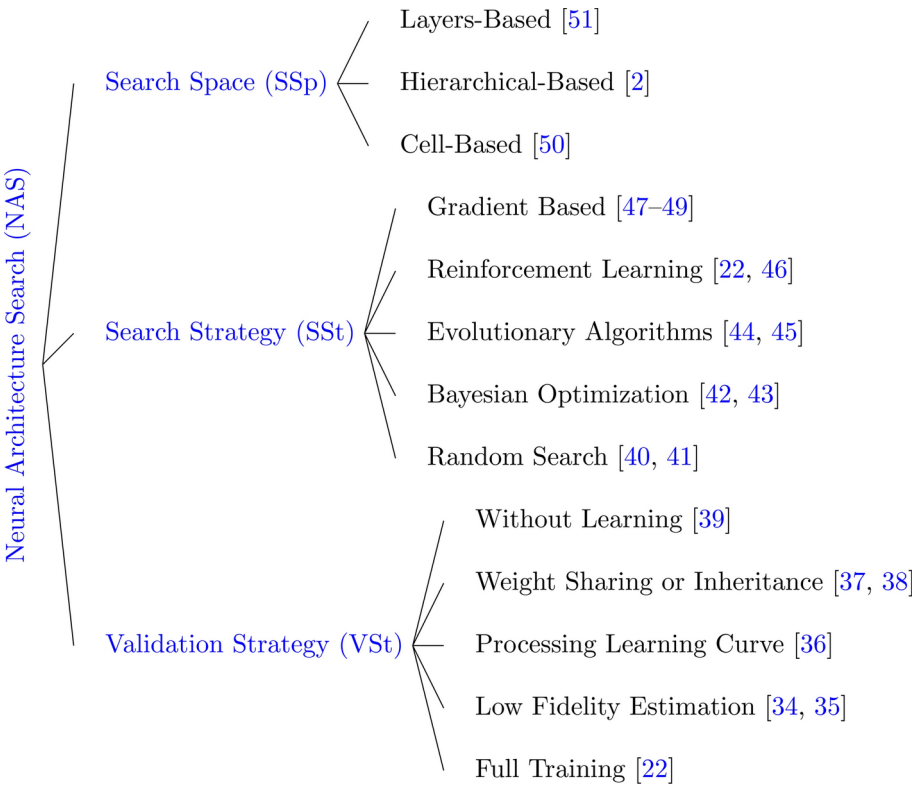

Fig. 6 Different methodologies adopted in each pillar of the NAS algorithms and each will be elaborated further

options, e.g. skip connections, convolutional layers with different kernel sizes, etc. Giving limited options with fixed layer types reduced the searching time by reducing the SSp.

Hierarchical-based Search Space (*figure 7b*):

In the *hierarchical-based* architecture besides in-series connections, a model can have parallel branches and skip connections, as shown in Fig. 7b. Therefore, this leads to a larger SSp and thus, some techniques need to be employed to reduce it and in return, to speed up the procedure Tan et al. (2019); Li et al. (202c0).

Cell-based Search Space (*figure 7c*):

In the *cell-based architecture* the selection unit is a cell that will be connected in series to create models with different sizes. However, the layers in each cell are connected in a hierarchical fashion (Fig. 7c). This means, in this approach, we have the benefits of hierarchical connections but with smaller SSp because of the cell-based selection Liu et al. (2018b).

Even though limiting the architecture's layers and connections can decrease the search time, some algorithms try to modify the SSp according to some restrictions (e.g. memory, processing, latency, accuracy limitations) while optimizing the model Lin et al. (2021, 2020); Liu et al. (2018b); Huang et al. (2021). Lin et al. Lin et al. (2021) showed that layers' size as well as their place or order in the network can change memory usage intensively. For instance, Fast Probabilistic NAS or FP-NAS Yan et al. (2021) was proposed to explore the

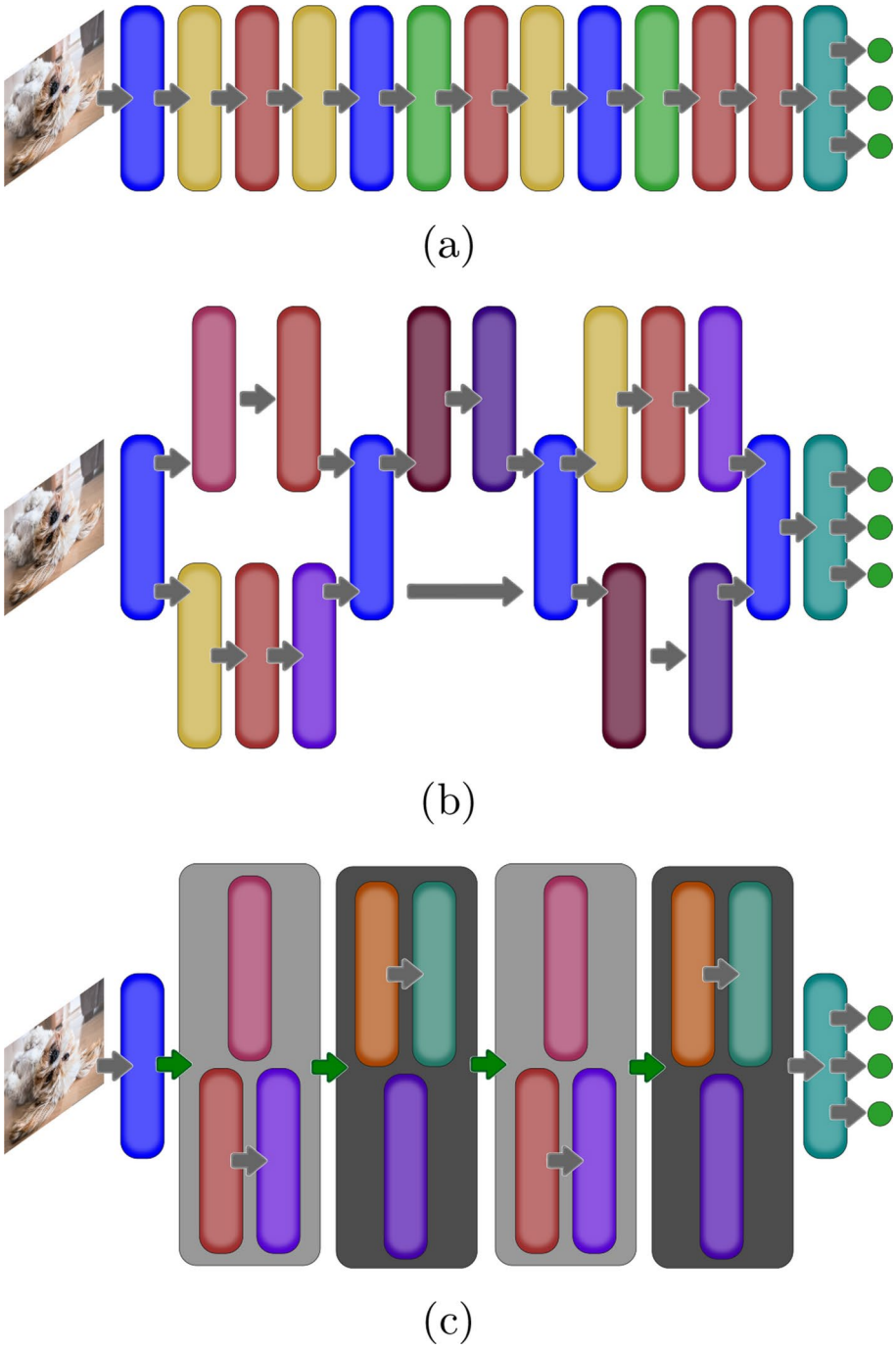

Fig. 7 Different Search Space (SSp) connection possibilities: **a** Layers-based Wu et al. (2019): layers in series (no parallel or skip), **b** Hierarchical-based Zhu and Newsam (2017): layers in series, parallel, or skip, **c** Cell-based Liu et al. (2018b): connections within one cell are Hierarchical-based but between the cells are in series (cells with same colors have the same structure – means that this architecture has two types of cells)

SSp adaptively using probabilistic learning together with an adaptive sampling algorithm that could shrink down the SSp and thus speed up the optimization. As a result, the optimized model needed less memory and fewer numbers of Floating Point Operations (FLOPs). CP-NAS (Child-Parent NAS) Zhang et al. (2021a) and DARTS (differentiable architecture search) Liu et al. (2018a) developed continuous variables for each connection between layers. The optimizer continuously changes the variables using gradient-based methods to find the best suitable connections by emphasizing the important connections with higher values. This method worked faster to find optimized connections in the dictated SSp. The problem is that restricting options like the number of layers or the types of layers can be problematic because it forces the algorithm to search for models with a specific level of complexity and depth. This can make the optimization process more challenging, as can be seen in DARTS Liu et al. (2018a). But some researchers believe that having complex cells and then pruning the SSp can speed up the process of finding the best architecture by using less computing power Liu et al. (2018b); Loni et al. (2020); Hong et al. (2021) instead of limiting the SSp. On the other side of the coin, using the limited SSp for the NAS algorithm has some benefits like helping the optimizer to find models for specific applications (like light models for different hardware) by defining some special operations like layers and connections Mao et al. (2021) to reach models with specific properties such as memory usage.

3.2 Search strategy

Search Strategy (SSt) defines how to explore the SSp. Its main goal is to construct the best model for a given dataset from a vast pool of models available in the SSp. To achieve this goal, a well-designed SSp in conjunction with a proper optimizer as SSt should be employed in NAS algorithms. Several popular SSts are available in the literature such as Random Search, Bayesian Optimizer, Evolutionary Algorithms, Reinforcement learning, Gradient-based optimizer, etc. that will be discussed in the following. As can be seen in figure 8, the popularity of different optimizers is changing over time. Bayesian Optimizer got popular because of their convergence speed but they need to reformulate the Search Space which makes the problem more complex. Evolutionary Algorithms is one of the most popular methods which uses a population-based optimization method and can ensure high effectiveness. Gradient-based optimizers are getting more popular because of their Graphics Processing Unit (GPU) friendly methodology. This method can be run on the GPU which is faster than Central Processing Unit (CPU) when using large datasets and matrix calculations. Random Search is easy to use but not efficient. Reinforcement learning based methods lost popularity because they need a lot of effort to find the best model for a dataset which needs a lot of time for validation but, by introducing cheap and fast validation techniques, it regained its position. Table 1 gives a high-level comparison between these techniques.

3.2.1 Random search

Random Search (RS) is a numerical method that is normally used on discrete problems Li and Talwalkar (2020); Wen et al. (2020). This optimizer is easy to implement but is the slowest one for the NAS algorithms, therefore, it found very limited applications in this field. This method is mainly used when the goal is to develop a Validation Strategy and its performance should be independent of SSt Abdelfattah et al. (2021).

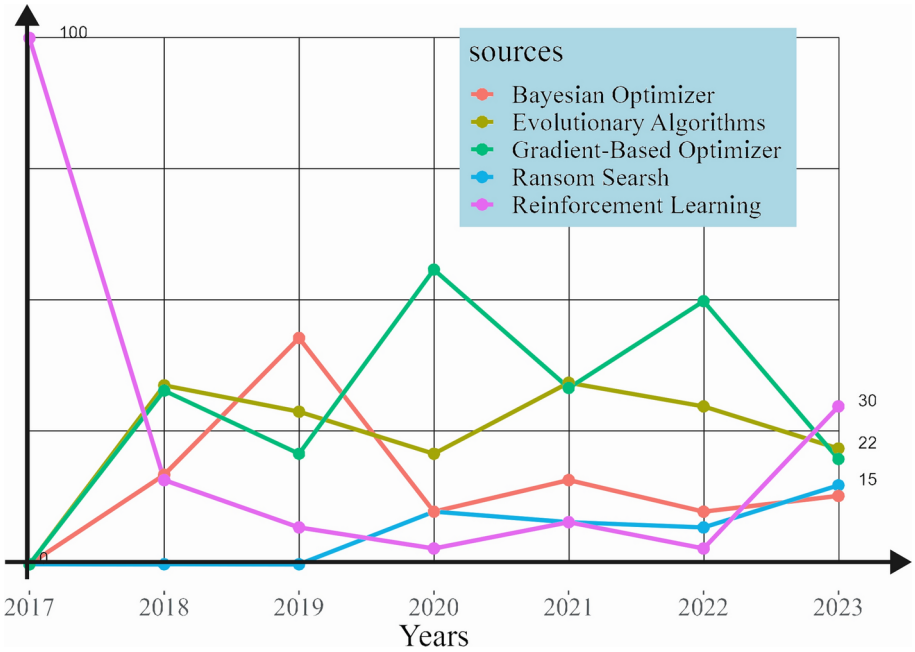

Fig. 8 Percentage of different optimizers used in SSt of NAS algorithms

Table 1 High-level comparison of different Search Strategies (D and C stand for Discrete and Continuous, respectively)

Optimizer	Simple	Fast	Accurate	SSp domain
Random Search [40, 41]	✓	—	—	D
Gradient Based [47–49]	—	✓	✓	C
Bayesian Optimization [42, 60]	—	✓	✓	D/C
Evolutionary Algorithms [44, 45]	✓	—	—	D/C
Reinforcement Learning [22, 46]	—	—	✓	D/C

3.2.2 Gradient-based optimizer

Gradient-based optimizer (GO) algorithms rely on the derivate or gradient of the Objective Function (OF) to update the parameters/hyperparameters iteratively in the direction of its maximum or minimum Pham et al. (2018). This will contribute to a faster exploration of the SSp for finding the optimal architecture by NAS. These methods can work well for differentiable objective functions where gradients can be readily computed. Therefore, this type of approach, such as DARTS Liu et al. (2018a), needs to compensate for the discrete nature of the SSp in searching for the optimized model.

Due to the iterative nature of the GO methods, they are considered computationally expensive algorithms. However, GPUs can be leveraged for solving this problem Dong and Yang (2019). As a result, GO algorithms are becoming popular because they are faster on a GPU than CPU.

3.2.3 Bayesian optimizer

The Bayesian Optimizer (BO) is a widely used optimization technique that utilizes probabilistic models to approximate the OF and select the next set of hyperparameters. It is particularly effective when the OF is not necessarily continuous and/or its evaluation is computationally expensive as it is in NAS; since it minimizes the number of evaluations needed to optimize the hyperparameters Jin et al. (2019).

3.2.4 Evolutionary algorithms

Evolutionary Algorithms (EA) are a class of optimizers inspired by principles from biological evolution and genetics. It was first used to obtain a proper architecture for neural networks in Elsken et al. (2018). To adapt EA to the NAS algorithms to find the best architecture the following steps should be taken:

- Population generation: some architectures should be chosen as an initial population;
- Performance evaluation: the algorithm will evaluate the population's performance based on some metrics such as accuracy, complexity, and resource requirement;
- Population evolution: some models will be chosen based on selection techniques, such as tournament selection Real et al. (2019b) or fitness proportionate selection Lopes and Alexandre (2022), to evolve and make the next generations;
- Termination criteria: this evolutionary procedure continues until a termination criterion, such as a predefined error or a maximum number of generations, is met.

3.2.5 Reinforcement learning

Reinforcement learning (RL) Kaelbling et al. (1996) algorithms work through the reward and penalties from the feedback which in NAS is provided by the Validation Strategy. The agent who is looking for the answer to the optimization problem tries to find the goal by gaining the most possible rewards in an interaction with the environment. In the scope of NAS, an RL agent interacts with the neural network environment in the SSp of the NAS algorithm Real et al. (2019b); Zoph and Le (2016); Cassimon et al. (2020) and learns to find the optimized architecture based on feedback that can be one of several or more evaluation metrics, such as accuracy, complexity, latency, etc. For instance, in Zoph and Le (2016), the problem was defined as an RL agent that tries to optimize a specific objective like accuracy by selecting architectural components. The agent modifies its policy based on how created architectures perform on VSt pillar. Due to the necessity of evaluating each generated architecture, RL-based NAS approaches demand a large amount of computational power. This approach is enabled to progressively refine its decision-making abilities and uncover novel architectural configurations that outperform existing models.

3.3 Validation strategy (or proxies)

The most time-consuming step in NAS algorithms is the Validation Strategy (VSt) pillar, which assesses the performance of a model through the Objective Function (OF) and gives it as feedback to SSt or SSp. This value indicates how good the architecture would perform

on the data at hand from the perspective of the OF. Several methods have been proposed in the literature as VSt. The most straightforward approach is to train each model from scratch on the training dataset for a pre-determined iteration (n) and then assess the model's performance on the validation dataset. This approach which will be referred to as training-based Zoph and Le (2016), is the most accurate one but computationally very expensive. Therefore, the research trend here is to reduce the computational cost of this stage by approximating the performance of a model on the validation dataset. This is why it is sometimes called proxies in the literature Mellor et al. (2021). The proposed methods can be categorized as full training (training-based) Zoph and Le (2016), partial training Li et al. (2017); Zoph et al. (2018); Zela et al. (2018); Falkner et al. (2018); Real et al. (2019a); Runge et al. (2018); Trofimov et al. (2020); Elsken et al. (2019); Pham et al. (2018); Liu et al. (2018a); Xie et al. (2018), adaptive training Swersky et al. (2014); Domhan et al. (2015); Klein et al. (2016); Baker et al. (2017); Jeong et al. (2022) and no training (zero-shot) Gracheva (2021); Mellor et al. (2021). Table 2 compares the required training iteration for these different methods schematically. You can find a comparison of some proxies in White et al. (2021b).

In the following section, we will elaborate on different VSt techniques.

3.3.1 Full training

In this approach, every single architecture proposed by the SSt will be fully trained (for n epochs as you can find in Table 2) before assessing its performance on the validation dataset. Therefore, it compares the exact performance of the models but with the cost of an exhaustively long training time Zoph and Le (2016).

3.3.2 Partial training

In this approach, the models will be trained for $r < n$ iterations (Table 2). The r iterations can be done for each model from scratch (type I) or the model could inherit its weights from another model and then be trained for extra r iterations (type II). The former is also called low fidelity and was introduced to reduce the computational burden and thus, speed up the searching procedure. For instance, the performance of each model on the validation dataset was assessed after its training for a few epochs in Zimmer et al. (2021); Trofimov et al. (2020). The second type, Weight Sharing or Inheritance, aims to reduce the training time by sharing the weights from one model to the next one, thus new models never need training from scratch to be validated for the NAS algorithm Liu et al. (2018a). This category can be implemented in four ways:

- One-time training of a huge architecture (SuperNet) and then using its different parts as

Table 2 Comparison of different Validation Strategy (VSt) methods from the training time perspective

Method	Training Epochs
Fully training (training-based) [22]	0 — n
Partial training - type I (low fidelity) [34, 35]	0 — n
Partial training - type II (weight sharing techniques) [37, 38, 49, 80]	0 — n
Adaptive training (depending on the learning curve) [36]	0 — n
No training (zero-shot) [39, 81]	0 — n

smaller models (SubNets) Liu et al. (2018a),
- Training small architectures (SubNets) and then sharing their weights with a bigger architecture that contains all the small models (SuperNet) Chen et al. (2021a),
- Sharing weights just in the generation moment of a model by the saved models Wan et al. (2020), and
- Transferring weights from one model to other ones which are more suitable with algorithms like EA optimizers in which the optimizer uses some previous models to generate new ones Wan et al. (2020).One of the most famous pieces of literature is ENAS Pham et al. (2018) which introduced a novel parameter-sharing technique. The proposed search strategy seeks to identify an optimized SubNet within the SuperNet architecture. By employing this method, connections can be shared among distinct SubNets, utilizing a single directed acyclic graph (DAG) Li et al. (2022).

3.3.3 Adaptive training (processing learning curve)

In this approach, the model trains from scratch but the number of iterations depends on the behavior of the learning curve. A learning curve is a curve that shows how the accuracy or loss of the model changes over training iterations. Analyzing this curve can help us predict whether the model is converging to a reasonable performance during the early stages of the training step. For instance, Jeong et al. Jeong et al. (2022) introduced performance metrics based on the depth and flatness of the loss value during the training step.

3.3.4 Zero-Shot (zero-cost)

All the VSt methods mentioned above require some level of training for a model. Since NAS algorithms normally handle hundreds or even thousands of models, even some training iterations for each model could lead to an untractable search time. Therefore, researchers are looking for methods that can provide a score for each model as an indicator of the model's performance. These methods which are mostly based on mathematical algorithms related to data and/or model architecture Mellor et al. (2021), can reduce the validation time of the NAS algorithm to close to zero seconds Lin et al. (2021); Fan et al. (2023).

4 Literature

For shortlisting the papers for this section we used Systematic Literature Review (SLR) because this method starts with wide open keywords in the field and filters papers step by step by logical conditions and can eliminate the human preferences in the paper collecting.

Using NAS for finding an optimized model normally needs lots of time, memory, and energy. Therefore, the recent trend is to minimize its cost by modifying at least one of its three pillars (search space, search strategy, and validation strategy as shown in Fig. 5) to increase its efficiency and effectiveness and decrease its computational demand. Also, these improvements are heading to the point that hardware specifications are taken into account such that one can optimize AI models by NAS algorithm for devices with limited computational power and memory size e.g. edge devices and microcontrollers.

4.1 Comparing NAS algorithms

Over the years, several different algorithms have been developed in the field of NAS that will be compared and discussed in this section. To compare their performance, some metrics and benchmarks need to be first defined. Performance metrics are essential tools for evaluating the effectiveness of models. They provide quantitative measures of a model's ability to achieve its intended purpose, such as classification accuracy, prediction error, or energy efficiency. The choice of performance metrics depends on the specific application and the desired outcomes. Benchmarks are evaluation frameworks that provide a structured and comparable basis for assessing NAS algorithm performance. They typically contain a defined SSp, VSt information, and performance metrics; allowing researchers to compare the performance of a model with any other model, systematically and objectively.

4.1.1 Performance metrics and menchmarks

Each benchmark provides several parameters that can be used as performance metrics. The following is the definition of these metrics.

- Accuracy: The proportion of correct predictions made by a model.
- Loss: The difference between the predicted values and the actual values.
- Latency: the amount of time that takes for a model to process and react to an input; lower latency indicates faster processing (it relates to real-time performance and responsiveness)
- FLOPs: number of Floating Point Operations that a model needs to process an input
- Energy: the required amount of energy needed to run a model on a specific hardware
- Training Time: the required time for training a model for a specific number of iterations
- Trained Parameters: the values of trained weights.These metrics can be considered as some specifications of benchmarks in the NAS algorithms. Table 3 demonstrates some NAS benchmarks available for image classification tasks. It also shows the metrics and features included in each benchmark.

For instance, in table 3, we can see that Nas-Bench-101 Ying et al. (2019) has a SSp consisting of 15×10^3 models with Cell-based architectures. These models are trained on CIFAR-10 and CIFAR-100 datasets but not on ImageNet and the following metrics are reported: 1. accuracy on Validation data (Val. Acc.) 2. accuracy on test data (Test Acc.) 3. latency of models 4. number of FLOPs 5. training time 6. number of training parameters 7. but the energy consumption is not available. Also, in the hardware part, you can see that this benchmark is evaluated on just one single hardware (GPU), which means there are no measured metrics of models on different hardware.

4.1.2 Methods

Historically, Zoph and Le Zoph and Le (2016) started Neural Architecture Search (NAS) algorithms by implementing Reinforcement learning as their Search Strategy to find the most accurate model for a specific dataset. Because their VSt was based on full training, it needed a huge amount of GPU hours to find the optimized model. Since then, researchers

Table 3 NAS Benchmarks for Image Classification (Cell-based (CB)/Hierarchical-based (HB))

Features		Benchmarks					
		NAS-bench-101 Ying et al. (2019)	NAS-bench-201 Dong and Yang (2020)	NATS-Bench Dong et al. (2021b)	NAS-bench-1shot1 Zela et al. (2020)	NAS-bench-301 Siems et al. (2020)	HW-NAS-BENCH Li et al. (2021b)
Search Space Size ($\times 10^3$)		423	15	15	363	10^{15}	10^{18}
Search Space Type		CB	CB	CB	CB	HB	CB
Dataset	CIFAR-10	✓	✓	✓	✓	✓	✓
	CIFAR-100	✓	✓	✓	✓	✗	✓
	ImageNet	✗	✗	✓	✓	✗	✓
Metrics	Val. Acc	✓	✓	✓	✓	✓	✓
	Test Acc	✓	✓	✓	✓	✓	✓
	Latency	✗	✓	✓	✓	✓	✓
	FLOPs	✗	✓	✓	✗	✗	✓
	Energy	✗	✗	✗	✗	✗	✓
	Training Time	✓	✓	✓	✓	✓	✓
	Train Parameters	✓	✓	✓	✗	✗	✓
Different hardware		✗	✗	✗	✗	✗	✓

have been trying to improve the performance of the NAS algorithm from different perspectives e.g. running time, energy efficiency, optimized model performance, etc. The rest of this section introduces and compares the available literature based on their SSt. Table 4 summarizes all the different methods to compare next to each other.

Reinforcement learning: As mentioned, the first NAS algorithm is introduced using Reinforcement learning (RL) Zoph and Le (2016). Within this SSt, the RL algorithm searches for the best architecture based on feedback such as accuracy. In general, RL enabled the algorithm to find architecture that is optimized in the sense of accuracy and latency, etc., and performed better than the human-designed models.

To dive into more detail, Zoph and Le Zoph and Le (2016) implemented a recurrent neural network (RNN) to find the best architecture. The RNN was trained with an RL algorithm to maximize the performance of the generated architecture. This algorithm was iterative, and in each iteration, the generated architecture was trained from scratch and then evaluated on the validation dataset. The obtained validation accuracy was fed to the controller, as the reward, to generate a new architecture for the next step. Although the process was very costly, about 800 GPUs of hours, it could outperform human-designed models.

In Mills et al. (2021), the algorithm was improved to be faster and more accurate by leveraging the concept of the SuperNet and SubNet. A SuperNet is a large, flexible network that contains a large number of small models, known as SubNets. In this way, the NAS algorithm can find the best model by evaluating the performance of different SubNets within the SuperNet.

Since NAS algorithms are normally computationally demanding, the main focus of the researchers shifted toward reducing the computational cost. In this regard, Zoph et al. Zoph et al. (2018) tackled this issue by limiting the Search Space for the algorithm and introducing the *Cell-based* architecture. This leads to reducing the training cost from 800 to 500 GPUs. In this way, instead of searching for the entire network, the algorithm needs to search for the layers and connections inside a cell. Two types of cells were created in this method: Normal cells and reduction cells. Normal cells extract the features holding the input size but the reduction cells reduce the output size while processing it. Later, they created a model by connecting these cells multiple times. Using this method, they were able to find the best cell with a smaller dataset, then transfer the cell to a larger and more extensive network, and train it on a large dataset. As was proven in Wen et al. (2021), this approach was extremely energy and time-efficient.

Ding et al. Ding et al. (2021) increased the speed of the NAS algorithm by implementing RL optimizers and broader architecture with flexible hyperparameters; as a result, they introduced BNAS to find a model that has less size and higher accuracy.

One of the known issues of BNAS Ding et al. (2021) was an unfair learning SubNet meaning that during the training step of the SuperNet, some SubNets tend to be trained better than others causing some problems at the end of the optimization process. The optimizer tended to find models that were well-trained instead of choosing models that were more suitable for the data. This issue was addressed in Ding et al. (2022) and led to BNAS-v2. To reach the aim, they adjusted the learning rate based on the gradient to prevent aggressive changes in architecture selection. To decrease the computation and memory consumption, they implemented a method to select just the active part of each layer during the process. Also, BNAS Chen et al. (2020) suffers from unstable training and over-fitting, therefore, a normalization method for the active SubNets in the SuperNet was applied.

Table 4 NAS Methods, C10:CIFAR-10, C100:CIFAR-100, ImgN: Imagenet, A.: Accuracy, #p: number of parameters, RT: GPU hours running time

Paper	Year	SSp	SSt	VSt	A. c10	A. c100	A. ImgN	#p c10	#p c100	#p ImgN	Run Time	Latency	GPU
Zoph and Le (2016)	2016	LB	RL	TB	96.35	_	_	37.4M	_	_	22,400	_	K40
Zoph et al. (2018)	2018	CB	RL	TB	97.6	_	82.7	27.6M	_	88.9M	2000	_	P100
Ding et al. (2021)	2021	CB	RL	one-shot TB	97.12	_	25.7	4.8M	_	8.5M	4.56	_	GTX 1080Ti
Liu et al. (2018a)	2018	CB	GO+ RL	TB	97.24	_	73.3	3.3M	_	4.7M	96	_	GTX 1080Ti
Wang et al. (2022)	2022	CB	GO+ RL	TB	97.64	_	75.8	3.35M	_	5.1M	12	_	Tesla-V100
Pham et al. (2018)	2018	CB + LB	GO	TB	97.42	_	_	13.1M	_	_	11	_	GTX 1080 Ti
Wu et al. (2019)	2019	LB	GO	TB	_	_	74.9	_	_	5.5M	216	28.1ms	_
Wan et al. (2020)	2020	LB	GO	W-Sh	_	_	77.2	_	_	325 M	600	_	Tesla V100
Howard et al. (2019)	2019	LB	RL	TB	_	_	75.2	_	_	219 M	_	44ms	_
Dong and Yang (2019)	2019	CB	GO	TB	97.18	81.87	72.5	2.5M	2.5M	44 M	4	_	Tesla V100
Chen et al. (2020)	2020	CB	RL	TB	96.53	_	71.3	4.6M	_	6.2M	2.5	_	GTX TITAN
Ding et al. (2022)	2022	CB	GO+ RL	TB	97.23	_	73.5	2.5M	_	4.1M	2	_	GTX 1080Ti
Yu et al. (2022)	2022	CB	GO+ cyclic	TB	97.52	85.92	76.3	3.9M	4 M	22.5M	7	41.2ms	Tesla V100
Song (2021)	2021	CB	GO+ clustering	TB	97.5	85.9	76.2	3.3M	3.9M	22.4M	4.8	_	Tesla P100
Zheng et al. (2021)	2021	NB201	GO	_	98.14	86.24	_	3.5M	3.5M	_	14.4	_	4? Tesla V100
Cai and Luo (2021)	2021	_	BO+ MOO	TB	95.4	_	_	6.99	_	_	48	_	GTX 2080TI
White et al. (2021)	2021	CB (DARTS)	BO	TB	97.36	_	_	_	_	_	283	_	Tesla V100
Saha et al. (2022)	2022	LB	BO	TB	_	_	_	_	_	_	_	_	_
Elsken et al. (2018)	2018	CB + LB	EA	Proxy	97.42	_	_	13.1M	_	_	1920	_	_
Real et al. (2019)	2019	CB	EA	TB	96.66	_	83.9	3.2 M	_	469 M	_	_	450 K40
Lopes et al. (2022)	2022	NB101/NB201	EA	Proxy	93.99	72.36	46.04	_	_	_	7.5	_	1080Ti
Louati et al. (2022)	2022	Man Made	EA	Proxy	98.24	88.83	92.83	1.8M	1.8M	2.1M	744	_	2080Ti
Chen et al. (2021)	2021	LB	EA+ SSp shrinking	TB	80	_	77.9	_	_	_	288	_	Tesla V100
Shang et al. (2022)	2022	LB	EA	TB	95.8	80.75	75.5	_	_	_	96	_	RTX2080Ti
Luo et al. (2018)	2018	CB	RS	TB + W-Sh	96.47	85.25	_	2.5M	128 M	_	7.2	_	V100
Chen et al. (2021)	2021	CB	EA	Proxy	94.19	82.01	_	_	_	_	48	_	2080

Table 4 (continued)

Paper	Year	SSp	SSt	VSt	A. c10	A. c100	A. ImgN	#p c10	#p c100	#p ImgN	Run Time	Latency	GPU
Jeong et al. (2022)	2022	DARTS/NB201	GO	learning curve	76.06	76.05	_	5.3M	5.2M	_	_	_	V100
Mellor et al. (2021)	2021	NB101/NB201	RS	Zero	93.1	69.1	45.08	_	_	_	0.06	_	GTX 1080 Ti
Wu et al. (2021)	2021	NB201	EA	Zero	93.7	71.57	45.18	_	_	_	0.03	_	RTX 3060
Chen et al. (2021)	2021	NB202	GO	Zero	93.9	71.24	42.38	3.8M	_	5.4M	0.5	_	GTX 1080Ti
Mokhtari et al. (2022)	2022	NB203	GO	zero	93.58	70.27	44.53	_	_	_	_	_	a RTX A3000
Luo et al. (2018)	2018	CB	GO	Proxy	96.47	85.25	_	2.5M	128 M	_	7.2	_	V100
Franken et al. (2022)	2022	NB201	GO	Proxy	_	_	_	_	_	_	0	_	NONE
Qian et al. (2022)	2022	NB201	tree	proxy	94.37	73.09	46.33	_	_	_	3.6	_	GTX2080Ti
Zhang et al. (2021)	2021	NB201	GO	proxy	93.45	70.71	43.7	5.5M	_	_	_	_	_
Zhang et al. (2022)	2022	NB202	GO+ BO	proxy	94.33	72.95	46.54	_	_	_	14.4	_	_
Zhang et al. (2021)	2023	NB201	RS	proxy	94.03	72.53	46.18	3.2M	_	_	1.2	_	GTX2080Ti

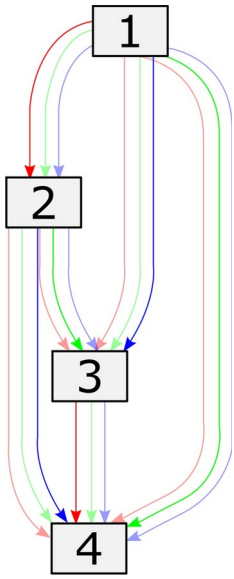

Fig. 9 A simplified SSp of differentiable architecture search (DARTS) Xue et al. (2022): non-faded connections (or layers) are selected for the architecture, blocks are different data stages, and different colors show different types of layers like CNN, skip, etc

In response to the increasing demand for more efficient NAS algorithms, Chen et al. developed an efficient NAS algorithm called Binarized Neural Architecture Search Chen et al. (2020). In this approach, the SSt explore only a portion of SSp that was randomly sampled. Similar to Zoph et al. (2018), they produced two cells (normal and reduction cells) to build up the main model. In this literature, along with channel sampling to reduce the number of parameters that need to be searched over, another method has been introduced called operation space reduction, which limits the search space in the sense of operations. To solve the problem of not converging to a well-optimized model, they implemented a performance-based SSt to ensure a high-performing architecture. Later, Zhang et al. Zhang et al. (2021a) improved the binarized algorithm to achieve a faster NAS algorithm by sharing trained weights from parents to child (generated) models during the optimization.

Gradient-based optimizer: Knowing that continuous optimizers, like Gradient-based optimizer (GO), are faster than discrete ones on GPUs, bringing them to a NAS algorithm as a SSt can speed it up and make it more applicable to the AI world. However, the main challenge would then be how to deal with the discrete nature of the Search Space.

This issue in NAS was first addressed by Differentiable Architecture Search (DARTS) Liu et al. (2018a) in which the search for the optimal architecture was performed on GPU. In this approach, a cell-based SSp was used that consisted of several types of layers such as convolution, skip, or pooling that were connected to each other. The SSt was a reinforcement agent that implemented the GO method to optimize the architecture of a model. In order to have continuous SSp and enable GO, all connections in architecture are assigned a variable. Then, the SSt optimizes the performance of the architecture by changing those variables that define the state of the connection (appearance in the network or not) at the end of the optimization Xue et al. (2022); Wan et al. (2022). As was shown in Fig. 9, only the connections with the highest values were retained, and the others were removed.

DARTS Liu et al. (2018a) could be unstable, and thus the search procedure could lead to sub-optimal solutions. Regularized Differentiable Architecture Search (RDARTS) Wang et

al. (2022b) overcomes this problem by utilizing regularization to encourage the RL agent to choose architectures that are capable of being more generalized and less overfitted. ENAS (Efficient NAS Pham et al. (2018)) introduced a method that lets the computer find the best architecture only using a single GPU in less than a day. In this method, sharing parameters played an important role in minimizing the optimization time. Further, they implemented Long short-term memory (LSTM) to find the hyperparameters of the model. In this method, the whole search space is defined as SuperNet and Subnets. The controller (which is the SSt in this approach) selects a suitable SubNet through the search space. As a result, they found out that using SubNet during training enhances the performance of the optimized model at the end of the process by preventing it from being trapped in local minima.

Dong and Yang addressed the issue of converting the discrete SSp to a continuous by introducing Gradient-based search using Differentiable Architecture Sampler or GDAS-Dong and Yang (2019). In this approach, they represented the SSp as a Directed Acyclic Graph (DAG) in which, each sub-graph can be sampled to be an architecture for the neural network. To limit the number of sub-graphs and speed up the search process, a differentiable sampler was developed. In this way, they were able to search for the optimized model efficiently benefiting the SSt presented in GDAS which was faster when using DAG SSp. GDAS guided the search toward architectures with lower validation loss.

Addressing the problem that NAS algorithms are computationally inefficient, CDARTS, which stands for a Cyclic Differentiable Architecture Search Yu et al. (2022), developed a two-step optimization to solve the problem. In every iteration of the algorithm, both the architecture and weights of the model are being trained: first, the weights of the model are trained; and then, the hyperparameters are evaluated to optimize the architecture. Until the evaluation method does not change the architecture, the iterations go on. CDARTS achieved the optimized architecture in less than 24 GPU hours for different datasets with a competitive accuracy.

Wu et al. developed a differentiable NAS technique in which besides optimizing the accuracy of the model, they considered hardware constraints to find an optimized network architecture for a mobile device. It is called FBnet: Facebook-Berkeley-Nets Wu et al. (2019). In this approach, layer-wise SSp with a predefined number of layers was implemented, i.e. 22 layers. The functions of each layer can be selected among 9 operations. This technique was applied to find a proper model on the ImageNet dataset. They managed to achieve 74.9% top-1 accuracy with 28.1ms latency. Also, the fastest architecture (for mobile devices) has 73% top-1 accuracy with 19.8ms latency on Samsung Galaxy S8.

In 2020, they introduced FBnetV2 Wan et al. (2020) which can perform NAS in a more efficient way on a larger SSp than FBnet Wu et al. (2019). In this method, to search for the most accurate architecture, first, a SuperNet was made and trained for one time. Then, the best SubNet is chosen as the optimized architecture. There were some issues like incompatible dimensions of weights, and increased memory consumption while sharing weights between different models. To solve them, they used a new zero padding method to make the kernels shareable between different models even with different sizes.

Some other approaches for NAS benefited GO optimizers and other improvements. For instance, Song et al. Song (2021) implemented a GO algorithm together with a clustering method to minimize the Search Space and speed up the search time; In Zheng et al. (2021); Wan et al. (2022), authors solved the challenge of achieving more accurate models by implementing a probabilistic GO method; Wu et al.Wu et al. (2021a) solve the limitation

of DARTS SSp by adding a Gradient-based searching algorithm and a Gradient-based pruning technique to decrease the running time as well as extending the SSp; Wei et al.Wei et al. (2022a) implemented multi-objective GO to optimize hardware-friendly models according to different metrics.

Bayesian optimizer: Evaluating models with different architecture in the NAS algorithm increases the search time. Bayesian Optimizer (BO) can find the optimized architecture by evaluating fewer models than the other SSts. BOs achieve this by building a probabilistic model of the OF, which allows them to estimate the performance of an architecture without actually evaluating it. They then use this probabilistic model to select the next architecture, focusing on the most promising regions of the SSp. This iterative process allows BOs to converge to the optimal architecture with fewer evaluations, significantly reducing the computational cost of NAS.

In essence, BOs work by modeling the relationship between the hyperparameters of an architecture and its performance. BO starts with an initial model, which could be a simple or a complex architecture. As they evaluate more architectures, BO updates its model to capture the most accurate and reliable relationship between the hyperparameters and the architectures' performance. This allows BOs to make more accurate predictions about the performance of the new one, guiding the SSt towards the optimal architecture. It should be bear in mind that, the complexity of BOs brings up three more complicated challenges in solving the NAS problem:

1. How to speed up the performance – Zhou et al. Zhou et al. (2019) solved this problem using DARTS SSp and Laplace Approximation,
2. How to quantify the posterior and likelihood to optimize the parameters or hyperparameters – this problem is solved by Shaw et al. Shaw et al. (2019) by introducing Bayesian neural networks (BNNs) and utilizing stochastic gradient descent to the probability distribution of neural network weights, and
3. How to explore the SSp to find the optimized architecture – Li et al. Li et al. (2020b) solve this problem by enabling direct performance prediction and exploring according to the estimated metrics.Kandasamy et al. Kandasamy et al. (2018) introduced BO in the NAS field, called NASBOT. This approach performed competitively with the other optimizers such as RS and EA. It was shown that BO makes more informed decisions about which architectures to evaluate and BO is less likely to get stuck in local optima. Further, They showed that BO are better at handling noisy objective functions.

BANANAS White et al. (2021a), Bayesian Optimization with Neural Architectures for Neural Architecture Search, introduced a specific encoding-decoding method between the SSps of the NAS and the BO SSt. Further, they implemented a NN based proxy to accelerate the VSt by skipping the time-consuming part of the validation step (training every architecture). As a result, this algorithm converged to the near-optimized architecture faster than other NAS methods. Later, to improve this approach, multi-objective BOs Yang et al. (2020a); Cai and Luo (2021) was introduced to search for the best algorithm while satisfying constraints like inference time, memory usage, etc. which are all hardware-related limitations.

It is easy to find BOs that are being used for different hardware and (IoT) devices for instance robotic and navigation applications Saha et al. (2022). They implemented BO and

a temporal convolutional network (TCN) backbone to satisfy their goal of finding a model for an edge device with lower latency compared to man-made models.

Evolutionary algorithms: One of the most important aspects of the EA algorithms is that they have the potential to find a model with high performance during the optimization. As mentioned previously, in every iteration of this optimization, the algorithm validates the population and keeps the best ones. Then, it generates offspring from the kept population. This way, the algorithm keeps the obtained performance of the found model or optimizes the model to a better architecture. At the end of the search, the close-to-optimal models are listed in the population.

LEMONADE (Lamarckian Evolutionary algorithm for Multi-Objective Neural Architecture DEsign) Elsken et al. (2018) is the first EA based NAS method. They developed a multi-objective EA optimizer (same as Lu et al. (2019)) to search for a model with better performance but with a small architecture size. They managed to speed up the algorithm benefiting a warm start method for the off-spring. As a result, this method achieved models with the same accuracy in comparison with MobileNet-V2 Sandler et al. (2018) but with fewer operations and less latency.

Like any other optimization algorithm, still there are chances for EA algorithm to stuck in local minima. Real et al. Real et al. (201b) introduced a new tournament selection in EA to favor the younger architecture in the population. The results showed that this algorithm found a model with better accuracy and lower computational cost using EA-based SSt in comparison with RL and RS methods.

Developing a zero-cost estimator in conjunction with the guided EA makes the NAS algorithm converge to the best architecture faster Lopes et al. (2022). The Jacobian covariance of the weights in the network is used to evaluate the accuracy of the network at the initial moment. As a result, this method found a model with a competitive accuracy within NAS-BENCH-101 and NAS-BENCH-201 SSps.

Searching for different architectures one by one was slow and there was a high chance of finding local optimal architecture. To solve these issues, NEAS, One-Shot Neural Ensemble Architecture Search Chen et al. (2021a), proposed *K-path* EA to find multiple best models instead of one single model. In this method, a SuperNet is defined as different paths of architecture instead of defining it as a giant architecture and SubNet was a path in the SuperNet. In this way, it became possible to implement a new weight-sharing technique (called layer-sharing technique) to merge different parts of architectures with each other to decrease memory consumption. This led to reduced search and training time.

Shang et al. addressed the issue of the inefficiency of the NAS algorithms and also the challenge of sharing trained weight with off-springs; as a result, they introduced EF-ENAS Shang et al. (2022), which is an improved method of a EA optimization algorithm. In this method, different off-spring generation methods were implemented to find the best architecture. This algorithm contains two parts: first, it defines some blocks that were more effective with previous models, and second, uses them as pre-defined blocks in the generation of offspring. These steps contribute to keeping the valuable and trained parts of the models for the off-springs. The proposed EA algorithm is called *correction-based* which is suitable for decreasing computational time in comparison with traditional EA algorithms.

Sometimes, the NAS algorithms tend to find complex and large-scale architectures with high accuracy which are not efficient for industrial or real-world applications. To solve this problem, Louati et al. Louati et al. (2022) used EA optimizer together with a pruning method

to make the network fit the hardware requirements like suitable memory size for health-care applications. This method works in two-step sequences: the low-level pruning action and a high-level design process, by implementing a co-evolutionary migration-based algorithm.

There are different methods for EA Zhu et al. (2019); Xue et al. (2021); Wen et al. (2021); Gottapu and Dagli (2020); He et al. (2021), like cartesian genetic programming (as a multi-objective NAS method) Pinos et al. (2022); Park and Yi (2022) focusing on the performance of the optimized architecture, genetic-based multi-objective optimizer Geraeinejad et al. (2021) focusing on not only the robustness of the model but also the memory efficiency, Aquila optimization together with a genetic algorithm to overcome more complex SSp Wang et al. (2022a), continuous evolutionary algorithm Chen and Xu (2022); Yang et al. (2020b) focusing on converge faster using weight sharing during searching on the Search Space, and a multi-objective EA method together with an online surrogate model to predict the performance Chen et al. (2022) focusing on solving the problem of having non-efficient NAS algorithms.

Random search: This approach has been mainly used in the NAS literature that focused on Validation Strategy (VSt). The reason is that their methodologies can be introduced and compared with the others independent of employed Search Strategy. Therefore, in the following, we sort the literature based on their VSt.

Partial training - type I: Low Fidelity In NAS algorithms, low-fidelity evaluation was used to reduce computational costs. However, these shallow evaluations can lead to suboptimal architectures due to their limited understanding of the true performance of architectures on the full dataset and with sufficient training. Addressing this issue, Trofimov et al. Trofimov et al. (2020) introduce knowledge distillation into the low-fidelity evaluation process. By transferring knowledge from a pre-trained model to a smaller newly generated network, they can obtain a more accurate estimate of the model's performance on the full dataset. This approach enhances the effectiveness of the NAS, leading to improved performance.

Partial training - type II: Weight Sharing or Inheritance

Neural Architecture Optimization Network (NAONet) Luo et al. (2018) used a weight-sharing method together with a proxy method to solve the NAS problem to find a lighter model on CIFAR-10. In this approach, they converted the discrete SSp to a continuous one by using an encoder-decoder algorithm. In every iteration, the algorithm encodes the architecture to the continuous space (SSp) that provides suitable space for predicting the performance. Later, SSt optimizes the architecture according to the predicted performance. The chosen architecture is decoded into an architecture as in original SSp.

LEMONADE Elsken et al. (2018) developed a technique to share parents' weights with the offspring by mapping the weights from one network to another. This sharing caused the model to inherit performances like accuracy, resource requirements, or other metrics from the old models before training.

Knowledge-Inherited Neural Architecture Search or ModuleNet Chen et al. (2021b) introduced a new technique to share trained parameters. In this method, knowledge of every trained model is transferred to the newly generated model by decomposing the trained model into parts and using them in the architecture generation process. This showed better performance in comparison with the SuperNet weight-sharing technique because of using well-trained parts instead of sharing every single weight in different optimizing iterations.

Some techniques have been developed to improve the performance of the weight-sharing method on a SuperNet. For instance, randomly activating some SubNets or connections on

the SuperNet can prevent the SuperNet from dictating the specific weight to some weights during the search Zhang et al. (2022a).

Adaptive training

GeNAS, Generalization-aware NAS Jeong et al. (2022), introduced a new metric for the NAS algorithms by processing the learning curve. In this method, the authors analyzed the flatness of the loss function during training. They realized that models with flatter loss curves have more generalization ability. Additionally, for more improvements, they mixed conventional cross-entropy loss metrics with flatness metrics, which led to architectures with better performance. This technique contributed to optimizing the architecture on a small dataset and training it on a large dataset (called transferability).

Zero-shot (zero-cost)

Zero-cost VSts or proxies can estimate the performance of the models before training them Zheng et al. (2020); Dai et al. (2021); Lu and Lyu (2021); Xu et al. (2021a); Gracheva (2021); Phan and Luong (2021); Hu et al. (2021). They can be divided into two classes: theory-driven methods that predict the performance of the models at the initial point of the optimizer without knowing about the performance of any model; and data-driven methods that predict the performance of the models using previous knowledge which is gained during the optimization process (they learn from the previous models' performance and later, predict the upcoming models' performance). In simple words, theory-driven methods can predict the performance of the model at any time without needing information from other models but the data-driven models need the performance of several models to learn the prediction and apply it to the new coming models. In the following paragraphs, we dive into more detail about these methods.

Formulating a VSt in the NAS algorithm in a way that does not need to train a model can improve the running time significantly so that the algorithm can be considered a zero processing cost method. NASWOT Mellor et al. (2021) managed to find an architecture with 92% testing accuracy on CIFAR-10 only in 3.05 seconds (using a single *Nvidia GTX 1080 Ti*) which, in comparison, takes more than 13,000 s for the ENAS Pham et al. (2018). The most important feature of this algorithm is that it works as a training-free algorithm that can be run on a single GPU. They implemented an estimation method to predict the validation accuracy at the initialization level. The scoring formula calculated an estimation value for the accuracy of the network without training using:

$$s = log \begin{vmatrix} N_A - d_H(c_1, c_1) & \dots & N_A - d_H(c_1, c_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(c_N, c_1) & \dots & N_A - d_H(c_N, c_N) \end{vmatrix} \quad (1)$$

In this equation, N_A is the number of *ReLu* functions in the model, c_i is a binarized code of the model for the i^th image in the mini-batch, and d_H is Hamming distance between c_is. The equation shows how the model can split different input data in a mini-batch from each other in the linear space of the features using ReLUs.

Genetic algorithm and noise immunity for neural architecture search without training or GA-NINASWOT Wu et al. (2021b) improved the efficiency and generalization of NASWOT Mellor et al. (2021) such that the optimized model has 0.6% higher test accuracy and 47% less time consumption. They used NASWOT's scoring algorithm together with a EA-based Search Strategy. Also, they introduced a look-up table for the validated networks to

store the parameters and validation metrics of the chosen model by the SSt to reduce memory consumption and improve computational efficiency. GA-NINASWOT outperformed NASWOT on all datasets including CIFAR-10, CIFAT-100, and ImageNet16–120 using NAS-BENCH-201 Search Space.

Training-free neural architecture search (TE-NAS) Chen et al. (2021c) introduced a NAS method that is train-free and label-free. This method uses analysis of the architecture in the spectrum of the neural tangent kernel (NTK) Jacot et al. (2018) and the number of linear regions to rank it. In this way, TE-NAS could measure the *trainability* and *expressivity*, which are heavily correlated with the test accuracy of the architecture. *Trainability* measures how much a model can be trained to learn from data. *Expressivity* is a metric that measures the ability of a model to represent a wide range of functions (complexity of the architecture). Also, this algorithm used a pruning mechanism called pruning-by-importance. This approach can enormously decrease the Search Space, to improve the trainability and lets the algorithm completely fine-tune the architecture without sacrificing expressivity. This method was able to find the best architecture on CIFAR-10 and ImageNet only in 0.5 and 4 GPU hours, respectively.

In the FAM method Mokhtari et al. (2022), the performance of the network was estimated by the Intra-Cluster Distance (ICD) score and a EA-based SSt was used. To convert the discrete SSp into a continuous one, they used an encoder to translate the layer types, kernel sizes, etc. into continuous parameters that are easier for the SSt to optimize.

Neural Architecture Optimization (NAO) Luo et al. (2018) is a 3-step NAS algorithm: I) Encoding discrete architecture to continuous space, II) Predicting accuracy and optimizing the architecture in the continuous space, III) Decoding the chosen continuous architecture to a model. Thus, they create a performance prediction function and a continuous SSp, instead of searching the SSp and training the generated model at each step. Similar to NAO Luo et al. (2018), EmProx Franken et al. (2022) developed an encoder-decoder method to map the architecture space to continuous SSp; the difference is that instead of Multilayer Perceptron (MLP), they used weighted k-nearest neighbors for the prediction. This strategy contains two sets of architectures: known and unknown. In this method, the accuracy is given for the first set. Then, the algorithm can learn from the given data to predict the accuracy for the second set of architectures using the kNN algorithm. Compared with other methods like NAO, SemiNAS, XGB, BANANAS, and MLP, this method works faster.

To find the best scoring methods and improve them, Abdelfatteh et al. Abdelfattah et al. (2021) used five more zero-cost metrics (*Snip* Lee et al. (2018), *Grasp*Wang et al. (2020a), *SynapticFlow* Tanaka et al. (2020), *Fisher* Turner et al. (2019), and $Grad - norm$Mellor et al. (2021)) to predict the performance of the models. After calculating these metrics, they used the Spearman Rank correlation coefficient (Spearman's ρ) to compare the correlation of these metrics with the validation accuracy of the models. Between the above-mentioned metrics, *SynapticFlow* showed the highest correlation between all investigated datasets (CIFAR-10, CIFAR-100, ImageNet16–120). They introduced a new metric called *vote*, which takes the majority *vote* between the three metrics: *SynpeticFlow*, *Snip*, and $Grad - norm$. they showed that *vote* performed better than each individual metric. Spearman's ρ was consistently above 0.8 on NAS-Bench-201. Also, the *vote* was able to predict the accuracy using just 3 mini-batches.

To introduce other proxy methods briefly, we can mention the following literature: NPE-NAS Wei et al. (2022b) used two different estimators: a BO acquisition function, which is

used as a graph-based uncertainty estimation network; and a graph-based neural network for predicting the performance of the architecture. Using a meta-learning framework, the NAS algorithm learned from a small number of samples in the SSp without probing the whole space Zhao et al. (2021b). Using few-shot NAS, they found more accurate models in comparison with one-shot models Bender et al. (2018). Fang et al. Fang et al. (2020) optimized the architecture for specific hardware with respect to accuracy and *FLOPs*/*latency*. Li et al. Li et al. (2020a) reformulated the validation step in a way to estimate the loss using weights of the model.

In the literature, there are some other methods to predict the performance of models without traditional training, such as the Firefly Algorithm method (FAM) Mokhtari et al. (2022) that predicts the weights for the model, EPE-NAS Lopes et al. (2021) that predicted the accuracy using *Jacobian* of weights of the model, and KNAS Xu et al. (2021b) that used the gradient of weights to estimate the performance.

Other methods: Besides the methods mentioned above, some other optimizers can be found in the literature that have been used to improve the performance of the NAS algorithms. Having known that searching for an optimized model on a SuperNet or in a big Search Space needs tons of processing power, it is more efficient to add layers one after another to the structure till converging to a model with acceptable performance. This optimization method is called growth-based strategy and has been implemented in the NAS algorithm Millán et al. (2018). Further, a Biology Inspired version of the Growth optimizer LaKemper et al. (2022) has been developed as a method for modifying the artificial neural network's structure. Introducing DensEMANN Garcia-Diaz and Bersini (2021) brought this opportunity to build architectures layer-by-layer and kernel-by-kernel during training with better performance. In this algorithm, it was possible to prune or add layers/kernels simultaneously during the optimization.

HotNAS Jiang et al. (2020a) developed a NAS algorithm for optimizing the pre-trained architectures instead of starting the algorithm from scratch. This method tried to improve the available models' performance and not generate a new model. In this way, they managed to reduce the searching time to 3 GPU hours on ImageNet and only 20 min on CIFAR-10 even without a proxy. They created a model library named 'Model Zoo' with 24 human-designed models for ImageNet among which only 4 models could satisfy the computational limitation of the chosen hardware. Later, they used this pre-trained model zoo considering some constraints like a range of 5*ms* latency on *Xilinx ZCU 102 FPGA* which came up with a model that reached the 87.50% accuracy. By changing these constraints the algorithm optimized a model with 90% accuracy with latency of $5 - 10ms$.

One other SSt for the NAS algorithms is the graph-based search algorithm Su et al. (2021); Wang et al. (2020b). This method brings a good balance between exploration and refinement to the SSt and speeds up the algorithm in comparison with RS and greedy search methods. Further, by introducing TNAS (NAS with trees) Qian et al. (2022), authors were able to prune the SSp, which can improve the search time extremely.

Knowing that during the training of a neural network, the most influential weights are subjected to bigger changes in comparison with the other weights, searching with Random Labels NAS or RLNAS Zhang et al. (2021b) developed an algorithm to find the optimized model by analyzing the changes. For this purpose, the SuperNet was trained with random labels. Then, by selecting the most changed weights in the network, they managed to choose

the SubNet. It was shown that the optimized model had better accuracy in comparison with previous methods.

Quantum-inspired NAS, or Q-NAS Szwarcman et al. (2019), tried to optimize the operations on fixed layer numbers. The quantum-inspired optimizer chooses a layer operation from a small SSp by quantum probabilities. Later, they improved their algorithm even further to touch the $+90\%$ accuracy on CIFAR-10 Szwarcman et al. (2022). The downside of this algorithm is its running time; it takes more than 50 GPU days to optimize the model.

Moreover, there are different methods, except the mentioned algorithm, in the literature like binarized neural networks Shen et al. (2019), swarm intelligence Byla and Pang (2019), greedy optimizers You et al. (2020); Li et al. (2020c), novelty search strategy Zhang et al. (2020a), attention-based search Nakai et al. (2021), slow-fast learning Tan et al. (2021), enhanced RL mixed with a new reward function Cassimon et al. (2020), etc.

Hybrid methods: Some methods mix different optimizers to increase the accuracy or the search speed. For instance, the Bayesian Learning rule to the architecture optimization in differentiable NAS (BaLeNAS) Zhang et al. (2022b) introduced to improve the baseline of the architectures to solve the issue that the DARTS Chen and Hsieh (2020) algorithm tends to find complex architecture because of shared weights from SuperNet. As a result, BaLeNAS reached a better accuracy in comparison with DARTSLiu et al. (2018a), and Zero-cost NAS Abdelfattah et al. (2021), etc. BaLeNAS-TF (train-free) achieved even better results with 94.33% test accuracy on CIFAR-10. Furthermore, to list hybrid methods briefly, we can mention: Fast Evolutionary NAS (FENAS) Shi et al. (2021) and Adaptive scalable NAS Zhang et al. (2021c). Also, there are some methods that are mixed of different optimization techniques and SSp structures for example Jing et al. Jing et al. (2022) used DAG and introduced a Neural Architecture Generator (NAG) to optimize the model. In this method, they utilized a generative adversarial network (GAN) framework to effectively explore the vast architecture space. GAN algorithms consist of two components: a generator that produces DAGs a.k.a model, and a discriminator that assesses the quality of the generated model, the discriminator compares the metrics of the generated model with models from the metrics dataset. Both of these networks benefited Gradient-based optimizer to be optimized; but, we classified this approach in this category, not in GO methods, knowing that the model searching process was a RS technique to generate noise as an input for the generator. In this approach, the algorithm needs to be trained once and used several times which can increase the efficiency of the NAS algorithm.

4.2 NAS for specific hardware

These days, NAS algorithms are finding their way to the edge and Internet of Things (IoT) devices, to bring neural networks close to the sensors. This allows (IoT) devices to have better performance and less latency. Before diving into the details, the following hardware-related terminologies should be introduced:

1. SRAM (Static Random-Access Memory): is a type of memory that stores data as long as power is applied and loses all the data when the power is off.
2. FLASH memory: is a type of memory that can store data even when power is lost.
3. Model size: is the amount of memory required to store a neural network model. The size of a model is determined by the number of parameters in the model.

4. Inference time: is the time it takes for a model to process an input and produce an output which can affect the latency.
5. Uptime: is the percentage of time that the device is ready to process the new data.Knowing that microcontroller units (MCUs) are cheap and self-contained, using the NAS algorithm will help (IoT) devices to work as stand-alone devices with acceptable inference time. Also, due to MCUs' low power consumption and ability to be an always-on system, they have been getting more attention recently in different applications like healthcare systems or low-cost Non-Destructive Testing (NDT)/Structural health monitoring (SHM) systems.

Sparse Architecture Search, so-called SpArSe Fedorov et al. (2019), introduced a method that is able to search for a network with small working memory, less model size, and better accuracy, which is able to be run on an MCU. They used BO-based multi-objective optimization to optimize the architecture with respect to 3 objectives: performance, model size, and inference time. They applied their method to MNIST, CIFAT-10, CUReT Dana et al. (1999), and Chars4k de Campos et al. (2009) and managed to find optimized models with a testing accuracy of 98.64%, 73.84%, 80.68%, and 77.78%, respectively. All the models needed less than $2KB$ RAM and had less than $2KB$ model size of an MCU. The algorithm requires 4 GPU days on average to find the best model.

Due to the importance of knowing the real-world latency of models on specific hardware for the NAS algorithms, using actual latency to calculate the OF of the NAS algorithm. For instance, MnasNet Tan et al. (2019) used a mobile phone to feed the exact real-world latency to the SSt. It was argued in this paper that the number of parameters and FLOPs are not accurate metrics to predict the latency on the actual device. This is mainly due to the difference in the running effort of different processing units even with the same number and type of operations. To generate and optimize models that are faster on the mobile device in comparison with other hardware, they defined a SSp using convolution layers, different kernels shape, two squeeze-and-excitation levels, skip connections, and so on. This helped them to maximize the accuracy while keeping the latency low. The SSt was a RL algorithm. In the end, they managed to find 3 architectures with $78 - 103ms$ latency in 4.5 days on 64 $TPUv2$. Moreover, it was able to achieve 75.2% top-1 accuracy with $78ms$ latency on a Google Pixel 1 phone, which is 2.3 times faster and 1.2% more accurate than NASnetQin and Wang (2019) and 1.8 times faster and 0.5% more accurate than MobileNetV2 Sandler et al. (2018).

MCUNet Lin et al. (2020) introduced a NAS algorithm that first, modifies and optimizes the SSp, then starts to search for the proper architecture in the modified SSp. As a result, MCUNet was able to find an architecture to achieve more than 70% top-1 accuracy on ImageNet for an MCU. This architecture used 3.5 times less SRAM and 5.7 times less flash memory compared to quantized MobileNetV2 Sandler et al. (2018). Also, their network was $2.4 - 3.4$ times faster with $3.7 - 4.1$ times smaller peak SRAM than MobileNetV2 Sandler et al. (2018). Later, MCUNetV2 Lin et al. (2021) is introduced to improve the efficiency and time consumption of MCUNet Lin et al. (2020). MCUNetV2 is $4 - 8$ times more efficient than MCUNet in peak memory usage and achieved 71.8% accuracy on ImageNet.

To optimize the model for Raspberry Pi 3 (RPi3), an RL optimizer has been used as well as a look-up table for the processing time that helps the algorithm to estimate the latency of the executing operations on the RPi3 Cassimon et al. (2020). They improved the ENAS

Pham et al. (2018) algorithm by defining two sets of constraints: hard constraints such as memory usage and latency, and soft constraints such as compression, cache usage, and network performance.

EdgeNAS Luo et al. (2020) developed a latency estimator and used a GO-based SSt to find the fastest model (model with lowest latency) for edge devices. Like MnasNet Tan et al. (2019), they provide some measurements that show how the latency and FLOPs are not perfectly correlated; therefore, they improved the latency estimation algorithm that is more correlated with the latency by which the NAS algorithm can find models with better performance and lower latency.

MicroNets Banbury et al. (2021) considered the opposite approach and decided to trust the number of operations as a metric for latency and energy consumption. A differentiable neural architecture search algorithm (DNAS), together with the number of operations as a OF was used to find suitable models with less latency and better performance/accuracy for MCUs. MCUs' hardware-related performance like the latency of the layers, the latency of the models, and the models' energy consumption were taken into account to define the hardware constraints for the optimizer. They also showed that energy consumption and latency have an acceptable linear relationship with the number of operations. Having the same amount of latency, MicroNets was able to reach better accuracy than MobileNetV2 Sandler et al. (2018). Furthermore, generated networks need less FLASH and SRAM memory in comparison with MobileNetV2.

There are other papers on the NAS algorithms for (IoT) and edge devices that we can mention briefly with: different optimization methods Zhao et al. (2020); Zhang et al. (2020b); Yang et al. (2021); Luo et al. (2021); different metrics Jiang et al. (2020b); Dong et al. (2021a); different Hardware Ipenburg et al. (2021); Li et al. (2021a); Dong et al. (2021a); Cardoso-Pereira et al. (2021); Liberis et al. (2021); and so on so forth. For finding more information about hardware-aware NAS, you can read Benmeziane et al. (2021). They gather valuable knowledge of the NAS using a hardware-aware perspective by considering two main challenges, the variety of data and the variety of hardware.

5 Discussion

This systematic literature review reveals that existing research is dominated by a few research questions:

1. Why NAS is beneficial and What are the underlying motivations for every transition in the field?
2. What is the current situation?
3. What are the benefits and the opportunities of NAS?In this section, we analyze and discuss present research in order to identify knowledge gaps and opportunities for future research.

5.1 Introduction and motivations in the NAS

Hoping to automate the AI models design and optimization led to the NAS algorithms. The NAS algorithm contains three pillars that contribute to bringing the automatic architecture

design into the practical world. In this journey, three different transitions happened: working on optimizing the whole architecture for a dataset, finding proper SSt for the algorithm that can lead to an optimized model, and focusing on predicting the performance of architectures.

The main goal of AI users was to find an architecture with the best performance for a dataset. This goal pushes the researchers to develop NAS framework to let everyone find the most accurate model without diving deep into the AI structures. Later, researchers used high-performance computational units to train and validate different architectures. However, the main lack in the process was the knowledge of choosing proper architecture and improving it. Therefore, researchers started to develop different SSts which head to the next barrier, time limitation. At this stage, the algorithms needed several days to find the optimized architecture. To solve this problem, they started to develop some algorithms to find the validation data faster. This trend led to the point that zero-cost VSts take the helm of the NAS algorithms and bring today's state-of-the-art performances.

5.2 Current status analysis

In this subsection, the main challenges and open issues in each of mentioned three pillars will be discussed.

The first pillar of the NAS algorithm is Search Space. Some defined spaces can be used for optimizing models for specific hardware or as benchmarks, but it is not possible to find a Search Space that is suitable for all types of data and scenarios because a general Search Space should contain all possible operations, layers, connections, etc. which make the space indefinite. On one side of the coin, the large size of the Search Space increases the searching time; but on the other side, larger Search Space guarantees more optimized architecture with better performance. Therefore, selecting the Search Space is kind of a trade-off between how much time needs to be spent and how accurate models we expect the NAS algorithm to find. Additionally, for optimizing models for specific hardware like edge devices, we need to limit the Search Space with some options that consume less time and energy on these devices.

The second pillar, Search Strategy, is an optimizer that works as a human in the NAS algorithm to choose the most efficient and most accurate architecture to overcome human-designed architectures. The Search Strategy can control the time consumption of the algorithm indirectly. The SSt, optimizer, uses a small portion of time and energy in comparison with training but the number of generated models that need to be validated during the process can make a huge difference at the end of the day. Therefore, researchers are looking for a Search Strategy that does not need a lot of iterations to find the optimized architecture. These days, Gradient-based optimizer (GO) and Bayesian Optimizer (BO) have become popular because they can satisfy the mentioned demand.

The last pillar is the Validation Strategy, which dictates the amount of time the NAS algorithm needs to validate a model. Many methods are introduced to decrease time consumption to have faster NAS algorithms like low fidelity or processing learning curve. That is true that some methods like weight sharing Pham et al. (2018) managed to increase the efficiency and speed of the algorithm but proxies/estimators are acting faster than any other methods that are based on training, no matter if it needs to train the model from scratch or not. The most important breakthrough in VSts is Neural Architecture Search without training method which is introduced as NASWOT Mellor et al. (2021).

5.3 Benefits and future direction

It should be mentioned that the astonishing performance of the architectures which are designed by NAS algorithms showed the benefits of this field. Continuously, better and more efficient techniques are being introduced and they are making the ground ready for the low-performance hardware to catch up with others, especially by increasing the popularity of edge and mobile devices. Focusing on the model latency on edge devices Lin et al. (2020) opens a new sub-field that combines both different hardware specifics and NAS techniques to create more efficient models with better performance.

The trend of the NAS algorithms going toward the improvement of the VSts that can predict the performance of the model on different datasets without training. The biggest challenge here is the variety of different architectures and also the variety in spaces and types of data.

6 Conclusion

In this study, we have systematically reviewed research articles on Neural Architecture Search (NAS). We analyzed the contributions with respect to specific research questions. This article contributes to research in several ways. First, it provides a systematic overview of existing research from 2017 to mid-2023. We have identified 160 significant contributions including journal articles and articles on conference proceedings. The contributions have been systematically introduced, analyzed, classified, and predicted the future of this emergent research field and will ease researchers' search for relevant studies. Second, through a thorough analysis, we have proposed potential areas and approaches for future studies.

The review concludes that the motives for every transition in NAS field, current status analysis, outcomes, benefits, research opportunities, and field direction are the most dominant topics in current research.

References

Abdelfattah M.S, Mehrotra A, Dudziak Ł, Lane N.D. (2021). Zero-cost proxies for lightweight NAS. arXiv preprint arXiv:2101.08134

Baker B, Gupta O, Raskar R, Naik N: (2017). Accelerating neural architecture search using performance prediction. arXiv preprint arXiv:1705.10823

Banbury C, Zhou C, Fedorov I, Matas R, Thakker U, Gope D, Janapa Reddi V, Mattina M, Whatmough P (2021) Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. Proc Machine Learn Syst 3:517–532

Barnell M, Raymond C, Smiley S, Isereau D, Brown D. (2022). Ultra low-power deep learning applications at the edge with Jetson Orin AGX hardware. In: 2022 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–4 IEEE
Bender G, Kindermans P.-J, Zoph B, Vasudevan V, Le Q. (2018). Understanding and simplifying one-shot architecture search. In: International Conference on Machine Learning, pp. 550–559. PMLR
Benmeziane H, Maghraoui KE, Ouarnoughi H, Niar S, Wistuba M, Wang N (2021) A comprehensive survey on hardware-aware neural architecture search. arXiv preprint arXiv:2101.09336
Byla E, Pang W: Deepswarm: (2019). Optimising convolutional neural networks using swarm intelligence. In: UK Workshop on Computational Intelligence, pp. 119–130 . Springer
Caglar B, Broggi G, Ali MA, Orgéas L, Michaud V (2022) Deep learning accelerated prediction of the permeability of fibrous microstructures. Composites Part Appl Sci Manufact 158:106973
Cai R, Luo J: (2021). Multi-task learning for multi-objective evolutionary neural architecture search. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 1680–1687. IEEE
Campos TE, Babu BR, Varma M (2009) Character recognition in natural images. Int Conf Computer Vision Theory Appl 1:273–280
Cardoso-Pereira I, Lobo-Pappa G, Ramos HS (2021) Neural architecture search for resource-constrained internet of things devices. In: 2021 IEEE Symposium on Computers and Communications (ISCC), pp. 1–6. IEEE
Cassimon T, Vanneste S, Bosmans S, Mercelis S, Hellinckx P (2020) Designing resource-constrained neural networks using neural architecture search targeting embedded devices. Int Things 12:100234
Chen X, Hsieh C-J (2020). Stabilizing differentiable architecture search via perturbation-based regularization. In: International Conference on Machine Learning, pp. 1554–1565. PMLR
Chen L, Xu H (2022) Mfenas: multifactorial evolution for neural architecture search. Proc Genet Evol Comput Conf Companion 10:631–634
Chen H, Zhang B, Zheng X, Liu J, Doermann D, Ji R (2020). Binarized neural architecture search. Proc AAAI Conf Artif Intell 34:10526–10533
Chen M, Fu J, Ling H (2021a) One-shot neural ensemble architecture search by diversity-guided search space shrinking. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 16530–16539
Chen Y, Gao R, Liu F, Zhao D (2021b) Modulenet: Knowledge-inherited neural architecture search. IEEE Transactions on Cybernetics
Chen W, Gong X, Wang Z (2021c). Neural architecture search on ImageNet in four GPU hours: A theoretically inspired perspective. arXiv preprint arXiv:2102.11535
Chen H, Huang H, Zuo X, Zhao X (2022) Robustness enhancement of neural networks via architecture search with multi-objective evolutionary optimization. Mathematics 10(15):2724
Cook S. (2012). CUDA Programming: a Developer's Guide to Parallel Computing with GPUs. Newnes .
Dai X, Wan A, Zhang P, Wu B, He Z, Wei Z, Chen K, Tian Y, Yu M, Vajda P: (2021). FBNetV3: Joint architecture-recipe search using predictor pretraining. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 16276–16285
Dana KJ, Van Ginneken B, Nayar SK, Koenderink JJ (1999). Reflectance and texture of real-world surfaces. ACM Trans Graph (TOG) 18(1):1–34
Del Valle AM, Mantovani RG Cerri R. (2023). A systematic literature review on automl for multi-target learning tasks. Artif Intell Rev 10:1–40
Dillon J.V, Langmore I, Tran D, Brevdo E, Vasudevan S, Moore D, Patton B, Alemi A, Hoffman M, Saurous R.A. (2017). Tensorflow distributions. arXiv preprint arXiv:1711.10604
Ding Z, Chen Y, Li N, Zhao D, Sun Z, Chen CP (2021) BNAS: Efficient neural architecture search using broad scalable architecture. IEEE Trans Neural Netw Learn Syst 10:12
Ding Z, Chen Y, Li N, Zhao D: BNAS-v2, (2022) Memory-efficient and performance-collapse-prevented broad neural architecture search. IEEE Trans Syst Man Cyber Syst 10:15
Ditty M: NVIDIA ORIN system-on-chip. In: 2022 IEEE Hot Chips 34 Symposium (HCS), pp. 1–17 (2022). IEEE Computer Society
Domhan T, Springenberg J.T, Hutter F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Twenty-fourth International Joint Conference on Artificial Intelligence
Dong X, Yang Y (2019) Searching for a robust neural architecture in four GPU hours. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1761–1770
Dong X, Yang Y: NAS-bench-201: (2020). Extending the scope of reproducible neural architecture search. arXiv preprint arXiv:2001.00326
Dong Z, Gao Y, Huang Q, Wawrzynek J, So H.K, Keutzer K (2021a). Hao: Hardware-aware neural architecture optimization for efficient inference. In: 2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 50–59. IEEE

Dong X, Liu L, Musial K, Gabrys B (2021b) NATS-Bench: Benchmarking nas algorithms for architecture topology and size. IEEE transactions on pattern analysis and machine intelligence
Efficient architecture search for deep neural networks (2020) Gottapu R.D., Dagli C.H. Proc Computer Sci 168:19–25
Elsken T, Metzen J.H, Hutter F (2018) Efficient multi-objective neural architecture search via Lamarckian evolution. arXiv preprint arXiv:1804.09081
Elsken T, Metzen JH, Hutter F (2019). Neural architecture search: a survey. J Machine Learn Res 20(1):1997–2017
Falkner S, Klein A, Hutter F: Bohb: (2018). Robust and efficient hyperparameter optimization at scale. In: International Conference on Machine Learning, pp. 1437–1446 . PMLR
Fan Y, Niu Z.-H, Yang Y.-B. (2023). Data-aware zero-shot neural architecture search for image recognition. In: ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1–5. IEEE
Fang J, Sun Y, Zhang Q, Li Y, Liu W, Wang X. (2020). Densely connected search space for more flexible neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10628–10637
Fedorov I, Adams RP, Mattina M, Whatmough P (2019) Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. Adv Neural Inform Process Syst 32:210
Franken G, Singh P, Vanschoren J: Emprox. (2022). Neural network performance estimation for neural architecture search. arXiv preprint arXiv:2206.05972
Garcia-Diaz A, Bersini H (2021) Densemann: Building a densenet from scratch, layer by layer and kernel by kernel. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–10 . IEEE
García-Holgado A, Marcos-Pablos S, García-Peñalvo F (2020) Guidelines for performing systematic research projects reviews
Geraeinejad V, Sinaei S, Modarressi M, Daneshtalab M: RoCo-NAS, (2021) Robust and compact neural architecture search. Int Joint Conf Neural Netw (IJCNN) 10:1–8
Giveki D, Karami M (2020) Scene classification using a new radial basis function classifier and integrated sift-lbp features. Pattern Anal Appl 23(3):1071–1084
Gracheva E (2021) Trainless model performance estimation based on random weights initialisations for neural architecture search. Array 12:100082
He C, Tan H, Huang S, Cheng R (2021) Efficient evolutionary neural architecture search by modular inheritable crossover. Swarm Evol Comput 64:100894
Hong M.-F, Chen H.-Y, Chen M.-H, Xu Y.-S, Kuo H.-K, Tsai Y.-M, Chen H.-J, Jou K. (2021). Network space search for pareto-efficient spaces. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3053–3062
Howard A, Sandler M, Chu G, Chen L.-C, Chen B, Tan M, Wang W, Zhu Y, Pang R, Vasudevan V: (2019). Searching for MobileNetV3. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1314–1324
Hu S, Cheng R, He C, Lu Z, Wang J, Zhang M (2021) Accelerating multi-objective neural architecture search by random-weight evaluation. Complex Intell Syst 10:1–10
Hu M, Pour Avval S.S, He J, Yue N, Groves RM (2024) Explainable artificial intelligence study on bolt loosening detection using lamb waves. Jian and Yue, Nan and Groves, Roger M., Explainable Artificial Intelligence Study on Bolt Loosening Detection Using Lamb Waves
Huang H, Ma X, Erfani SM, Bailey J (2021) Neural architecture search via combinatorial multi-armed bandit. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 . IEEE
Ipenburg Iv, Sapra D, Pimentel AD (2021) Exploring cell-based neural architectures for embedded systems. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 363–374. Springer
Jacot A, Gabriel F, Hongler C (2018) Neural tangent kernel: Convergence and generalization in neural networks. Adv Neural Inform Process Syst 31:225
Jeong J, Yu J, Han D, Yoo Y (2022) Neural architecture search with loss flatness-aware measure
Jiang W, Yang L, Dasgupta S, Hu J, Shi Y (2020a) Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. IEEE Trans Computer-Aided Design Integr Circuits Syst 39(11):4154–4165
Jiang W, Yang L, Sha EH-M, Zhuge Q, Gu S, Dasgupta S, Shi Y, Hu J (2020b) Hardware/software co-exploration of neural architectures. IEEE Trans Computer-Aided Design Integr Circuits Syst 39(12):4805–4815
Jin H, Song Q, Hu X (2019) Auto-Keras: An efficient neural architecture search system. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1946–1956
Jing K, Xu J, Zhang Z (2022) A neural architecture generator for efficient search space. Neurocomputing 486:189–199

Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
Kandasamy K, Neiswanger W, Schneider J, Poczos B, Xing EP (2018) Neural architecture search with bayesian optimisation and optimal transport. Adv Neural Inform Process Syst 31:210
Keele S (2007) Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, ver. 2.3 ebse technical report. ebse
Klein A, Falkner S, Springenberg JT, Hutter F (2016) Learning curve prediction with Bayesian neural networks
Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images
LaKemper C.A, Wang C, Yoder J.A: (2022). Biology inspired growth in meta-learning. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 63–64
Lange S, Riedmiller M: Deep auto-encoder neural networks in reinforcement learning. In: The 2010 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2010). IEEE
LeCun Y, Cortes C, Burges C: MNIST handwritten digit database. ATT Labs. http://yann.lecun.com/exdb/mnist **2**
Lee N, Ajanthan T, Torr P.H. (2018). Snip: Single-shot network pruning based on connection sensitivity. arXiv preprint arXiv:1810.02340
Li L, Talwalkar A (2020) Random search and reproducibility for neural architecture search. In: Uncertainty in Artificial Intelligence, pp. 367–377. PMLR
Li L, Jamieson K.G, DeSalvo G, Rostamizadeh A, Talwalkar A (2017) Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In: ICLR (Poster)
Li Y, Dong M, Wang Y, Xu C (2020a) Neural architecture search in a proxy validation loss landscape. In: International Conference on Machine Learning, pp. 5853–5862. PMLR
Li Z, Xi T, Deng J, Zhang G, Wen S, He R: Gp-nas (2020b) Gaussian process based neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11933–11942
Li G, Qian G, Delgadillo I.C, Muller M, Thabet A, Ghanem B (2020c) SGAS: Sequential greedy architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1620–1630
Li G, Mandal SK, Ogras UY, Marculescu R (2021a) Flash: Fast neural architecture search with hardware optimization. ACM Trans Embedded Comput Syst (TECS) 20(5s):1–26
Li C, Yu Z, Fu Y, Zhang Y, Zhao Y, You H, Yu Q, Wang Y, Lin Y (2021b) HW-NAS-Bench: Hardware-aware neural architecture search benchmark. arXiv preprint arXiv:2103.10584
Li S, Mao Y, Zhang F, Wang D, Zhong G (2022) DLW-NAS: Differentiable light-weight neural architecture search. Cognitive Computation, 1–11
Liberis E, Dudziak Ł, Lane ND (2021) μnas: Constrained neural architecture search for microcontrollers. In: Proceedings of the 1st Workshop on Machine Learning and Systems, pp. 70–79
Lin J, Chen W-M, Lin Y, Gan C, Han S (2020) MCUNET: Tiny deep learning on IoT devices. Adv Neural Inform Process Syst 33:11711–11722
Lin J, Chen W-M, Cai H, Gan C, Han S (2021a) Memory-efficient patch-based inference for tiny deep learning. Adv Neural Inform Process Syst 34:2346–2358
Lin M, Wang P, Sun Z, Chen H, Sun X, Qian Q, Li H, Jin R (2021b) Zen-NAS: A zero-shot NAS for high-performance image recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 347–356
Liu H, Simonyan K, Yang Y (2018a) DARTS: Differentiable architecture search. arXiv preprint arXiv:1806.09055
Liu C, Zoph B, Neumann M, Shlens J, Hua W, Li L-J, Fei-Fei L, Yuille A, Huang J, Murphy K (2018b) Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 19–34
Liu C, Chen L-C, Schroff F, Adam H, Hua W, Yuille A.L, Fei-Fei L (2019) Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 82–92
Loni M, Sinaei S, Zoljodi A, Daneshtalab M, Sjödin M (2020) Deepmaker: a multi-objective optimization framework for deep neural networks in embedded systems. Microproc Microsyst 73:102989
Lopes V, Alexandre LA (2022) Towards less constrained macro-neural architecture search. arXiv preprint arXiv:2203.05508
Lopes V, Alirezazadeh S, Alexandre LA (2021) EPE-NAS:Efficient performance estimation without training for neural architecture search. In: International Conference on Artificial Neural Networks, Springer, pp. 552–563

Lopes V, Santos M, Degardin B, Alexandre LA (2022) Efficient guided evolution for neural architecture search. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO '22, pp. 655–658. Association for Computing Machinery New York, NY, USA https://doi.org/10.1145/3520304.3528936
Louati H, Bechikh S, Louati A, Aldaej A, Said LB (2022) Joint design and compression of convolutional neural networks as a bi-level optimization problem. Neural Comput Appl 10:1–23
Lu L, Lyu B (2021) Reducing energy consumption of neural architecture search: an inference latency prediction framework. Sustain Cities Soc 67:102747
Lu Z, Whalen I, Boddeti V, Dhebar Y, Deb K, Goodman E, Banzhaf W (2019) NSGA-Net: neural architecture search using multi-objective genetic algorithm. Proc Genet Evol Comput Conf 10:419–427
Luo R, Tian F, Qin T, Chen E, Liu T-Y (2018) Neural architecture optimization. Adv Neural Inform Process Syst 31:210
Luo X, Liu D, Kong H, Liu W (2020) Edgenas: Discovering efficient neural architectures for edge systems. In: 2020 IEEE 38th International Conference on Computer Design (ICCD), pp. 288–295. IEEE
Luo X, Liu D, Huai S, Liu W (2021) Hsconas: Hardware-software co-design of efficient dnns via neural architecture search. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 418–421 . IEEE
Mao Y, Zhong G, Wang Y, Deng Z: Differentiable light-weight architecture search. In: 2021 IEEE International Conference on Multimedia and Expo (ICME), pp. 1–6 (2021). IEEE
Mellor J, Turner J, Storkey A, Crowley E.J: (2021). Neural architecture search without training. In: International Conference on Machine Learning, pp. 7588–7598. PMLR
Mey O, Neudeck W, Schneider A, Enge-Rosenblatt O: Machine learning-based unbalance detection of a rotating shaft using vibration data. In: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), vol. 1, pp. 1610–1617 (2020). IEEE
Millán A.P, Torres J, Johnson S, Marro J: (2018). Growth strategy determines network performance. arXiv preprint arXiv:1806.01878
Mills K.G, Salameh M, Niu D, Han F.X, Rezaei S.S.C, Yao H, Lu W, Lian S, Jui S. (2021). Exploring neural architecture search space via deep deterministic sampling. IEEE Access 9 110962–110974
Moher D, Liberati A, Tetzlaff J, Altman DG (2009) the PRISMA Group: Preferred reporting items for systematic reviews and meta-analyses: the prisma statement. Ann int Med 151(4):264–269
Mokhtari N, Nédélec A, Gilles M, De Loor P. (2022). Improving neural architecture search by mixing a firefly algorithm with a training free evaluation
Nakai K, Matsubara T, Uehara K (2021) Neural architecture search for convolutional neural networks with attention. IEICE Trans Inform Syst 104(2):312–321
Niu S, Wu J, Zhang Y, Guo Y, Zhao P, Huang J, Tan M (2021) Disturbance-immune weight sharing for neural architecture search. Neural Netw 144:553–564
Park G, Yi Y (2022) Condnas: Neural architecture search for conditional cnns. Electronics 11(7):1101
Phan QM, Luong NH (2021) Efficiency enhancement of evolutionary neural architecture search via training-free initialization. In: 2021 8th NAFOSTED Conference on Information and Computer Science (NICS), pp. 138–143 . IEEE
Pham H, Guan M, Zoph B, Le Q, Dean J (2018) Efficient neural architecture search via parameters sharing. In: International Conference on Machine Learning, pp. 4095–4104 . PMLR
Pinos M, Mrazek V, Sekanina L (2022) Evolutionary approximation and neural architecture search. Genet Program Evol Machines 12:1–24
Qian G, Zhang X, Li G, Zhao C, Chen Y, Zhang X, Ghanem B, Sun J: (2022). When NAS meets trees: An efficient algorithm for neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2782–2787
Qin X, Wang Z: Nasnet: (2019). A neuron attention stage-by-stage net for single image deraining. arXiv preprint arXiv:1912.03151
Rastegar H, Giveki D (2023) Designing a new deep convolutional neural network for content-based image retrieval with relevance feedback. Computers Electr Eng 106:108593
Rastegar H, Giveki D, Choubin M (2024) Eeg signals classification using a new radial basis function neural network and jellyfish meta-heuristic algorithm. Evol Intell 17(2):1197–1208
Real E, Aggarwal A, Huang Y, Le QV (2019a) Aging evolution for image classifier architecture search. AAAI Conf Artif Intell 2:2
Real E, Aggarwal A, Huang Y, Le QV (2019b) Regularized evolution for image classifier architecture search. In: Proceedings of the Aaai Conference on Artificial Intelligence, vol. 33, pp. 4780–4789
Runge F, Stoll D, Falkner S, Hutter F: (2018) Learning to design rna. arXiv preprint arXiv:1812.11951
Saha SS, Sandha SS, Garcia LA (2022) Srivastava M: Tinyodom: Hardware-aware efficient neural inertial navigation. Proc ACM Interactive Mobile Wearable Ubiquitous Technol 6(2):1–32

Salmanipour S, Aghdami A, Abdoli SM, Sokhansanj A (2023) Separation of a two binary-azeotrope acetonitrile-cyclohexane-toluene ternary mixture via continuous triple column extractive distillation with heat integration: design, simulation, and multi-objective genetic-algorithm (moga) optimization. Separation Sci Technol 58(14):2539–2555

Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C (2018) MobileNetV2: Inverted residuals and linear bottlenecks. Proc IEEE Conf Computer Vision Pattern Recognition. 10:4510–4520

Saxena S, Verbeek J (2016) Convolutional neural fabrics. Adv Neural Inform Process Syst 20:10

Shang R, Zhu S, Ren J, Liu H, Jiao L (2022) Evolutionary neural architecture search based on evaluation correction and functional units. Knowledge-Based Syst 10:109206

Shaw A, Wei W, Liu W, Song L, Dai B (2019) Meta architecture search. Adv Neural Inform Process Syst 23:10

Shen M, Han K, Xu C, Wang Y: (2019). Searching for accurate binary neural architectures. In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, pp. 0–0

Shi R, Luo J, Liu Q: (2021). Fast evolutionary neural architecture search based on Bayesian surrogate model. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 1217–1224. IEEE

Siems J, Zimmer L, Zela A, Lukasik J, Keuper M, Hutter F: (2020). NAS-bench-301 and the case for surrogate benchmarks for neural architecture search. arXiv preprint arXiv:2008.09777

Silberman N, Fergus R: NYU Depth V2 dataset. NYU Depth V2 Dataset (2011)

Song HY (2021) A method for gradient differentiable network architecture search by selecting and clustering candidate operations. Appl Sci 11(23):11436

Su X, Huang T, Li Y, You S, Wang F, Qian C, Zhang C, Xu C: (2021). Prioritized architecture sampling with monto-carlo tree search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10968–10977

Swersky K, Snoek J, Adams R.P. (2014). Freeze-thaw Bayesian optimization. arXiv preprint arXiv:1406.3896

Szwarcman D, Civitarese D, Vellasco M (2019) Quantum-inspired neural architecture search. In: 2019 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 . IEEE

Szwarcman D, Civitarese D, Vellasco M (2022) Quantum-inspired evolutionary algorithm applied to neural architecture search. Appl Soft Computing 120:108674

Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le QV (2019) MnasNet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2820–2828

Tan H, Cheng R, Huang S, He C, Qiu C, Yang F, Luo P (2021) Relativenas: relative neural architecture search via slow-fast learning. IEEE Trans Neural Netw Learn Syst 10:210

Tanaka H, Kunin D, Yamins DL, Ganguli S (2020) Pruning neural networks without any data by iteratively conserving synaptic flow. Adv Neural Inform Process Syst 33:6377–6389

Targ S, Almeida D, Lyman K: Resnets in resnets: Generalizing residual architectures. arXiv preprint arXiv:1603.08029 (2016)

Trofimov I, Klyuchnikov N, Salnikov M, Filippov A, Burnaev E. (2020). Multi-fidelity neural architecture search with knowledge distillation. arXiv preprint arXiv:2006.08341

Turner J, Crowley E.J, O'Boyle M, Storkey A, Gray G. (2019). Blockswap: Fisher-guided block substitution for network compression on a budget. arXiv preprint arXiv:1906.04113

Wan A, Dai X, Zhang P, He Z, Tian Y, Xie S, Wu B, Yu M, Xu T, Chen K: (2020) Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12965–12974

Wan X, Ru B, Esparança P.M, Carlucci F.M. (2022). Approximate neural architecture search via operation distribution learning. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 2377–2386

Wang C, Zhang G, Grosse R: (2020a). Picking winning tickets before training by preserving gradient flow. arXiv preprint arXiv:2002.07376

Wang L, Zhao Y, Jinnai Y, Tian Y, Fonseca R: (2020b). Neural architecture search using deep neural networks and monte carlo tree search. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 9983–9991

Wang T-T, Chu S-C, Hu C-C, Jia H-D, Pan J-S (2022a) Efficient network architecture search using hybrid optimizer. Entropy 24(5):656

Wang L, Xie L, Zhao K, Guo J, Tian Q (2022b) Regularized differentiable architecture search. IEEE Embedded Syst Lett 15:210

Wei H, Lee F, Hu C, Chen Q (2022a) Moo-dnas: Efficient neural network design via differentiable architecture search based on multi-objective optimization. IEEE Access 10:14195–14207

Wei C, Niu C, Tang Y, Wang Y, Hu H, Liang J: NPENAS. (2022b). Neural predictor guided evolution for neural architecture search. IEEE Transactions on Neural Networks and Learning Systems

Wen W, Liu H, Chen Y, Li H, Bender G, Kindermans P.-J: (2020) Neural predictor for neural architecture search. In: European Conference on Computer Vision, pp. 660–676. Springer
Wen Y-W, Peng S-H, Ting C-K (2021) Two-stage evolutionary neural architecture search for transfer learning. IEEE Trans Evol Comput 25(5):928–940
White C, Neiswanger W, Savani Y (2021a) BANANAS: Bayesian optimization with neural architectures for neural architecture search. Proc AAAI Conf Artif Intell 35:10293–10301
White C, Zela A, Ru R, Liu Y, Hutter F (2021b) How powerful are performance predictors in neural architecture search? Adv Neural Inform Process Syst 34:28454–28469
Wu B, Dai X, Zhang P, Wang Y, Sun F, Wu Y, Tian Y, Vajda P, Jia Y, Keutzer K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10734–10742
Wu Y, Liu A, Huang Z, Zhang S, Van Gool L (2021a) Neural architecture search as sparse supernet. Proc AAAI Conf Artif Intell 35:10379–10387
Wu M.-T, Lin H.-I, Tsai C.-W: (2021b). A training-free genetic neural architecture search. In: Proceedings of the 2021 ACM International Conference on Intelligent Computing and Its Emerging Applications, pp. 65–70
Xie S, Zheng H, Liu C, Lin L: Snas: (2018) stochastic neural architecture search. arXiv preprint arXiv:1812.09926
Xu Y, Wang Y, Han K, Tang Y, Jui S, Xu C, Xu C: ReNAS: (2021a) Relativistic evaluation of neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4411–4420
Xu J, Zhao L, Lin J, Gao R, Sun X, Yang H: (2021b). KNAS: green neural architecture search. In: International Conference on Machine Learning, pp. 11613–11625 PMLR
Xue Y, Wang Y, Liang J, Slowik A (2021) A self-adaptive mutation neural architecture search algorithm based on blocks. IEEE Computational Intell Magazine 16(3):67–78
Xue F, Qi Y, Xin J (2022) Rarts: An efficient first-order relaxed architecture search method. IEEE Access 10:65901–65912
Yaghoubi V, Kumru B (2024) Retrosynthetic life cycle assessment: a short perspective on the sustainability of integrating thermoplastics and artificial intelligence into composite systems. Adv Sustain Syst 10:2300543
Yaghoubi V, Cheng L, Van Paepegem W, Kersemans M (2022) An ensemble classifier for vibration-based quality monitoring. Mechan Syst Signal Process 165:108341
Yan Z, Dai X, Zhang P, Tian Y, Wu B, Feiszli M. (2021). FP-NAS: Fast probabilistic neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 15139–15148
Yang Y, Nam A, Nasr-Azadani M, Tung T: (2020a). Resource-aware pareto-optimal automated machine learning platform. In: 2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), pp. 1–6. IEEE
Yang Z, Wang Y, Chen X, Shi B, Xu C, Xu C, Tian Q, Xu C: (2020b). Cars: Continuous evolution for efficient neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1829–1838
Yang Z, Zhang S, Li R, Li C, Wang M, Wang D, Zhang M (2021) Efficient resource-aware convolutional neural architecture search for edge computing with pareto-bayesian optimization. Sensors 21(2):444
Ying C, Klein A, Christiansen E, Real E, Murphy K, Hutter F. (2019). NAS-bench-101: Towards reproducible neural architecture search. In: International Conference on Machine Learning, pp. 7105–7114 . PMLR
You S, Huang T, Yang M, Wang F, Qian C, Zhang C: Greedynas: (2020). Towards fast one-shot nas with greedy supernet. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1999–2008
Yu H, Peng H, Huang Y, Fu J, Du H, Wang L, Ling H (2022) Cyclic differentiable architecture search. IEEE Trans Pattern Anal Machine Intell 13:210
Zela A, Klein A, Falkner S, Hutter F: (2018). Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. arXiv preprint arXiv:1807.06906
Zela A, Siems J, Hutter F. (2020). NAS-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. arXiv preprint arXiv:2001.10422
Zhang M, Li H, Pan S, Liu T, Su S.W: (2020a). One-shot neural architecture search via novelty driven sampling. In: IJCAI, pp. 3188–3194
Zhang L.L, Yang Y, Jiang Y, Zhu W, Liu Y: (2020b). Fast hardware-aware neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 692–693

Zhang B, Chen H, Yang L, Chen C, Zhu Y, Doermann D. (2021a). Cp-nas: Child-parent neural architecture search for 1-bit cnns. In: Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, pp. 1033–1039

Zhang X, Hou P, Zhang X, Sun J: (2021b). Neural architecture search with random labels. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10907–10916

Zhang T, Lei C, Zhang Z, Meng X.-B, Chen C.P:(2021c). AS-NAS Adaptive scalable neural architecture search with reinforced evolutionary algorithm for deep learning. IEEE Transactions on Evolutionary Computation 25(5), 830–841

Zhang H, Jin Y, Hao K (2022a) Evolutionary search for complete neural network architectures with partial weight sharing. IEEE Trans Evol Comput 4:12

Zhang M, Pan S, Chang X, Su S, Hu J, Haffari G.R, Yang B: BaLeNAS: (2022b). Differentiable architecture search via the Bayesian learning rule. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11871–11880

Zhao Z, Zhang G.-e, Jiang M, Feng L, Tan K.C: (2020). Ednas: An efficient neural architecture design based on distribution estimation. In: 2020 2nd International Conference on Industrial Artificial Intelligence (IAI), pp. 1–6. IEEE

Zhao J, Lv W, Du B, Ye J, Sun L, Xiong G (2021a) Deep multi-task learning with flexible and compact architecture search. Int J Data Sci Anal 10:1–13

Zhao Y, Wang L, Tian Y, Fonseca R, Guo T. (2021b). Few-shot neural architecture search. In: International Conference on Machine Learning, pp. 12707–12718. PMLR

Zheng X, Ji R, Wang Q, Ye Q, Li Z, Tian Y, Tian Q. (2020). Rethinking performance estimation in neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11356–11365

Zheng X, Ji R, Chen Y, Wang Q, Zhang B, Chen J, Ye Q, Huang F, Tian Y (2021) MIGO-NAS: towards fast and generalizable neural architecture search. IEEE Trans Pattern Anal Machine Intell 43(9):2936–2952

Zhou H, Yang M, Wang J, Pan W: BayesNAS: (2019) A Bayesian approach for neural architecture search. In: International Conference on Machine Learning, pp. 7603–7613. PMLR

Zhu Y, Newsam S (2017) Densenet for dense flow. In: 2017 IEEE International Conference on Image Processing (ICIP), pp. 790–794. IEEE

Zhu H, An Z, Yang C, Xu K, Zhao E, Xu Y: (2019) Eena: efficient evolution of neural architecture. In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, pp. 0–0

Zimmer L, Lindauer M, Hutter F (2021) Auto-Pytorch: multi-fidelity metalearning for efficient and robust autodl. IEEE Trans Pattern Anal Machine Intell 43(9):3079–3090

Zoph B, Le QV (2016) Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578

Zoph B, Vasudevan V, Shlens J, Le QV (2018) Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8697–8710