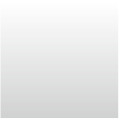
[](http://crossmark.crossref.org/dialog/?doi=10.1007/s10462-024-10726-1&domain=pdf)

**Automated machine learning: past, present and future**

**Mitra Baratchi1 · Can Wang1 · Stefen Limmer2 · Jan N. van Rijn1 · Holger Hoos1,3**

**·**

**Thomas Bäck1 · Markus Olhofer2**

Accepted: 10 February 2024 / Published online: 18 April 2024 © The Author(s) 2024

**Abstract**

Automated machine learning (AutoML) is a young research area aiming at making

high-performance machine learning techniques accessible to a broad set of users. This is achieved by identifying all design choices in creating a machine-learning model and addressing them automatically to generate performance-optimised models. In this article, we provide an extensive overview of the past and present, as well as future perspectives of AutoML. First, we introduce the concept of AutoML, formally defne the problems it aims to solve and describe the three components underlying AutoML approaches: the search space, search strategy and performance evaluation. Next, we discuss hyperparam- eter optimisation (HPO) techniques commonly used in AutoML systems design, followed by providing an overview of the neural architecture search, a particular case of AutoML for automatically generating deep learning models. We further review and compare available AutoML systems. Finally, we provide a list of open challenges and future research direc- tions. Overall, we ofer a comprehensive overview for researchers and practitioners in the area of machine learning and provide a basis for further developments in AutoML.

**Keywords** Automated machine learning · Neural architecture search · Hyperparameter optimisation · Search space · Search strategy

**1 Introduction**

From diagnosing diseases to translating languages or forecasting the weather, machine learning has changed our lives in many ways by giving computer systems new powers. The successes achieved by machine learning systems, however, highly rely on experienced machine learning experts who design specifc machine learning pipelines. These pipe- lines are typically composed of various components, such as data preprocessors, feature extractors and training algorithms. In order to create high-performing models using these pipelines, machine learning experts need to deal with the complex task of selecting suit- able components among many available options, as well as fnding performance-optimis- ing hyperparameter settings for each component. The many choices that have to be made in this context typically interact in ways even seasoned experts fnd difcult to predict; achieving good results, therefore, relies on experience, intuition and substantial amounts of experimentation. Furthermore, beyond the design of pipelines for ofine modelling, there

Extended author information available on the last page of the article

a 灬

is a need for automatic adaptation and generation of new models in autonomous systems with machine-learning components. Such systems should be able to automatically deal with changes in their environment, as well as with modifed or new system components, without the need for manually updating the underlying machine-learning pipelines.

Automated machine learning, in short AutoML, is a rapidly growing research area that strives to make machine learning available to non-machine learning experts by automating the steps that needs to be taken in creating high-performance machine-learning pipelines for given use cases. The topic of AutoML is strongly tied to the broader theme of auto- mating data science (De Bie et al. [2022](#bookmark1)), which aims at automating the full spectrum of tasks performed in the context of deriving insight from raw data. Generally, automating data science considers automation of four key steps: (1) data exploration to understand and interpret data, (2) data engineering to create a dataset ready to be fed into a machine learn- ing pipeline by cleaning and removing outliers, data augmentation and feature engineer- ing (3) model building by algorithm selection and hyperparameter optimisation, and (4) exploitation of data in decision-making. Among these, current AutoML systems mainly focus on automated model building for a given dataset. Therefore, in this survey, we mainly cover research in Step 3. More concretely, AutoML can be considered as the process of automatically selecting at least one of the following: (a) the structure of a machine learn- ing pipeline, (b) steps of the pipeline and (c) hyperparameters of steps of the pipeline. It should be mentioned that there are automated systems, such as the automated statistician (Steinruecken et al. [2019](#bookmark2)), that have focused on automating other data science tasks (e.g., automatically creating human-readable reports from raw data). However, these fall outside the scope of this article.

Foundations of modern AutoML systems were laid as early as 1976, when John Rice introduced the *algorithm selection problem* (Rice [1976](#bookmark3)). Earlier work that considered auto- mating steps of machine learning pipelines considered algorithm selection (Ali and Smith [2006](#bookmark4); Brazdil and Soares [2000](#bookmark5)) or hyperparameter optimisation (Bengio [2000](#bookmark6); Bergstra and Bengio [2012](#bookmark7)) separately. The real power of AutoML was unlocked through the defni- tion of the combined algorithm selection and hyperparameter optimisation problem,

and the subsequent demonstration that this problem can be solved efectively, with the development of Auto-WEKA (Thornton et al. [2013](#bookmark8)), which used this concept to select a model from a search space consisting of classic machine learning algorithms and their hyperparameters. More recently, AutoML research has also placed a strong focus on neural architecture search (NAS) (Liu et al. [2018a](#bookmark9)), an area that specifcally concerns the design of one class of machine-learning models, i.e., neural networks, by automatically designing them.

There has recently been broad and substantial impact of AutoML research within aca- demia and industry, leading to a steep increase in research activities in the area. Results of ongoing academic research in AutoML include systems such as auto-sklearn (Feurer et al. [2015](#bookmark10)), TPOT (Olson et al. [2016a](#bookmark11)) and AutoKeras (Jin et al. [2019](#bookmark12)). Inspired by these works, many industrial AutoML platforms have also be designed for large-scale deploy- ments, such as Google Cloud AutoML (Bisong [2019](#bookmark13)), Amazon SageMaker Autopilot (Das et al. [2020](#bookmark14)) and Oracle AutoML (Yakovlev et al. [2020](#bookmark15)). Thanks to these systems, high- performance machine learning models and pipelines can now be constructed with minimal efort from users.

Overall, we aim to provide a survey targeting novice researchers and practitioners on AutoML with knowledge on Machine Learning. We provide extensive background to understand the approach taken for research in both AutoML and NAS systems. Specif- cally, we

a 灬

• provide comprehensive background information needed to understand the advanced approaches taken in AutoML research;

• cover the topic of hyperparameter optimisation, a core technique used in most AutoML systems;

• provide an overview of the available state-of-the-art AutoML systems for classic machine learning and deep neural networks; and

• present and discuss a list of open challenges and future research directions.

Following previous work (see, e.g., Hutter et al. [2019](#bookmark16); Elsken et al. [2019b](#bookmark17)), we focus our discussion on three key components found in all AutoML systems: (1) the *search space* that incorporates all design choices in a given machine-learning pipeline (i.e., all algo- rithms in a pipeline, their hyperparameters, or architectural hyperparameters of a neural network; (2) the *search strategy* used to fnd good candidate solutions within the search space, i.e., performance-optimising instantiations of the design choices; and (3) the *perfor- mance evaluation (or estimation) technique* used to automatically assess the performance of candidate solutions on a fraction of a given dataset.

Using these components as a structuring principle, we provide a comprehensive over- view of the most relevant past and present work in AutoML. In this, we do not aim for completeness (which, in light of the large number of publications on the topic, would be hardly feasible), but rather focus on what can reasonably be considered the most impor- tant and impactful approaches. After introducing defnitions of the problems encountered and solved in the area of AutoML, we discuss the three key components in more detail, followed by brief general introductions to two additional topics of broad importance to AutoML research: benchmarks and meta-learning techniques.

We review the techniques used for both AutoML and NAS systems. We start by dis- cussing background topics for both categories of work by introducing the AutoML prob- lem defnitions, the search space and the search strategy. Next, we cover hyperparameter optimisation techniques used for tuning machine learning algorithms defned with a fxed search space and general performance estimation approaches. Following this background material, we present three categories of AutoML approaches: hyperparameter optimisation techniques, which assume that the only design choices to be made are hyperparameters of a given machine-learning pipeline or system; NAS techniques, which focus on design choices related to the architecture of deep neural networks; and broad-spectrum AutoML systems that consider design spaces containing broad classes of machine-learning algo- rithms, but are not specifcally focused on deep neural networks. Finally, we discuss sev- eral directions for future research in the area of AutoML that we believe are particularly promising.

We note that there are several earlier survey papers on AutoML (Yao et al. [2018](#bookmark18); ElShawi et al. [2019](#bookmark19); Guyon et al. [2016](#bookmark20)) as well as a book (Hutter et al. [2019](#bookmark16)) and we iden- tifed recent surveys that were published while our article presented here was under review (Karmaker et al. [2021](#bookmark21); Barbudo et al. [2023](#bookmark22); Del Valle et al. [2023](#bookmark23)). Our goal in writing a new survey paper was to complement existing work by covering newer methods, uncovered topics and provide necessary background information for researchers to improve and design AutoML systems. Since AutoML is a fast-moving research area, many recent approaches, notably in NAS and broad-spectrum AutoML systems, are not covered in these earlier surveys. There is a survey by Waring et al. ([2020](#bookmark24)), focusing mainly on the application of AutoML in the healthcare domain by reviewing of-the-shelf techniques and their possible use by medical professionals. Karmaker et al. ([2021](#bookmark21)) survey AutoML systems by classify- ing them based on the level of autonomy. Their survey does not discuss technical details of

a 灬

AutoML methods and techniques, but rather provides a practical guide for users to select an AutoML system. More recently, Barbudo et al. ([2023](#bookmark22)) published a survey on AutoML research with a focus on providing a uniform taxonomy of terms in AutoML, identifying the breadth of ML tasks covered by AutoML and research trends. This review does not cover fundamental topics in detail (HPO, NAS), nor does it provide details of the respec- tive methods. In contrast, we discuss a broad range of AutoML methods in more detail to provide the necessary background for AutoML researchers interested in developing these techniques further. He et al. ([2021](#bookmark25)), Salehin et al. ([2024](#bookmark26)) provide AutoML surveys mainly focused on NAS. In contrast, we cover hyperparameter optimisation, NAS and broad-spec- trum AutoML systems, emphasising connections between those sub-areas of AutoML. The survey article by Salehin et al. ([2024](#bookmark26)) provides a more general overview of NAS meth- ods, along with introductory information suitable for readers with limited machine learning knowledge (e.g., diferent tasks on machine learning pipelines). Compared to that survey, ours discusses relevant methods from the literature in more detail. We also provide a more detailed background on important topics, such as important benchmarks and libraries. The short survey of Zöller and Huber ([2021](#bookmark27)) mainly provides background information to under- stand their benchmarking task of a number of available AutoML systems. It covers hyper- parameter optimisation techniques but does not provide much detail about the underlying search spaces or about NAS systems. The survey by Escalante ([2021](#bookmark28)), as the name sug- gests, mainly provides a brief overview of the state of AutoML research without providing details on any of the techniques. Del Valle et al. ([2023](#bookmark23)) focus on reviewing AutoML solu- tions specifcally designed for multi-label and multi-target regression problems, while we provide a comprehensive and detailed overview of AutoML systems for a broad class of machine learning problems.

There are several other papers that focus purely on NAS (Elsken et al. [2019b](#bookmark17); Wistuba et al. [2019](#bookmark29)), mainly covering early research and the main components of NAS. Compared to the more recent survey by Ren et al. ([2022](#bookmark30)), our discussion of NAS is supported by extensive background information (e.g., hyperparameter optimisation approaches) to pro- vide a deeper understanding of the underlying topics. There are also a number of dedicated survey articles on topics such as evolutionary NAS (Liu et al. [2021](#bookmark31)) and reinforcement learning for NAS (Jaafra et al. [2019](#bookmark32)). While we cover important work on these topics in our survey, the scope of our work is diferent, as we aim to cover the broader research area of AutoML; thus, we consider these surveys as complementary to our work.

Overall, we aim to provide a useful entry point into the area of AutoML to researchers and practitioners new to the area, as well as a basis for further discussion and development to AutoML experts.

**2 Background**

In this section, we mainly focus on introducing the concepts and techniques that are referred to in the context of AutoML later in this paper.

**2.1 Problem defnitions**

Machine learning *models* are generated by machine learning *algorithms* by internally optimising a number of *parameters*. Machine learning algorithm typically have several *hyperparameters* that need to be externally set by users and control various aspects of

a 灬

the training process; their values usually do not change during training. For example, a neural network model can have millions of parameters, such as the weights and bias values. The learning algorithm that determines these parameters has several hyperpa- rameters, such as the learning rate, the optimisation schedule and how to perform data augmentation. Next to the choice of the algorithm, the choice of hyperparameters has a strong infuence on the fnal performance of the model and the efcacy of the learning process.

AutoML aims to produce the best model for a dataset of interest, given a large set of machine learning algorithms and their hyperparameter spaces. There are two major goals in fnding these models: (1) to obtain a model with the highest performance for a given machine learning task (e.g., classifcation or regression accuracy) on the dataset, and (2) to ensure that good “out of sample generalisation” of the model on unseen data is achieved. Hyperparameter optimisation (HPO) strategies facilitate reaching the frst goal, while towards the second, techniques such as cross-validation are used to compare models obtained using diferent hyperparameter confgurations.

There are three closely related AutoML problems, namely, (i) *algorithm (or model) selection*, which deals with multiple machine learning algorithms with fxed hyperparame- ter settings; (ii) *hyperparameter optimisation*, which deals with a single algorithm; and (iii) *combined algorithm selection and hyperparameter optimisation* (Thornton et al. [2013](#bookmark8)). To formally defne these problems, we use the following notation:

• **횲** = Λ1 ×Λ2 × … ×Λn denotes the confguration space of algorithm *A* induced by *n* hyperparameters, where Λ*i* denotes the domain of the *i*th hyperparameter.

• A = {A(1), … , A(R) } denotes a set of learning algorithms. The hyperparameters of each algorithm A(j) are defned over the confguration space Λ(j).

• D = {(**x**1, y1 ), … , (**x**n, yn )} denotes the training dataset, where **퐱퐢** and yi represent the values of features and target variables, respectively. To perform *k*-fold cross-validation,

D is further split into *k* equally-sized partitions composed of training (Dt(a)in , … , Dt()in)

and validation sets (Dv(id , … , Dv(id) such that Dt( = D∕ Dv()lid. Note that there is

usually also a test set, but this is considered to be completely external to the selection or optimisation procedure.

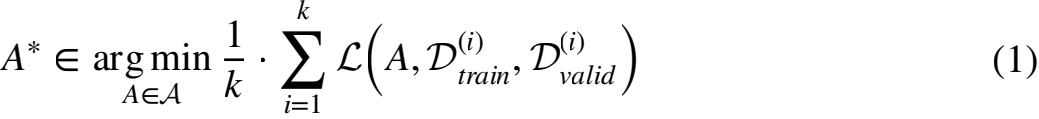
• Generalisation performance is evaluated by running training algorithm *A* on Dt( and

evaluating the resulting model on Dv()lid by measuring the loss using a performance

metric, such as classifcation accuracy, denoted by L(A Dt( Dv()lid ). This evaluation

approach corresponds to *k*-fold cross-validation, which has also been the method of choice in the AutoML literature thus far. In principle, however, other validation tech- niques can be used.

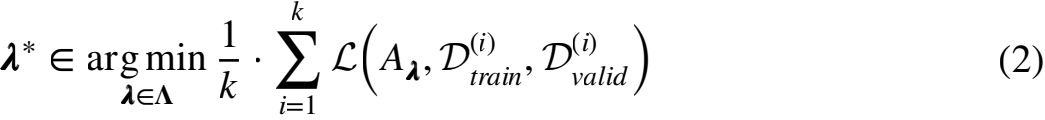
**Defnition 1** (Algorithm (or Model) Selection Problem) Given a set A of learning algo- rithms (associated with specifc classes of models), the algorithm selection problem can then be defned as fnding A\* ∈ A that yields the model with the best performance on the validation set (Thornton et al. [2013](#bookmark8)):



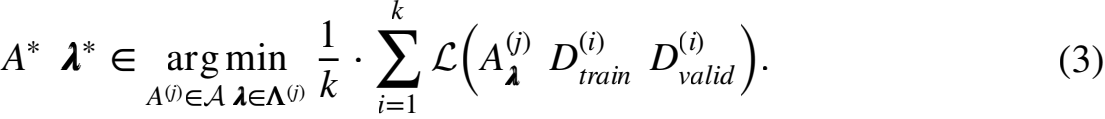
a 灬

We note that the A ∈ A can also correspond to the same learning algorithm with difer- ent hyperparameter settings or even just diferent sequences of random numbers; equiva- lently, the problem can also be formalised for sets of models (irrespectively how these were obtained) rather than sets of algorithms.

**Defnition 2** (Hyperparameter optimisation problem) Given a learning algorithm *A* with hyperparameters Λ, and letting A*휆* denote that *A* with the hyperparameter vector *휆* ∈ **횲**, the HPO problem can be defned as fnding *휆*\* that yields the model with the best performance on the validation set (Thornton et al. [2013](#bookmark8)):



**Defnition 3** (Combined Algorithm Selection and Hyperparameter Optimisation (CASH) problem) Given a set of learning algorithms A, where the hyperparameters of each algo- rithm A(j) ∈ A have the confguration space Λ(j). The CASH problem can be defned as fnding the algorithm A\* and its hyperparameter confguration *휆*\* that yields the model with the best performance on the validation set (Thornton et al. [2013](#bookmark8)):



Dealing with the algorithm selection problem requires fnding an approach for setting the hyperparameters of the algorithms. For instance, this can be done by using the default hyperparameter values or optimising the hyperparameters of each algorithm separately before performing algorithm selection. This approach is less relevant for modern AutoML research and thus will not be covered in this survey paper. The HPO problem is still fre- quently tackled in NAS systems, where only the hyperparameters of one type of learning algorithm are optimised.

HPO procedures are also used within approaches for solving the CASH problem. Sec- tion [3](#bookmark37) covers HPO strategies, and in Sect. [4](#bookmark38), we will discuss how these techniques are used in NAS. The CASH problem is the most general AutoML problem and typically consid- ers a much more diverse search space based on diferent machine learning algorithms and other elements of a machine learning pipeline, such as preprocessors. In Sect. [5](#bookmark39), we will review AutoML systems that are defned based on the CASH problem.

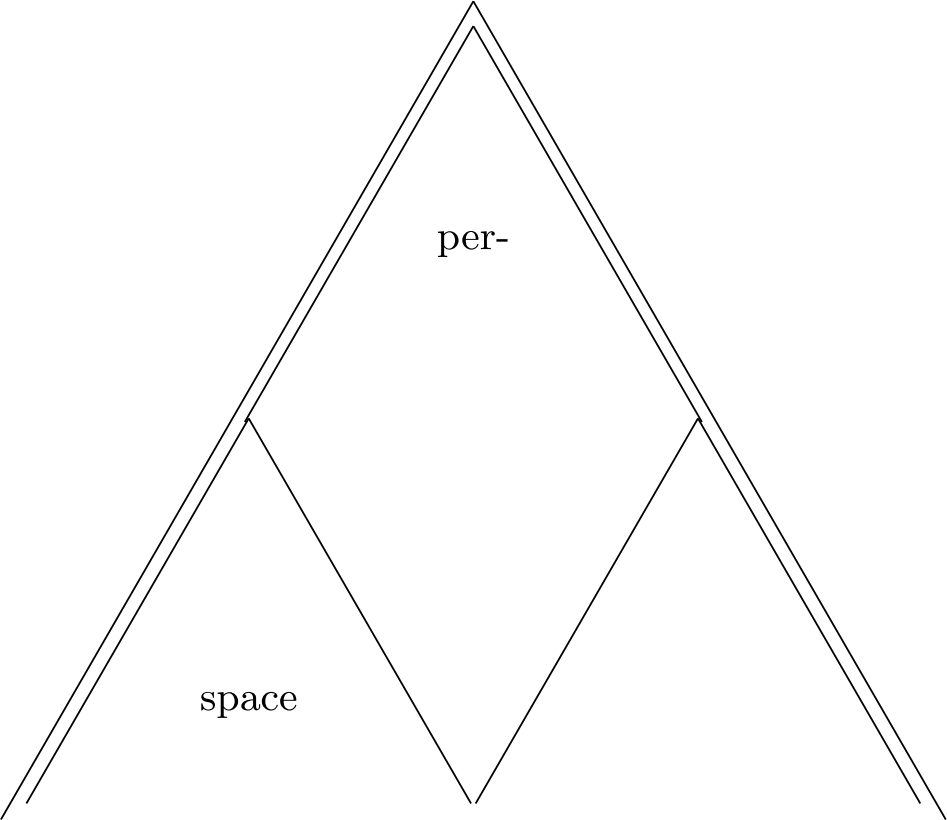
**2.2 Components of AutoML**

As outlined in Fig. [1](#bookmark41), AutoML approaches can be defned in terms of three key compo- nents: (i) *the search space*, (ii) *the search strategy (or algorithm)*, and (iii) *performance evaluation (or estimation)* (see, e.g., Hutter et al. [2019](#bookmark16); Elsken et al. [2019b](#bookmark17)). In the follow- ing, we explain these components in more detail.

**2.2.1 Search space**

The search space defnes all design choices in the machine learning solution space (pipe- line) that need to be made. These include the learning algorithms (and hence model classes) that can be selected and their hyperparameters (including, e.g., architectural

a 灬

form ance evaluation

Auto ML

search search

algorithm



**Fig. 1** Three important components of AutoML

hyperparameters of a neural network). By treating the choice of the algorithm as a hyper- parameter, the CASH problem can, in principle, be tackled using HPO techniques.

There are various types of hyperparameters, including categorical and numerical hyper- parameters. Categorical hyperparameters can take a discrete set of values. For example, support vector machines can use linear, Gaussian or sigmoid kernels. There is no notion of order among these values. On the other hand, numerical hyperparameters are defned on the numeric domain, e.g., the set of all positive integers or the set of all real values. These hyperparameters typically have a range predefned by the AutoML engineer. Examples are the learning rate of a neural network or the kernel width of the aforementioned Gaussian kernel. Hyperparameter optimisation methods and AutoML systems implicitly deal with these types of hyperparameters in a diferent way. While early research explicitly men- tioned how these methods handle the various types of hyperparameters (see, e.g., Berg- stra et al. [2011](#bookmark42); Hutter et al. [2011](#bookmark43)), in current work this is often handled implicitly, for example by using numerical encodings of categorical hyperparameters [see, e.g., auto- sklearn (Feurer et al. [2015](#bookmark10))].

There can be dependencies among hyperparameters in the search space as well. Some hyperparameters are only relevant if a certain other hyperparameter is set to a given value. For example, the kernel width hyperparameter is only relevant for certain types of kernels (e.g., Gaussian, sigmoid). Similarly, in neural networks, the number of layers determines the relevance of other hyperparameters for each layer. As men- tioned previously, the choice of the learning algorithm can be modelled using a cat- egorical hyperparameter, whose value determines which group of hyperparameters specifc to the various learning algorithm will be activated. These hyperparameters are referred to as conditional hyperparameters and lead to a tree-structured search space where some hyperparameters that appear in leaf nodes are only valid when their parent nodes take certain values. Therefore, when using HPO techniques, we have to consider that these hyperparameters are essentially diferent from the other hyperparameters. In some cases, it might be plausible to assume that the diference between small values of

a 灬

a given numerical hyperparameter has a higher impact on performance than the same diference between larger values. To make this more concrete, the diference between a learning rate of 0.01 and 0.02 is typically more signifcant than the diference between a learning rate of 10 and 10.01. When, for example, a random search operator samples uniformly over such a range, all values will be chosen with equal probability, neglecting this important aspect. In order to address this issue, a transformation can be applied to the hyperparameter range, such that sampling takes place in this transformed range. A common choice for this is the log-transformation operator. When applying the log trans- formation on a hyperparameter range, values are sampled uniformly within this logarith- mic space before being transformed back to the original space and subsequently being used to instantiate the confguration. This ensures that higher emphasis is being placed on smaller values. Snoek et al. ([2014](#bookmark44)) describe a method that automatically selects a fexible transformation to sample hyperparameter values from.

**2.2.2 Search algorithm**

The search algorithm specifes how the search space is explored in order to optimise the performance of the resulting model. Early approaches considered search spaces that were small enough to permit exhaustive enumeration. However, realistic applications of AutoML typically give rise to infnite search spaces induced by dozens of hyper- parameters. While, in principle, these can still be explored using random search or grid search (after discretisation), in practice, far better results can usually be obtained using more sophisticated search strategies. In Sects. [3.2](#bookmark45) and [4.2](#bookmark46), we will discuss various search algorithms that are proposed with the aim of optimising the hyperparameters of machine-learning algorithms and pipelines.

**2.2.3 Performance evaluation**

The search algorithm iteratively provides a series of candidate solutions (i.e., learning algorithms and/or hyperparameter settings) that could be used to solve the given machine- learning task. In order to determine which of these is the best, their performance needs to be evaluated. In addition, the fnal result (i.e., model) produced by an AutoML system also needs to be evaluated. It is important to ensure that the data used for this latter outer valida- tion is disjoint from that used for the former inner validation since otherwise, the degree to which the model generalises to new data is likely to be overestimated. Therefore, as men- tioned in Sect. [2.1](#bookmark34), standard cross-validation, based on splits into training and testing data, does not sufce for the general purpose of AutoML. This has been taken into account in defning the L function in the defnitions of Sect. [2.1](#bookmark34).

The most commonly employed method for dealing with this situation splits the data D that is used by the AutoML system to produce a fnal model into a training set (Dtrain ) and a validation set (D alid ). Any model considered by the search algorithm underlying the AutoML system is then trained on the training set and evaluated on the validation set, as mentioned earlier in Sect. [2.1](#bookmark34). This is typically done using a *k*-fold (inner) cross- validation scheme, by splitting D into *k* equally-sized partitions composed of training

(Dt(a)in , … , Dt()in ) and validation sets (Dv(id , … , Dv(id ) such that Dt( = D∕ Dv()lid.

a 灬

This overall procedure is sometimes referred to as nested evaluation or nested cross- validation (Varma and Simon [2006](#bookmark48)). In the remainder of this survey, we focus on the

inner validation, which forms a key component of the AutoML system.

**2.3 Benchmarks**

In (automated) machine learning, it is quite common to develop a method that does not only solve a single problem but is meant to solve a wide range of problems. For exam- ple, the methods described in Sect. [3](#bookmark37) all address HPO, which can be applied to many datasets. In order to justify claims about the performances of these systems, researchers and practitioners rely on benchmarks. Having good benchmarks is important, as they form the basis for a fair and accurate assessment of the state of the art, which in turn provides the basis for determining which approaches can readily be deployed in prac- tice and for deciding which methods receive attention in terms of further research. In Sects. [3](#bookmark37) and [4](#bookmark38), we will give an overview of important and widely used benchmarks for HPO and NAS, respectively.

**2.4 Meta‑learning**

The main goal of meta-learning is to observe how various machine-learning approaches perform on a range of diferent datasets and to use the meta-data collected from these observations to learn new tasks faster (Vanschoren [2018](#bookmark50); Brazdil et al. [2022](#bookmark51)).

In the context of AutoML, meta-learning is typically used for warm-starting the search process by recommending good initial hyperparameter settings [as used, e.g., in Auto-sklearn (Feurer et al. [2022](#bookmark52)) and OBOE (Yang et al. [2019](#bookmark53))]. There are also forms of meta-learning that immediately warm-start the parameters of a model [i.e., the weights of a neural network; see, e.g., MAML (Finn et al. [2017](#bookmark54)) and Reptile (Nichol et al. [2018](#bookmark55))]. As hyperparameters control the learning process, and parameters are the result of the learning process, there are usually far more parameters than hyperparame- ters, which results in specifc challenges for the latter type of meta-learning approaches. Furthermore, as these challenges are not specifcally linked to AutoML, we consider those approaches as falling outside of the scope of this survey and refer interested read- ers to existing literature (Hospedales et al. [2022](#bookmark56); Huisman et al. [2021](#bookmark57)).

In the following, we briefy describe how meta-learning can be used in automated machine-learning systems. It generally builds upon the algorithm selection framework by Rice ([1976](#bookmark3)). When applied to AutoML systems, meta-learning is composed of two important phases: (i) meta-feature extraction and (ii) generation and testing of a meta- model. Meta-features describe the properties of a given dataset and can be used to assess the similarities between datasets; they thus form the basis for knowledge transfer between similar datasets. Diferent meta-features have been proposed in the literature (e.g., simple, statistical, information-theoretic, model-based, and landmarking. For an extensive overview of meta-features used in literature, the reader is referred to Brazdil et al. ([2022](#bookmark51)), Pinto et al. ([2016](#bookmark58)), Rivolli et al. ([2022](#bookmark59)). Next to warm-starting, i.e., the setting of hyperparameters based on what is known to work well on similar datasets, meta-features can also be used to estimate the performance of algorithms through meta- models. Meta-models can be used to learn the relationship between meta-features and the performance of learning algorithms (based on, e.g., accuracy, training time, and

a 灬

learning curve). The predictions made by the meta-model can be used to warm-start the optimisation techniques. This allows judging whether an algorithm and/or hyperparame- ter setting is a good candidate to be evaluated in the context of a given search procedure.

**3 Hyperparameter optimisation**

In some cases, we are already committed to using a specifc machine-learning algorithm (e.g., a support vector machine) because of external constraints, and the goal is to optimise the hyperparameters of this specifc algorithm. In this case, we deal with a hyperparameter optimisation (HPO) problem. In this section, we will describe solutions to the HPO prob- lem, as defned by Defnition [2](#bookmark35) of Sect. [2](#bookmark33). As described in Sect. [2.2](#bookmark40), HPO procedures can be described in terms of a search space, a search strategy and an evaluation mechanism, which we address in Sects. [3.1](#bookmark60), [3.2](#bookmark45) and [3.3](#bookmark61), respectively. In Sect. [3.4](#bookmark62), we describe sev- eral popular libraries for HPO, and in Sect. [3.5](#bookmark63), we give an overview of commonly used benchmarks.

**3.1 Search spaces**

The search space of HPO typically consists of all hyperparameters that are relevant to be optimised (numerical as well as categorical, and in many cases conditional). To fnd a good hyperparameter confguration, the search space needs to contain at least some regions where such good confgurations can be found. When the search space is too large, the suc- cess of the optimisation procedure highly depends on the quality of the search algorithm and might take more time.

HPO frameworks often take as input a search space, requiring the user to bring in exper- tise on relevant and important hyperparameters and how to search for good values. Various research lines went in the direction of generating this knowledge (Snoek et al. [2014](#bookmark44); Hutter et al. [2014](#bookmark64); van Rijn and Hutter [2018](#bookmark65)), whereas other works aim to automatically construct a good search space based on historical data (Hoos [2012](#bookmark66); Wistuba et al. [2015a](#bookmark67); Pfsterer et al. [2021](#bookmark68); Perrone et al. [2019](#bookmark69)). In this chapter, we will review what we consider the most impactful work in this direction.

We note that relatively little is known about the search landscapes in which hyper- parameter optimisation and other AutoML tasks take place. Recently, Pushak and Hoos ([2022](#bookmark70)) have found evidence that most AutoML loss landscapes only have a small number of strongly interacting hyperparameters, giving rise to a convex, unimodal structure. The idea of using efciently computable proxies for an expensive objective function is underly- ing several classes of HPO methods (notably, Bayesian optimisation) and has also been explored in engineering optimisation (Long et al. [2022](#bookmark71)) and image processing (Tseng et al. [2019](#bookmark72)). Furthermore, exploratory landscape analysis has recently been used for identifying efciently computable proxy functions for HPO (Schneider et al. [2022](#bookmark73)). While all of those eforts aim at improving our understanding of HPO (and more general: AutoML) land- scape properties, they have yet to produce major practical impact in terms of guiding the design of search spaces or search strategies.

Important design criteria for the search spaces underlying HPO are the set of hyper- parameters to be optimised, the dependencies between these hyperparameters (i.e., conditional hyperparameters), the ranges of values to be considered for numerical

a 灬

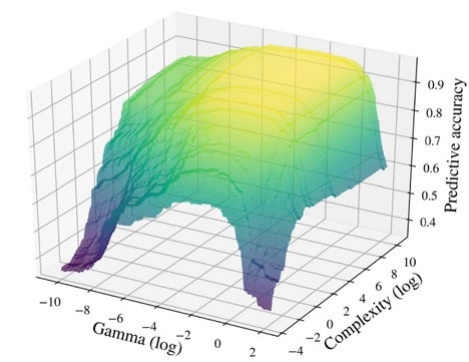
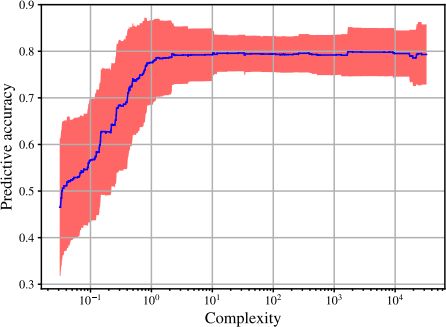
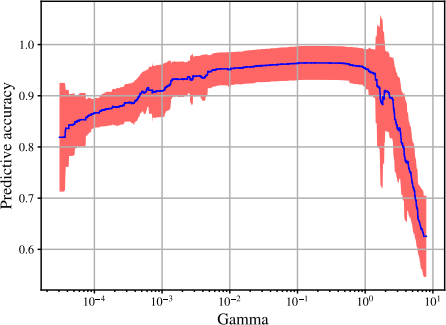
hyperparameters, and potential transformations. In Sects. [3.1.1](#bookmark74) and [3.1.2](#bookmark75) we will describe functional ANOVA and ablation analysis, respectively, for determining the importance of hyperparameters on a single dataset, as a post-hoc analysis after HPO. Section [3.1.3](#bookmark76) describes works that utilise this knowledge across datasets, which is an important step towards building confguration spaces. Section [3.1.4](#bookmark77) describes a technique to apply the transformation of a hyperparameter search space. Section [3.1.5](#bookmark78) outlines various philoso- phies behind designing a search space. Finally, Sect. [3.1.6](#bookmark79) covers several methods that bring all these components together and automatically design a search space.

**3.1.1 Hyperparameter importance through functional ANOVA**

When designing a search space, one important consideration is to determine which hyper- parameters should be considered and which can be safely ignored because they hardly infuence performance. Functional ANOVA is a statistical method commonly used for sen- sitivity analysis that can determine the contribution of a single hyperparameter (or combi- nation of hyperparameters) to the overall variance of the result (Sobol [1993](#bookmark80)). Functional ANOVA works based on the *marginal* for each subset of hyperparameters and determines their general infuence on the performance on a given dataset. It works on the premise that hyperparameters with a high variance in the marginal are important to optimise, and hyper- parameters with a low variance in the marginal are less important to optimise.

The marginal determines, for a given subset of hyperparameters and a given set of val- ues for these hyperparameters, what the *marginal performance* is. Informally, for a given hyperparameter, the marginal performance is the performance per value, averaging over all possible values of all other hyperparameters. This can be easily visualised for a small num- ber of hyperparameters. Figure [2](#bookmark81) shows examples of marginals for two hyperparameters of a support vector machine (left, middle), and also for the combination of these hyperparam- eters (right). The vertical axes present the average expected performance of a given value when averaging over all possible values of all other hyperparameters. As such, this is a much stronger statement about the performance than just varying a single hyperparameter over a given range, but this comes at the cost of additional efort of computing this.

Averaging over all possible values of all performance values of all other hyperparam- eters seems practically infeasible, as many search spaces are prohibitively large or even infnite. However, Hutter et al. ([2014](#bookmark64)) showed how the marginal performance values for machine learning hyperparameters could be calculated using surrogate models (Eggensper- ger et al. [2015](#bookmark82)). This requires data on the performance of a large number of confgurations,



**Fig. 2** Marginals of two hyperparameters (complexity and gamma) of a support vector machine, and the combination of both. The horizontal axes represent the value of a given hyperparameter, and the vertical axis represents the marginal performance, in terms of predictive accuracy (van Rijn and Hutter [2018](#bookmark65))

a 灬

which can be obtained by evaluating these confgurations on the current dataset. HPO methods typically perform various performance evaluation on various parts of the search space, and come with such performance evaluations as a side-product. As such, methods to determine hyperparameter importance can often be used in post-hoc analysis, after an HPO procedure has been used to optimise the hyperparameters on a given dataset. The surrogate model can then be trained to map hyperparameter confgurations to performance estimates. After it has been trained, it can predict the performance of previously unseen hyperpa- rameter confgurations. Note that these surrogate models might not be fully accurate, and some efort should be devoted to ensuring that the surrogate model is sufciently accurate. Hutter et al. ([2014](#bookmark64)) show specifcally how tree-based surrogate models can be used to ef- ciently calculate marginals for hyperparameters, even for machine learning algorithms with infnitely large confguration spaces.

Once we have determined the marginal for each hyperparameter and combination of hyperparameters, we can utilise functional analysis of variance (ANOVA) to determine the importance of these. Functional ANOVA calculates the variance of the marginal of a given subset of hyperparameters and relates it to the variance of the other marginals. This indicates how much a given subset of hyperparameters contributes to the variance in performance when taking into consideration all hyperparameters. Hyperparameters that yield marginals with high variance are deemed important and should likely be optimised. Hyperparameters that yield marginals with a low variance are deemed less important and could likely be skipped in the optimisation process. We note that since hyperparameter importance is determined retroactively after the optimisation process has been completed, this knowledge will typically be utilised for future optimisation processes. Section [3.1.3](#bookmark76) describes how several works have done this.

**3.1.2 Hyperparameter importance through ablation analysis**

Ablation analysis aims to determine the importance of hyperparameters by removing (ablating) them from the confguration space in order of importance (Fawcett and Hoos [2016](#bookmark83)). Ablation analysis is performed on the basis of two hyperparameter confgurations; in the context of HPO, it typically utilises the default confguration and an optimised con- fguration. Similar to functional ANOVA, it can therefore be used as a post-hoc analysis technique after a HPO method has determined the optimised confguration for a given dataset.

For each hyperparameter, there is a (possible) diference between the value in the default confguration and the optimised confguration. Ablation analysis starts with one of the confgurations (e.g., the optimised confguration) and determines which hyperparameter would impact performance most if it were to be set to the value of the other confguration (e.g., the default confguration). It does so by evaluating all the resulting confgurations that can be reached by changing a single hyperparameter from the current confguration to the default value. It selects the hyperparameter and subsequent confguration that impacts the performance most and continues the procedure from that confguration by determin- ing which of the other hyperparameters would impact performance most by setting it to the default value. This is iterated until all hyperparameters have been set sequentially to the default value. In this way, a so-called *ablation path* is computed, i.e., a list of hyperpa- rameters sorted by their impact on performance. This ablation path and the corresponding changes in performance can be used to assess hyperparameter importance. We note that

a 灬

ablation is a greedy procedure, and the impact of each parameter can sensitively depend on the order in which other parameter values have been changed.

In order to perform ablation analysis, we typically use the default confguration and an optimised confguration. The optimised confguration can be determined by running an HPO procedure. Note that on top of this HPO procedure, the ablation analysis also evalu- ates additional confgurations along the ablation path. This causes additional computational costs. Biedenkapp et al. ([2017](#bookmark84)) address this problem by introducing surrogate models, that (similar to the surrogate models used by functional ANOVA) are trained on a limited set of performance evaluations and that can estimate the performance of previously unseen confgurations. In principle, the surrogate model can be trained on performance evaluations that have been obtained during HPO. This alleviates the need for additional costly perfor- mance evaluations, although the surrogate model might be somewhat inaccurate. Similar to functional ANOVA, ablation analysis is a post-hoc method that is employed after the com- pletion of the optimisation process. The knowledge gained from ablation can be utilised in future runs of HPO methods, as detailed in the following section.

**3.1.3 Hyperparameter importance across datasets**

While functional ANOVA (Sect. [3.1.1](#bookmark74)) and ablation analysis (Sect. [3.1.2](#bookmark75)) are designed as post-hoc methods that are typically utilised to determine the importance of hyperparam- eters on a given dataset after the HPO procedure has been run, we are often interested in patterns that generalise across datasets. In other words, given an algorithm, we would like to know what are generally the most important hyperparameters, regardless of the dataset.

van Rijn and Hutter ([2018](#bookmark65)) combined the functional ANOVA framework with the experimental results that are available in OpenML (Vanschoren et al. [2013](#bookmark85)). They apply functional ANOVA on each dataset in OpenML100 (Bischl et al. [2021](#bookmark86)), using the perfor- mance results of confgurations that were already available in OpenML, and determine for various learning algorithms (i.e., random forests, AdaBoost and support vector machines) the importance of the hyperparameters. This work showed that, for these algorithms, in many cases the same hyperparameters are important. As such, knowledge of hyperparam- eter importance on a collection of datasets is likely to transfer to other datasets. This work was later also extended to neural networks. Sharma et al. ([2019](#bookmark87)) demonstrated similar results for residual neural networks, using a benchmark suite of image classifcation data- sets, and Moussa et al. ([2023](#bookmark88)) for quantum neural networks, using a benchmark suite of small datasets on which a quantum circuit can be simulated.

Alternatively, Probst et al. ([2019](#bookmark89)) proposed a new hyperparameter importance measure dubbed tunability, which calculates for each hyperparameter what would be the efect in terms of performance when it would be optimised. For this, it assumes a default value for all other hyperparameters. Tunability is related to ablation analysis yet difers in important aspects. While ablation analysis sequentially and greedily calculates an ablation path and determines parameter importance based on the changes in performances observed in the context of that sequence, tunability determines for each hyperparameter the performance increase if it were the frst step of the ablation path. Probst et al. ([2019](#bookmark89)) determined the tun- ability of six machine learning algorithms on a subset of binary classifcation datasets from the OpenML100. Similar to the conclusions of van Rijn and Hutter ([2018](#bookmark65)), often the same hyperparameters emerged as being important.

a 灬

**3.1.4 Transformation of input spaces**

For many HPO tasks, the hyperparameter space is defned by a range (minimum and maxi- mum value) per numerical hyperparameter, across which values for this hyperparameter can be sampled uniformly. Sometimes, the search range can be transformed a priori to infuence the way that confgurations are sampled and might emphasise or de-emphasise certain regions of the underlying search space. For example, a log transformation will ensure uniform sampling from a log-transformed space, emphasising lower values and de- emphasising higher values. While usually not detailed in scientifc publications, an inspec- tion of the source code reveals that AutoML systems such as auto-sklearn (Feurer et al. [2015](#bookmark10)) and ML-Plan (Mohr et al. [2018](#bookmark90)) rely on such transformations. Search space trans- formations are usually determined based on the experience of the user and can have a great impact on the performance of a given HPO procedure.

Snoek et al. ([2014](#bookmark44)) proposed a principled approach to these transformations by intro- ducing Bayesian input warping. Specifcally, their approach transforms each hyperparam- eter according to a beta conditional density function. The authors refer to this transforma- tion as *warping*. The beta distribution is a fexible parameterised probability distribution defned by two parameters, 훼 and 훽; depending on their values, a wide variety of trans- formation functions can be obtained, including the linear, exponential, logarithmic and sigmoid transformations. For a confguration space of *n* numerical hyperparameters, this results in n × 2 additional parameters that need to be optimised as well. While the exact details are beyond the scope of this survey, Snoek et al. ([2014](#bookmark44)) use input warping in combi- nation with a Gaussian process. They treat these additional parameters as hyperparameters to the covariance function, place a normally-distributed prior over these, and use a Markov chain Monte Carlo (MCMC) approach via slice sampling to optimise the covariance func- tion. Once this optimisation process has been performed, the search space has been trans- formed for each hyperparameter according to an optimised value of the beta distribution, allowing for uniform sampling in the transformed input space.

**3.1.5 Philosophies on search space design**

There are various approaches to search space design. The ‘programming by optimisation’ paradigm (Hoos [2012](#bookmark66)) is based on the idea of avoiding premature commitment to design choices; consequently, all design choices that cannot be made a priori on a strong basis are included in the search space, and it is up to the search algorithm to fnd good con- fgurations within the resulting, typically vast search spaces. Various techniques have been proposed that work very well with large search spaces, including ones that allow for fast pruning on parts that are unlikely to improve over the best confgurations encountered thus far (Wistuba et al. [2015a](#bookmark67); Wang et al. [2021a](#bookmark91)), but also Bayesian optimisation methods that have been demonstrated to be able to relatively quickly fnd good candidate confgurations within large search spaces (Bergstra et al. [2011](#bookmark42); Hutter et al. [2011](#bookmark43); Snoek et al. [2012](#bookmark92)).

In contrast, the paradigm of model-free optimisation utilises data from previous experi- ments to generate a set of *complementary* confgurations that can then be performed sequentially on the target dataset (Wistuba et al. [2015b](#bookmark93)). This approach aims to search for a small set of confgurations that together work well on a range of datasets. If a specifc con- fguration covers already good performance on a cluster of datasets, other confgurations will be searched to work well on other clusters of datasets. Pfsterer et al. ([2021](#bookmark68)) showed

a 灬

that the generation of an optimal set of confgurations is NP-complete and extended the previous greedy approach by surrogate models in order to make it applicable to machine learning algorithms with an arbitrary number of hyperparameters. Gijsbers et al. ([2021](#bookmark94)) further extended this approach by introducing the notion of symbolic confgurations, i.e., the list of confgurations can also include dataset-dependent operators [such as the well- known random forest default, where the number of attributes is equal to the squared root of the number of attributes (Breiman [2001](#bookmark95))]. The symbolic defaults employ a richer and more fexible search space. As such, it is harder to fnd a complementary set of symbolic defaults, but once found, they will achieve better performance.

The programming by optimisation and model-free optimisation paradigm can be com- bined, as was done by Feurer et al. ([2022](#bookmark52)) (see Sect. [5](#bookmark39)).

**3.1.6 Automated search space construction**

We now briefy outline a number of methods that autonomously construct a search space for HPO.

Perrone et al. ([2019](#bookmark69)) proposed a method aimed at determining a good search space based on historical data. Specifcally, they consider a scenario where a high number of evaluations from a large number of similar datasets is already available. The assumption is that regions in the search space that worked well for these historic datasets will also work well for the new dataset. We note that this can be considered a form of transfer learn- ing. The authors aim to defne a search space that contains the best solutions of all previ- ous tasks. They suggest two specifc forms, a low-volume bounding box and a low-volume ellipsoid, and provide optimisation methods that can utilise this historic data to construct such a search space.

Alternatively, the AMS system supports the machine learning expert in generating a search space (Cambronero et al. [2020](#bookmark96)). More specifcally, it will generate a search space based on a weak specifcation. In some cases, the machine learning expert wants to express some preference for a certain solution type, e.g., they suggest a logistic regression model. AMS utilises (1) the source code documentation to suggest similar model types and (2) a corpus of source code fles to suggest complementary preprocessing components as well as hyperparameters that should be explored. The corpus of source code fles will be scanned for similar pipelines, based on which other components are determined that are often used in combination with the models that were specifc in the weak specifcation. These are all added to the search space. The resulting search space can be explored using various search algorithms; notably, the authors demonstrate their approach to work well in combination with random search and genetic programming.

**3.2 Search strategy**

In this section, we describe various search strategies for HPO. In this section, we focus pri- marily on conceptual methods rather than complete systems, although some of the concep- tual methods do have well-maintained packages available. The key diference with com- plete systems is that they often combine various conceptual methods and come with an implementation (these are listed in Sect. [5](#bookmark39)). We list the packages that we are aware of as well. Table [1](#bookmark97) gives an overview of the methods that we survey per category.

a 灬

a 灬

**Table 1** Overview of methods that we survey in this chapter, categorised by their design philosophy

|  |  |  |
| --- | --- | --- |
| Category | Examples | Summary |
| Uninformed search | Grid search, random search | Uninformed search, typically used as baseline |
| Bayesian optimisation | SMAC,TPE, Spearmint | Uses surrogate model to infer promising regions in search space |
| Reinforcement learning | Hype-RL | Models the problem as a sequential decision making problem |
| Evolutionary algorithms | Genetic prog., CMA-ES | Combines descriptions of various confgurations to new ones |
| Monte-Carlo tree search | MOSAIC | Represents pipeline structure as tree that can be traversed |
| Gradient-based opt |  | Estimate gradient of validation loss with respect to hyperparameters |

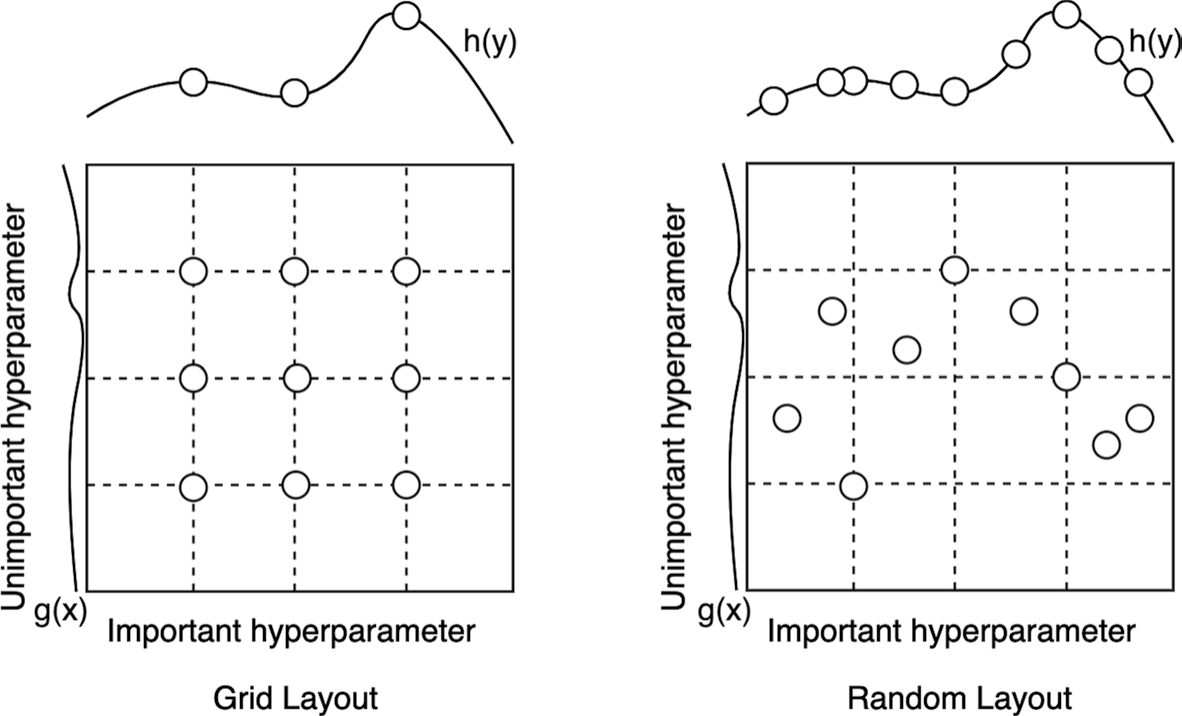
Citations are provided in the text, where the methods are explained. Gradient-based optimisation are more often used for neural architecture search

**122** Page 16 of 88 M Baratchi et a l

**3.2.1 Grid search and random search**

Grid search and random search are the earliest and simplest search techniques used for HPO. Grid search (also known as a parameter sweep) selects the best confguration by exhaustively evaluating all possible combinations of hyperparameter values. To use grid search on continuous hyperparameters, the respective domains have to be mapped to a set of discrete values. In general, grid search performs reasonably well in the low-dimensional search spaces induced by small numbers of hyperparameters. However, performing a grid search in high-dimensional search spaces or with high resolution for discretised hyper- parameters involves the evaluation of very large numbers of confgurations. To improve efciency in such scenarios, Larochelle et al. ([2007](#bookmark98)) proposed a multi-resolution approach in which, frst, a confguration is selected from a coarse-grained confguration space, and next, a higher resolution search is performed in the vicinity of the selected confguration.

As an alternative to grid search, random search samples confgurations from the search space at random; this does not require the discretisation of continuous hyperparameters. Grid and random search approaches share the advantage of simplicity, ease of implementa- tion and excellent parallelisability. Bergstra and Bengio ([2012](#bookmark7)) demonstrated that random search tends to outperform grid search in high-dimensional spaces when using the same computational budget. This is explained by the fact that each hyperparameter contributes diferently to the overall loss. Grid search unnecessarily allocates too many resources to the evaluation of unimportant hyperparameters. Figure [3](#bookmark99) provides an illustrative example comparing grid search and random search. Assuming that nine confgurations are eval- uated for optimising a function *f*, which is highly infuenced by a function *h*, it can be seen that with random search, all nine evaluations explore distinct values over this func- tion *h*, whereas grid search only explores three distinct values. Bergstra and Bengio ([2012](#bookmark7)) further compared various sampling strategies for random search and found that Sobol



**Fig. 3** The diference between random search and grid search. Each approach performs 9 evaluations for optimising a function (x y) = g(x)+ h(y) that depends on two parameters, *x* and *y*. The functions *g*(*x*) and *h*(*y*) are shown on the left and above the representation of the search space, respectively. Assume that *f*(*x*, *y*) is strongly infuenced by *h*(*y*) and only weakly by *g*(*x*). Grid search only evaluates *h*(*y*) for three distinct val- ues of *y*. However, with random search, all the nine evaluations explore distinct *y* and hence *h*(*y*), increasing the chance of also fnding a good value of*f*(*x*, *y*) (Bergstra and Bengio [2012](#bookmark7))

a 灬

sequences (Antonov and Saleev [1979](#bookmark100)) ofer a particularly efective way to perform random sampling. Sobol sequences have later been adopted by systems such as SMAC3 (Lindauer et al. [2022](#bookmark101)).

Random and grid search are both commonly used as baselines, and many popular machine learning libraries include implementations of these simple search techniques [see, e.g., Scikit-learn (Buitinck et al. [2013](#bookmark102)), Tune (Liaw et al. [2018](#bookmark103)), Talos (Talos [2019](#bookmark104)), and H2O (H2O.ai [2017](#bookmark105))].

**3.2.2 Bayesian optimisation and variants**

Evaluating a hyperparameter confguration can be a computationally expensive task since it requires training and validating a machine learning model. Both grid search and ran- dom search require a relatively large number of such function evaluations, especially when the number of hyperparameters is large. Bayesian optimisation (Garnett [2023](#bookmark107); Hutter et al. [2011](#bookmark43); Snoek et al. [2012](#bookmark92)), sometimes referred to as sequential model-based optimisation, aims to reduce the number of function evaluations by incorporating knowledge about the performance of confgurations encountered earlier in the search process. It uses concepts from Bayesian statistics, in particular in the way a statistical model is used to estimate the probability distribution over the possible performance (loss) values of a previously unseen hyperparameter confguration.

Bayesian optimisation utilises knowledge about the search space by ftting a model to the data collected from previously evaluated confgurations (the objective function *f*); this model is often referred to as a surrogate model or empirical performance model. The two key ingredients of Bayesian optimisation are the (1) surrogate model and the (2) acquisi- tion function. The surrogate model is a probabilistic model for approximating the objec- tive function *f* that maps hyperparameter confgurations to the respective performance values. Bayesian optimisation uses a prior belief of the shape of*f* and updates this prior

|  |
| --- |
| Target  observations  surrogate model  95% confidence interval |

1 . 50- 1 . 25-

× o . 75-

0 . 50- 0 . 25-

|  |
| --- |
| 〈 |

- 2 0 2 4 6 8 10



Acquisition Function Next Best Guess



2-

|  |
| --- |
|  |

2 0 2 4 6 8 1



**Fig. 4** Bayesian optimisation using a Gaussian process-based surrogate model and the upper confdence bound as acquisition function after seven function evaluations. The solid blue line (target) is the function that is to be optimised, and the dashed line and yellow surface are the mean performance and associated uncertainty derived from the surrogate model, respectively. There is no uncertainty at points where the tar- get function has been evaluated. The acquisition function determines the utility of the confgurations to be evaluated (Nogueira [2014](#bookmark109)). (color fgure online)

a 灬

with new evaluations of*f* (on selected hyperparameter settings) to achieve a posterior that better approximates *f* (Shahriari et al. [2016](#bookmark110); Jones et al. [1998](#bookmark111)). The acquisition function is used to guide the process of sampling from the hyperparameter space in the next round by evaluating the utility of candidate hyperparameter settings based on the surrogate model. The acquisition function, in principle, uses the mean and the uncertainty in the posterior distribution derived from the surrogate model to determine which hyperparameter confgu- ration should be evaluated next. Figure [4](#bookmark108) shows a snapshot from the Bayesian optimisation procedure after seven iterations.

Many diferent functions can be used as surrogate models and acquisition functions in Bayesian optimisation; for further details, we refer the interested reader to the survey on Bayesian optimisation by Shahriari et al. ([2016](#bookmark110)). Many researchers have developed Bayes- ian optimisation methods for HPO (Snoek et al. [2012](#bookmark92); Bergstra et al. [2011](#bookmark42); Hutter et al. [2011](#bookmark43)). The key diference between these approaches lies in the choice of the surrogate model and the acquisition function.

We briefy introduce widely used surrogate models and acquisition functions below. For a more detailed discussion on Bayesian optimisation, we refer the reader to the textbook on Bayesian optimisation (Garnett [2023](#bookmark107)), which provides extensive information on the mathe- matical basis of Gaussian processes as surrogate models, on decision-theoretical questions (in particular, the determination of data points for evaluation), as well as on theoretical results and practical considerations. As Bayesian optimisation can be used to optimise any form of black-box function, the book also deals extensively with topics related to realistic problem settings that are also applicable to machine learning, such as uncertainty in the observation space.

*Surrogate models* The two most widely used surrogate models are Gaussian processes and random forests; the use of the former has been explored by various authors (Gins- bourger et al. [2010](#bookmark112); Snoek et al. [2012](#bookmark92); Desautels et al. [2014](#bookmark113)). A Gaussian process is a non- parametric statistical model defned based on its prior mean and covariance functions. The posterior mean and covariance at any evaluated point represent the model’s prediction and its uncertainty. Such a model is suitable as a surrogate model in Bayesian optimisation, as it provides the uncertainty estimates that are needed for the acquisition function to choose the next evaluation point. The computational complexity of training a Gaussian process model is O(n3 ), the cost of inversion of its covariance matrix, where *n* is the number of observations. In the context of Bayesian optimisation, each observation is a function evalu- ation carried out earlier in the optimisation process. One of the main drawbacks of Gauss- ian processes arises from the fact that the computational cost of training increases with the number of evaluations that have been carried out. Gaussian processes with most standard kernels do not scale to high-dimensional datasets. However, methods such as sparsifying Gaussian processes (Seeger et al. [2003](#bookmark114)) have improved applicability to large datasets by reducing the rank of the covariance matrix.

Another limitation of Gaussian-process-based Bayesian optimisation methods is that they are only applicable to continuous hyperparameter spaces and cannot natively handle integer, categorical or conditional hyperparameters. Additional approximations need to be introduced to handle these commonly encountered types of hyperparameters. For instance, for integer-valued hyperparameters, continuous values are rounded to the closest integer after optimising the acquisition function. For categorical hyperparameters, an approxima- tion based on a one-hot encoding can be used. Garrido-Merchán and Hernández-Lobato ([2020](#bookmark115)) show that these approximations may lead to the failure of the Bayesian optimisation process, as they ignore that some confgurations are invalid and may put probability mass on points where *f* cannot be evaluated. They further provide a systematic transformation of

a 灬

the categorical and integer variables that permits the assumption that the value of*f* remains constant in certain areas of the underlying search space.

An alternative class of surrogate models are based on random forest regres- sion (Breiman [2001](#bookmark95)); these have been demonstrated to be more scalable than Gaussian processes (Hutter et al. [2011](#bookmark43)). A random forest model creates an ensemble of deci- sion trees that can collectively approximate the response surface of the given objective function *f*. Since training individual trees is parallelisable, and each tree is trained only on a sample of data, this technique scales much better to large datasets. Furthermore, it can quite easily handle diferent hyperparameter types, including conditional hyper- parameters. The drawbacks of random forest regression models are that the uncertainty estimation is known to be less accurate than those of a Gaussian process and that they do not extrapolate well outside the observed data points; the latter also applies to most Gaussian process models.

*Acquisition functions* The acquisition function determines, based on a given sur- rogate model, which point in the search space to evaluate next. Expected improve- ment is a commonly used acquisition function; it is composed of two terms relating to the (1) expected value at a given point of the function to be optimised and (2) the associated variance (or uncertainty). This combination naturally provides a trade-of between exploitation (around a promising point) and exploration (in unknown areas). By considering the expected value, the search is focused on regions of the search space containing good candidate solutions with high probability. In contrast, the uncertainty term encourages the exploration of regions that are weakly explored or in which can- didate solutions of highly variable quality have been observed (Jones et al. [1998](#bookmark111)). There are other acquisition functions, such as the upper confdence bound (Hofman and Shahriari [2014](#bookmark116)) of uncertainty for every query point, or information-theoretic approaches, such as entropy search (Hennig and Schuler [2012](#bookmark117)).

*Parallelism* Bayesian optimisation takes advantage of all information collected during the optimisation process by evaluating samples sequentially, one by one. This makes it much more data-efcient than the grid and random search approaches. However, as the surrogate model typically suggests only a single best confguration to evaluate next, it is not trivial to parallelise Bayesian optimisation. To address this issue, Ginsbourger et al. ([2011](#bookmark118)) extend the original Bayesian optimisation framework by proposing a multi-point expected improvement criterion for the simultaneous selec- tion of multiple points that are evaluated in parallel. In this approach, some evalua- tions can be performed while the results of previous evaluations are not yet fully avail- able. This is enabled by injecting partial knowledge of ongoing evaluations into the expected improvement formulation.

*Sequential model-based algorithm confguration (SMAC)* (Hutter et al. [2011](#bookmark43)) is a Bayesian optimisation procedure for general algorithm confguration. SMAC uses ran- dom forests as surrogate models and expected improvement as an acquisition func- tion. The random forest model allows SMAC to optimise conditional and categorical hyperparameters. Further improvements to SMAC have been proposed [e.g., pruning the search space to increase efciency (Li et al. [2022](#bookmark119))].

*Tree-structured Parzen estimator (TPE)* (Bergstra et al. [2011](#bookmark42)) is another approach based on Bayesian optimisation that uses expected improvement as an acquisition function and TPE to model the probabilities and distributions of the target function. Gaussian process approaches model p(y ∣ 휆) directly, where 휆 denotes the confgura- tions, and *y* indicates the performance observed for a given confguration 휆 . TPE, how- ever, calculates p(휆 ∣ y) and *p*(*y*). p(휆 ∣ y) is modelled by replacing the distributions

a 灬

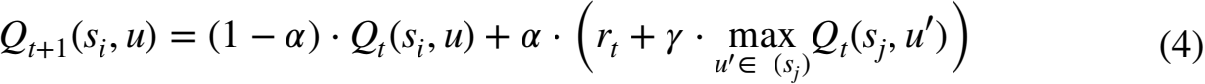
of the confguration prior with two non-parametric densities that make a distinction between good and bad confgurations: (1) the density of confgurations that have a loss below a given threshold (set to a quantile of observed *y* values) and (2) the density of those with higher loss. Through maintaining a sorted list of confgurations in memory, the run-time of TPE scales linearly with respect to the number of hyperparameters. TPE can also be used for conditional hyperparameters and tree-structured confgura- tion spaces. TPE is implemented in the HyperOpt library (Bergstra et al. [2013](#bookmark120)).

**3.2.3 Reinforcement learning‑based approaches**

Reinforcement learning concerns sequential decision processes in state space (Sutton and Barto [2018](#bookmark122)). The agent or a controller learns to fnd optimal paths as a Markov decision process with a number of states and an action space . At each state si ∈ , there are a number of actions (si ) ⊆ that can be selected by the agent. Taking an action u ∈ (si ) will create a state transition from state si to state sj with probability ps ∣su(sj ∣ si u). The agent interacts with the environment at points in time. At each timestamp *t*, the agent receives an immediate reward rt , based on its action *u* and the transition between si and sj. The goal in reinforcement learning is to determine a policy, i.e., a function that determines for each state a probability distribution over actions, such that a cumulative reward func- tion computed from the immediate rewards rt (in many cases, a weighted sum over *t*) is maximised.

Value-based and policy-based approaches are two well-known classes of reinforce- ment learning methods. In value-based approaches, the agent estimates the optimal value function *Q* that defnes which action to take in a particular state to achieve the maximum reward. The policy-based methods directly learn the optimal policy 휋 .

Q-learning (Watkins [1989](#bookmark123)) is an example of a value-based approach where the agent learns a look-up table of actions and states by iteratively updating the equation (Baker et al. [2017](#bookmark124)):



Here, 훼 is the Q-learning rate determining the weight of new information to old informa- tion, and 훾 is a discount factor which defnes the weight given to rewards depending on how far in the future they will be collected. Originally, Q-learning is defned for discrete spaces and which makes it useful for optimising discrete (or discretised) hyperparameters. There are, however, extension of Q-learning to continuous action spaces as well (Millán et al. [2002](#bookmark125)), which can potentially be used to optimise continuous hyperparameters.

Policy-based approaches have better convergence properties than Q-learning approaches and are suited for higher-dimensional and continuous actions (i.e., optimising continuous hyperparameters). The policy-gradient approach is an example of a policy-based approach that has been used for HPO. The approach taken is to directly learn the optimal policy 휋 (ut ∣ st ) given the history of state-action pairs (st ut ). Policies are defned by a number of parameters 휃 , and the general goal is to optimise the parameters of the policy such that the total expected reward is optimised. The REINFORCE algorithm (Williams [1992](#bookmark126)) is a well-

known algorithm to calculate the policy parameters 휃 by maximising the expected reward J(휃) = **피**P(u1∶T ∶휃)[R] (5)

a 灬

Here, P(u1:T) denotes the probability of a sequence of *T* actions that have resulted in reward *R* after *T* time steps.

The search for good hyperparameter settings is usually seen as a *sequential* decision process, in which the values of one or more hyperparameters are modifed in each step. Diferent approaches have been taken so far to formulate the HPO problem within an RL framework. Each state is commonly defned by hyperparameters and their values. The way states and actions are defned could allow step-wise changes to only a sin- gle hyperparameter (Wu et al. [2020](#bookmark127); Zoph and Le [2017](#bookmark128)), to a selected group of hyper- parameters (Baker et al. [2017](#bookmark124)) or to all hyperparameters at once (Jomaa et al. [2019](#bookmark129)). Additionally, the states could hold other information, such as dataset meta-features or the history of evaluated hyperparameter confgurations (Jomaa et al. [2019](#bookmark129)). Formu- lating states that allow changes to only a single hyperparameter at each step, rather than to a group, allows treating the confguration of individual hyperparameters as a sequential decision process as well. This allows to implicitly consider the condition- ality among hyperparameters in the search space by remembering previous decisions. This also impacts the size of the action space, which implicitly determines the search space. Let us assume that *n* hyperparameters are to be optimised, each with a domain by Λi . Treating the hyperparameters individually will reduce the action space size from **횲** = Λ1 ×Λ2 × … ×Λn to **횲** = Λ1 +Λ2 + … +Λn (Wu et al. [2020](#bookmark127)).

*Hyp-RL* (Jomaa et al. [2019](#bookmark129)) is a general HPO method based on Q-learning. In Hyp- RL each action corresponds to a hyperparameter confguration (to set all hyperparam- eters) being rewarded based on the validation loss of the confgured model rt . The total reward is calculated by accumulating the validation loss of the models confgured in a sequence of actions. The policy selects the actions that maximise the discounted cumu- lative reward through iterative Q-learning updates from an initial state (i.e., hyperpa- rameters initialised randomly or to their default value).

In the policy-based approach proposed by Zoph and Le ([2017](#bookmark128)), each action corre- sponds to setting one hyperparameter, and a sequence of *T* actions in a trajectory lead- ing to the confguration of all hyperparameters is rewarded by the validation accuracy of the confgured model *R*. The optimal policy is selecting the trajectory that maximises the expected reward *J* based on computing the policy gradient to update the controller parameters 휃 based on the reward.

In general, reinforcement learning methods have been used for HPO (Jomaa et al. [2019](#bookmark129); Wu et al. [2020](#bookmark127)) and more commonly in neural architecture search (examples for the latter will be given in Sect. [4.2.1](#bookmark130)).

**3.2.4 Evolutionary algorithms**

Evolutionary algorithms (Simon [2013](#bookmark132); Bäck et al. [1997](#bookmark133)) are population-based global optimisation algorithms inspired by biological evolution. They work on a set (*popula- tion*) of *P* solution candidates (*individuals*). Starting from an initial population, evolu- tionary algorithms iteratively vary the population (giving rise to a sequence of *genera- tions*), as illustrated in Algorithm 1 (Bäck et al. [2018](#bookmark134)). In each generation, a population *P*(*t*) (*parent individuals*) of size 휇 are selected, from which new individuals P (t) P (t) (*ofspring individuals*) are generated using variation operators; then, the next set of indi- viduals (*survivors*) are selected from the previous population and the ofspring. These selected individuals form the new population for the next generation. The selection of

a 灬

parent and survivor individuals is typically based on the*ftness* values **F**(t) (i.e., objec- tive values) of the individuals, where individuals with higher ftness are preferred over individuals with lower ftness. Typical variation operators are *crossover* and *mutation*. Crossover combines two or more parent individuals to transfer the benefcial features of the parents to the ofspring. The mutation operator applies small random changes to individuals to increase the diversity within the population.

**Algorithm 1** Evolutionary algorithm

Input: population size , mutation group size 入, termination, re comb ina-

tion, mutation and selection parameters or , or , om , Os

output: the best individual a\* or the best population P\* found during the run

1: t — 0;

2: P(t)  INrTIALISE(U);

3: F(t)  EVALUATE(P(t),P);

4: while (TERMNArion(p(t) , ot)  do

5: P' (t) (p(t) , or); 6: P" (t) MurArc(p' (t) , om)); 7: F(t)  EVALUArB(P" (t),入);

8: P(t + 1) (P" (t) , F(t),μ, OS ) 9: t  t + 1;

10: end while



In the context of HPO, mutation and crossover are analogous to exploitation and explo- ration, respectively. When applying an evolutionary algorithm to an optimisation problem, one has to decide how solutions are encoded as individuals. For example, an integer vari- able can be encoded as an integer but also as a list of binary variables.

In evolutionary algorithms, the *genome* encoding of an individual includes the repre- sentation of a possible solution to the problem, the actual solution after evolution is termed *phenotype* and its encoding is termed *genotype* (Bäck et al. [2018](#bookmark134)). Tani et al. ([2021](#bookmark135)) evalu- ated an evolutionary algorithm and particle swarm optimisation (Kennedy and Eberhart [1995](#bookmark136)) on two HPO tasks, concluding that both can outperform random search and gradient descent. There are two special types of evolutionary algorithms which are frequently used in the context of HPO and AutoML: genetic programming and CMA-ES. In the following, these approaches are briefy outlined.

*Genetic programming* (Koza [1994](#bookmark137)) is a form of an evolutionary algorithm that evolves programs composed of functions, which work on primary inputs and/or outputs of other functions. An example of such a program could be a mathematical expression, where the functions are mathematical operators (e.g., addition, sine, logarithm), and the actual opti- misation task could be to fnd an expression, which best fts some experimental data. Often a tree representation is used to represent programs in genetic programming. TPOT (Olson et al. [2016a](#bookmark11)) is an example of an AutoML system that uses genetic programming for the optimisation of machine learning pipelines and their hyperparameters (see Sect. [5.4](#bookmark138) for more details).

a 灬

*Covariance matrix adaptation evolution strategy (CMA-ES)* (Hansen et al. [2003](#bookmark139)) is an evolutionary algorithm, which has been demonstrated to be very efcient for a number of black-box optimisation tasks, including HPO (Loshchilov et al. [2012](#bookmark140); Watanabe and Roux [2014](#bookmark141); Loshchilov and Hutter [2016](#bookmark142); Jedrzejewski-Szmek et al. [2018](#bookmark143)). It samples candidate solutions from a multivariate normal distribution whose mean, and covariance matrix is updated based on the performance of the individuals in the population. CMA-ES works well on non-linear and non-convex optimisation tasks; it is typically used for problems with search spaces with three up to a hundred dimensions. CMA-ES has shown good per- formance compared to other black-box optimisers, such as Bayesian optimisation, on con- tinuous black-box optimisation benchmarks (Loshchilov et al. [2013](#bookmark144)). While Bayesian opti- misation is recommended for conditional search spaces, CMA-ES is recommended if the search space only contains continuous hyperparameters and the objective function is cheap, or the evaluation budget is large (Mendoza et al. [2016](#bookmark145)).

**3.2.5 Monte Carlo tree search**

Monte Carlo tree search (MCTS) is an approach for addressing state-space Markovian sequential decision problems (Kocsis and Szepesvári [2006](#bookmark146)) working based on a ran- domised evaluation of a search tree. This approach has been successfully used in game AI to predict the best moves to reach a winning position in a game (Chaslot et al. [2008a](#bookmark147)), such as Go (Silver et al. [2017](#bookmark148)) or Chess (Silver et al. [2018](#bookmark149)). For a given search problem, the MCTS algorithm builds a tree where each node (representing states) includes informa- tion about the current value of the position (usually the average of the results of simulated games visiting this node) and the visit count of this position. The MCTS algorithm repeats the following steps (Chaslot et al. [2008b](#bookmark150)): (1) traverse the tree through the selection of the best next node to move to (through balancing between exploitation and exploration based on the statistics stored), (2) expansion of the selected node by adding new child nodes to the tree to increase the options to win the game, (3) simulation, or playout, to fnish the game by traversing the search tree multiple ways in a random way and further assigning a reward based on calculating how close the output of random decisions was from the fnal winning output, and (iv) back-propagation to update each node that was traversed in the tree based on the result of the simulation.

The main power of MCTS-based approaches lies in addressing sequential problems and are thus better suited for optimising hyperparameters representing ordered decisions such as hyperparameters involved in creating a pipeline of a fxed number of components (e.g., frst selecting a data preprocessors, afterwards feature selectors, and fnally a machine learning algorithm). This works especially well in combination with pipelines with a fxed structure, as considered, for example, in MOSAIC (Rakotoarison et al. [2019](#bookmark151)) (see Sect. [5](#bookmark39)). When the pipeline structure is fxed, it can be represented as a search tree that can be traversed by MCTS. Internal nodes in the search tree represent *partial confgurations* in which only the frst preprocessing operators are fxed, whereas a leaf node represents a full confguration. A surrogate model that generalises over the full confguration space, which can also assess the quality of partial confgurations (as represented by internal nodes), can be employed to determine the performance of a given node (Rakotoarison et al. [2019](#bookmark151)). This can be done, for example, by means of sampling techniques, where a pre-defned number of leaf nodes (representing full confgurations) are being evaluated, and the average of those represents the quality of the partial confguration. Once a playout-operation has determined a suitable leaf-node, the confguration that belongs to the leaf node is instantiated and evaluated on

a 灬

the real data, and the measured performance is backpropagated into the internal tree rep- resentation. Also, the surrogate is being updated with this additional information. MCTS algorithms for HPO are only researched to a limited degree.

**3.2.6 Gradient‑based optimisation**

The gradient descent algorithm, classically used for setting the parameters of models, can be extended to jointly optimise the hyperparameters of the algorithms as well. As men- tioned before, random search has shown promising results in the context of optimising small numbers of hyperparameters. For moderately higher dimensions, more complex methods (e.g., Bayesian optimisation) are preferred. However, when dealing with neu- ral networks, besides a moderate amount of hyperparameters, there are also millions (if not billions) of parameters to optimise (i.e., the weights and bias values). Typically, the parameters of a neural network are optimised using a gradient descent method, whereas the hyperparameters are optimised by an HPO method. However, the optimised validation loss with respect to the hyperparameter can be estimated, allowing gradient descent methods to traverse the loss landscape with respect to hyperparameter values. Recently, gradient-based optimisation has shown promising results for optimising very large numbers of (hyper) parameters (see, e.g., Lorraine et al. [2020](#bookmark152)) and also in meta-learning (see, e.g., Rajeswaran et al. [2019](#bookmark153)).

One of the earlier works in gradient-based optimisation for diferentiable and continu- ous HPO was proposed by Bengio ([2000](#bookmark6)). This approach uses reverse mode diferentiation or backpropagation and focuses on small-scale continuous HPO based on diferentiable objective functions, such as squared loss and logistic loss, that can be optimised with gra- dient descent. One of the main limitations of this approach is that it requires intermediate variables to be maintained in memory for the reverse pass of the backpropagation proce- dure. This gives rise to prohibitively large memory requirements and limits the practical applicability of this method.

There has been more recent research on memory-efcient methods for approximating the hypergradients (i.e., the gradient of the validation loss with respect to hyperparame- ters). These methods generally fall into two categories: (1) iterative diferentiation (see, e.g., Franceschi et al. [2017](#bookmark154); Maclaurin et al. [2015](#bookmark155)), which approximates the hypergradi- ents by defning a sequence of functions that recursively approximate one another; and (2) approximate implicit diferentiation (see, e.g., Lorraine et al. [2020](#bookmark152)), which defnes an implicit function for the hypergradients through applying the implicit function theorem. Empirical results from several studies indicate that implicit diferentiation methods tend to be more memory-efcient (see, e.g., Grazzi et al. [2020](#bookmark156); Rajeswaran et al. [2019](#bookmark153)).

Gradient-based methods are only applicable to continuous hyperparameters and twice- diferentiable loss functions. It is also possible to extend the use of these techniques to dis- crete hyperparameters using continuous relaxation methods (see, e.g., Jang et al. [2017](#bookmark157)). For instance, Jang et al. ([2017](#bookmark157)) propose the Gumble-Softmax estimator to represent samples of one-hot encoded categorical variables with a diferentiable distribution. As explained in Sect. [4.2.5](#bookmark158), DARTS (Liu et al. [2019b](#bookmark159)) represents categorical hyperparameters by means of continuous variables, using the softmax operation in a similar manner.

a 灬

**3.3 Performance evaluation techniques**

HPO techniques evaluate various confgurations from the search space and, based on these evaluations, in each step, select the one deemed to be most promising. As was explained in Sect. [2.2.3](#bookmark47), this is often done using nested cross-validation (Varma and Simon [2006](#bookmark48)): the dataset is split into a training, a validation and a testing partition. All confgurations are trained on the training set and evaluated on the validation set. The testing partition is set aside and only used for the fnal evaluation of the best confguration. We thus fnd evaluation procedures at an inner and outer level. The HPO method employs the evaluation procedure at the inner level to assess how well a specifc confguration will perform on the given dataset. Eventually, the evaluation procedure at the outer level assesses how well the HPO method as a whole works. The inner evaluation procedure is under the control of the (user of the) HPO method, whereas this is not the case for the outer evaluation procedure. The main focus of this section is on the inner evaluation procedure. Evaluating various candidate confgurations can be computationally expensive, and therefore several proce- dures have been developed to speed up this process. We will review three general types of procedures: racing methods, methods that evaluate at lower budgets, and learning curve extrapolation methods.

**3.3.1 Multi‑objective evaluation metrics and trustworthiness**

In the performance evaluation process, the most critical question concerns the choice of the performance metric to be optimised for. Naturally, many HPO methods optimise for accuracy or the area under the ROC curve (see, e.g., Hutter et al. [2011](#bookmark43); Li et al. [2017](#bookmark160); Huisman et al. [2023](#bookmark161)); however, it is also possible to consider additional measures in a multi-objective fashion (Karl et al. [2022](#bookmark162)). Computational cost is a prominent factor to take into consideration in hyper-parameter optimisation, both for the sake of environmental con- cerns (Tornede et al. [2023b](#bookmark163)) as well as the fact that machine learning models are being deployed on various devices that have fewer compute power (Evchenko et al. [2021](#bookmark164)). HPO techniques iterate over a range of possible confgurations and are, therefore, typically com- putationally expensive. However, they can still search for a model that requires less cost at inference time. When a model has low complexity, it generally requires less computational cost when making predictions. This can, depending on the amount of predictions that are being made in production, drastically reduce the overall compute or power consump- tion. Lu et al. ([2020](#bookmark165)) and Zhang et al. ([2018](#bookmark166)) measure this amount of operations in FLOPS. Other measures that can be taken into consideration are

• *Class probabilities* and *uncertainty quantifcation* Many machine learning models pro- vide a measure of certainty per prediction, quantifying the amount of uncertainty that the prediction belongs to a certain class. However, it has been noted that these probabil- ities are not always well-calibrated (de Menezes et al. [2023](#bookmark167)). de Menezes et al. ([2023](#bookmark167)) survey techniques that can be used to better calibrate uncertainty estimations. König et al. ([2020](#bookmark168)) utilises HPO to build a wrapper around classifers that have better uncer- tainty quantifcation.

• *Robustness* Machine learning models, in particular deep learning models, are vulner- able to adversarial attacks (Goodfellow et al. [2015](#bookmark169)). An adversarial attack makes small perturbations to the input that will force the model to misclassify an otherwise correctly classifed observation. Various methods have been proposed to train robust models

a 灬

against input perturbations (Bai et al. [2021](#bookmark170)). This robustness can also be quantifed by a measure called adversarial accuracy (Zhang et al. [2019b](#bookmark171)). To quantify the robustness of a model, formal verifcation methods have been proposed (Meng et al. [2022](#bookmark172); Botoeva et al. [2020](#bookmark173); Wang et al. [2021c](#bookmark174)). As these techniques are computationally expensive, it is hard to integrate them in HPO methods. However, it is possible to utilise HPO to speed up robustness verifcation (König et al. [2022](#bookmark175), [2023](#bookmark176)), or aid the selection of machine learning models that are both accurate and robust (Bosman et al. [2023](#bookmark177)).

• *Fairness* and *bias mitigation* As the outcome of machine learning methods directly involves and afects human lives, care should be taken to ensure that the outcomes are fair, although the exact notion of fairness depends on the context of the decision situ- ation. While one of the key elements to this is human agency and oversight (European Commission High Level Expert Group AI [2018](#bookmark178)), various measures can be optimised to steer the modelling in a useful direction, such as equalised odds or equality of opportu- nity (Hardt et al. [2016](#bookmark179)). Weerts et al. ([2023](#bookmark180)) give an overview of challenges and oppor- tunities of maintaining fairness in AutoML.

For a complete overview of evaluation metrics to consider or multi-objective HPO, the reader is referred to the work by Karl et al. ([2022](#bookmark162)).

**3.3.2 Racing methods**

One way of speeding up the internal evaluation procedure is by racing. When the inter- nal evaluation procedure is subject to a cross-validation scheme, it will run each candidate various times, each time trained and tested on a diferent part of the data. In some cases, it becomes already clear after a few of those iterations that the candidate confguration will not be competitive with the best confguration found previously.

iRace implements this design criterion by employing a statistical test, specifcally, the Friedman test or the t-test (López-Ibáñez et al. [2016](#bookmark181)). When this test determines that the evaluations of a given candidate confguration are not showing statistically signifcantly better performance than the best confguration seen so far, no further evaluations are being conducted, and the evaluation procedure is stopped early.

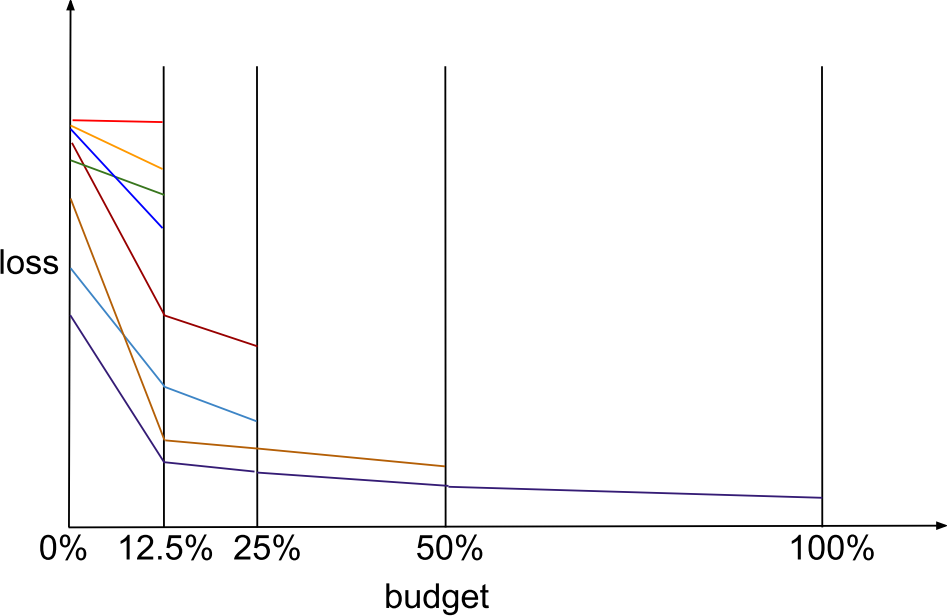
Statistical tests can be unnecessarily conservative, therefore wasting compute time on candidate confgurations that appear to be not competitive but have not shown to be sta- tistically signifcantly dominated. An alternative to this is random aggressive online rac- ing (ROAR), an extension to random search that applies the racing strategy in a more aggressive way (Hutter et al. [2011](#bookmark43)). It stops the evaluation of a candidate confguration after the average performance on recent validation folds is lower than the average perfor- mance of the current best algorithm. This way, many candidate confgurations can already be dropped after a single validation fold at the risk of occasionally eliminating a superior candidate solution.

**3.3.3 Evaluation using a lower budget**

One way to reduce the training time is by estimating the performance using a reduced training budget (Mohr and van Rijn [2022](#bookmark182)). This can be achieved in various ways: one can subsample the given set of data points, decrease the number of attributes, and lower image resolution in computer vision tasks (Chrabaszcz et al. [2017](#bookmark183)), or limit the training process to a few iterations (e.g., train a neural network with a given number of epochs) or up to a

a 灬

**122** Page 28 of 88 M. Baratchi et al.



**Fig. 5** Illustration of the sample schedule of successive halving. The x-axis represents the budget in terms of data points used to evaluate a given confguration, whereas the y-axis represents the performance in terms of loss (lower is better). Initially, eight confgurations are evaluated on 12.5% of the maximum num- ber of data points, after which the worst-performing four confgurations are dropped. The remaining four confgurations are evaluated on 25% of the maximum number of data points, after which the worst-perform- ing two confgurations are dropped. This procedure is repeated until only one confguration is evaluated on 100% of the data points (Feurer and Hutter [2019](#bookmark185))

given cutof in running time. The latter approach is sometimes called low-fdelity learn- ing (Binder et al. [2020](#bookmark186)), or multi-fdelity in cases where multiple training budgets are used (Hu et al. [2019](#bookmark187)).

When evaluating on a lower budget, one may wonder at which budget to sample. Obvi- ous answers to this could be using a percentage of the dataset, e.g., 10% or 50% of the train data. Provost et al. ([1999](#bookmark188)) addressed this problem by proposing *efcient progres- sive sampling*, which has later been used in various other approaches, such as the learning curve method by Leite and Brazdil ([2010](#bookmark189)) and successive halving (Jamieson and Talwalkar [2016](#bookmark190)). They motivate their approach by stating that when having access to a large dataset, not all data points need to be utilised, and propose a technique that detects an appropri- ate number of data points that should be used for a given confguration. They propose to evaluate confgurations using multiple budgets, using a geometrical schedule, e.g., using a dataset of size n = {64,128,256,512, …}, until convergence is detected. Here, conver- gence refers to the fact that the learning curve is saturated, and performance does not fur- ther improve when more data is provided. They prove that this type of schedule is asymp- totically optimal in terms of computation time. In other words, the authors claim that this procedure will select a dataset size that obtains maximised performance, utilising at most a constant factor more running time than when training the algorithm on all available data. However, often this procedure will actually be faster than training the algorithm on all data available, presumably in cases when an algorithm obtains a saturated performance with fewer data than all available data.

Successive halving (Jamieson and Talwalkar [2016](#bookmark190)) is a multi-armed bandit method that aims to reduce training time by allocating resources more efciently. This method also uses the geometrical schedule as proposed by Provost et al. ([1999](#bookmark188)). Initially, a ran- dom set of confgurations is sampled and evaluated at a certain budget (e.g., data samples). This budget represents only a fraction of the normally required training time of the can- didate confgurations, ensuring that less time is taken. The performance of the selected

a 灬

confgurations is evaluated, and only the top percentage (e.g., 50%) of the confgurations with the highest performance are selected for the next round, where they are evaluated on a larger data sample. This process is repeated until only one confguration is left, as illus- trated in Fig. [5](#bookmark184).

Successive halving has several hyperparameters that also need to be determined. For example, the minimal budget and the initial number of confgurations are important hyper- parameters that can signifcantly infuence performance. Setting the minimal budget too low might exclude certain confgurations from being considered since some confgurations may require a high budget to excel and might thus be dropped too early when the initial budget is too low. Li et al. ([2017](#bookmark160)) proposed hyperband, an extension to successive halving that aims to dynamically balance the number of confgurations and the initial budget allo- cated for evaluating the confgurations. Hyperband is essentially a loop around successive halving, invoking it multiple times with a diferent minimal budget and number of confgu- rations. The maximum budget per learning algorithm is fxed. Each iteration of successive halving is called a bracket. Generally, the confgurations per successive halving bracket are sampled completely at random from a larger confguration space. Hyperband starts with a bracket that evaluates a high number of confgurations with a low budget; in each subse- quent bracket, the number of initial confgurations is decreased while the initial budget is increased. Efectively, each subsequent bracket of successive halving will explore the same sample sizes as the previous bracket, except for the frst. As an edge case, the fnal bracket of successive halving is run with only a few confgurations, and the initial budget is the same as the maximum budget. Using this property, Li et al. ([2017](#bookmark160)) proved that hyperband is never more than a log factor slower than random search.

Successive halving and hyperband are performance evaluation methods that work well in combination with random search but can also be naturally integrated into other search strategies. Baker et al. ([2018](#bookmark191)) propose fast-hyperband, a method that employs a machine learning model to predict whether an evaluated confguration can improve over the best confguration found so far. Indeed, successive halving and hyperband are quite static in the way they drop candidate confgurations, and by employing a model, better-informed deci- sions can be made.

Various methods have been proposed that directly combine multi-fdelity methods with Bayesian optimisation. While most of these methods can be used ‘out of the box’ in com- bination with most search algorithms described in Sect. [3.2](#bookmark45), Bayesian optimisation is more complex, as it trains an internal surrogate model. Falkner et al. ([2018](#bookmark192)) combine hyperband with Bayesian optimisation to select new candidate confgurations. As hyperband usually samples uniformly at random, it can greatly beneft from focusing on good regions on the search space. Specifcally, Falkner et al. ([2018](#bookmark192)) use the TPE as a surrogate model (Bergstra et al. [2011](#bookmark42)). There is empirical evidence from the work of Zela et al. ([2018](#bookmark193)) that the cor- relation between the performance with a low training budget and high training budget is weak. In light of this, Li et al. ([2017](#bookmark160)) and Falkner et al. ([2018](#bookmark192)) suggest increasing the sam- ple size gradually.

**3.3.4 Early stopping by learning curve extrapolation**

Learning curves describe the performance of learning algorithms as a function of a given resource, e.g., the number of training iterations or the number of training examples, and are commonly used to extrapolate to the performance on the full budget (Mohr and van Rijn [2022](#bookmark182)). Various resources exist to obtain historic learning curves on many datasets, such as

a 灬

OpenML (Vanschoren et al. [2013](#bookmark85)), LCBench (Zimmer et al. [2021](#bookmark195)) and LCDB (Mohr et al. [2022](#bookmark196)).

Leite and Brazdil ([2005](#bookmark197)) proposed a method that leverages similarities in the learning curves observed for diferent datasets. Their work builds upon the assumption that if the datasets are similar, the confgurations will also perform or rank similarly. The method requires access to a set of learning curves on historic datasets; it uses a distance function between learning curves and a *k*-NN-based algorithm to determine to which historic data- sets a given dataset is most similar. Utilising this distance function, learning curves of the same confguration on diferent datasets are being identifed based on their shape simi- larity, assuming that similar datasets will lead to similar learning curves. After identify- ing similar datasets, knowledge of confgurations that worked well on these is applied to the current dataset. Later, Leite and Brazdil ([2010](#bookmark189)) extended this work by also taking into consideration the so-called meta-features, and van Rijn et al. ([2015](#bookmark198)) further extended the approach to also take into consideration a measure of running time. *Freeze–thaw Bayesian optimisation* allows to dynamically stop (freeze) and restart (thaw) the training procedure (Swersky et al. [2014b](#bookmark199)). The optimisation of hyperparameters stops when it seems unlikely that it will lead to fnding a model with a small loss. Then, another hyperparameter con- fguration will be evaluated. In case the chances for fnding a small loss for a stopped HPO process have increased, that process can be resumed.

Domhan et al. ([2015](#bookmark200)) proposed a technique for the early termination of unpromis- ing confgurations using a probabilistic model that predicts the performance distribution based on the frst part of a learning curve. The partially observed learning curve is mod- elled using a set of 11 parametric curve models. In order to yield accurate predictions, this method usually requires a relatively long partial learning curve. Later, Klein et al. ([2017c](#bookmark201)) improved this idea by proposing a neural network-based method, incorporating specifc learning curve operators to learn the prediction model across diferent learning curves of various algorithms on the same dataset. Both Domhan et al. ([2015](#bookmark200)) and Klein et al. ([2017c](#bookmark201)) assume that it is possible to model the learning curve by using a set of function families, such as the inverse power law (Brumen et al. [2021](#bookmark202)).

Klein et al. ([2017a](#bookmark203)) proposed FABOLAS, a Bayesian optimisation method that also models the improvement over various amounts of budget and uses this model to select a confguration.

Most early-stopping approaches require a validation set to estimate the performance of the ongoing training process. There are two drawbacks to this approach. Firstly, the evalu- ation of the model on the validation set at diferent intervals is computationally expen- sive. Secondly, it requires making a choice on the size of the validation set, considering the trade-of between low generalisation error and the use of sufcient amounts of training data. To address these problems, Mahsereci et al. ([2017](#bookmark204)) proposed using an early stopping strategy for gradient-optimisation tasks without a validation set. For this purpose, the infor- mation on local statistics of the computed gradients is used. Without a need for a hold-out validation set, this method allows the optimiser to use all available training data.

Mohr and van Rijn ([2023](#bookmark205)) introduced *learning curve-based cross-validation (LCCV)*, an extension to cross-validation that takes into account the evaluation of the learning curve of a given hyperparameter confguration. LCCV considers all confgurations in order and works with the concept of the best confguration encountered so far. The main assumption of their work is that learning curves are convex and provide empirical evidence that this holds for observation-based learning curves of many algorithms on most datasets. Using this convexity assumption, they make an optimistic estimation of what the maximum per- formance of a given confguration at a certain budget can be, similar to the formulations

a 灬

of Sabharwal et al. ([2016](#bookmark206)). Using this optimistic estimation, LCCV determines whether a given confguration will still be able to improve over the currently best confguration. If this is not the case, then the confguration can be discarded prematurely. Thus, similar to racing, LCCV takes a rather conservative approach and only discards a confguration when it is rather certain that it will not improve over the currently best confguration.

**3.4 HPO systems and libraries**

In this section, we review various well-known systems and libraries that can be used for HPO. We note that there is some overlap with the works that are described in the previous sections; here, our focus is on the implementation of the underlying methods and practical considerations regarding their use. Our descriptions are based on those given in scientifc publications. We note that in some cases, development eforts may have continued, leading to improvements in functionality and usability.

*SMAC* (Hutter et al. [2014](#bookmark64); Lindauer et al. [2022](#bookmark101)) is a general-purpose algorithm con- fgurator and HPO system based on Bayesian optimisation. One of the distinguishing features of SMAC is its use of a random forest as the underlying surrogate model, ren- dering the Bayesian optimisation procedure broadly applicable to various types of search spaces. SMAC is used at the core of various widely used AutoML systems, including AutoWEKA (Thornton et al. [2013](#bookmark8)) and Auto-sklearn (Feurer et al. [2015](#bookmark10)).

*HyperOpt* (Bergstra et al. [2013](#bookmark120)) implements various optimisation algorithms, including random search, TPE (Bergstra et al. [2011](#bookmark42)) and adaptive TPE. It can be parallelised using Apache Spark and MongoDB.

*Spearmint* (Snoek et al. [2012](#bookmark92)) is a Bayesian optimisation system. It uses Gaussian pro- cesses as a surrogate model and expected improvement as an acquisition function. Com- pared to vanilla Bayesian optimisation, Spearmint allows for efective parallelisation across multiple cores. Results of an empirical study comparing TPE, SMAC and Spearmint sug- gest that it is preferable to use SMAC and TPE when dealing with large and conditional search spaces, whereas Spearmint is recommended for low-dimensional and continuous problems (Eggensperger et al. [2013](#bookmark207)).

*Optuna* (Akiba et al. [2019](#bookmark208)) is an HPO system built upon TPE. Earlier optimisation frameworks, such as SMAC and HyperOpt, only allow a static defnition of the hyperpa- rameter space and cannot be used when no full description of the hyperparameter space is given by the user. Optuna, however, provides a defne-by-run API that allows users to dynamically defne and modify the search space. It also includes multi-fdelity strategies to speed up the optimisation process. Optuna covers several multi-fdelity strategies for per- formance evaluation, e.g., the asynchronous successive halving algorithm (Li et al. [2020a](#bookmark209)) and hyperband (Li et al. [2017](#bookmark160)).

*Bayesopt* (Martinez-Cantin [2014](#bookmark210)) is a fexible framework that supports the optimisa- tion of continuous, discrete and categorical hyperparameters. It allows users to select from many components relevant to Bayesian optimisation, such as the initialisation procedure, the acquisition function, the optimisation of the acquisition function, the surrogate model and the evaluation metric. Furthermore, Bayesopt implements an improvement to calculate the covariance matrix of Gaussian processes that, at each iteration, determines how many new elements will appear in the covariance matrix and how many will remain the same. This reduces the computational complexity of computing this matrix from O(n3 ) to O(n2 ).

*RoBO* (Klein et al. [2017b](#bookmark211)) is a package that implements various HPO algorithms based on Bayesian optimisation, including several methods focusing on multi-fdelity and

a 灬

auxiliary tasks, such as multi-task Bayesian optimisation (Swersky et al. [2013](#bookmark212)), Boham- ian (Springenberg et al. [2016](#bookmark213)), and Fabolos (Klein et al. [2017a](#bookmark203)).

*BOHB* (Falkner et al. [2018](#bookmark192)) implements, in addition to the equally named BOHB algo- rithm, various relevant baseline methods, such as successive halving and hyperband. The BOHB package supports parallel computing and aims to address various practical prob- lems that arise when running HPO algorithms in parallel on multiple CPUs.

*MOE* (Yelp [2014](#bookmark214)) (metric optimisation engine) is an HPO package based on Bayesian optimisation implemented in Python. It uses expected improvement as an acquisition func- tion and Gaussian processes as a surrogate model.

*Scikit-optimize* (Head et al. [2017](#bookmark215)) implements a sequential model-based approach to optimisation. It supports several methods, including sequential optimisation with decision trees/gradient boosted trees and Bayesian optimisation with Gaussian processes. For acqui- sition functions, it supports expected improvement, lower confdence bound, and probabil- ity of improvement.

*Optunity* (Claesen et al. [2014](#bookmark216)) covers several of the previously mentioned optimisers for HPO, including grid search, random search, particle swarm optimisation and CMA-ES.

*Syne Tune* (Salinas et al. [2022](#bookmark217)) is a package for distributed HPO. It includes a range of optimisers (including random search, Bayesian optimisation and evolutionary search) and multi-fdelity approaches (including BOHB and hyperband). In addition, it also supports more advanced methods for hyperparameter transfer learning, constrained HPO and multi- objective optimisation.

**3.5 HPO benchmarks**

The common way to create an HPO benchmark is to select a collection of datasets and apply a set of state-of-the-art methods to these. For a long, this has been the standard prac- tice in the feld. However, there are also drawbacks to this approach. Bischl et al. ([2021](#bookmark86)) argued that this makes it hard to compare results across studies, gives rise to cherry-picking results and leads to a publication bias. They addressed this by utilising OpenML to create benchmark suites. Benchmark suites are a collection of tasks designed to thoroughly evalu- ate (machine learning) algorithms. They further argue that good benchmarking is done on a large collection of tasks. They integrate their approach with OpenML (Vanschoren et al. [2013](#bookmark85)), in order to make use of the API and eco-system of OpenML, making it easy for users to create benchmark suites and to run algorithms of interest on these. By defning a standard set of benchmark tasks, the research community is encouraged to use the same type of benchmarks for similar types of algorithms. This makes it easier to compare results across studies and harder to cherry-pick results. Of course, there can be valid reasons to use a specifc subset of benchmarks, for instance, when a new algorithm is conjectured to address issues arising in the context of solving the respective tasks.

Bischl et al. ([2021](#bookmark86)) introduce two benchmark suites, OpenML100 and its successor, OpenML-CC18. OpenML Curated Classifcation 2018 (OpenML-CC18) is a collection of machine learning tasks that are carefully selected and curated from OpenML. For example, for each of the tasks, it has been verifed that a corresponding scientifc publication (or other documentation) existed, it has been determined whether the original task was indeed a classifcation task (and not, for example, a binarised regression task) and whether the data was real-world data (to exclude artifcial datasets).

Gijsbers et al. ([2019](#bookmark218)) developed the AutoML benchmark, by extending OpenML- CC18. The AutoML benchmark focusess on bigger and more complex datasets as the

a 灬

OpenML-CC18 does, and might for that reason be a better ft to benchmark AutoML systems. Beyond an extended benchmark suite, the authors also developed a platform on which AutoML systems can be uploaded. These will then be automatically evaluated on the benchmark, and compared with earlier uploaded benchmark suites.

Eggensperger et al. ([2021](#bookmark219)) developed HPOBench, a workbench for benchmarking HPO methods. HPOBench provides several HPO problems, including the optimisation of the hyperparameters of a support vector machine, gradient boosting, random forest and a mul- tilayer perceptron. HPOBench supports multi-fdelity approaches (see Sect. [3.3](#bookmark61)), which makes it a good choice for evaluating such methods. HPOBench includes so-called raw, tabular and surrogate benchmarks. Where the raw benchmark contains just the dataset and task defnition, implying that the actual confgurations need to all be ran (incurring a com- putational cost), the tabular benchmark contains pre-computed evaluations for all possible hyperparameter confgurations in the search space so that using it only incurs minor com- putational overhead. A surrogate benchmark also contains pre-computed evaluations, but does not do this for all possible confgurations (as the confguration space might be too large or even infnite). Instead, a surrogate model (Eggensperger et al. [2015](#bookmark82)) is trained and subsequently used to replace the actual function evaluation. Therefore, when running a sur- rogate benchmark, no actual evaluations will be performed. Overall, tabular and surrogate benchmarks provide a convenient way of speeding up the development of HPO algorithms by allowing the algorithm designer to test-run the HPO algorithm on various benchmarks with little computational efort.

**4 Neural architecture search**

In this section, we turn our attention to AutoML systems for automatically designing deep neural network architectures. Conventionally, neural networks are represented in the form of computational graphs (Goodfellow et al. [2016](#bookmark220)) of nodes that perform operations (e.g., addition, convolution, pooling, activation) on the input they receive from their parent nodes. The architecture of a neural network represents the parents of each node (i.e., struc- ture or node connections), as well as the operations performed by the nodes.

Training a neural network requires setting two sets of hyperparameters. The frst group are *training hyperparameters* that mainly afect the training process. These are hyperpa- rameters such as the learning rate, optimiser type or batch size. The second group, how- ever, are *architectural hyperparameters* that defne the network architecture, such as the number, sizes and operations of layers. Neural Architecture Search (NAS) research mainly considers optimising the latter category of hyperparameters. Therefore, in the rest of this section, the term hyperparameter mainly denotes architectural hyperparameter.

Network architecture engineering is still a time-consuming and expensive task that needs to be performed manually by experts. NAS methods aim at fnding good architec- tures by optimising architectural hyperparameters. Conceptually, NAS is a sub-feld of AutoML that builds, to a great extent, on the hyperparameter optimisation (HPO) tech- niques discussed in Sect. [3](#bookmark37). We can frame NAS as an optimisation problem with the goal of fnding an architecture that achieves the best possible performance in the target task within a predefned search space. We note that each layer within the network architecture defnes new hyperparameters to be set for its operations, which leads to a tree-structured

a 灬

search space. This makes NAS more complex than HPO for classic machine-learning algo- rithms without conditional hyperparameters.

Neuro-evolution of augmenting topologies (NEAT) (Stanley and Miikkulainen [2002a](#bookmark221)) is one of the earlier approaches proposed in the early 2000s, aiming at automating the pro- cess of designing neural network architectures by jointly optimising network topology and parameters. NEAT is based on the idea of evolving the neural network architecture (struc- ture and connection weights) using a genetic algorithm. However, being designed to work on the level of single neurons, this approach does not scale to deep network architectures with millions of neurons and diferent layer types. Zoph and Le ([2017](#bookmark128)) were among the frst who considered the idea of ‘neural architecture search’ with the goal of automating the design of modern deep neural networks. This initial attempt at NAS relied on the availabil- ity of very substantial computational resources (800 GPUs for 4 weeks). Many diferent approaches have been proposed to make NAS more computationally efcient while still attaining high performance.

NAS methods can be described in terms of the three components introduced in the con- text of HPO in Sect. [2.2](#bookmark40) (Elsken et al. [2019b](#bookmark17)). In NAS, *search spaces* can comprise archi- tectural and training hyperparameters. The efcient design of a search space using prior expert knowledge on the type of networks that perform best for a given class of problems can largely improve the efciency of the search. The *search strategy* determines how to fnd good hyperparameter confgurations (and hence, a good architecture) within a given, possibly vast, search space. *Performance estimation approaches* are used to decide which confgurations will achieve high performance on a given dataset without the need to per- form potentially very time-consuming, full training and validation.

In the following, we review prominent and important NAS methods focusing on the underlying search space design in Sect. [4.1](#bookmark222), and search strategy, in Sects. [4.2](#bookmark46) and [4.3](#bookmark223) cov- ers various performance evaluation approaches proposed for NAS. Finally, we will discuss available NAS libraries and benchmarks in Sects. [4.4](#bookmark224) and [4.5](#bookmark225), respectively.

Table [2](#bookmark226) gives an overview of prominent NAS methods based on their underlying search space and search strategy. As seen in the table, we distinguish micro-level, macro-level, and hierarchical search space design approaches. Widely used search strategies include methods based on reinforcement learning, Bayesian optimisation, gradient-based and evolutionary algorithms. We note that there are methods such as Lemonade (Elsken et al. [2019a](#bookmark227)) and RENAS (Chen et al. [2019c](#bookmark228)) that appear under more than one category in our table as their search strategy or search space covers more that one approach.

In what follows, we will describe the NAS techniques listed in Table [2](#bookmark226) in more detail.

**4.1 Search space**

The search space of a given NAS method represents the space of all possible neural net- work architectures. However, searching within a vast space of all possible hyperparam- eter settings is an extremely computationally expensive task. Most research in search space design has focused on imposing simplifying constraints to reduce the number of architec- tural hyperparameter confgurations. Two aspects have been considered in designing the search space of neural network architectures: (1) the design of the global architecture of the network, and (2) the design of sub-architectures that can be repeated to create a full neural network in a modular fashion. This directly leads to the concepts of macro-level and micro- level searches. The macro-level search considers optimising the entire network by search- ing for the operations and connections between layers (see, e.g., Zoph and Le [2017](#bookmark128); Pham

a 灬

Automated machine learning: past, present and future Page 35 of 88 **122**

**Table 2** An overview of NAS methods categorised based on the search strategy and the search space

a 灬

|  |  |  |
| --- | --- | --- |
| Search space | Search strategy | Name |
| Micro-level | Reinforcement learning Bayesian optimisation  Evolutionary algorithm  Monte Carlo tree search Gradient descent | Path-level transformation (Cai et al. [2018b](#bookmark229)), NASNet (Zoph et al. [2018](#bookmark230)), ENAS (Pham et al. [2018](#bookmark231)), Block- QNN (Zhong et al. [2018](#bookmark232)), RENAS (Chen et al. [2019c](#bookmark228), [d](#bookmark233)), FPNAS (Cui et al. [2019](#bookmark234))  PNAS (Liu et al. [2018a](#bookmark9)), BOGCN-NAS (Shi et al. [2020](#bookmark235)), NAS-BOWL (Ru et al. [2020a](#bookmark236)), Auto-pytorch (Zimmer et al. [2021](#bookmark195)) CSNAS (Nguyen and Chang [2022](#bookmark237)), BANANAS (White et al. [2021a](#bookmark238)), HOTNAS (Yang et al. [2023](#bookmark239))  CoDeepNEAT (Miikkulainen et al. [2019](#bookmark240)), AmoebaNet-A (Real et al. [2019](#bookmark241)), AmoebaNet-B (Real et al. [2019](#bookmark241)), Lemonade (Elsken et al. [2019a](#bookmark227)), RENAS (Chen et al. [2019c](#bookmark228)), MONCAE (Dimanov et al. [2021](#bookmark242)),  OSNAS (Zhang et al. [2022a](#bookmark243)), DFG-NAS (Zhang et al. [2022b](#bookmark244)) AlphaX (Wang et al. [2020](#bookmark245))  MaskConnect (Ahmed and Torresani [2018](#bookmark246)), GHN (Zhang et al. [2019a](#bookmark247)) NAO (Luo et al. [2018](#bookmark248)) DARTS (Liu et al. [2019b](#bookmark159)), Proxyless (Cai et al. [2019](#bookmark249)), SharpDARTS (Hundt et al. [2019](#bookmark250)), PDARTS (Chen et al. [2019b](#bookmark251)),  DARTS+ (Liang et al. [2019](#bookmark252)), PCDARTS (Xu et al. [2020](#bookmark253)), SNAS (Xie et al. [2019](#bookmark254)), FBNET (Wu et al. [2019](#bookmark255)), FAIR DARTS (Chu et al. [2020](#bookmark256)), DROP NAS (Hong et al. [2020](#bookmark257)), ABS (Hu et al. [2020](#bookmark258)), GDAS (Zhang et al. [2020](#bookmark259)), DOTS (Gu et al. [2021](#bookmark260)), IDARTS (Xue et al. [2021](#bookmark261)), EC-DARTS (Zhou et al. [2021](#bookmark262)), TNASP (Lu et al. [2021](#bookmark263)), MR-DARTS (Gao et al. [2022](#bookmark264)), B-DARTS (Ye et al. [2022](#bookmark265)), LISSNAS (Gopal et al. [2023](#bookmark266)) |
| Macro-level | Random search  Reinforcement learning  Bayesian optimisation Evolutionary algorithm  Monte Carlo tree search Gradient descent | RS NAS (Li and Talwalkar [2019](#bookmark267)), RandomNAS (Zhang et al. [2020](#bookmark259))  AutoNET (Mendoza et al. [2016](#bookmark145)), RL NAS (Zoph and Le [2017](#bookmark128)), MetaQNN (Baker et al. [2017](#bookmark124)), Deep-  Architect (Negrinho and Gordon [2017](#bookmark268)), ENAS (Pham et al. [2018](#bookmark231)), Net transformation (Cai et al. [2018a](#bookmark269)), PROXYLESS-NAS (Cai et al. [2019](#bookmark249))  DeepArchitect (Negrinho and Gordon [2017](#bookmark268)), Auto-Keras (Jin et al. [2019](#bookmark12)), NASBOT (Kandasamy et al. [2018](#bookmark270)), CSNAS (Nguyen and Chang [2022](#bookmark237); Nguyen et al. [2021](#bookmark271))  GeNet (Xie and Yuille [2017](#bookmark272); Real et al. [2017](#bookmark273)), CGP-CNN (Suganuma et al. [2018](#bookmark274)), NASH (Elsken et al. [2018](#bookmark275)), Neuro-cell-based evolution (Wistuba [2018](#bookmark276)), Lemonade (Elsken et al. [2019a](#bookmark227); Irwin-Harris et al. [2019](#bookmark277))  Monte Carlo planning (Wistuba [2017](#bookmark278))  Smash (Brock et al. [2018](#bookmark279)), TAS (Dong and Yang [2019](#bookmark280)) |

a 灬

**Table 2** (continued)

**122** Page 36 of 88 M Baratchi et a l

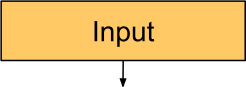
|  |  |  |
| --- | --- | --- |
| Search space | Search strategy | Name |
| Hierarchical | Reinforcement learning Evolutionary algorithm Gradient descent  Bayesian optimisation | MnasNet (Tan et al. [2019](#bookmark281))  Hierarchical (Liu et al. [2018b](#bookmark282))  (Shin et al. [2018](#bookmark283)), Auto-DeepLab (Liu et al. [2019a](#bookmark284)) NAGO (Ru et al. [2020b](#bookmark285)) |

et al. [2018](#bookmark231); Kandasamy et al. [2018](#bookmark270); Brock et al. [2018](#bookmark279)). The micro-level search focuses on optimising cells or blocks (see, e.g., Liu et al. [2018a](#bookmark9), [2019b](#bookmark159)) that will be stacked to con- struct the fnal network. There are also approaches that consider a hierarchical search space (see, e.g., Liu et al. [2018b](#bookmark282), [2019a](#bookmark284); Zhong et al. [2018](#bookmark232); Tan et al. [2019](#bookmark281)) by making use of a combination of the two previously mentioned approaches that leads to a hierarchically structured search space. In the following sections, we elaborate on these approaches.

**4.1.1 Macro‑level search spaces**

Macro-level search considers generating the entire structure of the network. Designing the architecture in a layer-wise manner greatly reduces the degrees of freedom within the global search space; at the same time, it typically still allows for an arbitrary sequence of layers, each with its own architectural hyperparameters. Macro-level search spaces typi- cally include hyperparameters such as the number of layers, conditional hyperparameters, such as the type of each layer (e.g., convolution, pooling) and the hyperparameters of these operations (e.g., flter size and stride of a convectional layer).

Previously, Baker et al. ([2017](#bookmark124)),Zoph and Le ([2017](#bookmark128)), Kandasamy et al. ([2018](#bookmark270)) and Brock et al. ([2018](#bookmark279)) have considered this type of search space by generating new networks (with a pre-defned maximum number of layers) in each iteration of the search process. Baker et al. ([2017](#bookmark124)) generated the network architecture by iteratively searching within the space of architectural hyperparameters for each layer (e.g., number of flters, flter height, and flter width). Zoph and Le ([2017](#bookmark128)) further proposed to widen the search space by allowing skip connections and branching layers. Figure [6](#bookmark286) shows two diferent examples of chain- structured networks that make up this type of search space. The network shown on the left is a simple example where every layer receives its input from the previous layer and transfers its outputs to the next. The network shown on the right has a more complex struc- ture, including multiple branches and skip connections. In a layer-wise architecture, the space of possible networks increases exponentially with the possible number of layers. Xie and Yuille ([2017](#bookmark272)) imposed further limitations on the design of layer-wise architectures by defning a search space of networks with a limited number of stages, where each stage is composed of a number of pre-defned layers.





















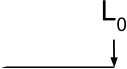
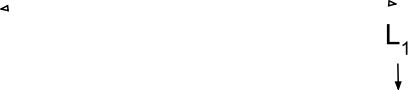
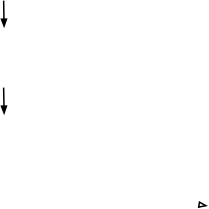
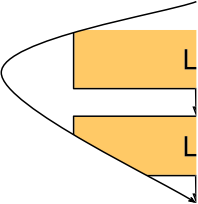
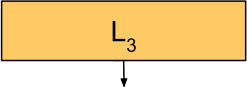








|  |
| --- |
|  |



|  |
| --- |
|  |

|  |
| --- |
|  |

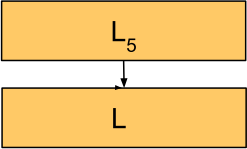
|  |
| --- |
| 2 |



|  |  |
| --- | --- |
|  |  |



|  |  |
| --- | --- |
|  |  |

7



|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| output |

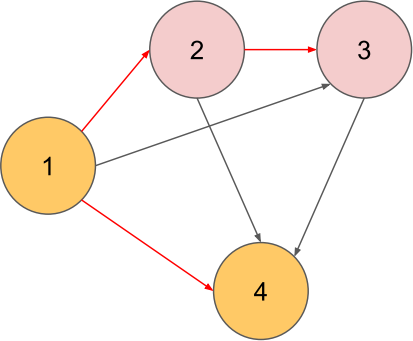
**Fig. 6** Examples of chain-structure neural networks. Each Li in the graph indicates a layer with a specifc operation (Elsken et al. [2019b](#bookmark17))

a 灬

**4.1.2 Micro‑level search spaces**

A macro-level search space provides fexibility in terms of defning a network by consider- ing a large set of architectural hyperparameters. However, by having a large search space, the task of fnding a good architecture will become computationally expensive in terms of training time and other resources. Inspired by the observation that well-known convolu- tional neural network (CNN) architectures such as ResNet and Inception include repeated motifs (Zoph et al. [2018](#bookmark230)), the second group of algorithms has considered a micro-level approach defned based on cell or block structures.

*Cell-based search* Cell structures are, in essence, mini-architectures composed of a number of layers and operations. These cells will be iteratively stacked to create a larger architecture. Searching for the best architecture will then be reduced to searching for the best cell structure (Zoph et al. [2018](#bookmark230)); the cells creating a larger network will all have the same architecture but diferent weights. In principle, by imposing additional structure on the search space, searching within a cell-based space is much simpler than searching within the space of all possible network structures. This structure will impose a limit on the maximum achievable performance by cell-based approaches. However, as shown by Zoph et al. ([2018](#bookmark230)), the micro-level search can still achieve higher accuracy than macro-level search in a much shorter amount of time by using better initial models for the cell search. The cell-based architecture can also potentially generalise better to other problems and thus allow better transfer of knowledge across datasets. The full networks designed using NAS in macro-level search spaces are task- and data-specifc, and they are typically difcult to transfer to other datasets when the input data sizes are diferent. In a cell-based structure, better transferability can be achieved by adding more downsampling operations before cells (Zhong et al. [2018](#bookmark232)) or by adding more copies of the cell (Zoph et al. [2018](#bookmark230)).

In micro-search, it is common to formalise the NAS search space of a cell as a directed acyclic graph (DAG) where nodes represent local operations, and directed edges represent the fow of information (see, e.g., Liu et al. [2018a](#bookmark9); Luo et al. [2018](#bookmark248); Pham et al. [2018](#bookmark231)). Cells typically have two inputs and a single output node. In convo- lutional cells, input nodes of the cell are acquired from the previous two layers of the network. In recurrent cells, the input nodes are defned based on the current state and the previous one (Liu et al. [2019b](#bookmark159)). Figure [7](#bookmark288) represents an example of a recurrent cell with four computational nodes.

|  |
| --- |
| ReLU |

|  |
| --- |
| ReLU |



avg

|  |
| --- |
| tanh (1) |

|  |
| --- |
| h[t-1] |

|  |
| --- |
| tanh (4) |

**Fig. 7** An example of a recurrent cell in a micro-level search space. Left: The search space is represented in the form of a DAG. The order of operations is represented by the red arrows. Right: The recurrent cell created by taking the subgraph of the DAG on the left induced by the red edges (Pham et al. [2018](#bookmark231)). In Sect. [4.3.1](#bookmark289), we explain how these subgraphs are used for improving the performance evaluation. (color fg- ure online)

a 灬

NASNET (Zoph et al. [2018](#bookmark230)), ENAS (Pham et al. [2018](#bookmark231)), DARTS (Liu et al. [2019b](#bookmark159)), SNAS (Xie et al. [2019](#bookmark254)), and PNAS (Liu et al. [2018a](#bookmark9)) are examples of NAS approaches based on cell-level search spaces. NASNET (Zoph et al. [2018](#bookmark230)) is one of the frst approaches in the design of cell-based NAS. NASNET focuses on NAS in image clas- sifcation by making use of a search space based on modular convolutional cells; *nor- mal cells* (with output and input in the same dimension), and *reduction cells* (with the output dimension being half of the input dimension, in both height and width). Based on the combination of these two types of cells, architectures can be built for process- ing images of any size. The structures of these cells can be searched to identify which operations are applicable to hidden states within cells and how to combine the outputs of pairs of hidden states into a new one. The ‘normal cell, reduction cell’ structure has been adopted by other researchers for CNNs (Real et al. [2019](#bookmark241); Liu et al. [2018a](#bookmark9), [2019b](#bookmark159)), along with additional simplifcations to the cell structure. For instance, PNAS (Liu et al. [2018a](#bookmark9)) formalises cell-structures for CNN NAS that can implicitly emulate the normal and reduction cells mentioned earlier; by pruning a number of operations that were not selected from the search space considered by Zoph et al. ([2018](#bookmark230)), a much smaller search space is obtained that can be searched more efciently.

Efcient neural architecture search (ENAS) (Pham et al. [2018](#bookmark231)) is another impactful search space that formalises both CNN and RNN cells along with an approach for weight sharing to speed up the performance evaluation (explained in Sect. [4.3.1](#bookmark289)). More generic computational cells have been proposed by Liu et al. ([2019b](#bookmark159)).

Some of the micro-level search spaces have used cell-level search within spaces of predefned chain-structured architectures (see, e.g., Liu et al. [2018a](#bookmark9), [2019b](#bookmark159); Pham et al. [2018](#bookmark231)). This approach initially tries to fnd a good cell structure by considering the connec- tion topology and operations tied to each connection. Next, a fxed number of such cells is stacked in a chain-structured network. In order to go beyond simple chain structures and beneft from multi-path structures (which are commonly used in the state-of-the-art CNNs) Cai et al. ([2018b](#bookmark229)) introduced a path-level network transformation operation per- mitting modifcations of the path topology of a given network that defnes the connection paths between layers. Using the path-level operations, Cai et al. ([2018b](#bookmark229)) constructed a gen- eralised multi-branch tree-structured search space that can encode predefned multi-branch structures based on advanced human-designed architectures [e.g., ResNets (He et al. [2016](#bookmark290)), DenseNets (Huang et al. [2017](#bookmark291)),PyramidNet (Han et al. [2017](#bookmark292))]. This approach signifcantly improves the efciency of the cell design compared to a simple chain structure.

*Block-based search* The networks obtained by repeatedly reusing optimised cells are of limited diversity. To overcome this limitation, blocks (i.e., cells with diverse structures) can be optimised. FPNAS (Cui et al. [2019](#bookmark234)) optimises various diferent blocks to generate the full network. This is done using a bi-level optimisation problem, where each block is optimised separately, while keeping all other blocks fxed. Similarly, FBNET (Wu et al. [2019](#bookmark255)) and ProxylessNAS (Cai et al. [2019](#bookmark249)) support layer-wise search for blocks to increase the diversity of networks found. Chen et al. ([2019d](#bookmark233)) proposed to use eight diferent types of blocks based on well-known network architectures such as ResBlocks and Inception.

**4.1.3 Hierarchical search spaces**

Hierarchical search spaces combine micro-level and macro-level approaches to improve layer diversity. One way to create a hierarchical search space is by means of recursion. For instance, Liu et al. ([2018b](#bookmark282)) proposed an approach starting from lower-level motifs as a

a 灬

|  |
| --- |
| stride width |

|  |
| --- |
| Filter Height |

|  |
| --- |
| Filter width |

|  |
| --- |
| stride Height |

|  |
| --- |
| Number of Filters |

|  |
| --- |
| Filter Height |

|  |
| --- |
| Number of Filters |

A



A

A

A

A

|  |
| --- |
|  |

|  |
| --- |
|  |

A



Layer N-1 Layer N Layer N+1

**Fig. 8** The controller is implemented in the form of an RNN. Here, this controller samples a feedforward neural network with only convolutional layers (empty squares) and predicts an architectural string com- posed of the hyperparameters of the convolutional layers (flter height, flter width, stride height, stride width and the number offlters) (Zoph and Le [2017](#bookmark128))

small set of primitive operations (convolution, depth-wise convolution, separable convolu- tion, max-pooling) at the bottom level of the hierarchy, from which higher-level motifs are then built recursively, such that the highest-level motif corresponds to the full architec- ture. At each level of this hierarchy, motifs are represented in the form of DAGs. A similar approach has been proposed by Ru et al. ([2020b](#bookmark285)) to create a three-level hierarchy: At the top level, there is a graph of cells. At the middle level, each cell is represented as a graph of nodes. At the bottom level, each node is represented as a graph of operations. By varying the graph generator hyperparameters at each level, a diverse range of architectures can be obtained.

MnasNet (Tan et al. [2019](#bookmark281)) makes use of a diferent hierarchical search space. In this approach, a full CNN architecture is factorised into diferent segments, each comprising a number of identical layers. For each segment, the operations and connections of a single layer, as well as the number of layers, are optimised. The optimised layer will be repeated to create a full segment. This approach was inspired by the idea that diferent parts of the network should be treated diferently, as they play diferent roles in the overall accuracy and inference latency (e.g., in a CNN architecture, earlier blocks impact the inference latency more as they process larger input sizes). Liu et al. ([2019a](#bookmark284)) took a diferent strat- egy in proposing a bi-level hierarchical search space that allows selecting the network-level structure by searching within a space of popular network designs.

**4.2 Search strategy**

After the search space has been decided, a search strategy is needed to fnd the best archi- tecture within this space. In the following, we give an overview of search strategies used in NAS.

**4.2.1 Reinforcement learning**

NAS can be addressed using reinforcement learning. In this case, an agent’s goal is to gen- erate a network from the action space of the search space. As outlined in Sect. [3](#bookmark37), the pol- icy-gradient approach is a well-known reinforcement learning technique that relies on opti- mising policies with respect to rewards. Using a policy-gradient-based method for NAS to

a 灬

design CNNs and RNNs was frst proposed by Zoph and Le ([2017](#bookmark128)). In this approach, the structure and connectivity of the elements of the neural network are encoded in the form of an architectural string (as illustrated in Fig. [8](#bookmark293)). These architectural strings are composed of a set of tokens (for CNNs, these are architectural hyperparameters such as flter height, fl- ter width, stride height, stride width, and the number offlters for one layer). The controller is implemented as an RNN that will generate a child network by predicting the hyperpa- rameters of such an architectural string one by one and stopping when a certain pre-defned number of layers has been generated. The hyperparameters that are predicted by the con- troller can be considered as a list of actions performed in the design of an architecture, and the reward corresponds to the accuracy achieved by the child network. The policy gradi- ent is calculated to update the controller using this reward signal with the aim of maxim- ising the expected accuracy of the generated architectures. The REINFORCE algorithm (Williams [1992](#bookmark126)) mentioned earlier in Sect. [3.2.3](#bookmark121) is used to optimise the parameters of the controller.

MetaQNN (Baker et al. [2017](#bookmark124)) and BlockQNN (Zhong et al. [2018](#bookmark232)) use Q-Learning, the other popular reinforcement learning algorithm. MetaQNN (Baker et al. [2017](#bookmark124)) defnes states as a group of hyperparameters and uses a learning agent to generate layers one by one, until the entire network has been generated. This approach assumes that a well-per- forming layer in one network will also perform well in another one. The generated network is then trained, and its accuracy is used as a reward for the agent. BlockQNN (Zhong et al. [2018](#bookmark232)) defnes an action space that allows block-wise (as opposed to layer-wise) network generation with an agent that is trained to choose layers within a block.

ENAS (Pham et al. [2018](#bookmark231)) and NASNet (Zoph et al. [2018](#bookmark230)) are examples of reinforce- ment learning NAS approaches with an action space that allows generating architectural cells (explained earlier in Sect. [4.1.2](#bookmark287)).

Defning an action space based on function-preserving transformations allows to gen- erate new architectures by transferring knowledge from previously trained networks and thus speeding up the evaluation process. Layer-level architecture transformations (Cai et al. [2018a](#bookmark269)) allow actions such as adding flters or layers, while path-level transformations (Cai et al. [2018b](#bookmark229)) allow actions that modify the path topology of the network.

**4.2.2 Bayesian optimisation**

As outlined in Sect. [3.2.2](#bookmark106), Bayesian optimisation employs a surrogate model and an acqui- sition function in the optimisation process. For hyperparameter optimisation, Gaussian processes and random forests are the two commonly used surrogate models and expected improvement is a popular acquisition function.

Some of the design choices in the NAS search space cannot be encoded in the form of continuous variables (e.g., the number of layers or types of activation functions). While Gaussian processes in their basic form are only applicable to continuous variables, by using newly developed kernel functions, they can also handle categorical and conditional hyperparameters (Swersky et al. [2014a](#bookmark295)). For instance, to address the NAS problem using Gaussian processes new distance metrics have been proposed for measuring the similarity between pairs of network architectures (see, e.g., Kandasamy et al. [2018](#bookmark270); Jin et al. [2019](#bookmark12)). Typically, these metrics are implemented in the form of edit distance between architec- ture encoding. For instance, Auto-Keras (Jin et al. [2019](#bookmark12)) proposes a Bayesian optimisa- tion approach that uses Gaussian processes as a surrogate model combined with a distance metric based on network morphism (Wei et al. [2016](#bookmark296)), which allows to morphologically

a 灬

transform the architecture of a neural network while keeping its functionality. The Auto- Keras framework uses edit-distance neural network kernels to estimate how many opera- tions need to be performed to change (morph) one neural network into another. GP-NAS (Li et al. [2020c](#bookmark297)) is another Gaussian process-based technique that uses a customised kernel function to measure the similarity between architecture encodings. Since relevant opera- tions (e.g., the number of channels, kernel size) considered for creating these encodings are diverse in type and incomparable, the kernel function used in GP-NAS works on groups of operations rather than on individual operations. NASBOT (Kandasamy et al. [2018](#bookmark270)) defnes new kernel functions, referred to as optimal transport metrics for architectures of neural networks (OTMANN), to measure the similarity between two neural networks. OTMANN measures the distance between neural networks using three factors that determine the per- formance of a neural network: (1) the operations performed at each layer, (2) the types of these operations, and (3) how the layers are connected. The OTMANN distance can be computed by solving an optimal transport problem, a well-studied optimisation problem for which several efective solvers exist (Peyré and Cuturi [2019](#bookmark298)). Yang et al. ([2023](#bookmark239)) pro- pose a hierarchical optimal transport metric to measure the cell-level similarity (consider- ing the similarity of nodes and information fow costs between node pairs) and network- level similarity (considering the cell-level similarity and the global position of each node within the network).

Ru et al. ([2020a](#bookmark236)) proposed Gaussian process-based Bayesian optimisation using graph kernels (Weisfeiler-Lehman subtree graph kernel) that can be applied to ENAS (Pham et al. [2018](#bookmark231)) cells (explained earlier in Sect. [4.1](#bookmark222)). Yang et al. ([2023](#bookmark239)) Neural predictor surro- gate models have also been used as the surrogate model of Bayesian optimisation for NAS; these are neural networks that are repeatedly trained on the architectures under evaluation to predict the accuracy achieved by previously unseen architectures. BANANAS (White et al. [2021a](#bookmark238)) uses neural predictors in combination with Bayesian optimisation. To make the Bayesian optimisation process more efcient, a variant of Thompson sampling (Thompson [1933](#bookmark299)) is used as the acquisition function; Thompson sampling is a heuristic technique for balancing exploration and exploitation, which is a crucial element of Bayes- ian optimisation. White et al. ([2021a](#bookmark238)) propose a new variant of Thompson sampling called *independent Thompson sampling*, which allows parallel Bayesian optimisation runs. Two options have been used for implementing neural predictors: MLPs and graph neural net- works. Compared to MLPs, graph neural networks, as studied by Shi et al. ([2020](#bookmark235)), are better suited to capture the topological structure of neural networks represented in the form of DAGs, and they can handle a variable number of nodes and scale to larger input archi- tectures. They, however, require large amounts of training data (in the form of empirically evaluated architectures).

Another approach to using Bayesian optimisation for NAS is using tree-based models (see Sect. [3.2.2](#bookmark106)), which do not require defning a distance function and can easily handle conditional and categorical hyperparameters. Auto-Net 1.0 and Auto-Net 2.0 (Mendoza et al. [2016](#bookmark145)) are two NAS approaches based on this idea. Auto-Net 1.0 uses the random- forest-based Bayesian optimisation system SMAC (Hutter et al. [2011](#bookmark43)) as an optimiser, and Lasagne (LasagneContributors [2022](#bookmark300)) as a deep-learning library. Auto-Net 1.0 is an exten- sion of auto-sklearn (which will be introduced in Sect. [5](#bookmark39)) (Feurer et al. [2015](#bookmark10)). Like auto- sklearn, it supports feature extraction, data preprocessing, and ensemble modelling. How- ever, fully-connected feed-forward neural networks are the only machine-learning models supported by Auto-Net 1.0. In Auto-Net 2.0, BOHB (Falkner et al. [2018](#bookmark192)), a combination of TPE and Hyperband (previously described in Sect. [3](#bookmark37)), is used as an optimiser, and PyTorch (LinuxFoundation [2022](#bookmark301)) is used instead of Lasagne. Similar to Auto-Net 1.0, Auto-Net

a 灬

2.0 covers all the preprocessing algorithms of auto-sklearn. Auto-pytorch tabular (Zimmer et al. [2021](#bookmark195)) further extends Auto-Net 2.0, targeting deep learning on tabular datasets by incorporating meta-learning and efcient micro-level search space.

**4.2.3 Evolutionary algorithms**

Evolutionary algorithms (see Sect. [3.2.4](#bookmark131)) have been used for evolving neural networks (a.k.a. neuro-evolution) for over two decades (see, e.g., Yao [1999](#bookmark302); Stanley and Miikku- lainen [2002a](#bookmark221), [b](#bookmark303); Angeline et al. [1994](#bookmark304)). Neuro-evolution focuses on jointly optimising the weights and hyperparameters (architectural and training hyperparameters) using evolution- ary algorithms. This approach only scales to small and medium-scale neural networks. Modern evolutionary NAS (EvNAS) research separates the two optimisation problems by employing evolutionary algorithms only for hyperparameter optimisation and leaving the optimisation of network weights to gradient-based optimisation, which is the recom- mended approach for training deep models (Bengio [2012](#bookmark305)). In the following, we briefy dis- cuss relevant aspects of both of these research lines. For a comprehensive survey, we refer interested readers to the survey by Liu et al. ([2021](#bookmark31)).

Before adding a network to the population of candidate solutions, we need to decide on genome encoding. The evolutionary algorithm will further modify the genome of individuals through mutation and crossover operations. There are two types of encod- ing approaches, direct and indirect. In direct encodings, parameters to be optimised are directly presented in a genome. In indirect encodings, a transformation is used to interpret the neural network from a genome (Templier et al. [2021](#bookmark306)). NEAT (Stanley and Miikku- lainen [2002a](#bookmark221)) is an example of a neuro-evolutionary approach that uses the direct encoding approach to design multilayer perceptrons (MLPs). In NEAT, both node and connection genes are directly encoded into genomes as a mixture of binary, discrete or continuous var- iables. The connection genes specify for each node the in- and out-node connections, the weight of those connections, whether the connection is expressed, and the innovation num- ber (which is meant to help keep topological innovations protected for a few generations before they disappear again from the population). The genome can be modifed through three types of mutation operations: (1) changing the weights of nodes, (2) adding con- nections between nodes, and (3) inserting a new node between a connection. By working on the level of single neurons, the direct encoding approach cannot scale to automatically design deep neural network architectures.

Indirect encoding schemes are later proposed to address this issue, using transforma- tions or generation rules for creating architectures in a more compact manner. Miikkulainen et al. ([2019](#bookmark240)) proposed an extension of NEAT for deep networks using an indirect encoding that allows each node in a genome to represent an entire layer rather than a single neuron. Similarly, HyperNEAT (Stanley et al. [2009](#bookmark307)) proposes an indirect encoding approach called connective compositional pattern-producing networks (CPPN), to create repeating motifs that represent spatial connectivity patterns as functions in Cartesian space.

One of the frst EvNAS approaches proposed by Real et al. ([2017](#bookmark273)) uses an indirect encoding scheme for representing simple single-layered networks with no convolutions, which are evolved into a far more complex network with high performance. Compared to NEAT, each mutation instead of changing one node can insert/remove layers of hundreds of nodes. Initiating the evolutionary process with trivial networks, as proposed by Real et al. ([2017](#bookmark273)) makes the process of fnding a good network slow. To speed up the search for better architectures, Liu et al. ([2018b](#bookmark282)) proposed a diversifcation scheme based on design

a 灬

patterns defned by human experts to initialise the search. They further proposed a hierar- chical encoding scheme and an action space for mutating hierarchical genotypes, where lower-level motifs are used as building blocks for constructing higher-level motifs.

After defning the genome encoding, efective evolutionary operators have to be chosen. As described in Sect. [3](#bookmark37), evolutionary algorithms iteratively select a number of parent indi- viduals based on a ftness function and generate ofspring by using crossover and mutation steps. Xie and Yuille ([2017](#bookmark272)) proposed using the classic roulette wheel selection approach, where the ftness of an individual is determined by training the corresponding neural net- work on a reference dataset, and its selection probability equals its ftness divided by the sum of all ftness values in the population. In this manner, better individuals have a higher chance to participate in reproduction. Tournament selection (Liu et al. [2018b](#bookmark282); Real et al. [2017](#bookmark273), [2019](#bookmark241)) is another parent selection operator, in which *p* randomly selected individu- als from the entire population (with or without replacement) participate in a tournament. The best individual in this group is selected as a parent (Goldberg and Deb [1990](#bookmark308)) and this process is repeated for a number of rounds until the mating pool is flled. Real et al. ([2017](#bookmark273)) used pairwise tournament selection ( = 2, which is a common choice). In pairwise selec- tion, the performance of only two randomly selected individuals are evaluated in each iter- ation, which makes the selection operator substantially more efcient than when a larger *p* is used. Real et al. ([2019](#bookmark241)) introduced an ageing process by associating an age parameter with each individual in order to track how long an individual has been within a population. This allows biasing the tournament selection for the next generation towards younger geno- types. The oldest individuals are removed at each cycle to keep the population size fxed. Discarding the oldest individuals rather than the worst ones allows for exploring more of the search space.

Mutation and crossover operators can be used to generate the individuals in the next generation of the population (ofspring networks). In general, the aim of the mutation is to search for the best individual around a single individual (Liu et al. [2021](#bookmark31)). For CNNs, mutation can involve adding/removing convolution operations, adding/removing skip con- nections or changing flter size/learning rate/weights (Real et al. [2017](#bookmark273)). For long short-term memory (LSTM) networks, mutation can involve adding/removing a connection of two LSTM layers, or adding/removing a skip connection of two LSTM nodes (Miikkulainen et al. [2019](#bookmark240)). Suganuma et al. ([2018](#bookmark274)) used point mutations that randomly change both the type and connections of a layer. Lorenzo and Nalepa ([2018](#bookmark309)) used a Gaussian process-based mutation operator that progressively refnes the individuals. Elsken et al. ([2019a](#bookmark227)) used network morphism and approximate network morphism as mutation operators. Network morphism extends the network (e.g., add skip connections, add more flters in a convolu- tion), while approximate network morphism reduces its size (e.g., remove skip connec- tions, remove some flters of a convolution). Network morphism only allows operators in the space of neural network architectures that preserve the function of the network. This removes the need to train ofspring networks from scratch.

Next to mutation, the crossover operation can also be used in generating individu- als for the next generation. The mutation operation modifes a single parent, resulting in ofspring that are somewhat similar to their parent. Crossover, in contrast, combines fea- tures from two diferent individuals and can thus create ofspring that difer substantially from their parents. One-point crossover is an example of an operator that has been used in EvNAS for combining two parents of the same length into an ofspring (Ahmed et al. [2019](#bookmark310)). This requires that the length of the two individuals (e.g., the depth of a network) has been defned beforehand. The variable-length encoding strategy proposed by Sun et al.

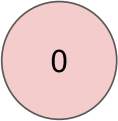
([2020](#bookmark311)) supports more efective architecture design processes by performing crossover on

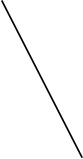
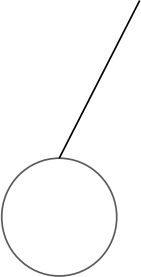
a 灬

individuals with diferent lengths. A number of EvNAS approaches do not make use of crossover operators, opting for a simpler approach (see, e.g., Real et al. [2017](#bookmark273); Liu et al. [2018b](#bookmark282)). Similar to earlier evolutionary approaches (see, e.g., Xie and Yuille [2017](#bookmark272); Sug- anuma et al. [2018](#bookmark274)) that predefne the depth of the network, the approach by Real et al. ([2017](#bookmark273)) removed the crossover operation and thus limits the search space to networks with a predefned depth. Other NAS approaches based on evolutionary algorithms include crosso- ver operators (Xie and Yuille [2017](#bookmark272); Irwin-Harris et al. [2019](#bookmark277); Lu et al. [2019](#bookmark312); Zhang et al. [2022a](#bookmark243)). Irwin-Harris et al. ([2019](#bookmark277)) have proposed an encoding strategy based on DAGs that uses crossover and mutation to construct CNN architectures of variable depth and with arbitrary graph structure. Xie and Yuille ([2017](#bookmark272)) made use of crossover and mutation oper- ations on architectures represented in multiple stages (where in each stage, the convolu- tional operators have a similar number of flters or channels). This structure allows encod- ing architectures into a fxed-length binary string. The mutation operation is performed by fipping a bit to preserve the quality of good individuals by slightly modifying them. The crossover operator applies to each stage in order to preserve the local structure within stages.

**4.2.4 Monte Carlo tree search (MCTS)**

Optimisation of neural architectures can also be viewed as a sequential decision process that can be solved using MCTS. The upper confdence bounds applied to trees (UCT) algo- rithm is a well-known Monte Carlo tree planning algorithm for tree-structured state-action spaces with a built-in exploration-exploitation mechanism that has been commonly used to search the space of network architectures (Negrinho and Gordon [2017](#bookmark268); Wistuba [2017](#bookmark278); Wang et al. [2020](#bookmark245)). MCTS works by defning a search tree, representing the search space that is traversed by visiting nodes based on a tree policy and a rollout policy. In the con- text of NAS, the internal nodes of this tree can represent hyperparameter values. A model

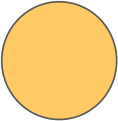


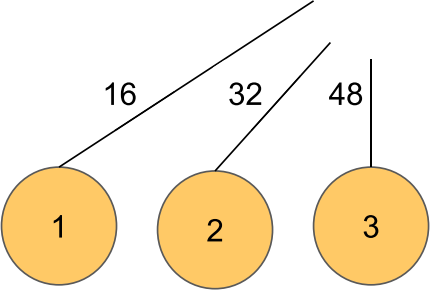
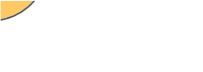
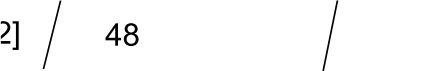
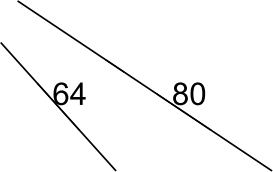
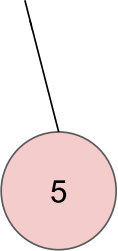
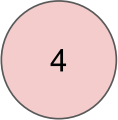
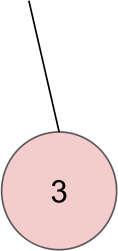
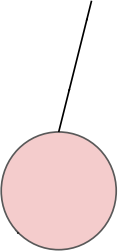
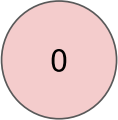
(a)

(b)

\[64,80]

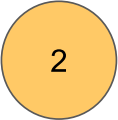
[16,32,48]/





32





**Fig. 9 a** A tree that encodes one hyperparameter and its possible values (16, 32, 48, 64, 80). **b** The same tree restructured with bisection. This allows more information to be shared between possible paths (e.g., sampling path to node 1 provides partial information about nodes 2 and 3) (Negrinho and Gordon [2017](#bookmark268))

a 灬

is defned when reaching a leaf node of the tree. Originally, in MCTS, when visiting a node in the tree, all children of that node need to be expanded before any of their children is expanded. This is not ideal for nodes that represent numerical hyperparameters with a large range of values that lead to a similar performance. To address this issue, Negrinho and Gordon ([2017](#bookmark268)) proposed combining UCT with a bisection approach that restructures the branches of the tree for such hyperparameters. At each node, instead of committing to a specifc value of a hyperparameter, a sequential committing process is followed. It is frst decided if the chosen hyperparameter value is in the frst or second half of the set of hyper- parameters. The bisection structure implicitly allows sharing information among similar paths over the search tree, enabling search in a large search space (see Fig. [9](#bookmark313)).

Techniques based on information sharing in diferent ways have been used to improve the performance of MCTS. Wistuba ([2017](#bookmark278)) defnes the NAS problem as a Markov deci- sion process where each state describes the current network architecture, and each action adds a layer to the network. To address this problem with MCTS, Wistuba ([2017](#bookmark278)) proposed two other variants of UCT that allow sharing information between branches of the search tree with the focus on reducing the time required for fnding a good architecture through information sharing: the frst of these shares information for the same action in similar states, while the second shares information between similar actions. This is achieved based on predictions of the fnal reward from previously selected actions. To improve the perfor- mance of MCTS, AlphaX (Wang et al. [2020](#bookmark245)) proposes to use a meta-deep neural network trained to estimate the accuracy achieved by candidate architectures.

**4.2.5 Gradient‑based methods**

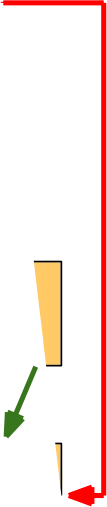
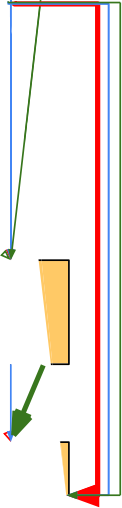
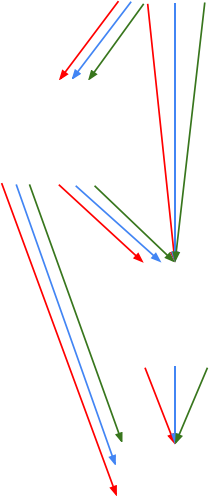
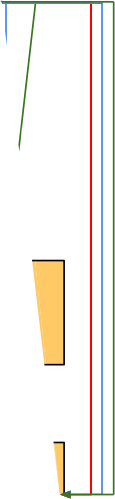
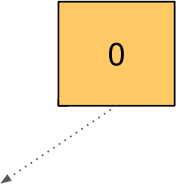
All the methods mentioned so far consider neural architecture search as a black-box opti- misation problem over a discrete search space. This approach involves the evaluation of the neural architectures with a large set of parameters that takes a signifcant amount of time and computational resources. Another category of approaches considers optimising the architecture using gradient descent. This way, much fewer data is needed for optimising the hyperparameters compared to the black-box optimisation approach (Liu et al. [2019b](#bookmark159)).

Since gradient-based optimisation is only applicable to continuous search spaces, con- tinuous relaxation approaches are used to transform the NAS search space to a continuous one. Some of the earlier approaches to creating continuous search spaces, such as (Ahmed and Torresani [2018](#bookmark246); Saxena and Verbeek [2016](#bookmark314); Shin et al. [2018](#bookmark283)), mainly focused on opti- mising a limited set of architectural hyperparameters. For instance, MaskConnect (Ahmed and Torresani [2018](#bookmark246)) focuses on optimising connectivity patterns by defning learna- ble masks in the form of binary vectors that are learnt jointly with network parameters, and Shin et al. ([2018](#bookmark283)) focused on optimising a number of CNN hyperparameters, such as flter size, number of channels and group convolution, by defning a continuous function based on these hyperparameters.

DARTS (Liu et al. [2019b](#bookmark159)) is a highly infuential approach that considers a continu- ous relaxation of a cell-based search space applicable to both recurrent and convolutional networks. It aims to fnd optimal sub-architectures for the normal and reduction cells explained in Sect. [4.1.2](#bookmark287). Consider a cell represented with a DAG composed of a set of nodes and edges. Each specifc node x(i) is a latent representation, and each edge (*i*, *j*) is associated with an operation (ij) that transforms the latent representation x(i) to x(j).

An intermediate node is calculated from its predecessors x(j) = Σi<j (o(ij) . x(i)).

a 灬

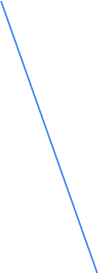
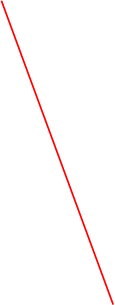
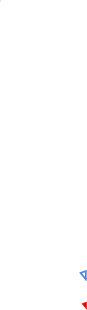


|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

?



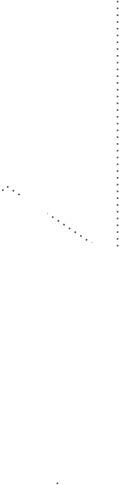
|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

?

|  |
| --- |
|  |

"?

?





|  |
| --- |
| 2 |

|  |
| --- |
| 2 |

|  |
| --- |
|  |





?







2

 2





?

3 ·

|  |
| --- |
| 3 |

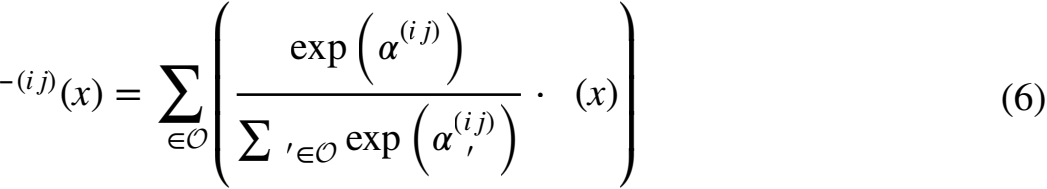
|  |
| --- |
| 3 |

|  |
| --- |
|  |

(a) (b) (c) (d)

**Fig. 10 a** Operations on the edges of the DAG are unknown. **b** The continuous relaxation approach of DARTS (Eq. [6](#bookmark316)) creates a mixture of operations. **c** Solving a bilevel optimisation problem allows jointly optimising of the mixing weights and the network weights. **d** The fnal architecture is determined from the learned mixing weights (Liu et al. [2019b](#bookmark159))

The DARTS approach, illustrated in Fig. [10](#bookmark315), relaxes the categorical choice of a particu- lar operation *o* on node x(i) (e.g., convolution, max pooling) to a softmax over all possible operations O. This will acquire a mixture of candidate operations for each edge denoted by ̄ (ij)(x) (Liu et al. [2019b](#bookmark159)):



where *o*(.) denotes an operation applied to node x(i), and the operation mixing weights of the pair of nodes x(i) and x(j) connected by the edge (*i*, *j*) is parameterised by a vector 훼(ij) with the size of ∣ O ∣. The goal of the architecture search process will be to identify an architecture encoding 훼 in the form of a set of continuous variables 훼 = {훼o(ij) }, each repre- senting the weight of an operation. A discrete architecture will be produced by replacing mixed operations ̄ (ij)(x) with a single operation that has the highest weight ( (ij) ∈ argmax 훼(ij)).

∈O

The architecture search process then aims to fnd 훼 \* that minimises the validation loss Lval (w 훼 \*), while the weights of the network w\* are determined by minimising the training

loss \* = argmin Ltrain ( , 훼 \*). This bi-level optimisation problem can be solved using gradient descent to jointly optimise *w* and 훼 .

There are a number of issues with this continuous relaxation approach that have been addressed by follow-up research. Increasing the depth of the candidate networks considered in DARTS exponentially increases the size of the space to be searched and, consequently, the GPU memory overhead. In light of this, Liu et al. ([2019b](#bookmark159)) originally restricted architec- ture search to a network of 8 cells and evaluated it on a network of 20 cells. Proxyless (Cai et al. [2019](#bookmark249)), addresses the memory inefciency of DARTS (Liu et al. [2019b](#bookmark159)) by defning

a 灬

proxy tasks (i.e., training on smaller datasets). This is achieved through a path binarisation approach for reducing the memory footprint. During the training of an overparameterised network, many paths remain active in memory. By defning binary gates instead of continu- ous gates, only one path will be retained active in memory at run-time.

Another issue with DARTS is caused by the diference in the behaviour of shallow and deep networks (i.e., faster gradient descent by shallow networks versus higher per- formance with deeper networks), leading into a so-called depth gap between search and evaluation. The networks preferred in the search process will not necessarily be opti- mal for evaluation (Chen et al. [2019b](#bookmark251)). To address this issue, PDARTS progressively increases the depth of candidate architectures while approximating the search space to prevent memory issues. The search is divided into multiple stages. At the end of each stage, the network depth increases, while the number of candidate operations is reduced using their scores (calculated based on 훼(ij)) in the search process so far as a criterion for selection. PCDARTS (Xu et al. [2020](#bookmark253)) uses another approximation scheme based on partial channel connection. During the continuous approximation, instead of sampling

all channels connecting two connected nodes within a cell, PCDARTS samples a ran- dom subset of channels and takes the computation on this subset as a surrogate for that on all channels.

Furthermore, the bi-level optimisation approach of DARTS sufers from so-called performance collapse (Liang et al. [2019](#bookmark252); Chu et al. [2020](#bookmark256); Chen et al. [2019b](#bookmark251)), i.e., per- formance decay of the discovered architecture as the number of search epochs increase, which is caused by the dramatic aggregation of skip-connections in the selected archi- tecture. PDARTS (Chen et al. [2019b](#bookmark251)) and B-DARTS (Ye et al. [2022](#bookmark265)) address this issue by means of new regularisation strategies. PDARTS incorporates design search space regularisation to alleviate the dominance of skip-connections during the search, which leads to more hyperparameters that need to be defned by the user. The BetaDecay regu- larisation used by Ye et al. ([2022](#bookmark265)) imposes constraints to prevent the value and variance of activated architecture parameters from getting overly large. A number of other studies have aimed to gain further insights into the cause of performance collapse. DARTS+ (Liang et al. [2019](#bookmark252)) shows that the number of skip-connections is linked to overftting, which can be addressed by an early stopping strategy for the search process.

On the other hand, Chu et al. ([2020](#bookmark256)) found evidence that the issue is caused by unfair competition among various operations in the bi-level optimisation process, which often leads to the dominance of skip-connections. They propose a cooperative mechanism implemented by additional activations for each parameter 훼(ij) to eliminate the competi- tion by allowing operations to be switched on or of without being suppressed. Diferent forms of unfair competition between simultaneously trained operations in DARTS, lead- ing to training stability issues, were identifed and more generally studied by Hong et al.

([2020](#bookmark257)), Gu et al. ([2021](#bookmark260)). Hong et al. ([2020](#bookmark257)) proposed to address this issue through the use of a new group drop-out operation, while (Gu et al. [2021](#bookmark260)) proposed a new continu- ous relaxation approach that completely decouples operations and topology search in diferentiable architecture search.

Neural Architecture Optimisation (NAO) Luo et al. ([2018](#bookmark248)) is another gradient-based NAS approach. While DARTS makes use of a continuous representation of the architec- ture, including its weights, NAO is based on a continuous embedding of the architec- ture search space only. In this approach, an auto-encoder is used to learn a continuous

a 灬

representation of neural network architectures. A surrogate model is trained on this con- tinuous representation to predict the performance of previously unseen candidate archi- tectures. In each iteration of the search process, the surrogate model is used in gradient- based optimisation to select a new architecture to be evaluated. This new architecture is obtained by mapping the continuous representation back to a discrete one using a decoder learned as part of the autoencoder model. The surrogate model and autoen- coder are updated at every iteration in order to minimise the encoder-decoder model loss and the performance prediction (surrogate) loss. While NAO uses gradient descent for hyperparameter optimisation, unlike DARTS, it does not consider a bi-level optimi- sation of weights and hyperparameters. However, NAO it can still use weight-sharing approaches (in Sect. [4.3.1](#bookmark289)) separately to speed up the evaluation of candidate networks.

Gradient-based approaches such as DARTS can be used in combination with gradient-based meta-learning approaches to further improve search efciency. Recently, Elsken et al. ([2020](#bookmark317)) have proposed MetaNAS, which integrates NAS with gradient-based meta-learning techniques. Their approach extends DARTS to optimise a meta-architecture with meta-weights during meta-learning; this facilitates the adapta- tion of the architecture to novel tasks.

**4.2.6 Random search and grid search**

As in the case of HPO, random search (Bergstra and Bengio [2012](#bookmark7); Li and Talwalkar [2019](#bookmark267)) and grid search (Zagoruyko and Komodakis [2016](#bookmark318)) have been used in NAS. Both approaches are very simple and easy to run in parallel, but they are usually not considered to be very efcient. Random search has been considered a competitive baseline for NAS (Yu et al. [2020](#bookmark319); Li and Talwalkar [2019](#bookmark267)). Comparing the performance of models on Penn Tree Band (PTB) (Marcus et al. [1994](#bookmark320)) language modelling and CIFAR-10 (Krizhevsky et al. [2010](#bookmark321)) image classifcation datasets, Yu et al. ([2020](#bookmark319)) demonstrated that, on average, a number of state-of-the-art NAS algorithms [ENAS (Pham et al. [2018](#bookmark231)), DARTS (Liu et al. [2019b](#bookmark159)) and NAO (Luo et al. [2018](#bookmark248))] have similar performance to random search when using the same search space for fnding a recurrent and convolutional cell.

Li and Talwalkar ([2019](#bookmark267)) evaluated random search with early stopping and weight-shar- ing strategies (explained in Sect. [4.3.1](#bookmark289)) on the search space of DARTS and demonstrated that it achieves performance competitive with that of DARTS and ENAS. This is not due to the poor performance of the latter NAS algorithms but mainly the consequence of the constraints they have imposed on the search space. In a well-constrained search space, even random search can perform well. Another reason could be the weight-sharing strategy that negatively impacts architecture ranking during the search.

**4.3 Performance evaluation in NAS**

A major performance bottleneck of NAS systems arises from the need to train each can- didate neural network. This is especially the case for deep neural networks, whose train- ing takes substantial amounts of time. Therefore, performance evaluation is a much more important topic in the context of NAS than in HPO methods for classic machine-learning algorithms. Early NAS systems, such as NASNet (Zoph et al. [2018](#bookmark230)) and AmobaNet (Real

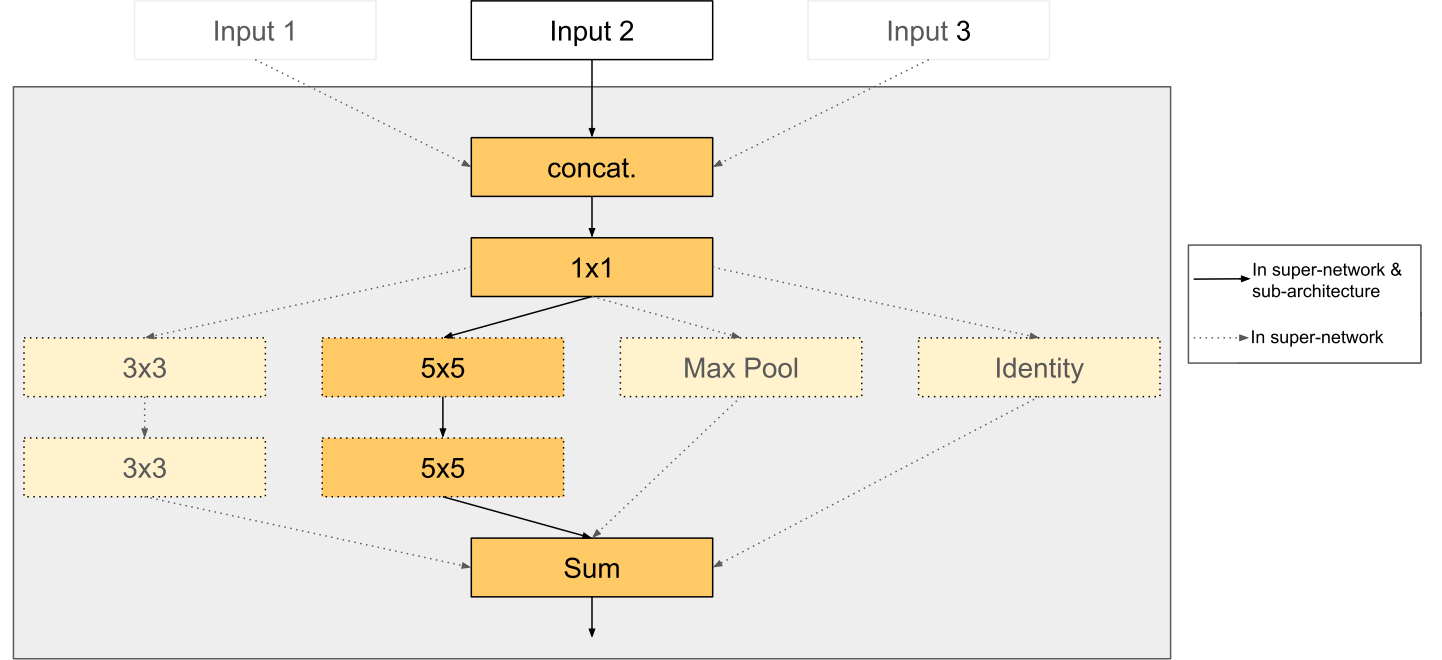
a 灬

et al. [2019](#bookmark241)), trained every candidate architecture from scratch, racking up thousands of days of GPU time. The design of cell-based search spaces, as discussed in Sect. [4](#bookmark38), can, to some extent, decrease the total cost of performance evaluation. The search process can also be sped up signifcantly using network morphisms and function-preserving transformations (Chen et al. [2016](#bookmark322)) that allow modifying the structure of a network without majorly chang- ing its predictions. Furthermore, efcient performance estimation strategies have recently become a major focus for research on NAS methods. In the following subsections, we dis- cuss approaches taken to speed up the performance evaluation in NAS.

**4.3.1 Parameter sharing**

Parameter (or weight) sharing or inheritance between network architectures is an approach that can reduce the computational demands of NAS methods. Parameter sharing can be performed by reusing the weights of previously optimised architectures or sharing compu- tations between diferent but related architectures.

Defning the search space that allows sampling sub-architectures from super-networks facilitates parameter sharing (Pham et al. [2018](#bookmark231); Cai et al. [2018a](#bookmark269); Elsken et al. [2018](#bookmark275)). ENAS (Pham et al. [2018](#bookmark231)), SMASH (Brock et al. [2018](#bookmark279)) and convolutional fabrics (Saxena and Verbeek [2016](#bookmark314)) use an over-parameterised super-network, defned in a discrete search space (also referred to as a one-shot model), that is a superposition of all possible sub- architectures, and enforce parameter sharing between sub-architectures. Once the weights of the super-network are trained, the performance of all sub-architectures on the valida- tion set can be ranked and compared without training by enabling and disabling edges (see Fig. [11](#bookmark323)). The most promising sub-architectures need to be further retrained. Generating and training a super-network for a given search space is itself a complicated task. Muñoz et al. ([2022](#bookmark324)) proposed a further approach, dubbed BootstrapNAS, that automates the generation and training of super-networks from pre-trained models.



**Fig. 11** An example of a one-shot cell that receives three inputs concatenates them, applies a 1 × 1 convo- lution operation, followed by multiple other operations and fnally sums the result together. A sub-archi- tecture, denoted by solid edges, can be evaluated by disabling inputs and operations by zeroing them out (Bender et al. [2018](#bookmark325))

a 灬

Creating diferentiable search spaces (see, e.g., Liu et al. [2019a](#bookmark284), [b](#bookmark159); Xu et al. [2020](#bookmark253)) using a continuous relaxation mechanism is another approach that implicitly allows parameter sharing. A diferentiable search space allows parameters and hyperparameters to be jointly optimised using gradient descent without a need for a candidate architecture to be itera- tively sampled and evaluated. The continuous representation proposed in DARTS (Liu et al. [2019b](#bookmark159)), for instance, benefts substantially from parameter sharing by defning a super-network that is diferentiable in both network weights and architectural hyperparam- eters at the expense of high GPU memory consumption. Combining search space shrink- age methods with parameter sharing allows further enhancement in the efciency of search algorithms. Hu et al. ([2020](#bookmark258)) proposed shrinking the search space to a region consisting of a high proportion of well-performing architectures. This is achieved by pruning low-rank operational choices by removing operations that are found in poor-performing architec- tures. The shrinkage method is implemented by measuring the diference in the initialised neural network and the fully trained network by calculating the angle between the respec- tive network weight vectors. This approach, however, ignores regions of the search space with a smaller proportion of well-performing networks. Gopal et al. ([2023](#bookmark266)) improve the shrinkage approach further by proposing a locality-based iterative search method. Locality, in this sense, points to the property that there is a correlation between structural similarity between architectures and similarity in their respective performance. This method initially samples diverse networks from the original search space and predicts their performance using a predictor trained on a NAS benchmark. Next, various samples are generated in the local neighbourhood of the original samples.

Limits of parameter sharing strategies have been studied by Yu et al. ([2020](#bookmark319)), Xie et al. ([2022a](#bookmark326)). Yu et al. ([2020](#bookmark319)) demonstrated that parameter sharing strategies degrade the rank- ing of candidate architectures. This is due the fact that estimating the performance of sub- architectures by copying their weights from a super-network may not refect the true per- formance of candidate architectures. Xie et al. ([2022a](#bookmark326)) identifed an instability issue arising in a scenario where individual runs of the search process will lead to networks of diferent quality. This instability is due to the optimisation gap between the super-network and its sub-architectures. It is not guaranteed that an optimised super-network creates an optimised sub-architecture. Diferent reasons have been identifed for this issue, including the unfair competition between parameters and hyperparameters, errors in calculating gradients in the continuous search and falling into local optima while optimising the super-network.

**4.3.2 Performance predictors**

Another line of work targeting more efcient performance evaluation in NAS aims at build- ing models to predict the performance of the neural networks in terms of the fnal accu- racy or ranking of candidate architectures. The performance predictor is once initialised at the start of the NAS process and subsequently queried with many architectures within the inner NAS optimisation loop. Benchmarks (extensively covered in Sect. [4.5](#bookmark225)) such as NAS- Bench-101 (Ying et al. [2019](#bookmark327)), NAS-Bench-201 (Dong and Yang [2020](#bookmark328)) and NATS-Bench (Dong et al. [2021](#bookmark329)) that include a large number of evaluated architectures provide great opportunities for creating performance prediction models. The methods mentioned earlier in Sect. [3.3.4](#bookmark194) for accuracy prediction through learning curve extrapolation do not have an

a 灬

initialisation phase. However, when used in NAS they still require the expensive process of training various candidate architectures.

One research direction to speed up the performance evaluation in NAS is to reduce the number of expensive training steps in the inner optimisation loop to zero. Deng et al. ([2017](#bookmark330)), Istrate et al. ([2019](#bookmark331)) proposed model-based approaches for the estimation of the accuracy achieved by a given network by analysing its structure. TAPAS (Istrate et al. [2019](#bookmark331)) initialises a predictive model for the performance of architectures based on data- set characteristics and a lifelong database of experiments on previously trained neural net- works. An accuracy predictor uses this model to predict the peak accuracy of each queried architecture after training. Akhauri and Abdelfattah ([2023](#bookmark332)) propose two separate neural network encodings for accuracy and latency prediction to support hardware-aware NAS that supports the optimisation of accuracy and latency objectives.

Recently, several zero-cost methods (see, e.g., Abdelfattah et al. [2021](#bookmark333); Mellor et al. [2021](#bookmark334)) have been proposed that further reduce both the initialisation and query cost of pre- dictors by exploiting more fundamental architectural properties of a network from its initial state by just a single forward/backward propagation pass on a single mini-batch of data. For instance, Mellor et al. ([2021](#bookmark334)) proposed a scoring method that predicts the performance of an untrained architecture by estimating the overlap of activations of rectifer linear units between data points in untrained networks and previously trained networks. This approach is motivated by the idea that the more similar the activations between two inputs, the more difcult it is for the network to learn to separate them. Abdelfattah et al. ([2021](#bookmark333)) proposed a number of zero-cost proxies using network pruning as an initialisation strategy. These proxies can be used to rank network architectures in NAS.

White et al. ([2021b](#bookmark335)) performed an empirical comparison of performance predictors in the context of diferent NAS frameworks. Their results suggest that even a simple combina- tion of strategies mentioned before (zero-cost, model-based and learning curve methods) can substantially improve performance by exploiting the complementary power of diferent strategies.

**4.4 NAS systems and libraries**

In this section, we review important systems and libraries for NAS that can be useful to practitioners and researchers. Some of these [e.g., AutoKeras (Jin et al. [2019](#bookmark12)), Auto- PyTorch (Zimmer et al. [2021](#bookmark195))] are essentially an implementation of a specifc NAS approach, which can be used by practitioners or further extended by researchers. These systems, in principle, follow a similar purpose to the AutoML systems covered in Sect. [5](#bookmark39). However, since they address the NAS problem as opposed to the CASH problem, we cover them here. Another group of libraries [e.g., NNI (Microsoft [2021](#bookmark336)), NASlib (White et al. [2021b](#bookmark335))] focuses on providing a homogeneous codebase that can also support further NAS research. These libraries empower researchers to customise and build upon available meth- ods, as well as to evaluate new NAS methods against important baselines. Prominent NAS systems and libraries include the following:

*AutoKeras* (Jin et al. [2019](#bookmark12)) is a NAS system built upon Keras supporting regression and classifcation tasks on image, text and tabular datasets. The proposed search strat- egy of AutoKeras is a combination of Bayesian optimisation and network morphism (Jin et al. [2019](#bookmark12)) (explained in Sect. [4.2.2](#bookmark294)). However, in the default setting, this system uses an optimisation method based on random search. AutoKeras supports optimising a macro

a 灬

search space and allows building networks using several block types, such as Xception and ResNet.

*Auto-PyTorch* (Zimmer et al. [2021](#bookmark195)) is a NAS system built on top of the PyTorch frame- work. As a search strategy, auto-PyTorch uses a combination of Bayesian optimisation and hyperband to speed up the training process (explained in Sect. [3.2.2](#bookmark106)). It also uses meta- learning and ensembling to produce stronger models. Auto-PyTorch focuses on optimis- ing the full machine-learning pipeline, including pre-processing, NAS, network training and regularisation. It supports two macro-scale search spaces: a smaller one for optimis- ing MLPs, and a bigger one for optimising deeper architectures that can include residual blocks. Originally, the main focus of Auto-PyTorch was to achieve state-of-the-art perfor- mance on tabular datasets by incorporating relevant pre-processing tasks into the search space. However, its usefulness has also been demonstrated in object detection tasks.

*NNI* (Microsoft [2021](#bookmark336)) is a Python library for HPO and NAS which facilitates users in developing and using NAS techniques. It includes implementations of many HPO tech- niques (e.g., random and grid search, Bayesian optimisation, evolutionary algorithms, and Hyperband) as well as state-of-the-art one-shot NAS methods (e.g., DARTS, ENAS, FBNET, ProxlessNAS). The deep-learning framework supported by NNI is PyTorch, and APIs are provided to construct and explore diferent search spaces. NNI also supports Kubernetes, an important system for cloud-native deployments. Other than HPO and NAS, NNI covers model compression and feature engineering as well.

*NASLib* (White et al. [2021b](#bookmark335)) is another Python library based on PyTorch and mainly focused on supporting NAS researchers. It systematically follows a modular and fexible approach that facilitates the reuse of search spaces and evaluation pipelines. NASLib cov- ers cell-based and hierarchical search spaces, various search strategies (e.g., random search, Bayesian optimisation, evolutionary, and gradient-based search), and performance evalua- tion approaches (e.g., one-shot, zero-cost, weight-sharing, learning curve-based). Further- more, it includes an extensive collection of NAS benchmarks that can help researchers evaluate and further develop performance prediction approaches.

*Katib* (George et al. [2020](#bookmark337)) is a library for HPO and NAS, mainly aimed at providing a scalable, cloud-native and production-ready system. It supports Kubernetes and is agnostic to the underlying machine-learning framework, supporting codes written by users in difer- ent frameworks (e.g., PyTorch, TensorFlow, ApacheMXNET, XGBoost). Katib includes implementations of various search strategies (e.g., random search, grid search, Bayesian Optimisation,TPPE, multivariate TPE, CMA-ES, hyperband and evolutionary algorithms); in terms of NAS approaches, only implementations of ENAS and DARTS are included.

*Hypernets* (DataCanvas [2021](#bookmark338)) is a library supporting various deep-learning frameworks (Tensorfow, Keras, PyTorch). It supports macro search spaces and the ENAS micro-search space, as well as Monte-Carlo tree search (MCTS), evolutionary search and random search strategies.

**4.5 NAS benchmarks**

While NAS-generated architectures have shown promising performance across difer- ent domains, performing thorough and fair comparisons against state-of-the-art NAS approaches still remains an issue. To some extent, this issue can be alleviated by follow- ing the suggested best practices (see, e.g., Lindauer and Hutter [2020](#bookmark339)) for making NAS codes and models available. However, even when the code is available, large computational resources are needed to reproduce the results of NAS experiments, and those resources are

a 灬

not broadly available to NAS researchers. A common solution to the challenges arising from this situation has been to directly use results reported in the literature as the basis for comparative evaluations. However, this approach can be misleading since those results can be strongly infuenced by factors such as the type of GPUs or the parallelisation strategies employed.

To address this reproducibility issue, in the past few years, there has been a number of eforts to create benchmarks for facilitating NAS research. These benchmarks were ini- tially created in the form of tabular datasets, whose entries provide relevant information about fully trained neural network models within a predefned search space. Having access to information such as running time and training/validation/test accuracy of all networks within a given search space, NAS researchers can faithfully simulate runs of a NAS proce- dure. This leads to tremendous speed-ups in the overall process of developing NAS algo- rithms. Naturally, the creation of these benchmarks tends to be computationally expensive; it involves the following steps: (1) defnition of the search space, (2) sampling candidate architectures over this search space using a well-defned strategy and removing duplicates, (3) training and evaluation of the architectures from the resulting set of architectures using diferent objective functions (or possibly even in the context of downstream tasks).

HPO-Bench (Klein and Hutter [2019](#bookmark340)) and NAS-bench-101 (Ying et al. [2019](#bookmark327)) are among the frst tabular benchmark datasets created in this manner. HPO-Bench covers 144 candidate architectures of a feedforward neural network and can be used for empirical evaluation of HPO methods. However, it is insufcient to evaluate NAS algorithms that focus on automatic design of advanced deep-learning architectures. NAS-Bench-101 is the frst public dataset suitable for benchmarking NAS research. It covers all 423,000 unique convolutional archi- tectures of a cell-based search space defned based on a DAG of seven nodes and three opera- tions. Each architecture was trained multiple times on CIFAR-10 leading to a large dataset of over 5 million trained models. This benchmark can be used for comparing HPO methods as well as certain NAS algorithms that do not use parameter sharing or network morphisms.

NAS-Bench-201 (Dong and Yang [2020](#bookmark328)) is tailored towards the evaluation of more NAS algorithms on more image datasets but within a smaller space based on a DAG of 4 nodes and 5 operations, resulting in 15,625 neural cell candidates in total. NAS-Bench-1Shot1 (Zela et al. [2020](#bookmark341)) reuses the NAS-Bench-101 dataset with some modifcations tailored to the evalu- ation of one-shot NAS methods. While benchmarks such as NAS-Bench-101, HPO-Bench, NAS-Bench-1Shot1 focus on solely architecture topology without considering the architec- ture size, NATS-Bench (Dong et al. [2021](#bookmark329)) covers both these factors by creating a benchmark of two sets of architectures with 15,625 neural cell candidates (4 nodes and 5 operations) to evaluate the architecture topology and 32,768 candidates for architecture sizes that can be used to evaluate all NAS methods. There have also been eforts to create more specialised benchmarks to evaluate more properties of a specifc group of NAS systems. For instance, BenchENAS (Xie et al. [2022b](#bookmark342)) focuses on evaluating EvNAS algorithms along with provid- ing a platform for running them in a fair manner, including a special parallel processing com- ponent for evaluating populations of candidate architectures. This benchmark currently covers 9 EvNAS algorithms that cover fxed-length encoding and variable-length encoding strategies, as well as single- versus multi-objective optimisation. NAS-Bench-NLP (Klyuchnikov et al. [2022](#bookmark343)) focuses on benchmarking NAS for natural language processing by using RNN cells as opposed to the convolutional cells used in the previous NAS benchmarks. It covers 14,000 trained architectures along with evaluation results using metrics from language modelling and relevant downstream tasks (e.g., sentence tasks, similarity and paraphrase tasks).

The above-mentioned tabular NAS benchmarks rely on the computationally expensive task of an exhaustive evaluation of all architectures within a given search space. Therefore, to

a 灬

ensure feasibility, their search spaces are limited to very small architectural spaces compared to those usually considered in the NAS literature. This might compromise the generalisability of the models evaluated using existing tabular NAS benchmarks. To enlarge this search space, NAS-Bench-301 (Zela et al. [2022](#bookmark344)) provides a surrogate NAS benchmark that covers 108 archi- tectures and is thus much larger than previous benchmarks. The underlying idea is to estimate the performance of candidate architectures using a surrogate model rather than actual evalu- ations. The authors of NAS-Bench-301 then created a new benchmark by sampling 60000 architectures from a much more complex search space over which they had trained the sur- rogate model. While the degree to which the resulting benchmark is realistic depends on the accuracy of the surrogate model, empirical results by Zela et al. ([2022](#bookmark344)) demonstrate that a surrogate model can yield a better model of the architecture performance when compared to a tabular benchmark. The reason for this is that tabular benchmarks tend to present few train- ing runs for each architecture acquired using a mini-batch procedure, which itself can have a variance. Thus, a tabular benchmark gives only a simple estimate of the true performance of an architecture. A surrogate model trained on a smaller subset of networks has the potential to yield a much more accurate estimation by taking into account extra information, such as the similarity between architectures.

**5 Broad‑spectrum automated machine‑learning systems**

In Sect. [4.4](#bookmark224), we covered AutoML systems such as AutoKeras (Jin et al. [2019](#bookmark12)) and Auto-PyTorch (Zimmer et al. [2021](#bookmark195)) that address the NAS problem. In this section, we review a number of broadly known and widely used AutoML systems based on classic machine-learning algorithms realised through addressing the CASH problem.

Although deep learning for certain data forms, such as natural images, has facilitated automated feature learning, most machine-learning models still crucially rely on some degree of feature engineering and preprocessing (Chen et al. [2019a](#bookmark345)). Therefore, many AutoML systems do not only focus on hyperparameter optimisation but on constructing full machine-learning pipelines that include data preprocessors and feature extractors, as well as diferent machine-learning models. To achieve this goal, a common approach (see, e.g., Feurer et al. [2015](#bookmark10); Thornton et al. [2013](#bookmark8)) is combining all design choices into a single large search space and then using HPO methods described in Sect. [3](#bookmark37) (see, e.g., Hutter et al. [2011](#bookmark43); Bergstra et al. [2011](#bookmark42), [2013](#bookmark120)) to search for the best parameter setting. It should, however, be noted that not all HPO methods can scale to large hyperparameter spaces or correctly handle all relevant types of hyperparameters, notably the conditional hyperparameters, that are relevant in the context of pipeline generation. In this case, the search space usually includes two types of information: the choice of algorithms and their hyperparameters. The idea mentioned above was used by Thornton et al. ([2013](#bookmark8)) to formally defne the *combined algorithm selection and hyperparameter optimisation (CASH) problem* as a single optimisation problem, as per Defnition [3](#bookmark36) in Sect. [2.1](#bookmark34).

Most AutoML systems address the CASH problem to automate supervised learning tasks by minimising the cross-validation loss based on classifcation or regression accu- racy. By taking a diferent approach to validation, AutoML systems for other machine- learning tasks can be realised. For instance, to address the scarcity of labelled data, Li et al. ([2019](#bookmark346)) defned the automated semi-supervised learning (AutoSSL) problem. In AutoSSL, instead of using cross-validation to evaluate and optimise performance, the relative performance of semi-supervised learning algorithms compared to a baseline

a 灬

supervised learning algorithm is considered. Unsupervised learning tasks of descriptive nature (as opposed to predictive) that work on unlabelled instances are, however, not yet fully addressed by AutoML systems.

Existing AutoML systems can be diferentiated with respect to the structure of the pipeline they construct. Some systems construct pipelines of a fxed and pre-specifed structure, for example, one data preprocessor followed by a single machine-learning model. Other systems do not rely on a fxed structure but can construct fexible pipe- lines, which can theoretically consist of an arbitrary number of preprocessors and mod- els. Many of the AutoML systems are able to incorporate ensembles in the constructed pipelines. Ensemble models can generally achieve higher performance and tend to be more robust against overftting than single models (Guyon et al. [2010](#bookmark347); Lacoste et al. [2014](#bookmark348)). Particularly, systems such as AutoGluon (Erickson et al. [2020](#bookmark349)) and Autostacker (Chen et al. [2018](#bookmark350)) use multi-layer stacking, where the output of the stacking models are fed to higher-level stackers to create large ensembles. Furthermore, some systems, like Auto-sklearn (Feurer et al. [2015](#bookmark10)), construct ensembles by combining multiple of the evaluated pipelines in a postprocessing step. The ensemble selection technique to select a subset of models can also be treated as a design choice in an AutoML sys- tem. Purucker and Beel ([2023](#bookmark351)) propose Assembled-OpenML, a computationally ef- cient ensemble-selection method using meta-learning.

As outlined in Sect. [2.4](#bookmark49), meta-learning approaches can be used to predict the perfor- mance of models and hyperparameter settings for a given task based on experience on other tasks. A number of AutoML systems employ meta-learning as a strategy to speed up the search for performance-optimised machine-learning pipelines.

Table [3](#bookmark352) provides an overview of prominent AutoML systems. Other than the underly- ing search strategy and pipeline structure, this table lists the machine-learning library that provides the basis for constructing the search space of the AutoML system, along with an indication of whether an implementation is publicly available and whether meta-learning is employed. In the following subsections, we provide more detailed descriptions of the AutoML systems in Table [3](#bookmark352), categorised based on the underlying optimisation technique.

**5.1 Systems based on random search and grid search**

In Sects. [3](#bookmark37) and [4](#bookmark38), we introduced the random and grid search approaches for HPO and NAS, respectively. Here, we briefy review AutoML systems that make use of these approaches. Since these search strategies do not scale to large search spaces, most research in this direc- tion has worked on a strategy to improve search efciency.

*AutoCompete* (Thakur and Krohn-Grimberghe [2015](#bookmark353)) makes use of random search and grid search for the selection of a model, a corresponding hyperparameter setting and optional data preprocessing steps. It supports classifcation and regression tasks based on a search space defned over models and data preprocessors implemented in Scikit-learn (Pedregosa et al. [2011](#bookmark354)). In order to speed up the search process using these inefcient search strategies, their approach simplifes the CASH problem by limiting the search space based on the encountered dataset type. Only specifc algorithms are selected per dataset type and certain hyperparameters of those are tuned.

*H2O AutoML* (LeDell and Poirier [2020](#bookmark355)) is an AutoML system for the optimisation of machine-learning pipelines, including a post-processing step for creating an ensemble of models. This system is implemented based on the H2O machine-learning library (H2O. ai [2017](#bookmark105)); while H2O AutoML ofers the same automated preprocessing steps available in

a 灬

Automated machine learning: past, present and future Page 57 of 88 **122**

**Table 3** Overview of existing AutoML systems

a 灬

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Search strategy | Pipeline structure | Library | Implementn. Available | Meta-learning |
| ADMM AutoML (Liu et al. | Alternating direction method | Fixed | Scikit-learn | No | No |
| [2020](#bookmark356)) | of multipliers + Bayesian optimisation |  |  |  |  |
| Alpine meadow (Shang et al. | Multi-armed bandit + Bayesian | Fixed | Northstar | Yes (proprietary) | Yes |
| [2019](#bookmark357)) | optimisation |  |  |  |  |
| ATM (Swearingen et al. [2017](#bookmark358)) | Multi-armed bandit + Bayesian optimisation | Fixed | Scikit-learn | Yes | Yes |
| Auto-sklearn (Feurer et al. | Bayesian optimisation | Fixed + postprocessing | Scikit-learn | Yes | Yes |
| [2015](#bookmark10)) |  | ensemble |  |  |  |
| Auto-WEKA (Thornton et al. [2013](#bookmark8)) | Bayesian optimisation | Fixed | WEKA | Yes | No |
| AutoCompete (Thakur and Krohn-Grimberghe [2015](#bookmark353)) | Random search & Grid search | Fixed | Scikit-learn | No | No |
| AutoGluon-Tabular (Erickson | Fixed order + Bayesian opti- | Flexible | Scikit-learn, XGBoost, Light- | Yes | No |
| et al. [2020](#bookmark349)) | misation |  | GBM, CatBoost, self-imple- mented neural networks |  |  |
| AutoML-DSGE (Assunção et al. [2020](#bookmark359)) | Genetic programming | Flexible | Scikit-learn | Yes | No |
| Autostacker (Chen et al. [2018](#bookmark350)) | Evolutionary algorithm | Flexible | Scikit-learn, XGBoost | No | No |
| FEDOT (Nikitin et al. [2022](#bookmark360)) | Genetic programming + Bayes- ian optimisation | Flexible | Scikit-learn, XGBoost, Light- GBM, CatBoost | Yes | No |
| FLAML (Wang et al. [2021b](#bookmark361)) | Estimated-cost-for-improvement- based sampling + direct search | Fixed | Scikit-learn, XGBoost, Light- GBM, CatBoost | Yes | No |
| GAMA (Gijsbers and Van- | Genetic programming | Flexible + postprocessing | Scikit-learn | Yes | No |
| schoren [2020](#bookmark362)) |  | ensemble |  |  |  |
| H2O AutoML (LeDell and | Fixed order + random search | Fixed + postprocessing | H2O | Yes | No |
| Poirier [2020](#bookmark355)) |  | ensemble |  |  |  |
| Hyperopt-sklearn (Komer et al. [2014](#bookmark363)) | Bayesian optimisation | Fixed | Scikit-learn | Yes | No |

a 灬

**Table 3** (continued)

**122** Page 58 of 88 M Baratchi et a l

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Search strategy | Pipeline structure | Library | Implementn. Available | Meta-learning |
| ML-Plan (Mohr et al. [2018](#bookmark90)) | Hierarchical planning | Fixed | Scikit-learn, WEKA, MEKA | Yes | No |
| MOSAIC (Rakotoarison et al. | Monte-Carlo tree search + | Fixed | Scikit-learn | Yes | No |
| [2019](#bookmark151)) | Bayesian optimisation |  |  |  |  |
| Naïve AutoML (Mohr and Wever [2022](#bookmark364)) | Fixed order + random search | Fixed | Scikit-learn, WEKA | Yes | No |
| Oboe (Yang et al. [2019](#bookmark53)) | Meta-learning | Fixed + postprocessing ensemble | Scikit-learn | Yes | Yes |
| Oracle AutoML (Yakovlev et al. | Meta-learning + gradient | Fixed | Scikit-learn, XGBoost, Light- | Yes | Yes |
| [2020](#bookmark15)) | descent |  | GBM |  |  |
| RECIPE (de Sá et al. [2017](#bookmark365)) | Genetic programming | Flexible | Scikit-learn | Yes | No |
| TabPFN (Hollmann et al. [2023](#bookmark366)) | Meta-learning + pre-trained neural network | Fixed | None | Yes | Yes |
| TPOT (Olson et al. [2016a](#bookmark11)) | Genetic programming | Flexible | Scikit-learn | Yes | No |

H2O, it does not select or optimise their hyperparameters within the full pipeline. H2O AutoML frst evaluates user-selected models with pre-specifed hyperparameters and in a pre-defned order, adding them to a leaderboard that keeps the ranking of models based on performance. Next, it tunes the hyperparameters of the models on the leaderboard using random search, where for some pre-specifed models, more time is allocated than for oth- ers. The pre-specifed models are those which are most promising in the developers’ opin- ion. In the fnal step, H2O AutoML constructs a stacking ensemble using a pre-specifed meta-model that is trained on the outputs of the optimised base models with the goal of fnding the best combination of models.

*Naïve AutoML* (Mohr and Wever [2022](#bookmark364)) is, according to its authors, a very simple solu- tion to AutoML, which can be considered as a baseline for more complicated black-box solvers and even sometimes outperform them. The general idea of this system is to imitate the sequence and analytical process of optimising a pipeline by humans in diferent stages rather than creating a large search space of all design choices that can be optimised at the same time. It assumes machine-learning pipelines consisting of a sequence of a fxed num- ber of data transformers (that transform one representation of data to another) followed by a predictor (that predicts the label of the input data). Pipelines are optimised in a series of optimisation stages, where each stage is responsible for constructing a certain part of the pipeline, e.g., one stage to select a predictor, a second stage to select a feature selector, and a third stage to set the hyperparameters of the predictor. Pipelines consisting of feature scaling, feature selection and a predictor are optimised in a specifc series of stages based on the naïve assumption that each component can be optimised locally and independently. For instance, algorithm selection is performed on algorithms that are parameterised with their default hyperparameter setting. With the exception of the hyperparameter optimisa- tion stage, which employs random search, all other stages evaluate solution candidates in a fxed order.

**5.2 Systems based on Bayesian optimisation**

Bayesian optimisation, introduced earlier in Sect. [3](#bookmark37), is a popular method used for realis- ing AutoML systems. Bayesian optimisation based on Gaussian process models is mainly applicable for low-dimensional problems with relatively few numerical hyperparameters. In contrast, Bayesian optimisation based on tree models is more suitable for high-dimen- sional, structured, and partially discrete problems, such as the CASH problem, and has been prominently used in AutoML systems (see, e.g., Thornton et al. [2013](#bookmark8)). In this section, we briefy describe a number of AutoML systems based on Bayesian optimisation.

*Auto-WEKA* (Thornton et al. [2013](#bookmark8)) is one of the earliest proposed AutoML systems. It tackles the CASH problem for a search space based on the algorithms available in the WEKA (Hall et al. [2009](#bookmark367)) machine-learning environment, covering base classifers, feature selection and meta-models for ensembling (voting, bagging, stacking). Auto-WEKA con- structs machine-learning pipelines consisting of an optional feature selector and a (meta-) model. Apart from feature selection, no further data preprocessing is supported. Auto- WEKA employs Bayesian optimisation based on SMAC (Hutter et al. [2011](#bookmark43)) (see Sect. [3.4](#bookmark62)) to optimise machine-learning pipelines and their corresponding hyperparameters. Auto- WEKA 2.0 (Kotthof et al. [2017](#bookmark368)) is the most recent version, in which supervised classifca- tion, as well as regression algorithms, are supported.

*HyperOpt-sklearn* (Komer et al. [2014](#bookmark363)) supports a search space based on the models and preprocessors in the Python library Scikit-learn (Buitinck et al. [2013](#bookmark102)). Series of multiple

a 灬

preprocessing steps are also supported by HyperOpt-sklearn, but such series have to be explicitly specifed in the search space. Hyperopt (Bergstra et al. [2013](#bookmark120)) (see Sect. [3.4](#bookmark62)) is used by HyperOpt-sklearn for the optimisation process.

*Auto-sklearn* (Feurer et al. [2015](#bookmark10)) is another AutoML system for generating machine- learning pipelines by addressing the CASH problem using algorithms available in Scikit-learn (Buitinck et al. [2013](#bookmark102)). Auto-sklearn uses SMAC (Hutter et al. [2011](#bookmark43)) to build machine-learning pipelines consisting of a data preprocessing step and a model. Compared to Auto-WEKA and HyperOpt-sklearn, auto-sklearn uses two additional techniques to increase the efciency and accuracy: (1) Auto-sklearn supports the warm-starting of the Bayesian optimisation procedure from promising candidate solutions identifed via meta- learning in order to accelerate the optimisation process. Meta-learning in Auto-sklearn is based on a set of simple information-theoretic and statistical meta-features. (2) Auto- sklearn also supports the construction of a voting ensemble consisting of several pipelines that were evaluated in the optimisation process. The ensemble construction used in auto-

sklearn optimises the weights of the models in the ensemble in a greedy fashion, start- ing from an empty set and adding new models to it as long as it increases the validation accuracy.

*Portfolio successive halving (PoSH) auto-sklearn* (Feurer et al. [2018](#bookmark369)) is an extension of auto-sklearn with the aim of yielding good performance under tight time constraints. It introduces a more efcient meta-learning strategy and the option to use successive halving in the evaluation of pipelines in order to reduce the time spent in evaluating poorly per- forming candidate pipelines.

*Auto-sklearn 2.0* (Feurer et al. [2022](#bookmark52)) further extends PoSH auto-sklearn by adding the possibility to automatically select the evaluation strategy (holdout or cross-validation, the number of folds in cross-validation, and whether to use successive halving or not) based on meta-learning.

*Alternating direction method of multipliers (ADMM) AutoML* (Liu et al. [2020](#bookmark356)) focuses on addressing the CASH problem to optimise pipelines composed of a data-preprocessor, feature selector and a machine-learning model using ADMM (Boyd et al. [2011](#bookmark370)), an opti- misation framework for solving complex constrained mixed integer problems based on decomposition. ADMM is employed to decompose and simplify the CASH problem into easier subproblems (e.g., diferentiating between hyperparameter optimisation and algo- rithm selection) with a smaller number of hyperparameters that can be addressed using Bayesian optimisation. Reducing the number of hyperparameters can signifcantly speed up the convergence of the overall black-box optimisation process. The ADMM framework also supports constraints that, for instance, restrict the prediction latency or the memory consumption of the machine-learning pipeline.

*AutoGluon* (Erickson et al. [2020](#bookmark349)) is an AutoML system with a focus on designing pipe- lines that generates complex model ensembles, excluding preprocessing steps. Its search space is defned over multiple models from Scikit-learn, XGBoost (Chen and Guestrin [2016](#bookmark371)), LightGBM (Ke et al. [2017](#bookmark372)), CatBoost (Dorogush et al. [2018](#bookmark373)) and neural networks directly implemented in AutoGluon-Tabular. AutoGluon-Tabular takes a multi-layered ensembling approach, where multiple base models of the same type are frst combined through a bagging ensembling approach. Multiple bagging ensembles can then be com- bined into a voting ensemble. The voting ensemble can be further extended by AutoGluon- Tabular by prepending one or multiple stacking layers consisting of multiple bagging ensembles each. The sequence in which models (base models and ensembles) are evaluated is fxed and predefned; essentially, more and more complex ensembles are constructed and evaluated during the search process. AutoGluon-Tabular provides diferent strategies for

a 灬

the optimisation of hyperparameters, but by default, Bayesian optimisation is employed. Shchur et al. ([2023](#bookmark374)) extends AutoGluon to probabilistic time-series forecasting tasks based on a search space composed of statistical and machine-learning based forecasting algorithms.

**5.3 Systems based on multi‑armed bandits**

*Auto tune models (ATM)* (Swearingen et al. [2017](#bookmark358)) is a distributed, collaborative, scal- able AutoML system. ATM uses a hybrid Bayesian and multi-armed bandit optimisation method for optimising pipelines and ofers model recommendations using meta-learning. It iteratively selects a model using a multi-armed bandit approach and executes one Bayesian optimisation step on the hyperparameters of the selected model. ATM supports only pipe- lines consisting of single models without any data preprocessing. The focus of this system is on the support of multiple users in parallel, i.e., in the form of a cloud service.

*Alpine meadow* (Shang et al. [2019](#bookmark357)) is an AutoML system integrated into the North- star data science platform (Kraska [2018](#bookmark375)). It optimises fxed pipelines consisting of data preprocessors and a model. This system focuses on providing interaction opportunities with the user. The system iteratively searches and recommends pipelines to the user and adapts its search space to the task using expert rules defned by the user. In each itera- tion of the search process, Alpine Meadow selects promising pipelines (i.e., steps of pipe- lines) through a multi-armed bandit approach and optimises the hyperparameters of these pipelines with Bayesian optimisation. Meta-learning is used to warm-start the multi-armed bandit approach.

**5.4 Systems based on evolutionary algorithms**

As described in Sect. [3.2.4](#bookmark131), evolutionary algorithms are population-based optimisation algorithms. They work on sets (populations) of solution candidates (individuals). Genetic programming (Koza [1994](#bookmark137)) is a particular evolutionary approach that is often employed as the search strategy in AutoML systems as it allows for a fexible description of ML pipe- lines (see Sect. [3.2.4](#bookmark131)). EAs are easy to parallelise, as the individuals in a population can be evaluated in parallel.

*The tree-based pipeline optimisation tool (TPOT)* (Olson et al. [2016a](#bookmark11), [b](#bookmark376)) uses genetic programming to optimise tree-structured machine-learning pipelines based on the search space defned over Scikit-learn. The leaves of the TPOT’s tree-structured pipelines repre- sent hyperparameters and (copies of) the input data, while the inner nodes represent pipe- line operators (preprocessors, decomposition, feature selection, models). Taking each oper- ator in a pipeline as a primitive, TPOT can construct arbitrarily complex pipelines using genetic programming. However, this may also lead to infeasible pipelines (e.g., a classifca- tion pipeline that does not include a classifer). Furthermore, the complexity of the result- ing pipelines increases the risk of overftting. To address this latter issue, TPOT maintains a Pareto front of previously evaluated pipelines with respect to the two objectives of pre- diction accuracy and complexity (in terms of the number of pipeline operators), allowing the user to choose a reasonable trade-of.

*Layered TPOT* (Gijsbers et al. [2017](#bookmark377)) and *TPOT-SH* (Parmentier et al. [2019](#bookmark378)) are both extensions of TPOT that focus on accelerating the optimisation process on large datasets. This is achieved by frst evaluating pipelines on smaller subsets of training data and select- ing only the promising pipelines for training on larger subsets. Both of these approaches

a 灬

induce the risk of missing viable pipelines that perform poorly on a subset of the data but achieve high accuracy when trained on the entire dataset.

*REsilient ClassifIcation Pipeline Evolution (RECIPE)* (de Sá et al. [2017](#bookmark365)) is an AutoML system focused on constructing classifcation pipelines that include preprocessing and clas- sifcation models (from Scikit-learn) using grammar-based genetic programming (GGP) (McKay et al. [2010](#bookmark379)). Compared to TPOT’s fexible genetic programming approach, that may lead to infeasible pipelines, using this grammar-based approach, RECIPE constrains the genetic programming process (more specifcally, the crossover and mutation opera- tions) by employing background knowledge on the structure of feasible pipelines. Similar to TPOT, RECIPE does not rely on a fxed pipeline structure and can construct complex pipelines. In contrast to TPOT, RECIPE does not support the union of multiple preprocess- ing steps, but only sequences of preprocessing steps. On the other hand, it supports voting ensembles, which are not supported by TPOT.

*Autostacker* (Chen et al. [2018](#bookmark350)) is an AutoML system that employs a basic evolution- ary algorithm (diferent from genetic programming) to optimise machine-learning pipe- lines with a search space composed of models (excluding preprocessors) from Scikit-learn (Buitinck et al. [2013](#bookmark102)) and XGBoost (Chen and Guestrin [2016](#bookmark371)). In contrast to TPOT and RECIPE, Autostacker constructs pipelines with a special stacking structure. These pipe- lines consist of multiple layers each, including multiple machine-learning models. The models in the frst layer take the raw input data as input, and models in subsequent layers can additionally make use of the outputs of models in preceding layers.

*General automated machine learning assistant (GAMA)* (Gijsbers and Vanschoren [2020](#bookmark362)) is an AutoML system that uses genetic programming to generate machine-learn- ing pipelines based on a given input dataset. GAMA is implemented based on the search space of Scikit-learn and automatically constructs pipelines that include preprocessing and machine-learning models. GAMA supports building an ensemble of the evaluated pipe- lines. The main distinguishing feature of GAMA compared to other AutoML systems, such as auto-sklearn and TPOT, is its focus on transparency to serve AutoML researchers by producing extensive log fles about the behaviour of the population of pipelines during the optimisation process.

*AutoML-DSGE* (Assunção et al. [2020](#bookmark359)) is another AutoML system that employs gram- mar-based genetic programming to optimise classifcation pipelines consisting of Scikit- learn data preprocessors and models. AutoML-DSGE uses dynamic structured gram- matical evolution (DSGE) (Lourenço et al. [2018](#bookmark380)), which is an extension of grammatical evolution (GE). GE uses a linear representation, i.e., individuals are represented in the form of lists of integers. Compared to tree-based representations, as used by RECIPE, a linear representation has the advantage of being easier to operate on and permitting the application of a wider range of evolutionary operators (McKay et al. [2010](#bookmark379)). DSGE uses an improved encoding with a higher locality (i.e., small changes in the genotype yield also only small changes in the phenotype) and a lower redundancy (i.e., diferent genotypes yield the same phenotype) compared to GE.

*FEDOT* (Nikitin et al. [2022](#bookmark360)) is an EA-based AutoML system for optimising pipelines, which operates in two phases: composition and hyperparameter tuning. During the com- position phase, analogously to TPOT, FEDOT optimises the structure of machine-learning pipelines represented as DAGs with preprocessors and models from Scikit-learn, XGBoost, LightGBM, and CatBoost, with the help of a tree-based genetic algorithm. However, in contrast to TPOT, it does not optimise the hyperparameters at this stage but considers only default hyperparameter settings. In a second phase, it tunes the hyperparameters of the best

a 灬

pipeline found by the genetic algorithm by means of Bayesian optimisation, using Hyper- opt (Bergstra et al. [2013](#bookmark120)) (see also Sect. [3.4](#bookmark62)).

**5.5 Systems based on Monte Carlo tree search**

Monte Carlo tree search (MCTS) is a heuristic search approach that has been used widely in turn-based games, such as Chess or Go. As outlined in Sects. [3](#bookmark37) and [4](#bookmark38), MCTS has also been used in hyperparameter optimisation and neural architecture search; furthermore, there are a few systems for the optimisation of machine-learning pipelines based on MCTS.

*MOSAIC* (Rakotoarison et al. [2019](#bookmark151)) is an AutoML system designed for optimising machine-learning pipelines with a predefned number of steps based on a search space defned over Scikit-learn (ML models and preprocessors). MOSAIC employs a hybrid opti- misation approach combining MCTS and Bayesian optimisation. The authors of MOSAIC motivate this hybrid approach by pointing out that MCTS can efciently solve sequential decision problems (in this case, the sequence of components that form a pipeline), whereas Bayesian optimisation is well-suited for solving expensive optimisation problems (in this case, hyperparameter settings of the components of the pipeline). The two extreme solu- tions one could consider for jointly optimising the structure and hyperparameters of a pipe- line are: (i) to optimise hyperparameters of every possible pipeline structure or (ii) to esti- mate the performance of a pipeline structure by sampling a few hyperparameter settings of each pipeline structure. MOSAIC adopts an intermediate solution by coupling these two optimisation approaches tightly via a shared surrogate model, which, on the one hand, is incorporated in the MCTS approach by approximating the average performance of pipe- lines, and on the other, allows hyperparameter optimisation of given pipelines using Bayes- ian optimisation. Similar to Bayesian optimisation approaches, MOSAIC builds a surro- gate model from ML pipelines and their hyperparameter confgurations evaluated earlier in the search process. Next, this surrogate model is used to derive a second surrogate model on pipeline structures from a number of hyperparameter confgurations sampled for each structure. The second model is used to guide the MCTS.

*Oracle AutoML* (Yakovlev et al. [2020](#bookmark15)) is an iteration-free AutoML system for designing pipelines based on meta-learning and a gradient descent approach. Algorithm selection, adaptive data reduction, and hyperparameter optimisation are the three main components in the Oracle AutoML system. In the algorithm selection component, the Oracle AutoML system uses meta-learning to predict the relative performance of algorithms on subsets of the given data and selects the best algorithm according to these predictions. The adaptive data reduction component selects a representative subset of the data. On this subset, the hyperparameters of the selected algorithm are optimised in parallel, using a combination of gradient-based optimisation (for continuous and discrete parameters) and a brute-force approach (for categorial parameters).

**5.6 Systems based on other methods**

In this section, we discuss AutoML systems based on approaches other than those covered in previous subsections.

*ML-Plan* (Mohr et al. [2018](#bookmark90)) is an AutoML system for constructing fxed AutoML pipe- lines (composed of a preprocessor and a classifer) based on a search space defned over Weka and Scikit-learn. ML-Plan does not use any of the HPO approaches mentioned previ- ously but rather regards AutoML pipeline construction as an AI planning task. Hierarchical

a 灬

planning can be used to organise the search space, and the resulting AI planning problem can be solved using the well-known approach of hierarchical task networks (HTNs) (Erol et al. [1994](#bookmark381)). In this case, AutoML tasks can be understood as graph search problems. A plan is a sequence of actions (e.g., selection of a specifc algorithm) that can be organised in a hierarchically structured network that represents the dependencies of the actions. A lack of representative data for the search process leads to overftting for most AutoML sys- tems. ML-Plan addresses this problem by applying a two-phase search mechanism (search and selection). In the frst phase, a global best-frst graph search algorithm is used to iden- tify good candidate pipelines on a part of the training data. In the second phase, the fnal pipeline is selected by applying Monte Carlo cross-validation on the full training data to the candidate pipelines identifed in the frst phase.

*ML2-Plan* (Wever et al. [2018](#bookmark382)) was introduced to extend ML-Plan for multi-label clas- sifcation. While most AutoML systems focus on single-label classifcation or regression, ML2-plan builds upon the search space of MEKA (Read et al. [2016](#bookmark383))—a multi-label exten- sion of WEKA. Since the preprocessors in ML-Plan are for single-label classifcation, they were not included in ML2-Plan.

*Oboe* (Yang et al. [2019](#bookmark53)) is an AutoML system that optimises machine-learning pipe- lines to create model ensembles with the help of meta-learning. Oboe employs collabo- rating fltering to select models for new datasets based on their performance on similar datasets. In an ofine stage, it evaluates all candidate models (in the form of algorithms and corresponding hyperparameter settings) on a set of datasets and creates a matrix of cross-validation errors for those models. As opposed to explicitly extracting meta-features, ftting a low-rank model on this matrix allows learning latent meta-features for models and datasets that best describe the cross-validated error. To predict the performance of mod- els on new datasets, Oboe introduces a time-constrained matrix completion method based on the optimal experiment design approach (Pukelsheim [2006](#bookmark384)), a classic method to defne optimal experiments according to statistical criteria. In this case, the optimal experiment is inferring the meta-features of the datasets, and the statistical criterion is minimising the covariance of the estimated meta-features.

*TensorOboe* (Yang et al. [2020](#bookmark385)) is an extension of Oboe that allows the optimisation of full pipelines consisting of data preprocessing steps, like imputation and scaling, and a model.

*Fast and Lightweight AutoML (FLAML)* (Wang et al. [2021b](#bookmark361)) is an AutoML system that other than model accuracy, focuses on optimising computational resources for an ef- cient search process. FLAML optimises machine-learning pipelines consisting of a single model without data preprocessing with a search space defned over models from Scikit- learn, XGBoost, LightGBM and CatBoost. For the hyperparameter optimisation, a direct search approach, as proposed by Wu et al. ([2021](#bookmark386)), is employed. The goal of the search is to minimise the total CPU time for fnding a model with a small error. To achieve this goal, in each search iteration, next to selecting a model and a corresponding hyperparameter con- fguration to evaluate, FLAML also determines the training sample size used in the evalu- ation as well as the evaluation strategy (hold-out or cross-validation). Based on previous evaluations, FLAML estimates for each model the CPU time offnding a confguration that results in an error lower than the currently best one. These estimations are used to select models and training sample sizes for each iteration of the search. This way, starting from cheap trials and low-performing models, the search will gradually move to expensive trials with more accurate models.

*TabPFN* (Hollmann et al. [2023](#bookmark366)) is a transformer-based AutoML system based on meta- learning (Brazdil et al. [2022](#bookmark51)). The underlying transformer model has been trained on many

a 灬

synthetic tabular datasets drawn from a given prior to approximate Bayesian inference. The meta-learning component learns across datasets and, when presented with a new dataset, makes predictions for the test set without additional training or hyperparameter optimisa- tion, as the transformer model has been trained to perform this dynamically. One of the current limitations of this system is that there is a practical limit to the size of datasets for which it can make predictions, as the transformer model scales quadratically with the input size. This specifc learning architecture was pioneered by Hochreiter et al. ([2001](#bookmark387)) on LSTMs, and later, Huisman et al. ([2023](#bookmark161)) revisited this LSTM-based architecture on few- shot learning image classifcation benchmarks.

**5.7 Selecting an AutoML system**

The choice between a large number of diferent AutoML systems can pose signifcant chal- lenges for practitioners. This choice can be made based on diferent criteria, an important one being the performance in fnding good machine learning pipelines. When proposing a new AutoML system, researchers typically rely on empirical performance comparisons against other AutoML systems. However, it is questionable whether these comparisons are free from bias. Recently, independent benchmarks of AutoML systems have been pub- lished to facilitate more objective evaluations. An overview of existing benchmarks is pro- vided in Table [4](#bookmark388).

As noted in the table, all these benchmarks rely on problems from OpenML (Van- schoren et al. [2013](#bookmark85)), and for none of them, a single AutoML system consistently outper- formed all others on all problems. Furthermore, as pointed out by Zöller and Huber ([2021](#bookmark27)) and Gijsbers et al. ([2022](#bookmark389)), such benchmarks have diferent limitations, including the fol- lowing: (i) The systems have incompatible interfaces (e.g., which stopping criterion can be set). (ii) The setting of their respective hyperparameters has an impact on their perfor- mance. (iii) Diferent time budgets might yield diferent results – one system might per- form poorly with a low time budget, but outperform the other systems when given more time, and vice versa. (iv) The systems compared have diferent search spaces. However, most systems allow adaptation of the search space, and future studies should consider com- paring AutoML systems based on similar or identical search spaces. These and other issues render a fair comparison difcult; thus, the results of the current benchmarks can only give a rough indication of the relative performances of various AutoML systems.

Besides performance, other criteria that can play an important role in the selec- tion of an AutoML system, including ease of use, the capability of automatically han- dling missing data, support for diferent feature types, and availability of a graphical user interface. Scriven et al. ([2022](#bookmark390)) evaluated features of 19 open source and 25 commercial AutoML systems, which are related to criteria including ease of use, explainability and handling of imperfect data. Among the open-source systems studied in this research, TPOT and PyCaret (Ali [2020](#bookmark391)), and among the commercial ones, Dataiku ([2023](#bookmark392)) and DataRobot ([2023](#bookmark393)) achieved the highest scores. Their study also concludes that the commercial systems outperform the open-source ones. Notably, these systems are superior in terms of explain- ability, ease of use, deployment and data security. However, the open source AutoML systems are generally better in terms of handling imperfect data and coverage of machine learning tasks (classifcation, regression, time series prediction, image-based problems, etc.). Furthermore, the authors hypothesise that the open-source systems are likely more performant than the commercial systems.

a 灬

a 灬

**Table 4** Overview of benchmarks of AutoML systems

|  |  |  |  |
| --- | --- | --- | --- |
| Work | Compared systems | Datasets | Conclusion |
| Balaji and Allen ([2018](#bookmark394)) Truong et al. [(2019](#bookmark395))  Zöller and Huber  [(2021](#bookmark27))  Eldeeb et al. [(2022](#bookmark396)) Gijsbers et al. [(2022](#bookmark389)) | Auto-sklearn, auto\_ml, H2O, TPOT  Auto-sklearn, AutoKeras, Darwin, H2O, Ludwig, TPOT  Auto-sklearn, ATM, H2O, Hyperopt-sklearn, TPOT  Auto-sklearn, Auto-WEKA, ATM, SmartML, RECIPE, TPOT  Auto-sklearn, AutoGluon, FLAML, GAMA, H2O, LightAutoML, MLJAR, TPOT | 57 classifcation and 30 regression problems from OpenML  200 classifcation and 100 regression problems from OpenML  73 classifcation problems from OpenML  100 classifcation problems from OpenML  71 classifcation and 33 regression problems from OpenML | Auto-sklearn performed best for classifcation and TPOT performed best for regression  H2O, Auto-sklearn and AutoKeras performed best  TPOT and Auto-sklearn performed best ATM and TPOT performed best  AutoGluon performed best |

**122** Page 66 of 88 M Baratchi et a l

**Table 5** Dates of last releases and number of github stars of diferent open source AutoML systems as of September 20,

2023

|  |  |  |
| --- | --- | --- |
| System | Last release | GitHub stars |
| H2O | No ofcial release last com- mit in GitHub: September 2023 | 6.5k |
| TPOT | August 2023 | 9.2k |
| AutoGluon | July 2023 | 6.2k |
| FEDOT | July 2023 | 0.5k |
| GAMA | June 2023 | 0.09k |
| Auto-sklearn | February 2023 | 7.1k |
| Auto-WEKA | March 2022 | 0.3k |
| RECIPE | No ofcial release last com- mit in GitHub: July 2021 | 0.05k |
| ATM | July 2019 | 0.5k |

When selecting an AutoML system, one might also consider how actively the software is further developed and how big the user base is. Table [5](#bookmark397) lists the dates of the most recent releases of diferent open source AutoML systems according to the information provided by their corresponding github repositories as of 20 September 2023. Furthermore, the number of stars of the corresponding github repositories is listed, which is used by Eldeeb et al. ([2022](#bookmark396)) as an indicator for popularity.

The nature of the datasets at hand might also play a role in the choice of an AutoML system, based on the following considerations:

• *Search space* For practical reasons, a user might prefer to use certain machine-learning algorithms or libraries. The type of datasets encountered in a specifc use context also determines the preprocessing steps needed in the pipeline. In that case, it should be considered if the search space of the AutoML system contains the respective methods.

• *Ensembling strategy* An advanced ensembling strategy can be expected to confgure models with much higher accuracy than achievable by a single algorithm; however, this typically comes at the cost of additional computational resources in case ensembling is performed during the search rather than post-hoc (i.e., using the previously trained models, the costs are much lower).

• *Meta-learning* A user should consider if there is meta-learning implemented in the AutoML system, but more importantly, if the datasets at hand are similar to the datasets considered for meta-learning by the AutoML system and are expected to beneft from meta-learning.

**6 Outlook and conclusions**

In this section, we cover open challenges and directions for future research in the area of AutoML that we believe to be particularly relevant.

*AutoML for unstructured data* Most research in automated machine learning has focused on supervised learning (classic regression and classifcation tasks) for tabular data. In NAS, there is a major focus on optimising convolutional and recurrent neural networks. More

a 灬

recently, graph neural networks have also attracted attention (see, e.g., Gao et al. [2020](#bookmark398); Li et al. [2021](#bookmark399); Ding et al. [2021](#bookmark400)). Support for other types of structured data that are relevant in many practical applications, such as time-series and spatio-temporal data, is still lim- ited. To develop AutoML methods for these types of data, more specialised search spaces need to be defned by adding other hyperparameters or preprocessing elements. The search space is where human expertise in designing specialised algorithms can be most easily incorporated in order to render the search process more efcient. Specialised search spaces have been created and used successfully for time-series (see, e.g., Wang et al. [2022a](#bookmark401), [b](#bookmark402)) and spatio-temporal datasets (see, e.g., Li et al. [2020b](#bookmark403)); however, there is substantial room for further work in this area.

*Automated machine learning with limited amounts of labelled data* For many machine learning models, especially for deep neural networks, the size of the labelled training dataset has a signifcant infuence on the fnal performance. Platforms such as Google Cloud AutoML provides the service of generating high-quality training data through Google’s human labelling service. However, many data instances cannot be eas- ily labelled without domain expertise (e.g., medical diagnoses) or are very difcult to label even with such expertise (e.g., multi-spectral satellite images). Transfer learning can increase the efciency of AutoML systems and reduce the need for labelled datasets (Wong et al. [2018](#bookmark404)). For instance, Salinas et al. ([2021](#bookmark405)) study how incorporating trans- fer learning in AutoML systems can let NAS systems developed for natural images be used for remote sensing images. Öztürk et al. ([2022](#bookmark406)) study how to efciently select a pretrained model to fne-tune for a given dataset through meta-learning from a large number of datasets and deep learning pipelines. Active learning and semi-supervised learning are other related techniques that can help train models with smaller amounts of labelled data. Supporting these tasks within AutoML systems could be achieved by considering objective functions that can assess the quality of the trained models with a much smaller number of labelled instances. Changes to the objectives might render model selection approaches such as cross-validation obsolete, as demonstrated by Chen and Wujek ([2020](#bookmark407)) for the case of active learning. Furthermore, some of the techniques used to improve the performance of AutoML systems for supervised learning may not be applicable anymore [e.g., meta-features, as demonstrated by Li et al. ([2019](#bookmark346))]. There may also be instabilities in performance improvement when using a mixture of labelled and unlabelled instances in semi-supervised learning.

*Multi-objective AutoML* A majority of AutoML systems focus on single-objective opti- misation based on regression or classifcation accuracy. Considering more than a single objective can increase the potential of fnding models or pipelines that are better suited for specifc applications. Solutions to most machine learning problems are often best assessed using multiple performance indicators, such as precision and recall. There are also ML problems of an inherently multi-objective nature, such as early time-series classifcation. As demonstrated by Ottervanger et al. ([2021](#bookmark408)), applying a multi-objective optimisation approach to algorithm selection and hyperparameter optimisation for such problems can provide opportunities to present a more diverse set of Pareto-optimal solutions. Many of the HPO approaches mentioned previously based on reinforcement learning or evolution- ary algorithms support multi-objective optimisation. However, in order to become appli- cable to general AutoML problems, these approaches need to be expanded to support tree- structured search spaces and conditional hyper-parameters.

*Cost-aware and cost-efcient search* For NAS, multi-objective optimisation has been considered to create neural network architectures that run on resource-constrained devices, such as mobile phones (see, e.g., Tan et al. [2019](#bookmark281)). These solutions mainly consider the

a 灬

running time of the models as an objective. With the increasing interest in training expen- sive models (e.g., large language models, vision models), efcient search and performance evaluation is becoming more relevant. In order to beneft from AutoML advances in these areas, research on cost-aware and cost-efciency of NAS and AutoML systems (e.g., through zero-shot learning) is more relevant than ever.

*Trustworthy AutoML* Next to the objectives that target computational performance met- rics, it is increasingly desirable to consider objectives or constraints related to the trust- worthiness and fairness of ML systems (e.g., König et al. [2020](#bookmark168), [2022](#bookmark175); Perrone et al. [2021](#bookmark409)). Goals such as trustworthiness are not yet crisply defned; therefore, automatically assessing the performance of ML systems based on these factors requires additional work on new performance metrics, whose optimisation increases widely accepted notions of trustworthi- ness. Trustworthiness and related requirements such as bias and fairness are socio-tech- nical topics. Therefore, the research in AutoML addressing these topics should consider both the design of the AutoML systems and their users. Weerts et al. ([2023](#bookmark180)) point to the challenges of incorporating fairness in the design of machine learning systems and discuss the potentials and limitations of fairness-aware AutoML. Fairness-aware AutoML systems can accelerate the integration of fairness criteria in ML pipelines. For instance, Fefer et al. ([2023](#bookmark410)) show that this can be achieved through searching within the search space of bias mitigators that improve group fairness (e.g., through adversarial debiasing or removing desperate and prejudice impacts) and ensembling strategies employed to achieve a trade-of between fairness and accuracy.

*Larger search spaces and additional design choices* Although research in AutoML sup- ports training a varied set of diferent types of model families, the main focus is still on optimising hyperparameters based on a specifc search space. Still, a user should decide which kind of search space is relevant. Future research should consider extending these search spaces to allow a model to be trained automatically without much user involvement. Automating the process of confguring models in such a broad search space will make the process even more challenging and thus require even more focus on increasing the ef- ciency of the search strategy.

*More efcient search space design for NAS* As mentioned previously, a modular cell- based design can largely constrain the search space. However, a post-hoc analysis of vari- ous existing cell-based search spaces by Wan et al. ([2022](#bookmark411)) still demonstrates a large amount of redundancy in the hyperparameters considered. Their analysis concludes that only a lim- ited number of operations within cells are important while observing that commonly used cell types, such as reduction cells, are unimportant in increasing the performance. This points to opportunities for further optimising the search space, which will hopefully allow for fnding better architectures much faster.

*AutoML for lifelong learning* So far, most AutoML systems consider a stable data gen- eration process, and AutoML approaches for lifelong machine learning remain largely unexplored. For many datasets collected over time or in a streaming setting, it is possible that the best model found at a specifc time is not the best one overall. Being able to iden- tify changes in the data generation process can beneft machine learning models when used for streaming settings. For example, Celik and Vanschoren ([2021](#bookmark412)) have tested six diferent concept drift adaptation strategies that can detect and adapt to the changes in the data gen- eration process. Tetteroo et al. ([2022](#bookmark413)) have implemented and compared these approaches in the context of Covid-19 time-series forecasting, where the data generation process under- goes various changes. We believe that AutoML systems can beneft from these kinds of strategies by incorporating them into the search process. Celik et al. ([2023](#bookmark414)) have recently

a 灬

proposed an adaptive AutoML framework for online learning built upon a search space of online learning algorithms.

*Automated data science* As discussed by De Bie et al. ([2022](#bookmark1)), to automate data science, many steps, from the point of data generation to decision-making need to be optimised. These steps include (1) data exploration, (2) data engineering, (3) model building, and (4) exploitation. As evident in our extensive review, although the area of AutoML is thriving and has attracted much attention in recent years, most AutoML research so far has focused on model building and mainly targeted supervised learning problems. To achieve the most ambitious goals of AutoML, problems beyond supervised learning and steps other than model building should be better supported by future AutoML systems. We also note that currently, the use of many AutoML approaches still involves choices that critically depend on expert knowledge, such as the selection of an AutoML system and, in many cases, its confguration for a given use case. Ideally, the levels of expertise required for carrying out these tasks should be reduced by further automation while maintaining a meaningful level of human insight and control.

In general, HPO can be used to automate diferent learning problems with a well- defned search space. Another related feld to AutoML that uses HPO is Automated Reinforcement Learning (AutoRL). AutoRL focuses on HPO on the search space com- posed of design choices in constructing reinforcement learning pipelines. For instance, in deep reinforcement learning, the policy is implemented in the form of a neural net- work, the parameters of which are improved based on diferent objective functions. The objective function itself can be considered as a hyperparameter and subjected to HPO. Mohan et al. ([2023](#bookmark415)) analyses the hyperparameter landscapes (that map hyper- parameter values to a given performance metric) in reinforcement learning pipelines, demonstrating the non-stationary of reinforcement learning that necessitates a more sophisticated and dynamic approach to HPO compared to AutoML. This makes AutoRL a much more challenging problem than AutoML. We refer interested readers to recently published work by Mohan et al. ([2023](#bookmark415)), Parker-Holder et al. ([2022](#bookmark416)).

*AutoML and generative AI* In the past few years, various generative AI models have been developed, supporting the use of natural language for the recognition and gen- eration of text and images. The impressive performance of these models stems from being trained on large datasets using complex transformer architectures. There is much potential in exploring synergies between these models and AutoML systems. Tornede et al. ([2023a](#bookmark417)) already speculate about various ways in which large language models and AutoML can be synergistic: (i) automated machine learning can be used to optimise various components of large language models, and (ii) large language models can be used as an interface to automated machine learning models. As an example of the lat- ter, Shen et al. ([2023](#bookmark418)) propose using an LLM-based agent for carrying various expert- level AI tasks based on user prompts. Their proposed system, HuggingGPT, can parse user requests and decompose them into standard tasks, select models for each task and execute these. The model selection capability of HuggingGPT currently mainly consid- ers user-defned instructions provided in the form of a list of models and well-defned model descriptions. Future research can enhance the models generated by such a system by incorporating automated model selection and hyperparameter optimisation. Another research direction involves improving the efciency of transformer models using AutoML. Transformer architectures have been shown to give rise to neural network models that perform impressively well on a variety of diferent tasks. However, this comes at the price of considerable computational cost in running these models. NAS

a 灬

research can focus on making these models more efcient, for instance, by identifying smaller or sparser architectures.

Overall, AutoML is still a relatively recent and very fast-growing research area. Many researchers are now working on addressing prominent open challenges, and we expect to see rapid progress in the directions outlined earlier in this section. This will render AutoML approaches and systems even more powerful, versatile and usable. We expect that, driven by academic research and commercial oferings, the real-world use of AutoML systems will increase sharply in the near future. We hope that our survey can help facilitate this development, by providing an overview to researchers and practitioners new to the area of AutoML.

We believe that as AutoML gains further traction, it is very important to not lose sight of the importance of human understanding and oversight in many application situations. Unless it is applied with care, automation can be detrimental in this respect; we, there- fore, see a great need for a focus on the design and principled deployment of AutoML sys- tems and techniques that address this challenge. Work in this direction will certainly play a major role for the future not only of the area of AutoML, but also of the broader felds of

machine learning, data science and artifcial intelligence.

**Acknowledgements** This work is part of the research programme *C2D–Horizontal Data Science for Evolv- ing Content*, under the DACCOMPLI project (Project Number 628.011.002) and the “Physics-aware Spatio- temporal Machine Learning for Earth Observation Data” project (Project Number OCENW.KLEIN.425) of the research programme Open Competition ENW, which are partly fnanced by the Dutch Research Council (NWO). This research was also partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

**Author contributions** M.B., C.W, S.L and J.v.R had the lead in writing the manuscript and carried out most of the background research. C.W. prepared most of the fgures and the Bibliography Section. H.H., T.B. and M.O. made major contributions in defning the scope and structure of the survey as well as to the fnal text.

**Confict of interest** The authors declare no confict of interest.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Com- mons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

**References**

Abdelfattah MS, Mehrotra A, Dudziak L et al (2021) Zero-cost proxies for lightweight NAS. In: Proceed- ings of the 9th international conference on learning representations, virtual event, Austria, 3–7 May 2021

Ahmed K, Torresani L (2018) MaskConnect: connectivity learning by gradient descent. In: Proceedings of the 15th European conference on computer vision, Munich, Germany, 8–14 September 2018. pp 362–378. <https://doi.org/10.1007/978-3-030-01228-1_22>

Ahmed AA, Darwish SMS, El-Sherbiny MM (2019) A novel automatic CNN architecture design approach based on genetic algorithm. In: Proceedings of the international conference on advanced intelligent systems and informatics, Cairo, Egypt, 26–28 October 2019. pp 473–482

Akhauri Y, Abdelfattah MS (2023) Multi-predict: few shot predictors for efcient neural architecture search. In: Proceedings of the AutoML conference, Potsdam, Germany, 12–15 September 2023

a 灬

Akiba T, Sano S, Yanase T et al (2019) Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM international conference on knowledge discovery & data mining,

Anchorage, AK, USA, 4–8 August 2019. pp 2623–2631. <https://doi.org/10.1145/3292500.3330701> Ali M (2020) PyCaret: an open source, low-code machine learning library in Python. PyCaret version 1.0

Ali S, Smith KA (2006) On learning algorithm selection for classifcation. Appl Soft Comput 6(2):119–138 Angeline PJ, Saunders GM, Pollack JB (1994) An evolutionary algorithm that constructs recurrent neural

networks. IEEE Trans Neural Netw 5(1):54–65. <https://doi.org/10.1109/72.265960>

Antonov IA, Saleev V (1979) An economic method of computing lp휏-sequences. USSR Comput Math Math Phys 19(1):252–256

Assunção F, Lourenço N, Ribeiro B et al (2020) Evolution of scikit-learn pipelines with dynamic struc- tured grammatical evolution. In: Proceedings of the 23rd international conference on the applications of evolutionary, Seville, Spain, 15–17 April 2020. pp 530–545. [https://doi.org/10.1007/978-3-030-](https://doi.org/10.1007/978-3-030-43722-0_34) [43722-0\_34](https://doi.org/10.1007/978-3-030-43722-0_34)

Bäck T, Fogel DB, Michalewicz Z (1997) Handbook of evolutionary computation. Release 97(1):B1

Bäck T, Fogel DB, Michalewicz Z (2018) Evolutionary computation 1: basic algorithms and operators. CRC Press, Boca Raton

Bai T, Luo J, Zhao J et al (2021) Recent advances in adversarial training for adversarial robustness. In: Zhou Z (ed) Proceedings of the thirtieth international joint conference on artifcial intelligence, virtual  event/Montreal, Canada, 19–27 August 2021. pp 4312–4321. <https://doi.org/10.24963/ijcai.2021/591> Baker B, Gupta O, Naik N et al (2017) Designing neural network architectures using reinforcement learn- ing. In: Proceedings of the 5th international conference on learning representations, Toulon, France,

24–26 April 2017

Baker B, Gupta O, Raskar R et al (2018) Accelerating neural architecture search using performance predic- tion. In: Workshop track proceedings of the 6th international conference on learning representations, Vancouver, BC, Canada, 30 April–3 May 2018

Balaji A, Allen A (2018) Benchmarking automatic machine learning frameworks. [https://doi.org/10.48550/](https://doi.org/10.48550/ARXIV.1808.06492) [ARXIV.1808.06492](https://doi.org/10.48550/ARXIV.1808.06492)

Barbudo R, Ventura S, Romero JR (2023) Eight years of AutoML: categorisation, review and trends. Knowl Inf Syst 65(12):5097–5149

Bender G, Kindermans P, Zoph B et al (2018) Understanding and simplifying one-shot architecture search. In: Proceedings of the 35th international conference on machine learning, Stockholmsmässan, Stock- holm, Sweden, 10–15 July 2018. pp 549–558

Bengio Y (2000) Gradient-based optimization of hyperparameters. Neural Comput 12(8):1889–1900

Bengio Y (2012) Practical recommendations for gradient-based training of deep architectures. In: Neural networks: tricks of the trade. Springer, pp 437–478

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13:281–305

Bergstra J, Bardenet R, Bengio Y et al (2011) Algorithms for hyper-parameter optimization. In: Proceedings of the 24th international conference on neural information processing systems, Sierra Nevada, Spain, 16–17 December 2011. pp 2546–2554

Bergstra J, Yamins D, Cox DD (2013) Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: Proceedings of the 30th international conference on machine learning, Atlanta, GA, USA, 16–21 June 2013. pp 115–123

Biedenkapp A, Lindauer M, Eggensperger K et al (2017) Efcient parameter importance analysis via abla- tion with surrogates. In: Proceedings of the thirty-frst AAAI conference on artifcial intelligence, San Francisco, CA, USA, 4–9 February 2017. pp 773–779

Binder M, Pfsterer F, Bischl B (2020) Collecting empirical data about hyperparameters for data driven AutoML. In: Proceedings of the 7th international conference on machine learning workshop on auto- mated machine learning, virtual event, 13–18 July 2020

Bischl B, Casalicchio G, Feurer M et al (2021) OpenML benchmarking suites. In: Proceedings of the neural information processing systems track on datasets and benchmarks 1, virtual event, December 2021

Bisong E (2019) Google AutoML: cloud vision. In: Building machine learning and deep learning models on Google cloud platform. Springer, pp 581–598

Bosman AW, Hoos HH, van Rijn JN (2023) A preliminary study of critical robustness distributions in neural network verifcation. In: Proceedings of the 6th workshop on formal methods for ML-enabled autono- mous systems

Botoeva E, Kouvaros P, Kronqvist J et al (2020) Efcient verifcation of ReLU-based neural networks via dependency analysis. In: Proceedings of the 34th AAAI conference on artifcial intelligence, New York, NY, USA, 7–12 February 2020. pp 3291–3299

a 灬

Boyd S, Parikh N, Chu E et al (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. Found Trends Mach Learn 3(1):1–122. [https://doi.org/10.1561/22000](https://doi.org/10.1561/2200000016) [00016](https://doi.org/10.1561/2200000016)

Brazdil P, Soares C (2000) A comparison of ranking methods for classifcation algorithm selection. In: Proceedings of the 11th European conference on machine learning, Barcelona, Catalonia, Spain, 31 May–2 June. pp 63–74. <https://doi.org/10.1007/3-540-45164-1_8>

Brazdil P, van Rijn JN, Soares C et al (2022) Metalearning: applications to automated machine learning and data mining, 2nd edn. Springer, Cham

Breiman L (2001) Random forests. Mach Learn 45(1):5–32. <https://doi.org/10.1023/A:1010933404324>

Brock A, Lim T, Ritchie JM et al (2018) SMASH: one-shot model architecture search through hypernet- works. In: Proceedings of the 6th international conference on learning representations, Vancouver, BC, Canada, 30 April–3 May 2018

Brumen B, Cernezel A, Bosnjak L (2021) Overview of machine learning process modelling. Entropy 23(9):1123

Buitinck L, Louppe G, Blondel M et al (2013) API design for machine learning software: experiences from the scikit-learn project. In: Proceedings of the European conference on machine learning and prin- ciples and practice of knowledge discovery in databases workshop: languages for data mining and machine learning, Prague, Czech Republic, 23–27 September. pp 108–122

Cai H, Chen T, Zhang W et al (2018a) Efcient architecture search by network transformation. In: Proceed- ings of the thirty-second AAAI conference on artifcial intelligence, New Orleans, LA, USA, 2–7 February 2018. pp 2787–2794

Cai H, Yang J, Zhang W et al (2018b) Path-level network transformation for efcient architecture search. In: Proceedings of the 35th international conference on machine learning, Stockholmsmässan, Stock- holm, Sweden, 10–15 July 2018. pp 677–686

Cai H, Zhu L, Han S (2019) ProxylessNAS: direct neural architecture search on target task and hardware. In: Proceedings of the 7th international conference on learning representations, New Orleans, LA, USA, 6–9 May 2019

Cambronero JP, Cito J, Rinard MC (2020) AMS: generating AutoML search spaces from weak specifca- tions. In: Proceedings of the 28th ACM joint European software engineering conference and sympo- sium on the foundations of software engineering, virtual event, 8–13 November 2020. pp 763–774

Celik B, Vanschoren J (2021) Adaptation strategies for automated machine learning on evolving data. IEEE Trans Pattern Anal Mach Intell 43(9):3067–3078. <https://doi.org/10.1109/TPAMI.2021.3062900>

Celik B, Singh P, Vanschoren J (2023) Online AutoML: an adaptive AutoML framework for online learn- ing. Mach Learn 112(6):1897–1921

Chaslot G, Bakkes S, Szita I et al (2008a) Monte-Carlo tree search: a new framework for game AI. In: Pro- ceedings of the artifcial intelligence and interactive digital entertainment. pp 216–217

Chaslot GMJ, Winands MH, Herik HJvd et al (2008b) Progressive strategies for Monte-Carlo tree search. New Math Nat Comput 4(03):343–357

Chen T, Guestrin C (2016) XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd interna- tional conference on knowledge discovery and data mining, San Francisco, CA, USA, 13–17 August 2016. pp 785–794. <https://doi.org/10.1145/2939672.2939785>

Chen T, Goodfellow IJ, Shlens J (2016) Net2Net: accelerating learning via knowledge transfer. In: Proceed- ings of the 4th international conference on learning representations, San Juan, Puerto Rico, 2–4 May 2016

Chen B, Wu H, Mo W et al (2018) Autostacker: a compositional evolutionary learning system. In: Pro- ceedings of the genetic and evolutionary computation conference, Kyoto, Japan, 15–19 July 2018. pp 402–409. <https://doi.org/10.1145/3205455.3205586>

Chen X, Qiao B, Zhang W et al (2019a) Neural feature search: a neural architecture for automated fea- ture engineering. In: Proceedings of the 2019 IEEE international conference on data mining, Beijing, China, 8–11 November 2019. pp 71–80. <https://doi.org/10.1109/ICDM.2019.00017>

Chen X, Xie L, Wu J et al (2019b) Progressive diferentiable architecture search: bridging the depth gap between search and evaluation. In: Proceedings of the IEEE international conference on computer vision, Seoul, Korea (South), 27 October–2 November 2019. pp 1294–1303

Chen Y, Meng G, Zhang Q et al (2019c) RENAS: reinforced evolutionary neural architecture search. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Long Beach, CA, USA, 16–20 June 2019. pp 4787–4796

Chen Z, Zhou Y, Huang Z (2019d) Auto-creation of efective neural network architecture by evolutionary algorithm and ResNet for image classifcation. In: Proceedings of the 2019 IEEE international confer- ence on systems, man and cybernetics, Bari, Italy, 6–9 October 2019. pp 3895–3900

a 灬

Chen X, Wujek B (2020) AutoDAL: distributed active learning with automatic hyperparameter selection. In: Proceedings of the thirty-fourth AAAI conference on artifcial intelligence, New York, NY, USA, 7–12 February 2020. pp 3537–3544

Chrabaszcz P, Loshchilov I, Hutter F (2017) A downsampled variant of ImageNet as an alternative to the CIFAR datasets. arXiv. <http://arxiv.org/abs/1707.08819>

Chu X, Zhou T, Zhang B et al (2020) Fair DARTS: eliminating unfair advantages in diferentiable architec- ture search. In: Proceedings of the European conference on computer vision, Glasgow, UK, 23–28 August 2020. pp 465–480

Claesen M, Simm J, Popovic D et al (2014) Easy hyperparameter search using Optunity. [http://arxiv.org/abs/](http://arxiv.org/abs/1412.1114) [1412.1114](http://arxiv.org/abs/1412.1114)

Cui J, Chen P, Li R et al (2019) Fast and practical neural architecture search. In: Proceedings of the IEEE international conference on computer vision, Seoul, Korea (South), 27 October–2 November 2019. pp 6509–6518

Das P, Ivkin N, Bansal T et al (2020) Amazon SageMaker Autopilot: a white box AutoML solution at scale. In: Proceedings of the fourth international workshop on data management for end-to-end machine learning, Portland, USA, 14 June 2020. pp 1–7

DataCanvas (2021) Hypernets. <https://github.com/DataCanvasIO/Hypernets>. Accessed 4 Nov 2021

Dataiku (2023) Dataiku. <https://www.dataiku.com/>. Accessed 20 Sept 2023

DataRobot (2023) DataRobot AI platform. <https://www.datarobot.com/>. Accessed 20 Sept 2023

De Bie T, De Raedt L, Hernández-Orallo J et al (2022) Automating data science. Commun ACM 65(3):76–

87. <https://doi.org/10.1145/3495256>

de Menezes T, Filho S, Song H et al (2023) Classifer calibration: a survey on how to assess and

improve predicted class probabilities. Mach Learn 112(9):3211–3260. [https://doi.org/10.1007/](https://doi.org/10.1007/s10994-023-06336-7)

[s10994-023-06336-7](https://doi.org/10.1007/s10994-023-06336-7)

de Sá AGC, Pinto WJGS, Oliveira LOVB et al (2017) RECIPE: a grammar-based framework for automati- cally evolving classifcation pipelines. In: Proceedings of the genetic programming European confer- ence, Amsterdam, The Netherlands, 19–21 April 2017. pp 246–261

Del Valle AM, Mantovani RG, Cerri R (2023) A systematic literature review on AutoML for multi-target learning tasks. Artif Intell Rev. <https://doi.org/10.1007/s10462-023-10569-2>

Deng B, Yan J, Lin D (2017) Peephole: predicting network performance before training. arXiv. [http://](http://arxiv.org/abs/1712.03351) [arxiv.org/abs/1712.03351](http://arxiv.org/abs/1712.03351)

Desautels T, Krause A, Burdick JW (2014) Parallelizing exploration-exploitation tradeofs in Gaussian process bandit optimization. J Mach Learn Res 15(119):4053–4103

Dimanov D, Balaguer-Ballester E, Singleton C et al (2021) MONCAE: multi-objective neuroevolution of convolutional autoencoders. In: Proceedings of the 9th international conference on learning representations workshop on neural architecture search, virtual event, 3–7 May 2021

Ding Y, Yao Q, Zhao H et al (2021) DifMG: diferentiable meta graph search for heterogeneous graph neural networks. In: Proceedings of the 27th international conference on knowledge discovery & data mining, virtual event, Singapore, 14–18 August 2021. pp 279–288

Domhan T, Springenberg JT, Hutter F (2015) Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Proceedings of the twenty-fourth international joint conference on artifcial intelligence, Buenos Aires, Argentina, 25–31 July 2015. pp 3460–3468

Dong X, Yang Y (2019) Network pruning via transformable architecture search. Adv Neural Inf Process Syst 32:760–771

Dong X, Yang Y (2020) NAS-Bench-201: extending the scope of reproducible neural architecture search. In: Proceedings of the 8th international conference on learning representations, Addis Ababa, Ethiopia, 26–30 April 2020

Dong X, Liu L, Musial K et al (2021) NATS-Bench: benchmarking NAS algorithms for architecture topology and size. IEEE Trans Pattern Anal Mach Intell 44:3634–3646

Dorogush AV, Ershov V, Gulin A (2018) CatBoost: gradient boosting with categorical features support. arXiv. <http://arxiv.org/abs/1810.11363>

Eggensperger K, Feurer M, Hutter F et al (2013) Towards an empirical foundation for assessing Bayes- ian optimization of hyperparameters. In: Proceedings of the conference on neural information processing systems workshop on Bayesian optimization in theory and practice, Lake Tahoe, NV, USA, 5–8 December 2013. p 3

Eggensperger K, Hutter F, Hoos H et al (2015) Efcient benchmarking of hyperparameter optimizers via surrogates. In: Proceedings of the twenty-ninth AAAI conference on artifcial intelligence, Austin, Texas, USA, 25–30 January 2015. pp 1114–1120

a 灬

Eggensperger K, Müller P, Mallik N et al (2021) HPOBench: a collection of reproducible multi-fdelity benchmark problems for HPO. In: Proceedings of the neural information processing systems track on datasets and benchmarks 1, virtual event, December 2021

Eldeeb H, Maher M, Matsuk O et al (2022) AutoMLBench: a comprehensive experimental evaluation of automated machine learning frameworks. <https://doi.org/10.48550/ARXIV.2204.08358>

ElShawi R, Maher M, Sakr S (2019) Automated machine learning: state-of-the-art and open challenges. arXiv. <http://arxiv.org/abs/1906.02287>

Elsken T, Metzen JH, Hutter F (2018) Simple and efcient architecture search for convolutional neural networks. In: Workshop track proceedings of the 6th international conference on learning repre- sentations, Vancouver, BC, Canada, 30 April–3 May 2018

Elsken T, Metzen JH, Hutter F (2019a) Efcient multi-objective neural architecture search via Lamarck- ian evolution. In: Proceedings of the 7th international conference on learning representations, New Orleans, LA, USA, 6–9 May 2019

Elsken T, Metzen JH, Hutter F (2019b) Neural architecture search: a survey. J Mach Learn Res 20(1):1997–2017

Elsken T, Stafer B, Metzen JH et al (2020) Meta-learning of neural architectures for few-shot learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Seattle, WA, USA, 13–19 June 2020. pp 12365–12375

Erickson N, Mueller J, Shirkov A et al (2020) AutoGluon-Tabular: robust and accurate AutoML for structured data. In: Proceedings of the 7th international conference on machine learning workshop on automated machine learning, virtual event, 13–18 July 2020

Erol K, Hendler JA, Nau DS (1994) UMCP: a sound and complete procedure for hierarchical task-net- work planning. In: Proceedings of the second international conference on artifcial intelligence planning systems, University of Chicago, Chicago, IL, USA, 13–15 June 1994. pp 249–254

Escalante HJ (2021) Automated machine learning—a brief review at the end of the early years. In: Auto- mated design of machine learning and search algorithms. Natural computing series. Springer, pp 11–28. <https://doi.org/10.1007/978-3-030-72069-8_2>

European Commission High Level Expert Group AI (2018) Ethics guidelines for trustworthy AI: high- level expert group on artifcial intelligence. [https://digital-strategy.ec.europa.eu/en/library/ethics-](https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai) [guidelines-trustworthy-ai](https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai). 8 Apr 2019

Evchenko M, Vanschoren J, Hoos HH et al (2021) Frugal machine learning. arXiv. [http://arxiv.org/abs/](http://arxiv.org/abs/2111.03731) [2111.03731](http://arxiv.org/abs/2111.03731)

Falkner S, Klein A, Hutter F (2018) BOHB: robust and efcient hyperparameter optimization at scale. In: Proceedings of the 35th international conference on machine learning, Stockholmsmässan, Stockholm, Sweden, 10–15 July 2018. pp 1436–1445

Fawcett C, Hoos HH (2016) Analysing diferences between algorithm confgurations through ablation. J Heuristics 22(4):431–458

Fefer M, Hirzel M, Hofman SC et al (2023) Searching for fairer machine learning ensembles. In: Pro- ceedings of the AutoML conference, Potsdam, Germany, 12–15 September 2023

Feurer M, Hutter F (2019) Hyperparameter optimization. In: Automated machine learning. Springer, pp 3–33

Feurer M, Klein A, Eggensperger K et al (2015) Efcient and robust automated machine learning. In: Advances in neural information processing systems 28: annual conference on neural information processing systems 2015, Montreal, QC, Canada, 7–12 December 2015. pp 2962–2970

Feurer M, Eggensperger K, Falkner S et al (2018) Practical automated machine learning for the AutoML challenge 2018. In: Proceedings of the international conference on machine learning workshop on automated machine learning, Stockholm, Sweden, 10–15 July 2018

Feurer M, Eggensperger K, Falkner S et al (2022) Auto-Sklearn 2.0: hands-free AutoML via meta-learn- ing. J Mach Learn Res 23(261):1–61

Finn C, Abbeel P, Levine S (2017) Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the international conference on machine learning, Sydney, NSW, Australia, 6–11 August 2017. pp 1126–1135

Franceschi L, Donini M, Frasconi P et al (2017) Forward and reverse gradient-based hyperparameter optimization. In: Proceedings of the international conference on machine learning, Sydney, NSW, Australia, 6–11 August 2017. pp 1165–1173

Gao Y, Yang H, Zhang P et al (2020) Graph neural architecture search. In: Proceedings of the twenty- ninth international joint conference on artifcial intelligence. pp 1403–1409. [https://doi.org/10.](https://doi.org/10.24963/ijcai.2020/195) [24963/ijcai.2020/195](https://doi.org/10.24963/ijcai.2020/195)

Gao F, Song B, Wang D et al (2022) MR-DARTS: restricted connectivity diferentiable architecture search in multi-path search space. Neurocomputing 482:27–39

a 灬

Garnett R (2023) Bayesian optimization. Cambridge University Press, Cambridge

Garrido-Merchán EC, Hernández-Lobato D (2020) Dealing with categorical and integer-valued vari- ables in Bayesian optimization with Gaussian processes. Neurocomputing 380:20–35. [https://doi.](https://doi.org/10.1016/j.neucom.2019.11.004) [org/10.1016/j.neucom.2019.11.004](https://doi.org/10.1016/j.neucom.2019.11.004)

George J, Gao C, Liu R et al (2020) A scalable and cloud-native hyperparameter tuning system. arXiv.

<http://arxiv.org/abs/2006.02085>

Gijsbers P, Vanschoren J (2020) GAMA: a general automated machine learning assistant. In: Proceed- ings of the European conference on machine learning and knowledge discovery in databases, Ghent, Belgium, 14–18 September 2020. pp 560–564. [https://doi.org/10.1007/978-3-030-67670-](https://doi.org/10.1007/978-3-030-67670-4_39) [4\_39](https://doi.org/10.1007/978-3-030-67670-4_39)

Gijsbers P, Vanschoren J, Olson RS (2017) Layered TPOT: speeding up tree-based pipeline optimiza- tion. In: Proceedings of the international workshop on automatic selection, confguration and composition of machine learning algorithms co-located with the European conference on machine learning & principles and practice of knowledge discovery in databases, Skopje, Macedonia, 22 September 2017. pp 49–68

Gijsbers P, LeDell E, Thomas J et al (2019) An open source AutoML benchmark. In: Proceedings of the 6th international conference on machine learning workshop on automated machine learning, Long Beach, CA, USA, 9–15 June 2019

Gijsbers P, Pfsterer F, van Rijn JN et al (2021) Meta-learning for symbolic hyperparameter defaults. In: Proceedings of the genetic and evolutionary computation conference, companion volume, Lille, France, 10–14 July 2021. pp 151–152. <https://doi.org/10.1145/3449726.3459532>

Gijsbers P, Bueno MLP, Coors S et al (2022) AMLB: an AutoML benchmark. arXiv. [http://arxiv.org/abs/](http://arxiv.org/abs/2207.12560) [2207.12560](http://arxiv.org/abs/2207.12560)

Ginsbourger D, Riche RL, Carraro L (2010) Kriging is well-suited to parallelize optimization. In: Pro- ceedings of the computational intelligence in expensive optimization problems. pp 131–162

Ginsbourger D, Janusevskis J, Le Riche R (2011) Dealing with asynchronicity in parallel Gaussian Pro- cess based global optimization. Research report, Mines Saint-Etienne

Goldberg DE, Deb K (1990) A comparative analysis of selection schemes used in genetic algorithms. In: Proceedings of the frst workshop on foundations of genetic algorithms. Bloomington Campus, IN, USA, 15–18 July 1990. pp 69–93. <https://doi.org/10.1016/b978-0-08-050684-5.50008-2>

Goodfellow IJ, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples. In: Proceed- ings of the 3rd international conference on learning representations, San Diego, CA, USA, 7–9 May 2015. pp 1–11

Goodfellow I,Bengio Y, Courville A (2016) Deep learning. MIT Press. <http://www.deeplearningbook.org>

Gopal B, Sridhar A, Zhang T et al (2023) LISSNAS: locality-based iterative search space shrinkage for neural architecture search. In: Proceedings of the thirty-second international joint conference on arti- fcial intelligence, Macao, SAR, China, 19–25 August 2023. pp 773–781. [https://doi.org/10.24963/](https://doi.org/10.24963/ijcai.2023/86) [ijcai.2023/86](https://doi.org/10.24963/ijcai.2023/86)

Grazzi R, Franceschi L, Pontil M et al (2020) On the iteration complexity of hypergradient computation. In: Proceedings of the 37th international conference on machine learning, virtual event, 13–18 July 2020. pp 3748–3758

Gu Y, Wang L, Liu Y et al (2021) DOTS: decoupling operation and topology in diferentiable architecture search. In: IEEE conference on computer vision and pattern recognition, virtual event, 19–25 June 2021. pp 12311–12320. <https://doi.org/10.1109/CVPR46437.2021.01213>

Guyon I, Safari A, Dror G et al (2010) Model selection: beyond the Bayesian/frequentist divide. J Mach Learn Res 11:61–87

Guyon I, Chaabane I, Escalante HJ et al (2016) A brief review of the ChaLearn AutoML challenge: any- time any-dataset learning without human intervention. In: Proceedings of the 2016 workshop on auto- matic machine learning, AutoML 2016, co-located with 33rd international conference on machine learning, New York City, NY, USA, 24 June 2016. pp 21–30

H2O.ai (2017) H2O AutoML. <http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html>. Accessed 4 Nov 2022

Hall M, Frank E, Holmes G et al (2009) The WEKA data mining software: an update. SIGKDD Explor Newslett 11(1):10–18. <https://doi.org/10.1145/1656274.1656278>

Han D, Kim J, Kim J (2017) Deep pyramidal residual networks. In: Proceedings of the 2017 IEEE confer- ence on computer vision and pattern recognition, Honolulu, HI, USA, 21–26 July 2017. pp 6307– 6315. <https://doi.org/10.1109/CVPR.2017.668>

Hansen N, Müller SD, Koumoutsakos P (2003) Reducing the time complexity of the derandomized evolu- tion strategy with covariance matrix adaptation (CMA-ES). Evol Comput 11(1):1–18. [https://doi.org/](https://doi.org/10.1162/106365603321828970) [10.1162/106365603321828970](https://doi.org/10.1162/106365603321828970)

a 灬

Hardt M, Price E, Srebro N (2016) Equality of opportunity in supervised learning. In: Lee DD, Sugiyama M, von Luxburg U et al (eds) Proceedings of the advances in neural information processing systems 29: annual conference on neural information processing systems, Barcelona, Spain, 5–10 December 2016. pp 3315–3323

He K, Zhang X, Ren S et al (2016) Deep residual learning for image recognition. In: Proceedings of the

2016 IEEE conference on computer vision and pattern recognition, Las Vegas, NV, USA, 27–30 June

2016. pp 770–778. <https://doi.org/10.1109/CVPR.2016.90>

He X, Zhao K, Chu X (2021) AutoML: a survey of the state-of-the-art. Knowl Based Syst 212(106):622.

<https://doi.org/10.1016/j.knosys.2020.106622>

Head T, Kumar M, Nahrstaedt H et al (2017) Sequential model-based optimization in Python. [https://scikit-](https://scikit-optimize.github.io/stable) [optimize.github.io/stable](https://scikit-optimize.github.io/stable)

Hennig P, Schuler CJ (2012) Entropy search for information-efcient global optimization. J Mach Learn Res 13:1809–1837. <https://doi.org/10.5555/2503308.2343701>

Hochreiter S, Younger AS, Conwell PR (2001) Learning to learn using gradient descent. In: Proceedings of the international conference on artifcial neural networks, Vienna, Austria, 21–25 August 2001. pp 87–94

Hofman MW, Shahriari B (2014) Modular mechanisms for Bayesian optimization. In: Proceedings of the conference on neural information processing systems workshop on Bayesian optimization, Montréal, QC, Canada, 8–13 December 2014. pp 1–5

Hollmann N, Müller S, Eggensperger K et al (2023) TabPFN: a transformer that solves small tabular clas- sifcation problems in a second. In: Proceedings of the eleventh international conference on learning representations, Kigali, Rwanda, 1–5 May 2023

Hong W, Li G, Zhang W et al (2020) DropNAS: grouped operation dropout for diferentiable architecture search. In: Proceedings of the twenty-ninth international joint conference on artifcial intelligence, IJCAI 2020. pp 2326–2332. <https://doi.org/10.24963/ijcai.2020/322>

Hoos HH (2012) Programming by optimization. Commun ACM 55(2):70–80. [https://doi.org/10.1145/](https://doi.org/10.1145/2076450.2076469)

[2076450.2076469](https://doi.org/10.1145/2076450.2076469)

Hospedales TM, Antoniou A, Micaelli P et al (2022) Meta-learning in neural networks: a survey. IEEE

Trans Pattern Anal Mach Intell 44(09):5149–5169. <https://doi.org/10.1109/TPAMI.2021.3079209>

Hu YQ, Yu Y, Tu WW et al (2019) Multi-fdelity automatic hyper-parameter tuning via transfer series

expansion. In: Proceedings of the AAAI conference on artifcial intelligence, Honolulu, HI, USA, [27](#bookmark419)

January–1 February 2019. pp 3846–3853

Hu Y, Liang Y, Guo Z et al (2020) Angle-based search space shrinking for neural architecture search. In: Proceedings of the 16th European conference on computer vision, Glasgow, UK, 23–28 August 2020. pp 119–134

Huang G, Liu Z, van der Maaten L et al (2017) Densely connected convolutional networks. In: Proceedings of the 2017 IEEE conference on computer vision and pattern recognition, Honolulu, HI, USA, 21–26 July 2017. pp 2261–2269. <https://doi.org/10.1109/CVPR.2017.243>

Huisman M, van Rijn JN, Plaat A (2021) A survey of deep meta-learning. Artif Intell Rev 54(6):4483–4541

Huisman M, Moerland TM, Plaat A et al (2023) Are LSTMs good few-shot learners? Mach Learn. [https://](https://doi.org/10.1007/s10994-023-06394-x) [doi.org/10.1007/s10994-023-06394-x](https://doi.org/10.1007/s10994-023-06394-x)

Hundt A, Jain V, Hager GD (2019) sharpDARTS: faster and more accurate diferentiable architecture search. arXiv. <http://arxiv.org/abs/1903.09900>

Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm confguration. In: Proceedings of the 5th international conference on learning and intelligent optimi- zation, Lille, France, 12–15 January 2015. pp 507–523. [https://doi.org/10.1007/978-3-642-25566-3\_](https://doi.org/10.1007/978-3-642-25566-3_40) [40](https://doi.org/10.1007/978-3-642-25566-3_40)

Hutter F, Hoos HH, Leyton-Brown K (2014) An efcient approach for assessing hyperparameter impor- tance. In: Proceedings of the international conference on machine learning, Beijing, China, 21–26 June 2014. pp 754–762

Hutter F, Kotthof L, Vanschoren J (2019) Automated machine learning: methods, systems, challenges. Springer, Cham

Irwin-Harris W, Sun Y, Xue B et al (2019) A graph-based encoding for evolutionary convolutional neural network architecture design. In: Proceedings of the 2019 IEEE congress on evolutionary computation, Wellington, New Zealand, 10–13 June 2019. pp 546–553

Istrate R, Scheidegger F, Mariani G et al (2019) TAPAS: train-less accuracy predictor for architecture search. In: Proceedings of the thirty-third AAAI conference on artifcial intelligence, Honolulu, HI, USA, 27 January–1 February 2019. pp 3927–3934. <https://doi.org/10.1609/aaai.v33i01.33013927>

Jaafra Y, Luc Laurent J, Deruyver A et al (2019) Reinforcement learning for neural architecture search: a review. Image Vis Comput 89:57–66. <https://doi.org/10.1016/j.imavis.2019.06.005>

a 灬

Jamieson KG, Talwalkar A (2016) Non-stochastic best arm identifcation and hyperparameter optimization. In: Proceedings of the international conference on artifcial intelligence and statistics, Cadiz, Spain, 9–11 May 2016. pp 240–248

Jang E, Gu S, Poole B (2017) Categorical reparameterization with Gumbel-Softmax. In: Proceedings of the 5th international conference on learning representations, Toulon, France, 24–26 April 2017

Jedrzejewski-Szmek Z, Abrahao KP, Jedrzejewska-Szmek J et al (2018) Parameter optimization using covariance matrix adaptation-evolutionary strategy (CMA-ES), an approach to investigate diferences in channel properties between neuron subtypes. Front Neuroinform 12:47. [https://doi.org/10.3389/](https://doi.org/10.3389/fninf.2018.00047) [fninf.2018.00047](https://doi.org/10.3389/fninf.2018.00047)

Jin H, Song Q, Hu X (2019) Auto-Keras: an efcient neural architecture search system. In: Proceedings of the 25th international conference on knowledge discovery & data mining, Anchorage, AK, USA, 4–8 August 2019. pp 1946–1956. <https://doi.org/10.1145/3292500.3330648>

Jomaa HS, Grabocka J, Schmidt-Thieme L (2019) Hyp-RL: hyperparameter optimization by reinforcement learning. arXiv. <http://arxiv.org/abs/1906.11527>

Jones DR, Schonlau M, Welch WJ (1998) Efcient global optimization of expensive black-box functions. J Glob Optim 13(4):455–492. <https://doi.org/10.1023/A:1008306431147>

Kandasamy K, Neiswanger W, Schneider J et al (2018) Neural architecture search with Bayesian optimisa- tion and optimal transport. In: Proceedings of the advances in neural information processing systems 31: annual conference on neural information processing systems, Montréal, Canada, 3–8 December 2018. pp 2020–2029

Karl F, Pielok T, Moosbauer J et al (2022) Multi-objective hyperparameter optimization—an overview. arXiv. <http://arxiv.org/abs/2206.07438>

Karmaker SK, Hassan MM, Smith MJ et al (2021) AutoML to date and beyond: challenges and opportuni- ties. ACM Comput Surv (CSUR) 54(8):1–36

Ke G, Meng Q, Finley T et al (2017) LightGBM: a highly efcient gradient boosting decision tree. In: Pro- ceedings of annual conference on neural information processing systems, Long Beach, CA, USA, 4–9 December 2017. pp 3146–3154

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of international conference on neural networks, Perth, WA, Australia, 27 November–1 December 1995. pp 1942–1948

Klein A, Hutter F (2019) Tabular benchmarks for joint architecture and hyperparameter optimization. arXiv. <http://arxiv.org/abs/1905.04970>

Klein A, Falkner S, Bartels S et al (2017a) Fast Bayesian optimization of machine learning hyperpa- rameters on large datasets. In: Proceedings of the 20th international conference on artifcial intel- ligence and statistics, Fort Lauderdale, FL, USA, 20–22 April 2017. pp 528–536

Klein A, Falkner S, Mansur N et al (2017b) RoBO: a fexible and robust Bayesian optimization frame- work in Python. In: Proceedings of the conference on neural information processing systems Bayesian optimization workshop, Long Beach, CA, USA, 4–9 December 2017

Klein A, Falkner S, Springenberg JT et al (2017c) Learning curve prediction with Bayesian neural net- works. In: Proceedings of the 5th international conference on learning representations, Toulon, France, 24–26 April 2017

Klyuchnikov N, Trofmov I, Artemova E et al (2022) NAS-Bench-NLP: neural architecture search

benchmark for natural language processing. IEEE Access 10:45736–45747. [https://doi.org/10.](https://doi.org/10.1109/ACCESS.2022.3169897)

[1109/ACCESS.2022.3169897](https://doi.org/10.1109/ACCESS.2022.3169897)

Kocsis L, Szepesvári C (2006) Bandit based Monte-Carlo planning. In: Proceedings of the machine

learning: 17th European conference on machine learning, Berlin, Germany, 18–22 September

2006, proceedings. pp 282–293. <https://doi.org/10.1007/11871842_29>

Komer B, Bergstra J, Eliasmith C (2014) HyperOPT-Sklearn: automatic hyperparameter confguration

for scikit-learn. In: Proceedings of the international conference on machine learning workshop on

automated machine learning, Beijing, China, 21–26 June 2014. p [50](#bookmark420)

König M, Hoos HH, van Rijn JN (2020) Towards algorithm-agnostic uncertainty estimation: predicting classifcation error in an automated machine learning setting. In: Proceedings of the 7th interna- tional conference on machine learning workshop on automated machine learning, virtual event, 13–18 July 2020

König M, Hoos HH, van Rijn JN (2022) Speeding up neural network robustness verifcation via algo- rithm confguration and an optimised mixed integer linear programming solver portfolio. Mach Learn 111(9):4565–4584

König M, Bosman AW, Hoos HH et al (2023) Critically assessing the state of the art in CPU-based local robustness verifcation. In: Proceedings of the workshop on artifcial intelligence safety, Washing- ton, DC, USA, 7–14 February 2023

a 灬

Kotthof L, Thornton C, Hoos HH et al (2017) Auto-WEKA 2.0: automatic model selection and hyper- parameter optimization in WEKA. J Mach Learn Res 18(25):1–5

Koza JR (1994) Genetic programming as a means for programming computers by natural selection. Stat Comput 4(2):87–112. <https://doi.org/10.1007/BF00175355>

Kraska T (2018) Northstar: an interactive data science system. Proc VLDB Endow 11(12):2150–2164.

<https://doi.org/10.14778/3229863.3240493>

Krizhevsky A, Nair V, Hinton G (2010) CIFAR-10 (Canadian Institute for Advanced Research). Master’s Thesis, Department of Computer Science, University of Toronto. p 1

Lacoste A, Marchand M, Laviolette F et al (2014) Agnostic Bayesian learning of ensembles. In: Pro- ceedings of the 31th international conference on machine learning, Beijing, China, 21–26 June 2014. pp 611–619

Larochelle H, Erhan D, Courville A et al (2007) An empirical evaluation of deep architectures on prob- lems with many factors of variation. In: Proceedings of the 24th international conference on machine learning, Corvallis, Oregon, USA, 20–24 June 2007. pp 473–480

LasagneContributors (2022) Lasagne. <https://github.com/Lasagne/Lasagne>. Accessed 4 Nov 2022

LeDell E, Poirier S (2020) H2O AutoML: scalable automatic machine learning. In: Proceedings of the 7th international conference on machine learning workshop on automated machine learning, vir- tual event, 13–18 July 2020

Leite R, Brazdil P (2005) Predicting relative performance of classifers from samples. In: Machine learn- ing, proceedings of the twenty-second international conference, Bonn, Germany, 7–11 August 2005. pp 497–503. <https://doi.org/10.1145/1102351.1102414>

Leite R, Brazdil P (2010) Active testing strategy to predict the best classifcation algorithm via sampling and metalearning. In: Proceedings of the 19th European conference on artifcial intelligence, Lis- bon, Portugal, 16–20 August 2010. pp 309–314

Li L, Talwalkar A (2019) Random search and reproducibility for neural architecture search. In: Proceed- ings of the thirty-ffth conference on uncertainty in artifcial intelligence, Tel Aviv, Israel, 22–25 July 2019. p 129

Li L, Jamieson KG, DeSalvo G et al (2017) Hyperband: a novel bandit-based approach to hyperparameter optimization. J Mach Learn Res 18(185):1–52

Li YF, Wang H, Wei T et al (2019) Towards automated semi-supervised learning. In: Proceedings of the AAAI conference on artifcial intelligence, virtual event, 2–9 February 2021. pp 4237–4244

Li L, Jamieson KG, Rostamizadeh A et al (2020a) A system for massively parallel hyperparameter tuning. In: Proceedings of the machine learning and systems, Austin, TX, USA, 2–4 March 2020

Li T, Zhang J, Bao K et al (2020b) AutoST: efcient neural architecture search for spatio-temporal predic- tion. In: Proceedings of the 26th international conference on knowledge discovery & data mining, virtual event, 23–27 August 2020. pp 794–802

Li Z, Deng J, Zhang G et al (2020c) GP-NAS: Gaussian process based neural architecture search. In: 2020 IEEE conference on computer vision and pattern recognition, Seattle, WA, USA, 13–19 June 2020. pp 11930–11939. <https://doi.org/10.1109/CVPR42600.2020.01195>

Li Y, Wen Z, Wang Y et al (2021) One-shot graph neural architecture search with dynamic search space. In: Thirty-ffth AAAI conference on artifcial intelligence, virtual event, 2–9 February 2021. pp 8510–8517

Li H, Liang Q, Chen M et al (2022) Pruning SMAC search space based on key hyperparameters. Concurr Comput Pract Exp. <https://doi.org/10.1002/cpe.5805>

Liang H, Zhang S, Sun J et al (2019) DARTS+: improved diferentiable architecture search with early stop- ping. arXiv. <http://arxiv.org/abs/1909.06035>

Liaw R, Liang E, Nishihara R et al (2018) Tune: a research platform for distributed model selection and training. arXiv. <http://arxiv.org/abs/1807.05118>

Lindauer M, Hutter F (2020) Best practices for scientifc research on neural architecture search. J Mach Learn Res 21(243):1–18

Lindauer M, Eggensperger K, Feurer M et al (2022) SMAC3: a versatile Bayesian optimization package for hyperparameter optimization. J Mach Learn Res 23(54):1–9

LinuxFoundation (2022) PyTorch. <https://pytorch.org>. Accessed 4 Nov 2022

Liu C, Zoph B, Neumann M et al (2018a) Progressive neural architecture search. In: Proceedings of the 15th European conference on computer vision, Munich, Germany, 8–14 September 2018. pp 19–35.

<https://doi.org/10.1007/978-3-030-01246-5_2>

Liu H, Simonyan K, Vinyals O et al (2018b) Hierarchical representations for efcient architecture search. In: Proceedings of the 6th international conference on learning representations, Vancouver, BC, Canada, 30 April–3 May 2018

a 灬

Liu C, Chen LC, Schrof F et al (2019a) Auto-DeepLab: hierarchical neural architecture search for semantic image segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recogni- tion, Long Beach, CA, USA, 16–20 June 2019. pp 82–92. <https://doi.org/10.1109/CVPR.2019.00017>

Liu H, Simonyan K, Yang Y (2019b) DARTS: diferentiable architecture search. In: Proceedings of the 7th international conference on learning representations, New Orleans, LA, USA, 6–9 May 2019

Liu S, Ram P, Vijaykeerthy D et al (2020) An ADMM based framework for AutoML pipeline confguration. In: Proceedings of the thirty-fourth AAAI conference on artifcial intelligence, New York, NY, USA, 7–12 February 2020. pp 4892–4899

Liu Y, Sun Y, Xue B et al (2021) A survey on evolutionary neural architecture search. IEEE Trans Neural Netw Learn Syst. <https://doi.org/10.1109/TNNLS.2021.3100554>

Long FX, van Stein B, Frenzel M et al (2022) Learning the characteristics of engineering optimization prob- lems with applications in automotive crash. In: Fieldsend JE, Wagner M (eds) GECCO ’22: genetic and evolutionary computation conference, Boston, MA, USA, 9–13 July 2022. pp 1227–1236. [https://](https://doi.org/10.1145/3512290.3528712) [doi.org/10.1145/3512290.3528712](https://doi.org/10.1145/3512290.3528712)

López-Ibáñez M, Dubois-Lacoste J, Cáceres LP et al (2016) The irace package: iterated racing for automatic algorithm confguration. Oper Res Perspect 3:43–58. <https://doi.org/10.1016/j.orp.2016.09.002>

Lorenzo PR, Nalepa J (2018) Memetic evolution of deep neural networks. In: Proceedings of the genetic and evolutionary computation conference, Kyoto, Japan, 15–19 July 2018. pp 505–512

Lorraine J, Vicol P, Duvenaud D (2020) Optimizing millions of hyperparameters by implicit diferentiation. In: Proceedings of the international conference on artifcial intelligence and statistics, virtual event, 26–28 August 2020. pp 1540–1552

Loshchilov I, Hutter F (2016) CMA-ES for hyperparameter optimization of deep neural networks. arXiv.

<http://arxiv.org/abs/1604.07269>

Loshchilov I, Schoenauer M, Sebag M (2012) Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In: Proceedings of the genetic and evolutionary computation conference, Philadel- phia, PA, USA, 7–11 July 2012. pp 321–328. <https://doi.org/10.1145/2330163.2330210>

Loshchilov I, Schoenauer M, Sebag M (2013) Bi-population CMA-ES algorithms with surrogate models and line searches. In: Proceedings of the genetic and evolutionary computation conference, Amster- dam, The Netherlands, 6–10 July 2013, companion material proceedings. pp 1177–1184. [https://doi.](https://doi.org/10.1145/2464576.2482696) [org/10.1145/2464576.2482696](https://doi.org/10.1145/2464576.2482696)

Lourenço N, Assunção F, Pereira FB et al (2018) Structured grammatical evolution: a dynamic approach. In: Handbook of grammatical evolution. Springer, pp 137–161

Lu Z, Whalen I, Boddeti V et al (2019) NSGA-Net: neural architecture search using multi-objective genetic algorithm. In: Proceedings of the genetic and evolutionary computation conference, Prague, Czech Republic, 13–17 July 2019. pp 419–427

Lu Z, Whalen I, Dhebar YD et al (2020) NSGA-Net: neural architecture search using multi-objective genetic algorithm (extended abstract). In: Proceedings of the twenty-ninth international joint conference on artifcial intelligence, virtual event, 7–8 January 2021. pp 4750–4754. [https://doi.org/10.24963/ijcai.](https://doi.org/10.24963/ijcai.2020/659) [2020/659](https://doi.org/10.24963/ijcai.2020/659)

Lu S, Li J, Tan J et al (2021) TNASP: a transformer-based NAS predictor with a self-evolution framework. In: Proceedings of the advances in neural information processing systems 34: annual conference on neural information processing systems, virtual event, 6–14 December 2021. pp 15125–15137

Luo R, Tian F, Qin T et al (2018) Neural architecture optimization. In: Proceedings of the advances in neu- ral information processing systems 31: annual conference on neural information processing systems, Montréal, Canada, 3–8 December 2018. pp 7827–7838

Maclaurin D, Duvenaud D, Adams R (2015) Gradient-based hyperparameter optimization through revers- ible learning. In: Proceedings of the international conference on machine learning, Lille, France, 6–11 July 2015. pp 2113–2122

Mahsereci M, Balles L, Lassner C et al (2017) Early stopping without a validation set. arXiv. [http://arxiv.](http://arxiv.org/abs/1703.09580) [org/abs/1703.09580](http://arxiv.org/abs/1703.09580)

Marcus M, Kim G, Marcinkiewicz MA et al (1994) The Penn treebank: annotating predicate argument structure. In: Human language technology: proceedings of a workshop held at Plainsboro, New Jer- sey, 8–11 March 1994

Martinez-Cantin R (2014) BayesOPT: a Bayesian optimization library for nonlinear optimization, experi- mental design and bandits. J Mach Learn Res 15(1):3735–3739

McKay RI, Hoai NX, Whigham PA et al (2010) Grammar-based genetic programming: a survey. Genet Pro- gram Evol Mach 11(3–4):365–396

Mellor J, Turner J, Storkey A et al (2021) Neural architecture search without training. In: Proceedings of the international conference on machine learning, virtual event, 18–24 July 2021. pp 7588–7598

a 灬

Mendoza H, Klein A, Feurer M et al (2016) Towards automatically-tuned neural networks. In: Proceedings of the workshop on automatic machine learning, New York, NY, USA, 24 June 2016. pp 58–65

Meng MH, Bai G, Teo SG et al (2022) Adversarial robustness of deep neural networks: a survey from a formal verifcation perspective. IEEE Trans Depend Secur Comput. [https://doi.org/10.1109/TDSC.](https://doi.org/10.1109/TDSC.2022.3179131) [2022.3179131](https://doi.org/10.1109/TDSC.2022.3179131)

Microsoft (2021) Microsoft neural network intelligence. <https://github.com/microsoft/nni>. Accessed 4 Nov 2021

Miikkulainen R, Liang J, Meyerson E et al (2019) Evolving deep neural networks. In: Artifcial intelligence in the age of neural networks and brain computing. Elsevier, pp 293–312

Millán JDR, Posenato D, Dedieu E (2002) Continuous-action q-learning. Mach Learn 49(2):247–265.

<https://doi.org/10.1023/A:1017988514716>

Mohan A, Benjamins C, Wienecke K et al (2023) AutoRL hyperparameter landscapes. In: Proceedings of the AutoML conference, Potsdam, Germany, 12–15 September 2023

Mohr F, van Rijn JN (2022) Learning curves for decision making in supervised machine learning—a survey. arXiv. <http://arxiv.org/abs/2201.12150>

Mohr F, van Rijn JN (2023) Fast and informative model selection using learning curve cross-validation. IEEE Trans Pattern Anal Mach Intell 45(8):9669–9680

Mohr F, Wever M (2022) Naive automated machine learning. Mach Learn. [https://doi.org/10.1007/](https://doi.org/10.1007/s10994-022-06200-0) [s10994-022-06200-0](https://doi.org/10.1007/s10994-022-06200-0)

Mohr F, Wever M, Hüllermeier E (2018) ML-Plan: automated machine learning via hierarchical planning. Mach Learn 107(8):1495–1515. <https://doi.org/10.1007/s10994-018-5735-z>

Mohr F, Viering TJ, Loog M et al (2022) LCDB 1.0: an extensive learning curves database for classifcation tasks. In: Proceedings of the joint European conference on machine learning and knowledge discov- ery in databases, Grenoble, France, 19–23 September 2022

Moussa C, Patel YJ, Dunjko V et al (2023) Hyperparameter importance and optimization of quantum neural networks across small datasets. Mach Learn. <https://doi.org/10.1007/s10994-023-06389-8>

Muñoz JP, Lyalyushkin N, Lacewell CW et al (2022) Automated super-network generation for scalable neu- ral architecture search. In: Proceedings of the international conference on automated machine learn- ing, Baltimore, MD, USA, 25–27 July 2022, Johns Hopkins University. pp 5/1–15

Negrinho R, Gordon GJ (2017) DeepArchitect: automatically designing and training deep architectures. arXiv. <http://arxiv.org/abs/1704.08792>

Nguyen V, Le T, Yamada M et al (2021) Optimal transport kernels for sequential and parallel neural archi- tecture search. In: Proceedings of the international conference on machine learning, virtual event, 18–24 July 2021. pp 8084–8095

Nguyen N, Chang JM (2022) CSNAS: contrastive self-supervised learning neural architecture search via sequential model-based optimization. IEEE Trans Artif Intell 3(4):609–624. [https://doi.org/10.1109/](https://doi.org/10.1109/TAI.2021.3121663) [TAI.2021.3121663](https://doi.org/10.1109/TAI.2021.3121663)

Nichol A, Achiam J, Schulman J (2018) On frst-order meta-learning algorithms. arXiv. [http://arxiv.org/abs/](http://arxiv.org/abs/1803.02999) [1803.02999](http://arxiv.org/abs/1803.02999)

Nikitin NO, Vychuzhanin P, Sarafanov M et al (2022) Automated evolutionary approach for the design of composite machine learning pipelines. Future Gener Comput Syst 127:109–125. [https://doi.org/10.](https://doi.org/10.1016/j.future.2021.08.022) [1016/j.future.2021.08.022](https://doi.org/10.1016/j.future.2021.08.022)

Nogueira F (2014) Bayesian optimization: open source constrained global optimization tool for Python. <https://github.com/fmfn/BayesianOptimization>. Accessed 4 Nov 2022

Olson RS, Bartley N, Urbanowicz RJ et al (2016a) Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of the genetic and evolutionary computation conference, Denver, CO, USA, 20–24 July 2016. pp 485–492. <https://doi.org/10.1145/2908812.2908918>

Olson RS, Urbanowicz RJ, Andrews PC et al (2016b) Automating biomedical data science through tree- based pipeline optimization. In: Proceedings of the applications of evolutionary computation: 19th European conference, Porto, Portugal, 30 March–1 April 2016. pp 123–137. [https://doi.org/10.1007/](https://doi.org/10.1007/978-3-319-31204-0_9) [978-3-319-31204-0\_9](https://doi.org/10.1007/978-3-319-31204-0_9)

Ottervanger G, Baratchi M, Hoos HH (2021) MultiETSC: automated machine learning for early time series classifcation. Data Min Knowl Disc 35(6):2602–2654

Öztürk E, Ferreira F, Jomaa H et al (2022) Zero-shot AutoML with pretrained models. In: Proceedings of the international conference on machine learning, Baltimore, MD, USA, 17–23 July 2022. pp 17138–17155

Parker-Holder J, Rajan R, Song X et al (2022) Automated reinforcement learning (AutoRL): a survey and open problems. J Artif Intell Res 74:517–568

Parmentier L, Nicol O, Jourdan L et al (2019) TPOT-SH: a faster optimization algorithm to solve the AutoML problem on large datasets. In: Proceedings of the 31st IEEE international conference on

a 灬

tools with artifcial intelligence, Portland, OR, USA, 4–6 November 2019. pp 471–478. [https://doi.](https://doi.org/10.1109/ICTAI.2019.00072) [org/10.1109/ICTAI.2019.00072](https://doi.org/10.1109/ICTAI.2019.00072)

Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830

Perrone V, Shen H, Seeger MW et al (2019) Learning search spaces for Bayesian optimization: another view of hyperparameter transfer learning. In: Proceedings of the advances in neural information processing systems 32: annual conference on neural information processing systems, Vancouver, BC, Canada, 8–14 December 2019. pp 12751–12761

Perrone V, Donini M, Zafar MB et al (2021) Fair Bayesian optimization. In: Proceedings of the 2021 AAAI/ ACM conference on AI, ethics, and society, virtual event, 19–21 May 2021. pp 854–863

Peyré G, Cuturi M (2019) Computational optimal transport. Found Trends Mach Learn 11(5–6):355–607.

<https://doi.org/10.1561/2200000073>

Pfsterer F, van Rijn JN, Probst P et al (2021) Learning multiple defaults for machine learning algorithms. In: Genetic and evolutionary computation conference, companion volume, Lille, France, 10–14 July 2021. pp 241–242. <https://doi.org/10.1145/3449726.3459523>

Pham H, Guan M, Zoph B et al (2018) Efcient neural architecture search via parameters sharing. In: Pro- ceedings of the 35th international conference on machine learning, Stockholmsmässan, Stockholm, Sweden, 10–15 July 2018. pp 4095–4104

Pinto F, Soares C, Mendes-Moreira J (2016) Towards automatic generation of metafeatures. In: Advances in knowledge discovery and data mining—20th Pacifc-Asia conference, PAKDD 2016, Auckland, New Zealand. pp 215–226

Probst P, Boulesteix A, Bischl B (2019) Tunability: importance of hyperparameters of machine learning algorithms. J Mach Learn Res 20:1934–1965

Provost FJ, Jensen DD, Oates T (1999) Efcient progressive sampling. In: Proceedings of the ffth inter- national conference on knowledge discovery and data mining, San Diego, CA, USA, 15–18 August 1999. pp 23–32

Pukelsheim F (2006) Optimal design of experiments, SIAM Classics edn. SIAM

Purucker L, Beel J (2023) Assembled-OpenML: creating efcient benchmarks for ensembles in AutoML with OpenML. arXiv

Pushak Y, Hoos H (2022) AutoML loss landscapes. ACM Trans Evol Learn Optim. [https://doi.org/10.](https://doi.org/10.1145/3558774) [1145/3558774](https://doi.org/10.1145/3558774)

Rajeswaran A, Finn C, Kakade SM et al (2019) Meta-learning with implicit gradients. In: Proceedings of the advances in neural information processing systems 32: annual conference on neural infor- mation processing systems 2019, NeurIPS 2019, 8–14 December 2019, Vancouver, BC, Canada. pp 113–124

Rakotoarison H, Schoenauer M, Sebag M (2019) Automated machine learning with Monte-Carlo tree search. In: Proceedings of the twenty-eighth international joint conference on artifcial intelli-  gence, Macao, China, 10–16 August 2019. pp 3296–3303. <https://doi.org/10.24963/ijcai.2019/457> Read J, Reutemann P, Pfahringer B et al (2016) MEKA: a multi-label/multi-target extension to Weka. J

Mach Learn Res 17(21):1–5

Real E, Moore S, Selle A et al (2017) Large-scale evolution of image classifers. In: Proceedings of the 34th international conference on machine learning, Sydney, NSW, Australia, 6–11 August 2017. pp 2902–2911

Real E, Aggarwal A, Huang Y et al (2019) Regularized evolution for image classifer architecture search. In: Proceedings of the thirty-third AAAI conference on artifcial intelligence, Honolulu, HI, USA, 27 January–1 February 2019. pp 4780–4789. <https://doi.org/10.1609/aaai.v33i01.33014780>

Ren P, Xiao Y, Chang X et al (2022) A comprehensive survey of neural architecture search: challenges and solutions. ACM Comput Surv 54(4):76:1-76:34. <https://doi.org/10.1145/3447582>

Rice JR (1976) The algorithm selection problem. Adv Comput 15:65–118

Rivolli A, Garcia LP, Soares C et al (2022) Meta-features for meta-learning. Knowl Based Syst 240(108):101 Ru BX, Wan X, Dong X et al (2020a) Neural architecture search using Bayesian optimisation with Weis-

feiler-Lehman kernel. arXiv. <http://arxiv.org/abs/2006.07556>

Ru R, Esperanca P, Carlucci FM (2020b) Neural architecture generator optimization. Adv Neural Inf Process Syst 33:12057–12069

Sabharwal A, Samulowitz H, Tesauro G (2016) Selecting near-optimal learners via incremental data allocation. In: Proceedings of the thirtieth AAAI conference on artifcial intelligence, Phoenix, AZ, USA, 12–17 February 2016. pp 2007–2015

Salehin I, Islam MS, Saha P et al (2024) AutoML: a systematic review on automated machine learning with neural architecture search. J Inf Intell 2(1):52–81. <https://doi.org/10.1016/j.jiixd.2023.10.002>

a 灬

Salinas NRP, Baratchi M, van Rijn JN et al (2021) Automated machine learning for satellite data: inte- grating remote sensing pre-trained models into AutoML systems. In: Proceedings of the joint European conference on machine learning and knowledge discovery in databases, virtual event, 13–17 September 2021. pp 447–462

Salinas D, Seeger MW, Klein A et al (2022) Syne Tune: a library for large scale hyperparameter tun- ing and reproducible research. In: International conference on automated machine learning, Johns Hopkins University, Baltimore, MD, USA, 25–27 July 2022. pp 16/1–23

Saxena S, Verbeek J (2016) Convolutional neural fabrics. Adv Neural Inf Process Syst 29:4053–4061

Schneider L, Schäpermeier L, Prager RP et al (2022) HPO × ELA: investigating hyperparameter opti- mization landscapes by means of exploratory landscape analysis. In: Rudolph G, Kononova AV, Aguirre HE et al (eds) Parallel problem solving from nature—PPSN XVII—17th international conference, PPSN 2022, Dortmund, Germany, 10–14 September 2022, proceedings, part I. pp 575–589. <https://doi.org/10.1007/978-3-031-14714-2_40>

Scriven A, Kedziora DJ, Musial K et al (2022) The technological emergence of AutoML: a survey of performant software and applications in the context of industry. arXiv. [https://doi.org/10.48550/](https://doi.org/10.48550/arXiv.2211.04148) [arXiv.2211.04148](https://doi.org/10.48550/arXiv.2211.04148)

Seeger MW, Williams CK, Lawrence ND (2003) Fast forward selection to speed up sparse Gaussian process regression. In: Proceedings of the international workshop on artifcial intelligence and statistics, Key West, FL, USA, 3–6 January 2003. pp 254–261

Shahriari B, Swersky K, Wang Z et al (2016) Taking the human out of the loop: a review of Bayesian optimization. Proc IEEE 104(1):148–175. <https://doi.org/10.1109/JPROC.2015.2494218>

Shang Z, Zgraggen E, Buratti B et al (2019) Democratizing data science through interactive curation of ML pipelines. In: Proceedings of the 2019 international conference on management of data. pp 1171–1188. <https://doi.org/10.1145/3299869.3319863>

Sharma A, van Rijn JN, Hutter F et al (2019) Hyperparameter importance for image classifcation by residual neural networks. In: Discovery science—22nd international conference, Split, Croatia, 28–30 October 2019, proceedings. pp 112–126. <https://doi.org/10.1007/978-3-030-33778-0_10>

Shchur O, Turkmen AC, Erickson N et al (2023) AutoGluon–TimeSeries: AutoML for probabilistic time series forecasting. In: Proceedings of the AutoML conference, Potsdam, Germany, 12–15 Septem- ber 2023

Shen Y, Song K, Tan X et al (2023) HuggingGPT: solving AI tasks with ChatGPT and its friends in HuggingFace. arXiv. <http://arxiv.org/abs/2303.17580>

Shi H, Pi R, Xu H et al (2020) Bridging the gap between sample-based and one-shot neural architecture search with BONAS. In: Proceedings of the advances in neural information processing systems 33: annual conference on neural information processing systems, virtual event, 6–12 December 2020

Shin R, Packer C, Song D (2018) Diferentiable neural network architecture search. In: Workshop track proceedings of the 6th international conference on learning representations, Vancouver, BC, Can- ada, 30 April–3 May 2018

Silver D, Schrittwieser J, Simonyan K et al (2017) Mastering the game of go without human knowledge. Nature 550(7676):354–359

Silver D, Hubert T, Schrittwieser J et al (2018) A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science 362(6419):1140–1144

Simon D (2013) Evolutionary optimization algorithms. Wiley Online Library, Hoboken

Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algo- rithms. In: Proceedings of the 25th international conference on neural information processing sys- tems, Lake Tahoe, NV, USA, 3–6 Dec 2012. pp 2951–2959

Snoek J, Swersky K, Zemel RS et al (2014) Input warping for Bayesian optimization of non-station- ary functions. In: Proceedings of the 31th international conference on machine learning, Beijing, China, 21–26 June 2014. pp 1674–1682

Sobol IM (1993) Sensitivity estimates for nonlinear mathematical models. Math Model Comput Exp 1(4):407–414

Springenberg JT, Klein A, Falkner S et al (2016) Bayesian optimization with robust Bayesian neural networks. In: Proceedings of the advances in neural information processing systems 29: annual conference on neural information processing systems, Barcelona, Spain, 5–10 December 2016. pp 4134–4142

Stanley KO, Miikkulainen R (2002a) Efcient reinforcement learning through evolving neural network topologies. In: Proceedings of the genetic and evolutionary computation conference, New York, NY, USA, 9–13 July 2002. p 9

a 灬

Stanley KO, Miikkulainen R (2002b) Evolving neural networks through augmenting topologies. Evol Comput 10(2):99–127

Stanley KO, D’Ambrosio DB, Gauci J (2009) A hypercube-based encoding for evolving large-scale neu- ral networks. Artif Life 15(2):185–212

Steinruecken C, Smith E, Janz D et al (2019) The automatic statistician. In: Automated machine learn- ing. Springer, pp 161–173

Suganuma M, Shirakawa S, Nagao T (2018) A genetic programming approach to designing convolu- tional neural network architectures. In: Proceedings of the twenty-seventh international joint con- ference on artifcial intelligence, Stockholm, Sweden, 13–19 July 2018. pp 5369–5373. [https://doi.](https://doi.org/10.24963/ijcai.2018/755) [org/10.24963/ijcai.2018/755](https://doi.org/10.24963/ijcai.2018/755)

Sun Y, Xue B, Zhang M et al (2020) Automatically designing CNN architectures using the genetic algo- rithm for image classifcation. IEEE Trans Cybern 50(9):3840–3854

Sutton RS, Barto AG (2018) Reinforcement learning: an introduction. MIT Press, Cambridge

Swearingen T, Drevo W, Cyphers B et al (2017) ATM: a distributed, collaborative, scalable system for automated machine learning. In: Proceedings of the 2017 IEEE international conference on Big Data, Boston, MA, USA, 11–14 December 2017. pp 151–162. [https://doi.org/10.1109/BigData.](https://doi.org/10.1109/BigData.2017.8257923) [2017.8257923](https://doi.org/10.1109/BigData.2017.8257923)

Swersky K, Snoek J, Adams RP (2013) Multi-task Bayesian optimization. In: Proceedings of the advances in neural information processing systems 26: 27th annual conference on neural informa- tion processing systems. Lake Tahoe, NV, United States, 5–8 December 2013. pp 2004–2012

Swersky K, Duvenaud D, Snoek J et al (2014a) Raiders of the lost architecture: kernels for Bayesian optimization in conditional parameter spaces. arXiv. <https://doi.org/10.48550/arxiv.1409.4011>

Swersky K, Snoek J, Adams RP (2014b) Freeze-thaw Bayesian optimization. CoRR. [http://arxiv.org/abs/](http://arxiv.org/abs/1406.3896) [1406.3896](http://arxiv.org/abs/1406.3896)

Talos A (2019) Autonomio Talos. Talos

Tan M, Chen B, Pang R et al (2019) MnasNet: platform-aware neural architecture search for mobile. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Long Beach, CA, USA, 16–20 June 2019

Tani L, Rand D, Veelken C et al (2021) Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics. Eur Phys J C 81(2):1–9

Templier P, Rachelson E, Wilson DG (2021) A geometric encoding for neural network evolution. In: Pro- ceedings of the genetic and evolutionary computation conference. pp 919–927. [https://doi.org/10.](https://doi.org/10.1145/3449639.3459361) [1145/3449639.3459361](https://doi.org/10.1145/3449639.3459361)

Tetteroo J, Baratchi M, Hoos HH (2022) Automated machine learning for covid-19 forecasting. IEEE Access 10:94718–94737. <https://doi.org/10.1109/ACCESS.2022.3202220>

Thakur A, Krohn-Grimberghe A (2015) AutoCompete: a framework for machine learning competition. In: Proceedings of the 2th international conference on machine learning workshop on automated machine learning, Lille, France, 6–11 July 2015

Thompson WR (1933) On the likelihood that one unknown probability exceeds another in view of the evi- dence of two samples. Biometrika 25(3/4):285–294

Thornton C, Hutter F, Hoos HH et al (2013) Auto-WEKA: combined selection and hyperparameter optimi- zation of classifcation algorithms. In: Proceedings of the 19th international conference on knowledge discovery and data mining, Chicago, IL, USA, 11–14 August 2013. pp 847–855. [https://doi.org/10.](https://doi.org/10.1145/2487575.2487629) [1145/2487575.2487629](https://doi.org/10.1145/2487575.2487629)

Tornede A, Deng D, Eimer T et al (2023a) AutoML in the age of large language models: current challenges, future opportunities and risks. CoRR abs/2306.08107. <https://doi.org/10.48550/ARXIV.2306.08107>. <http://arxiv.org/abs/2306.08107>

Tornede T, Tornede A, Hanselle J et al (2023b) Towards green automated machine learning: status quo and future directions. J Artif Intell Res 77:427–457. <https://doi.org/10.1613/jair.1.14340>

Truong A, Walters A, Goodsitt J et al (2019) Towards automated machine learning: evaluation and com- parison of AutoML approaches and tools. In: Proceedings of the 31st international conference on tools with artifcial intelligence, Portland, OR, USA, 4–6 November 2019. [https://doi.org/10.1109/](https://doi.org/10.1109/ictai.2019.00209) [ictai.2019.00209](https://doi.org/10.1109/ictai.2019.00209)

Tseng E, Yu F, Yang Y et al (2019) Hyperparameter optimization in black-box image processing using dif- ferentiable proxies. ACM Trans Graph. <https://doi.org/10.1145/3306346.3322996>

van Rijn JN, Hutter F (2018) Hyperparameter importance across datasets. In: Proceedings of the 24th inter- national conference on knowledge discovery & data mining, London, UK, 19–23 August 2018. pp 2367–2376. <https://doi.org/10.1145/3219819.3220058>

a 灬

van Rijn JN, Abdulrahman SM, Brazdil P et al (2015) Fast algorithm selection using learning curves. In: Advances in intelligent data analysis XIV—14th international symposium, Saint Etienne, France,

22–24 October 2015, proceedings. pp 298–309. <https://doi.org/10.1007/978-3-319-24465-5_26>

Vanschoren J (2018) Chapter 2—meta-learning. In: Automated machine learning. Springer, pp 39–68

Vanschoren J, van Rijn JN, Bischl B et al (2013) OpenML: networked science in machine learning. SIG-

KDD Explor 15(2):49–60. <https://doi.org/10.1145/2641190.2641198>

Varma S, Simon R (2006) Bias in error estimation when using cross-validation for model selection. BMC Bioinform 7:91. <https://doi.org/10.1186/1471-2105-7-91>

Wan X, Ru B, Esperança PM et al (2022) On redundancy and diversity in cell-based neural architecture search. In: The tenth international conference on learning representations, virtual event, 25–29 April 2022

Wang L, Zhao Y, Jinnai Y et al (2020) Neural architecture search using deep neural networks and Monte Carlo tree search. In: The thirty-fourth AAAI conference on artifcial intelligence, New York, NY, USA, 7–12 February 2020. pp 9983–9991

Wang C, Wang H, Zhou C et al (2021a) Experiencethinking: constrained hyperparameter optimization based on knowledge and pruning. Knowl Based Syst 223(106):602

Wang C, Wu Q, Weimer M et al (2021b) FLAML: a fast and lightweight AutoML library. In: Proceedings of machine learning and systems 2021, virtual event, 5–9 April 2021

Wang S, Zhang H, Xu K et al (2021c) Beta-CROWN: efcient bound propagation with per-neuron split constraints for neural network robustness verifcation. In: Advances in neural information processing systems 34: annual conference on neural information processing systems 2021, virtual event, 6–14 December 2021. pp 29909–29921

Wang C, Baratchi M, Bäck T et al (2022a) Towards automated machine learning for time-series forecasting. Under review

Wang C, Baratchi M, Bäck T et al (2022b) Towards time-series feature engineering in automated machine learning for multi-step-ahead forecasting. Eng Proc. <https://doi.org/10.3390/engproc2022018017>

Waring J, Lindvall C, Umeton R (2020) Automated machine learning: review of the state-of-the-art and opportunities for healthcare. Artif Intell Med 104(101):822

Watanabe S, Roux JL (2014) Black box optimization for automatic speech recognition. In: Proceedings of the IEEE international conference on acoustics, speech and signal processing, Florence, Italy, 4–9 May 2014. pp 3256–3260. <https://doi.org/10.1109/ICASSP.2014.6854202>

Watkins CJCH (1989) Learning from delayed rewards. PhD Thesis, University of Cambridge

Weerts H, Pfsterer F, Feurer M et al (2023) Can fairness be automated? Guidelines and opportunities for fairness-aware AutoML. <http://arxiv.org/abs/2303.08485>

Wei T, Wang C, Rui Y et al (2016) Network morphism. In: Proceedings of the international conference on machine learning. pp 564–572

Wever M, Mohr F, Hüllermeier E (2018) Automated multi-label classifcation based on ML-Plan. arXiv.

<http://arxiv.org/abs/1811.04060>

White C, Neiswanger W, Savani Y (2021a) BANANAS: Bayesian optimization with neural architectures for neural architecture search. In: Proceedings of the thirty-ffth AAAI conference on artifcial intelligence, virtual event, 2–9 February 2021. pp 10293–10301

White C, Zela A, Ru R et al (2021b) How powerful are performance predictors in neural architecture search? In: Advances in neural information processing systems 34: annual conference on neural information processing systems 2021, virtual event, 6–14 December 2021. pp 28454–28469

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8(3):229–256

Wistuba M (2017) Finding competitive network architectures within a day using UCT. arXiv. [http://](http://arxiv.org/abs/1712.07420) [arxiv.org/abs/1712.07420](http://arxiv.org/abs/1712.07420)

Wistuba M (2018) Deep learning architecture search by neuro-cell-based evolution with function-pre- serving mutations. In: Proceedings of the machine learning and knowledge discovery in data- bases—European conference, Dublin, Ireland, 10–14 September 2018. pp 243–258. [https://doi.](https://doi.org/10.1007/978-3-030-10928-8_15) [org/10.1007/978-3-030-10928-8\_15](https://doi.org/10.1007/978-3-030-10928-8_15)

Wistuba M, Schilling N, Schmidt-Thieme L (2015a) Hyperparameter search space pruning—a new com- ponent for sequential model-based hyperparameter optimization. In: Proceedings of the European conference on machine learning and knowledge discovery in databases, Porto, Portugal, 7–11 Sep- tember 2015. pp 104–119

Wistuba M, Schilling N, Schmidt-Thieme L (2015b) Sequential model-free hyperparameter tuning. In: Proceedings of the 2015 IEEE international conference on data mining, Atlantic City, NJ, USA, 14–17 November 2015. pp 1033–1038

a 灬

Wistuba M, Rawat A, Pedapati T (2019) A survey on neural architecture search. arXiv. [http://arxiv.org/](http://arxiv.org/abs/1905.01392) [abs/1905.01392](http://arxiv.org/abs/1905.01392)

Wong C, Houlsby N, Lu Y et al (2018) Transfer learning with neural AutoML. In: Proceedings of the 32nd international conference on neural information processing systems, Montréal, Canada, 3–8 December 2018. pp 8366–8375

Wu B, Dai X, Zhang P et al (2019) FBNet: hardware-aware efcient convnet design via diferentiable neural architecture search. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Long Beach, CA, USA, 16–20 June 2019. pp 10734–10742

Wu J, Chen S, Liu X (2020) Efcient hyperparameter optimization through model-based reinforcement learning. Neurocomputing 409:381–393. <https://doi.org/10.1016/j.neucom.2020.06.064>

Wu Q, Wang C, Huang S (2021) Frugal optimization for cost-related hyperparameters. In: Proceed- ings of the AAAI conference on artifcial intelligence, virtual event, 2–9 February 2021. pp 10347–10354

Xie L, Yuille A (2017) Genetic CNN. In: Proceedings of the IEEE international conference on computer vision, Venice, Italy, 22–29 October 2017. pp 1379–1388

Xie S, Zheng H, Liu C et al (2019) SNAS: stochastic neural architecture search. In: Proceedings of the 7th international conference on learning representations, New Orleans, LA, USA, 6–9 May 2019

Xie L, Chen X, Bi K et al (2022a) Weight-sharing neural architecture search: a battle to shrink the opti- mization gap. ACM Comput Surv 54(9):183:1-183:37. <https://doi.org/10.1145/3473330>

Xie X, Liu Y, Sun Y et al (2022b) BenchENAS: a benchmarking platform for evolutionary neural archi- tecture search. IEEE Trans Evol Comput 26(6):1473–1485. [https://doi.org/10.1109/TEVC.2022.](https://doi.org/10.1109/TEVC.2022.3147526) [3147526](https://doi.org/10.1109/TEVC.2022.3147526)

Xu Y, Xie L, Zhang X et al (2020) PC-DARTS: partial channel connections for memory-efcient archi- tecture search. In: Proceedings of the 8th international conference on learning representations, Addis Ababa, Ethiopia, 26–30 April 2020

Xue S, Wang R, Zhang B et al (2021) IDARTS: interactive diferentiable architecture search. In: Pro- ceedings of the 2021 IEEE international conference on computer vision, Montreal, QC, Canada, 10–17 October 2021. pp 1143–1152. <https://doi.org/10.1109/ICCV48922.2021.00120>

Yakovlev A, Moghadam HF, Moharrer A et al (2020) Oracle AutoML: a fast and predictive AutoML pipeline. Proc VLDB Endow 13(12):3166–3180

Yang C, Akimoto Y, Kim DW et al (2019) OBOE: collaborative fltering for AutoML model selection. In: Proceedings of the 25th international conference on knowledge discovery & data mining, Anchorage, AK, USA, 4–8 August 2019. pp 1173—-1183. [https://doi.org/10.1145/3292500.33309](https://doi.org/10.1145/3292500.3330909) [09](https://doi.org/10.1145/3292500.3330909)

Yang C, Fan J, Wu Z et al (2020) AutoML pipeline selection: efciently navigating the combinatorial space. In: Proceedings of the 26th international conference on knowledge discovery & data min- ing. pp 1446–1456. <https://doi.org/10.1145/3394486.3403197>

Yang J, Liu Y, Xu H (2023) HOTNAS: hierarchical optimal transport for neural architecture search. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, Vancouver, BC, Canada, 17–24 June 2023. pp 11990–12000

Yao X (1999) Evolving artifcial neural networks. Proc IEEE 87(9):1423–1447. [https://doi.org/10.](https://doi.org/10.1109/5.784219) [1109/5.784219](https://doi.org/10.1109/5.784219)

Yao Q, Wang M, Escalante HJ et al (2018) Taking human out of learning applications: a survey on auto- mated machine learning. arXiv. <http://arxiv.org/abs/1810.13306>

Ye P, Li B, Li Y et al (2022) b-DARTS: beta-decay regularization for diferentiable architecture search. In: Proceedings of the IEEE conference on computer vision and pattern recognition, New Orleans, LA, USA, 19–20 June 2022. pp 10874–10883

Yelp (2014) Metric optimization engine. <https://github.com/Yelp/MOE>

Ying C, Klein A, Christiansen E et al (2019) Nas-bench-101: towards reproducible neural architecture search. In: Proceedings of the international conference on machine learning, Long Beach, CA, USA, 9–15 June 2019. pp 7105–7114

Yu K, Sciuto C, Jaggi M et al (2020) Evaluating the search phase of neural architecture search. In: Pro- ceedings of the 8th international conference on learning representations, Addis Ababa, Ethiopia, 26–30 April 2020

Zagoruyko S, Komodakis N (2016) Wide residual networks. In: Proceedings of the British machine vision conference 2016, York, UK, 19–22 September 2016

Zela A, Klein A, Falkner S et al (2018) Towards automated deep learning: efcient joint neural architec- ture and hyperparameter search. arXiv. <http://arxiv.org/abs/1807.06906>

a 灬

Zela A, Siems J, Hutter F (2020) NAS-Bench-1Shot1: benchmarking and dissecting one-shot neural architecture search. In: Proceedings of the 8th international conference on learning representa- tions, Addis Ababa, Ethiopia, 26–30 April 2020

Zela A, Siems JN, Zimmer L et al (2022) Surrogate NAS benchmarks: going beyond the limited search spaces of tabular NAS benchmarks. In: The tenth international conference on learning representa- tions, virtual event, 25–29 April 2022

Zhang X, Zhou X, Lin M et al (2018) ShufeNet: an extremely efcient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recogni- tion, Salt Lake City, UT, USA, 18–22 June 2018. pp 6848–6856. [https://doi.org/10.1109/CVPR.](https://doi.org/10.1109/CVPR.2018.00716) [2018.00716](https://doi.org/10.1109/CVPR.2018.00716)

Zhang C, Ren M, Urtasun R (2019a) Graph hypernetworks for neural architecture search. In: Proceed- ings of the 7th international conference on learning representations, New Orleans, LA, USA, 6–9 May 2019

Zhang H, Yu Y, Jiao J et al (2019b) Theoretically principled trade-of between robustness and accu- racy. In: Chaudhuri K, Salakhutdinov R (eds) Proceedings of the 36th international conference on machine learning, ICML 2019, 9–15 June 2019, Long Beach, CA, USA. pp 7472–7482

Zhang M, Li H, Pan S et al (2020) Overcoming multi-model forgetting in one-shot NAS with diversity maximization. In: Proceedings of the IEEE conference on computer vision and pattern recogni- tion, Seattle, WA, USA, 13–19 June 2020. pp 7809–7818

Zhang H, Jin Y, Hao K (2022a) Evolutionary search for complete neural network architectures with partial weight sharing. IEEE Trans Evol Comput 26(5):1072–1086. [https://doi.org/10.1109/TEVC.](https://doi.org/10.1109/TEVC.2022.3140855) [2022.3140855](https://doi.org/10.1109/TEVC.2022.3140855)

Zhang W, Lin Z, Shen Y et al (2022b) Deep and fexible graph neural architecture search. In: Proceed- ings of the 39th international conference on machine learning, Baltimore, MD, USA, 17–23 July 2022. pp 26362–26374

Zhong Z, Yan J, Wu W et al (2018) Practical block-wise neural network architecture generation. In: Pro- ceedings of the 2018 IEEE conference on computer vision and pattern recognition, Salt Lake City, UT, USA, 18–22 June 2018. pp 2423–2432. <https://doi.org/10.1109/CVPR.2018.00257>

Zhou Q, Zheng X, Cao L et al (2021) EC-DARTS: inducing equalized and consistent optimization into DARTS. In: Proceedings of the IEEE international conference on computer vision, Montréal, QC, Canada, 10–17 October 2021. pp 11986–11995

Zimmer L, Lindauer M, Hutter F (2021) Auto-PyTorch tabular: multi-fdelity metalearning for efcient and robust AutoDL. IEEE Trans Pattern Anal Mach Intell 43(9):3079–3090

Zöller MA, Huber MF (2021) Benchmark and survey of automated machine learning frameworks. J Artif Intell Res 70:409–472

Zoph B, Le QV (2017) Neural architecture search with reinforcement learning. In: Proceedings of the 5th international conference on learning representations, Toulon, France, 24–26 April 2017

Zoph B, Vasudevan V, Shlens J et al (2018) Learning transferable architectures for scalable image recogni- tion. In: Proceedings of the 2018 IEEE conference on computer vision and pattern recognition, Salt Lake City, UT, USA, 18–22 June 2018. pp 8697–8710. <https://doi.org/10.1109/CVPR.2018.00907>

**Publisher’s Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional afliations.

**Authors and Afliations**

**Mitra Baratchi1 · Can Wang1 · Stefen Limmer2 · Jan N. van Rijn1 · Holger Hoos1,3 Thomas Bäck1 · Markus Olhofer2**

**·**

 Mitra Baratchi

m.baratchi@liacs.leidenuniv.nl

Can Wang

c.wang@liacs.leidenuniv.nl

Stefen Limmer

stefen.limmer@honda-ri.de

a 灬

Jan N. van Rijn

j.n.van.rijn@liacs.leidenuniv.nl

Holger Hoos hh@liacs.nl

Thomas Bäck

t.h.w.baeck@liacs.leidenuniv.nl

Markus Olhofer

markus.olhofer@honda-ri.de

1 Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands

2 Honda Research Institute Europe, Ofenbach, Germany

3 Chair for AI Methodology (AIM), RWTH Aachen University, 10587 Aachen, Germany

a 灬