Automated Machine Learning: Techniques and Frameworks

Radwa Elshawi(✉) and Sherif Sakr

Data Systems Group, University of Tartu, Tartu, Estonia
{radwa.elshawi,sherif.sakr}@ut.ee

Abstract. Nowadays, machine learning techniques and algorithms are employed in almost every application domain (e.g., financial applications, advertising, recommendation systems, user behavior analytics). In practice, they are playing a crucial role in harnessing the power of massive amounts of data which we are currently producing every day in our digital world. In general, the process of building a high-quality machine learning model is an iterative, complex and time-consuming process that involves trying different algorithms and techniques in addition to having a good experience with effectively tuning their hyper-parameters. In particular, conducting this process efficiently requires solid knowledge and experience with the various techniques that can be employed. With the continuous and vast increase of the amount of data in our digital world, it has been acknowledged that the number of knowledgeable data scientists can not scale to address these challenges. Thus, there was a crucial need for automating the process of building good machine learning models (AutoML). In the last few years, several techniques and frameworks have been introduced to tackle the challenge of automating the machine learning process. The main aim of these techniques is to reduce the role of humans in the loop and fill the gap for non-expert machine learning users by playing the role of the domain expert. In this chapter, we present an overview of the state-of-the-art efforts in tackling the challenges of machine learning automation. We provide a comprehensive coverage for the various tools and frameworks that have been introduced in this domain. In addition, we discuss some of the research directions and open challenges that need to be addressed in order to achieve the vision and goals of the AutoML process.

1 Introduction

Due to the increasing success of machine learning techniques in several application domains, they have been attracting a lot of attention from the research and business communities. In general, the effectiveness of machine learning techniques mainly rests on the availability of massive datasets. Recently, we have been witnessing a continuous exponential growth in the size of data produced by various kinds of systems, devices and data sources. It has been reported that there are 2.5 quintillion bytes of data is being created every day where 90% of

R.-D. Kutsche and E. Zimányi (Eds.): eBISS 2019, LNBIP 390, pp. 40–69, 2020.
https://doi.org/10.1007/978-3-030-61627-4_3

stored data in the world, has been generated in the past two years only[1]. On the one hand, the more data that is available, the richer and the more robust the insights and the results that machine learning techniques can produce. Thus, in the Big Data Era, we are witnessing many leaps achieved by machine and deep learning techniques in a wide range of fields [1,2]. On the other hand, this situation is raising a potential *data science crisis*, similar to the software crisis [3], due to the crucial need of having an increasing number of data scientists with strong knowledge and good experience so that they are able to keep up with harnessing the power of the massive amounts of data which are produced daily. In particular, it has been acknowledged that *data scientists can not scale*[2] and it is almost impossible to balance between the number of qualified data scientists and the required effort to manually analyze the increasingly growing sizes of available data. Thus, we are witnessing a growing focus and interest to support automating the process of building machine learning pipelines where the presence of a human in the loop can be dramatically reduced, or preferably eliminated.

In general, the process of building a high-quality machine learning model is an iterative, complex and time-consuming process that involves a number of steps. In particular, a data scientist is commonly *challenged* with a large number of choices where informed decisions need to be taken. For example, the data scientist needs to select among a wide range of possible algorithms including classification or regression techniques (e.g. Support Vector Machines, Neural Networks, Bayesian Models, Decision Trees, etc.) in addition to tuning numerous hyper-parameters of the selected algorithm. In addition, the performance of the model can also be judged by various metrics (e.g., accuracy, sensitivity, specificity, F1-score). Naturally, the decisions of the data scientist in each of these steps affect the performance and the quality of the developed model [4–6]. For instance, in `yeast dataset`[3], different parameter configurations of a Random Forest classifier result in different range of accuracy values, around 5%[4]. Also, using different classifier learning algorithms leads to widely different performance values, around 20%, for the fitted models on the same dataset. Although making such decisions require solid knowledge and expertise, in practice, increasingly, users of machine learning tools are often non-experts who require *off-the-shelf* solutions. Therefore, there has been a growing interest to *automate* and *democratize* the steps of building the machine learning pipelines.

In the last years, several techniques and frameworks have been introduced to tackle the challenge of automating the process of Combined Algorithm Selection and Hyper-parameter tuning (CASH) in the machine learning domain. These techniques have commonly formulated the problem as an optimization problem that can be solved by a wide range of techniques [7–9]. In general, the $CASH$ problem is described as follows:

[1] Forbes: How Much Data Do We Create Every Day? May 21, 2018.
[2] https://hbr.org/2015/05/data-scientists-dont-scale.
[3] https://www.openml.org/d/40597.
[4] https://www.openml.org/t/2073.

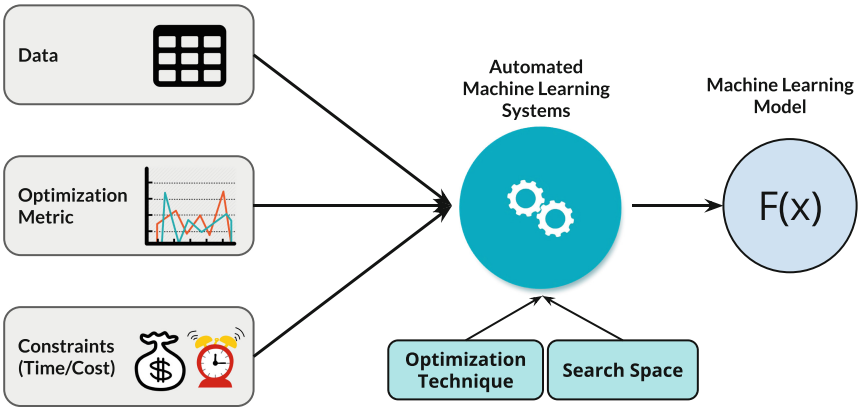

Fig. 1. The general workflow of the AutoML process.

Given a set of machine learning algorithms $\mathbf{A} = \{A^{(1)}, A^2, ...\}$, and a dataset D divided into disjoint training D_{train}, and validation $D_{validation}$ sets. The goal is to find an algorithm $A^{(i)^*}$ where $A^{(i)} \in \mathbf{A}$ and $A^{(i)^*}$ is a tuned version of $A^{(i)}$ that achieves the highest generalization performance by training $A^{(i)}$ on D_{train}, and evaluating it on $D_{validation}$. In particular, the goal of any CASH optimization technique is defined as:

$$A^{(i)^*} \in \underset{A \,\epsilon\, \mathbf{A}}{argmin}\ L(A^{(i)}, D_{train}, D_{validation})$$

where $\mathrm{L}(A^{(i)}, D_{train}, D_{validation})$ is the loss function (e.g.: error rate, false positives, etc.). In practice, one constraint for CASH optimization techniques is the *time budget*. In particular, the aim of the optimization algorithm is to select and tune a machine learning algorithm that can achieve (near)-optimal performance in terms of the user-defined evaluation metric (e.g., accuracy, sensitivity, specificity, F1-score) within the user-defined *time budget* for the search process (Fig. 1).

In this chapter, we present an overview of the state-of-the-art efforts for the techniques and framework in the automated machine learning domain. The remainder of this chapter is organized as follows. Section 2 covers the various techniques and frameworks that have been introduced to tackle the challenge of the automated machine learning process while Sect. 3 covers the automated deep learning process. We discuss some of the research directions and open challenges that need to be addressed in order to achieve the vision and goals of the AutoML process in Sect. 4 before we finally conclude the chapter in Sect. 5.

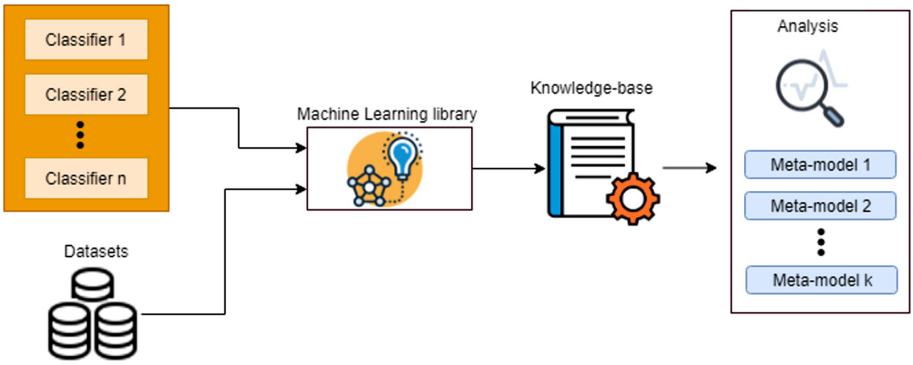

Fig. 2. An overview of meta-learning process.

2 Automated Machine Learning

In general, meta-learning can be described as the process of learning from previous experience gained during applying various learning algorithms on different kinds of data, and hence reducing the needed time to learn new tasks [10]. In the context of machine learning, several *meta learning*-techniques have been introduced as an effective mechanism to tackle the challenge of warm start for optimization algorithms. Figure 2 illustrates an overview of the meta-learning process. These techniques can generally be categorized into three broad groups [11]: *learning based on task properties*, *learning from previous model evaluations* and *learning from already pretrained models* (Fig. 3).

One group of meta-learning techniques has been based on learning from task properties using the *meta-features* that characterize a particular dataset [9]. Generally speaking, each prior task is characterized by a feature vector, of k features, $m(t_j)$. Simply, information from a prior task t_j can be transferred to a new task t_{new} based on their similarity, where this similarity between t_{new} and t_j can be calculated based on the distance between their corresponding feature vectors. In addition, a meta learner L can be trained on the feature vectors of prior tasks along with their evaluations $\mathbf{P}$ to predict the performance of configurations θ_i on t_{new}.

Some of the commonly used meta features for describing datasets are simple meta features including number of instances, number of features, statistical features (e.g., skewness, kurtosis, correlation, co-variance, minimum, maximum, average), landmark features (e.g., performance of some landmark learning algorithms on a sample of the dataset), and information theoretic features (e.g., the entropy of class labels) [11]. In practice, the selection of the best set of meta features to be used is highly dependent on the application [12]. When computing the similarity between two tasks represented as two feature vectors of meta data, it is important to normalize these vectors or apply dimensionality reduction techniques such as principle component analysis [12,13]. Another way to

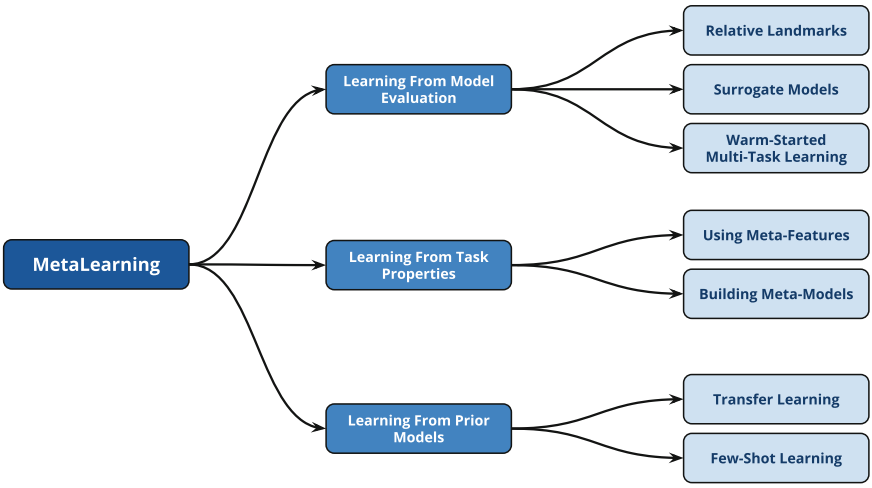

Fig. 3. A taxonomy of meta-learning techniques.

extract meta-features is to learn a joint distribution representation for a set of tasks.

Another meta-learning approach is to learn from prior tasks properties is through building *meta-models*. In this process, the aim is to build a meta model L that learns complex relationships between meta features of prior tasks t_j. For a new task t_{new}, given the meta features for task t_{new}, model L is used to recommend the best configurations. There exists a rich literature on using meta models for model configuration recommendations [14–18]. Meta models can also be used to rank a particular set of configurations by using the $K-$nearest neighbour model on the meta features of prior tasks and predicting the top k tasks that are similar to new task t_{new} and then ranking the best set of configurations of these similar tasks [19,20]. Moreover, they can also be used to predict the performance of new task based on a particular configuration [21,22]. This gives an indication about how good or bad this configuration can be, and whether it is worth evaluating it on a particular new task.

Another group of meta-learning techniques are based on *learning from previous model evaluation*. In this context, the problem is formally defined as follows.

Given a set of machine learning tasks $t_j \in T$, their corresponding learned models along their hyper-parameters $\theta \in \Theta$ and $P_{i,j} = P(\theta_i, t_j)$, the problem is to learn a meta-learner L that is trained on meta-data $\mathbf{P} \cup \mathbf{P}_{new}$ to predict recommended configuration Θ^*_{new} for a new task t_{new}, where T is the set of all prior machine learning tasks. Θ is the configuration space (hyper-parameter setting, pipeline components, network architecture, and network hyper-parameter), Θ_{new} is the configuration space for a new machine learning task t_{new}, $\mathbf{P}$ is the set of all prior evaluations $P_{i,j}$ of configuration θ_i on a prior task t_j, and $\mathbf{P}_{new}$ is a set of evaluations $P_{i,new}$ for a new task t_{new}.

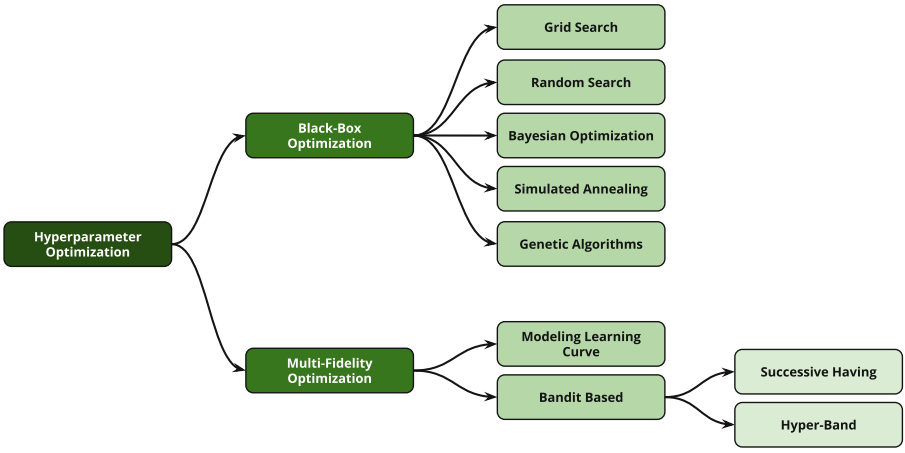

Fig. 4. A taxonomy for the hyper-parameter optimization techniques.

Learning from prior models can be done using *Transfer learning* [23], which is the process of utilization of pretrained models on prior tasks t_j to be adapted on a new task t_{new}, where tasks t_j and t_{new} are similar. Transfer learning has received lots of attention especially in the area of neural network. In particular, neural network architecture and neural network parameters are trained on prior task t_j that can be used as an initialization for model adaptation on a new task t_{new}. Then, the model can be fine-tuned [24–26]. It has been shown that neural networks trained on big image datasets such as ImageNet [17] can be transferred as well to new tasks [27,28]. Transfer learning usually works well when the new task to be learned is similar to the prior tasks, otherwise transfer learning may lead to unsatisfactory results [29]. In addition, prior models can be used in *Few-Shot Learning* where a model is required to be trained using a few training instances given the prior experience gained from already trained models on similar tasks.

2.1 Hyper-parameter Optimization

In general, several hyper-parameter optimization techniques have been based and borrowed ideas from the domains of statistical model selection and traditional optimization techniques [30–32]. In principle, the automated hyper-parameter tuning techniques can be classified into two main categories: *black-box optimization techniques* and *multi-fidelity optimization techniques* (Fig. 4).

Black-Box Optimization. *Grid search* is a simple basic solution for the hyper-parameter optimization [33] in which all combinations of hyper-parameters are evaluated. Thus, grid search is computationally expensive, infeasible and suffers from the *curse of dimensionality* as the number of trails grows exponentially with

the number of hyper-parameters. Another alternative is *random search* in which it samples configurations at random until a particular budget B is exhausted [34]. Given a particular computational budget B, random search tends to find better solutions than grid search [33]. One of the main advantages of random search, and grid search is that they can be easily parallelized over a number of workers which is essential when dealing with big data.

Bayesian Optimization is one of the state-of-the-art black-box optimization techniques which is tailored for expensive objective functions [35,36]. Bayesian optimization has received huge attention from the machine learning community in tuning deep neural networks for different tasks including classification tasks [37, 38], speech recognition [39] and natural language processing [40]. Bayesian optimization consists of two main components which are surrogate models for modeling the objective function and an acquisition function that measures the value that would be generated by the evaluation of the objective function at a new point. Gaussian processes have become the standard surrogate for modeling the objective function in Bayesian optimization [38,41]. One of the main limitations of the Gaussian processes is the cubic complexity to the number of data points which limits their parallelization capability. Another limitation is the poor scalability when using the standard kernels. Random forests [42] are another choice for modeling the objective function in Bayesian optimization. First, the algorithm starts with growing B regression trees, each of which is built using n randomly selected data points with replacement from training data of size n. For each tree, a split node is chosen from d algorithm parameters. The minimum number of points are considered for further split are set to 10 and the number of trees B to grow is set be 10 to maintain low computational overhead. Then, the random forest predicted mean and variance for each new configuration is computed. The random forests' complexity of the fitting and predicting variances are $O(n \log n)$ and $O(\log n)$ respectively which is much better compared to the Gaussian process. Random forests are used by the Sequential Model-based Algorithm Configuration (SMAC) library [43]. In general Tree-structured Parzen Estimator (TPE) [44] does not define a predictive distribution over the objective function but it creates two density functions that act as generative models for all domain variables. Given a percentile α, the observations are partitioned into two sets of observations (good observations and bad observations) where simple Parzen windows are used to model the two sets. The ratio between the two density functions reflects the expected improvement in the acquisition function and is used to recommend new configurations for hyper-parameters. Tree-Structured Parzen estimator (TPE) has shown great performance for hyper-parameter optimization tasks [44–48].

Simulated Annealing is a hyper-parameter optimization approach which is inspired by the metallurgy technique of heating and controlled cooling of materials [49]. This optimization technique goes through a number of steps. First, it randomly chooses a single value (current state) to be applied to all hyper-parameters and then evaluates model performance based on it. Second, it randomly updates the value of one of the hyper-parameters by picking a value from the immediate neighborhood to get neighboring state. Third, it evaluates the model performance

based on the neighboring state. Forth, it compares the performance obtained from the current and neighbouring states. Then, the user chooses to reject or accept the neighbouring state as a current state based on some criteria.

Genetic Algorithms (GA) are inspired by the process of natural selection [50]. The main idea of genetic-based optimization techniques is simply applying multiple genetic operations to a population of configurations. For example, the *crossover* operation simply takes two parent *chromosomes* (configurations) and combines their genetic information to generate new *offspring*. More specifically, the two parents configurations are cut at the same crossover point. Then, the sub-parts to the right of that point are swapped between the two parents chromosomes. This contributes to two new *offspring* (child configuration). Mutation randomly chooses a chromosome and mutates one or more of its parameters that results in a totally new chromosome.

Multi-fidelity Optimization. Multi-fidelity optimization is an optimization technique which focuses on decreasing the evaluation cost by combining a large number of cheap low-fidelity evaluations and a small number of expensive high-fidelity evaluation [51]. In practice, such an optimization technique is essential when dealing with big datasets as training one hyper-parameter may take days. More specifically, in multi-fidelity optimization, we can evaluate samples in different levels. For example, we may have two evaluation functions: *high-fidelity* evaluation and *low-fidelity* evaluation. The high-fidelity evaluation outputs precise evaluation from the whole dataset. On the other hand, the low-fidelity evaluation is a cheaper evaluation from a subset of the dataset. The idea behind the multi-fidelity evaluation is to use many low-fidelity evaluation to reduce the total evaluation cost. Although the low fidelity optimization results in cheaper evaluation cost that may suffer from optimization performance, but the speedup achieved is more significant than the approximation error.

Modeling learning curves is an optimization technique that models learning curves during hyper-parameter optimization and decides whether to allocate more resources or to stop the training procedure for a particular configuration. For example, a curve may model the performance of a particular hyper-parameter on an increasing subset of the dataset. Learning curve extrapolation is used in predicting early termination for a particular configuration [36]; the learning process is terminated if the performance of the predicted configuration is less than the performance of the best model trained so far in the optimization process. Combining early predictive termination criterion with Bayesian optimization leads to more reduction in the model error rate than the vanilla Bayesian black-box optimization. In addition, such a technique resulted in speeding-up the optimization by a factor of 2 and achieved the state-of-the-art neural network on CIFAR-10 dataset [52].

Bandit-based algorithms have shown to be powerful in tackling deep learning optimization challenges. In the following, we consider two strategies of the bandit-based techniques which are the *Successive halving* and *HyperBand*. *Successive halving* is a bandit-based powerful multi-fidelity technique in which

given a budget B, first, all the configurations are evaluated. Next, they are ranked based on their performance. Then, half of these configurations that performed worse than the others are removed. Finally, the budget of the previous steps is doubled and repeated until only one algorithm remains. It is shown that the successive halving outperforms the uniform budget al.location technique in terms of the computation time, and the number of iterations required [53]. On the other hand, successive halving suffer from the following problem. Given a time budget B, the user has to choose, in advance, whether to consume the larger portion of the budget exploring a large number of configurations while spending a small portion of the time budget on tuning each of them or to consume the large portion of the budget on exploring few configurations while spending the larger portion of the budget on tuning them.

HyperBand is another bandit-based powerful multi-fidelity hedging technique that optimizes the search space when selecting from randomly sampled configurations [54]. More specifically, partition a given budget B into combinations of number of configurations and budget assigned to each configuration. Then, call successive halving technique on each random sample configuration. Hyper-Band shows great success with deep neural networks and performs better than random search and Bayesian optimization.

2.2 AutoML Tools and Frameworks

In this section, we provide a comprehensive overview of several tools and frameworks that have been implemented to automate the process of combined algorithm selection and hyper-parameter optimization process. In general, these tools and frameworks can be classified into two main categories: *centralized* and *distributed*.

Centralized Frameworks. Several tools have been implemented on top of widely used *centralized* machine learning packages which are designed to run in a *single* node (machine). In general, these tools are suitable for handling small and medium sized datasets. For example, *Auto-Weka*[5] is considered as the first and pioneer machine learning automation framework [7]. It was implemented in Java on top of `Weka`[6], a popular machine learning library that has a wide range of machine learning algorithms. `Auto-Weka` applies Bayesian optimization using Sequential Model-based Algorithm Configuration (`SMAC`) [43] and tree-structured parzen estimator (TPE) for both algorithm selection and hyper-parameter optimization (Auto-Weka uses SMAC as its default optimization algorithm but the user can configure the tool to use TPE). In particular, SMAC tries to draw the relation between algorithm performance and a given set of hyper-parameters by estimating the predictive mean and variance of their performance along the trees of a random forest model. The main advantage of using SMAC is its robustness by having the ability to discard low performance parameter configurations

[5] https://www.cs.ubc.ca/labs/beta/Projects/autoweka/.
[6] https://www.cs.waikato.ac.nz/ml/weka/.

quickly after the evaluation on a low number of dataset folds. SMAC shows better performance on experimental results compared to TPE [43].

$Auto-MEKA_{GGP}$ [55] focuses on the AutoML task for multi-label classification problem [56] that aims to learn models from data capable of representing the relationships between input attributes and a set of class labels, where each instance may belong to more than one class. Multi-label classification has lots of applications especially in medical diagnosis in which a patient may be diagnosed with more than one disease. $Auto-MEKA_{GGP}$ is a grammar-based genetic programming framework that can handle complex multi-label classification search space and simply explores the hierarchical structure of the problem. $Auto-MEKA_{GGP}$ takes as input both of the dataset and a grammar describing the hierarchical search space of the hyper-parameters and the learning algorithms from `MEKA`[7] framework [57]. $Auto-MEKA_{GGP}$ starts by creating an initial set of trees representing the multi-label classification algorithms by randomly choosing valid rules from the grammar, followed by the generation of derivation trees. Next, map each derivation tree to a specific multi-label classification algorithm. The initial trees are evaluated on the input dataset by running the learning algorithm, they represent, using MEKA framework. The quality of the individuals are assessed using different measures such as fitness function. If a stopping condition is satisfied (e.g. a quality criteria), a set of individuals (trees) are selected in a tournament selection. Crossover and mutation are applied in a way that respects the grammar constraints on the selected individuals to create a new population. At the end of the evolution, the best set of individuals representing the well performing set of multi-label tuned classifiers are returned.

Auto-Sklearn[8] has been implemented on top of `Scikit-Learn`[9], a popular Python machine learning package [8]. `Auto-Sklearn` introduced the idea of meta-learning in the initialization of combined algorithm selection and hyperparameter tuning. It used SMAC as a Bayesian optimization technique too. In addition, ensemble methods were used to improve the performance of output models. Both meta-learning and ensemble methods improved the performance of *vanilla* SMAC optimization. *hyperopt-Sklearn* [58] is another AutoML framework which is based on Scikit-learn machine learning library. Hyperopt-Sklearn uses `Hyperopt` [59] to define the search space over the possible Scikit-Learn main components including the learning and preprocessing algorithms. Hyperpot supports different optimization techniques including random search, and different Bayesian optimizations for exploring the search spaces which are characterized by different types of variables including categorical, ordinal and continuous.

TPOT[10] framework represents another type of solution that has been implemented on top of `Scikit-Learn` [60]. It is based on genetic programming by exploring many different possible pipelines of feature engineering and learning algorithms. Then, it finds the best one out of them. *Recipe* [61] follows the same optimization procedure as `TPOT` using genetic programming, which in turn

[7] http://waikato.github.io/meka/.
[8] https://github.com/automl/auto-sklearn.
[9] https://scikit-learn.org/.
[10] https://automl.info/tpot/.

exploits the advantages of a global search. However, it considers the unconstrained search problem in `TPOT`, where resources can be spent into generating and evaluating invalid solutions by adding a grammar that avoids the generation of invalid pipelines, and can speed up optimization process. Second, it works with a bigger search space of different model configurations than *Auto-SkLearn* and `TPOT`.

ML-Plan[11] has been proposed to tackle the composability challenge on building machine learning pipelines [62]. In particular, it integrates a super-set of both `Weka` and `Scikit-Learn` algorithms to construct a full pipeline. ML-Plan tackles the challenge of the search problem for finding optimal machine learning pipeline using a hierarchical task network algorithm where the search space is modeled as a large tree graph where each leaf node is considered as a goal node of a full pipeline. The graph traversal starts from the root node to one of the leaves by selecting some random paths. The quality of a certain node in this graph is measured after making n such random complete traversals and taking the minimum as an estimate for the best possible solution that can be found. The initial results of this approach has shown that the composable pipelines over `Weka` and `Scikit-Learn` do not significantly outperform the outcomes from `Auto-Weka` and `Auto-Sklearn` frameworks because it has to deal with larger search space.

SmartML[12] has been introduced as the first `R` package for automated model building for classification tasks [9]. In the algorithm selection phase, SmartML uses a meta-learning approach where the meta-features of the input dataset is extracted and compared with the meta-features of the datasets that are stored in the framework's knowledge base, populated from the results of the previous runs. The similarity search process is used to identify the similar datasets in the knowledge base, using a nearest neighbor approach, where the retrieved results are used to identify the best performing algorithms on those similar datasets in order to nominate the candidate algorithms for the dataset at hand. The hyper-parameter tuning of SmartML is based on SMAC Bayesian Optimisation [43]. SmartML maintains the results of the new runs to continuously enrich its knowledge base with the aim of further improving the accuracy of the similarity search and thus the performance and robustness for future runs.

Autostacker [63] is an AutoML framework that uses an evolutionary algorithm with hierarchical stacking for efficient hyper-parameters search. Autostacker is able to find pipelines, consisting of preprocessing, feature engineering and machine learning algorithms with the best set of hyper-parameters, rather than finding a single machine learning model with the best set of hyper-parameters. Autostacker generates cascaded architectures that allow the components of a pipeline to "correct mistakes made by each other" and hence improves the overall performance of the pipeline. Autostacker simply starts by selecting a set of pipelines randomly. Those pipelines are fed into an evolutionary algorithm that generates the set of winning pipelines.

AlphaD3M [64] has been introduced as an AutoML framework that uses meta reinforcement learning to find the most promising pipelines. AlphaD3M finds

[11] https://github.com/fmohr/ML-Plan.
[12] https://github.com/DataSystemsGroupUT/SmartML.

patterns in the components of the pipelines using recurrent neural networks, specifically long short term memory (LSTM) and Monte-Carlo tree search in an iterative process which is computationally efficient in large search space. In particular, for a given machine learning task over a certain dataset, the network predicts the action's probabilities which lead to sequences that describe the whole pipeline. The predictions of the LSTM neural network are used by Monte-Carlo tree search by running multiple simulations to find the best pipeline sequence.

OBOE[13] is an AutoML framework for time constrained model selection and hyper-parameter tuning [65]. OBOE finds the most promising machine learning model along with the best set of hyper-parameters using collaborative filtering. OBOE starts by constructing an *error matrix* for some base set of machine learning algorithms, where each row represents a dataset and each column represents a machine learning algorithm. Each cell in the matrix represents the performance of a particular machine learning model along with its hyper-parameters on a specific dataset. In addition, OBOE keeps track of the running time of each model on a particular dataset and trains a model to predict the running time of a particular model based on the size and the features of the dataset. Simply, a new dataset is considered as a new row in the error matrix. In order to find the best machine learning algorithm for a new dataset, OBOE runs a particular set of models corresponding to a subset of columns in the error matrix which are predicted to run efficiently on the new dataset. In order to find the rest of the entries in the row, the performance of the models that have not been evaluated are predicted. The good thing about this approach is that it infers the performance of lots of models without the need to run them or even computing meta-features and that is why OBOE can find a well performing model within a reasonable time budget.

The *PMF*[14] AutoML framework is based on collaborative filtering and Bayesian optimization [66]. More specifically, the problem of selecting the best performing pipeline for a specific task is modeled as a collaborative filtering problem that is solved using probabilistic matrix factorization techniques. PMF considers two datasets to be similar if they have similar evaluations on a few set of pipelines and hence it is more likely that these datasets will have similar evaluations on the rest of the pipelines. This concept is quite related to collaborative filtering for movie recommendation in which users that had the same preference in the past are more likely to have the same preference in the future. In particular, the PMF framework trains each machine learning pipeline on a sample of each dataset and then evaluates such pipeline. This results in a matrix that summarizes the performance (accuracy or balanced accuracy for classification tasks and RMSE for regression tasks) of each machine learning pipeline of each dataset. The problem of predicting the performance of a particular pipeline on a new dataset can be mapped into a matrix factorization problem.

VDS [67] has been recently introduced as an *interactive* automated machine learning tool, that followed the ideas of a previous work on the `MLBase`

[13] https://github.com/udellgroup/oboe/tree/master/automl.
[14] https://github.com/rsheth80/pmf-automl.

framework [68]. In particular, it uses a meta learning mechanism (knowledge from the previous runs) to provide the user with a quick feedback, in few seconds, with an initial model recommendation that can achieve a reasonable accuracy while, on the back-end, conducting an optimization process so that it can recommend to the user more models with better accuracies, as it progresses with the search process over the search space. The VDS framework combines cost-based Multi-Armed Bandits and Bayesian optimizations for exploring the search space while using a rule-based search-space as query optimization technique. VDS prunes unpromising pipelines in early stages using an adaptive pipeline selection algorithm. In addition, it supports a wide range of machine learning tasks including classification, regression, community detection, graph matching, image classification, and collaborative filtering. *ATMSeer*[15] is an interactive visualization tool that has been introduced to support users for refining the search space of AutoML and analyzing the results [69]. Table 1 shows a summary of the main features of the centralized state-of-the-art AutoML frameworks.

Several cloud-based solutions have been introduced to tackle the automated machine learning problem using the availability of high computational power on cloud environments to try a wide range of models and configurations. For example, *Google AutoML*[16] supports training a wide range of machine learning models in different domains with minimal user experience. *Azure AutoML*[17] is a cloud-based service that can be used to automate building machine learning pipeline for both classification and regression tasks. AutoML Azure uses collaborative filtering and Bayesian optimization to search for the most promising pipelines efficiently [66] based on a database that is constructed by running millions of experiments of evaluation of different pipelines on many datasets. *Amazon Sage Maker*[18] provides its users with a wide set of most popular machine learning, and deep learning frameworks to build their models in addition to automatic tuning for the model parameters.

Distributed Frameworks. As the size of the dataset increases, solving the *CASH* problem in a centralized manner turns out to be infeasible due to the limited computing resources (e.g., Memory, CPU) of a single machine. Thus, there is a clear need for distributed solutions that can harness the power of computing clusters that have multiple nodes to tackle the computational complexity of the problem. *MLbase*[19] has been the first work to introduce the idea of developing a distributed framework of machine learning algorithm selection and hyperparameter optimization [68]. MLbase has been based on `MLlib` [70], a

[15] https://github.com/HDI-Project/ATMSeer.
[16] https://cloud.google.com/automl/.
[17] https://docs.microsoft.com/en-us/azure/machine-learning/service/.
[18] https://aws.amazon.com/machine-learning/.
[19] http://www.mlbase.org/.

Table 1. Summary of the main features of centralized AutoML frameworks

	Release date	Core language	Training framework	Optimization technique	ML task	Meta learning	User interface	Automatic feature extraction	Open source
AutoWeka	2013	Java	Weka	Bayesian optimization	Single-label classification regression	×	✓	✓	✓
AutoSklearn	2015	Python	Scikit-learn,	Bayesian optimization	Single-label classification regression	✓	×	✓	✓
TPOT	2016	Python	Scikit-learn	Genetic algorithm	Single-label classification regression	×	×	✓	✓
SmartML	2019	R	mlr, RWeka & Other R packages	Bayesian optimization	Single-label classification	✓	✓	×	✓
Auto-MEKA$_{GGP}$	2018	Java	Meka	Grammar-based genetic algorithm	Multi-label classification	✓	×	×	✓
Recipe	2017	Python	Scikit-learn	Grammar-based genetic algorithm	Single-label classification	✓	×	✓	✓
MLPlan	2018	Java	Weka and Scikit-learn	Hierachical task planning	Single-label classification	×	×	✓	✓
Hyperopt-sklearn	2014	Python	Scikit-learn	Bayesian optimization& Random search	Single-label classification regression	×	×	✓	✓
Autostacker	2018	–	–	Genetic algorithm	Single-label classification	×	×	✓	×
VDS	2019	-	-	Cost-based multi-armed bandits and Bayesian optimization	Single-label classification regression image classification audio classification graph matching	✓	✓	✓	×
AlphaD3M	2018	–	–	Reinforcement learning	Single-label classification regression	✓	×	✓	×
OBOE	2019	Python	Scikit-learn	Collaborative filtering	Single-label classification	✓	×	×	✓
PMF	2018	Python	Scikit-learn	Collaborative filtering & Bayesian optimization	Single-label classification	✓	×	✓	✓

Spark-based ML library. It attempted to reused cost-based query optimization techniques to prune the search space at the level of *logical learning plan* before transforming it into a *physical learning plan* to be executed.

Auto-Tuned Models (ATM) framework[20] has been introduced as a parallel framework for fast optimization of machine learning modeling pipelines [71]. In particular, this framework depends on parallel execution along multiple nodes with a shared model hub that stores the results out of these executions and tries to enhance the selection of other pipelines that can outperform the current chosen ones. The user can decide to use either of ATM's two searching methods, a hybrid Bayesian and multi-armed bandit optimization system, or a model recommendation system that works by exploiting the previous performance of modeling techniques on a variety of datasets.

TransmogrifAI[21] is one of the most recent modular tools written in Scala. It is built using workflows of feature preprocessors, and model selectors on top of Spark with minimal human involvement. It has the ability to reuse the selected work-flows. Currently, `TransmogrifAI` supports eight different binary classifiers and five regression algorithms. *MLBox*[22] is a Python-based AutoML framework for distributed preprocessing, optimization and prediction. MLBox supports model stacking where a new model is trained from the combined predictors of multiple previously trained models. It uses `hyperopt`[23], a distributed asynchronous hyper-parameter optimization library, in Python, to perform the hyper-parameter optimisation process.

Rafiki[24] has been introduced as a distributed framework which is based on the idea of using previous models that achieved high performance on the same tasks [72]. In this framework, regarding the data and parameter storage, the data uploaded by user to be trained is stored in a Hadoop Distributed File System (HDFS). During training, there is a database for each model storing the best version of parameters from hyper-parameter tuning process. This database is kept in memory as it is accessed and updated frequently. Once the hyper-parameter tuning process is finished, the database is dumped to the disk. The types of parameters to be tuned are either related to model architecture like number of Layers, and Kernel or related to the training algorithm itself like weight decay, and learning rate. All these parameters can be tuned using a random search or Bayesian optimization. Table 2 shows a summary of the main features of the distributed AutoML frameworks.

[20] https://github.com/HDI-Project/ATM.
[21] https://transmogrif.ai/.
[22] https://github.com/AxeldeRomblay/MLBox.
[23] https://github.com/hyperopt/hyperopt.
[24] https://github.com/nginyc/rafiki.

Table 2. Summary of the main features of distributed AutoML frameworks

	Release date	Core language	Optimization technique	Training framework	Meta-learning	User interface	Open source
MLBase	2013	Scala	Cost-based multi-armed bandits	Spark MLlib	×	×	×
ATM	2017	Python	Hybrid Bayesian, and multi-armed bandits optimization	Scikit-learn	✓	×	✓
MLBox	2017	Python	Distributed random search, Tree-Parzen estimators	Scikit-learn Keras	×	×	✓
Rafiki	2018	Python	Distributed random search, Bayesian optimization	TensorFlow Scikit-learn	×	✓	✓
TransmogrifAI	2018	Scala	Bayesian optimization, and random search	SparkML	×	×	✓

Table 3. Summary of the main features of the neural architecture search frameworks

	Release date	Open source	Optimization technique	Supported frameworks	Interface
Auto Keras	2018	✓	Network morphism	Keras	✓
Auto Net	2016	✓	SMAC	PyTorch	×
NNI	2019	✓	Random search different Bayesian optimizations annealing network morphism hyper-band naive evolution grid search	PyTorch, TensorFlow, Keras, Caffe2, CNTK, Chainer Theano	✓
enas	2018	✓	Reinforcement learning	Tensorflow	×
NAO	2018	✓	Gradient based optimization	Tensorflow, PyTorch	×
DARTS	2019	✓	Gradient based optimization	PyTorch	×
LEAF	2019	×	Evolutionary algorithms	–	×

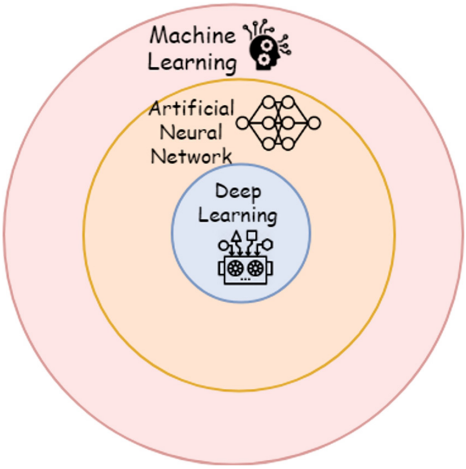

Fig. 5. The relationship between machine learning and deep learning.

3 Automated Deep Learning

3.1 Neural Architecture Search for Deep Learning

In general, deep learning techniques [73] represent a subset of machine learning methodologies that are based on artificial neural networks (ANN) which are mainly inspired by the neuron structure of the human brain (Fig. 5). It is described as *deep* because it has more than one layer of nonlinear feature transformation. Neural Architecture Search (NAS) is a fundamental step in automating the machine learning process and has been successfully used to design the model architecture for image and language tasks [74–78]. Broadly, NAS techniques falls into five main categories including *random search*, *reinforcement learning*, *gradient-based methods*, *evolutionary methods*, and *Bayesian optimization* (Fig. 6).

Random search is one of the most naive and simplest approaches for network architecture search. For example, Hoffer et al. [79] have presented an approach to find good network architecture using a random search combined with well-trained set of shared weights. Li and Talwalkar [80] proposed new network architecture search baselines that are based on a random search with early-stopping for hyper-parameter optimization. Results show that random search along with early-stopping achieves the state-of-the-art network architecture search results on two standard NAS bookmarkers which are PTB and CIFAR-10 datasets.

Reinforcement learning [81] is another approach that has been used to find the best network architecture. Zoph and Le [74] used a recurrent neural network (LSTM) with reinforcement to compose neural network architecture. More specifically, recurrent neural network is trained through a gradient based search algorithm called `REINFORCE` [82] to maximize the expected accuracy of the generated neural network architecture. Baker et al. [83] introduced a meta-modeling

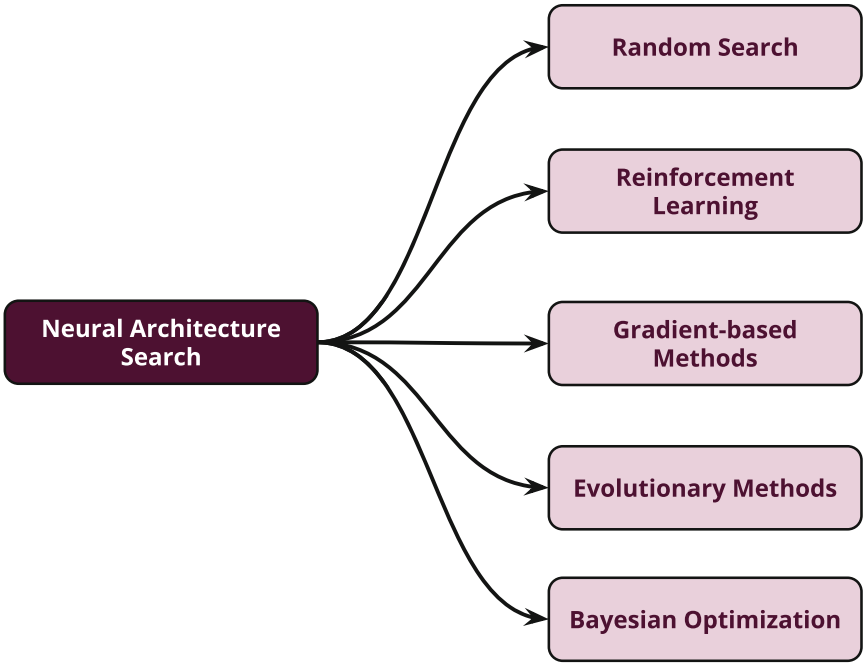

Fig. 6. A taxonomy for the Neural Network Architecture Search (NAS) techniques

algorithm called `MetaQNN` based on reinforcement learning to automatically generate the architecture of a convolutional neural network for a new task. The convolutional neural network layers are chosen sequentially by a learning agent that is trained using $Q-$learning with $\epsilon-$greedy exploration technique. Simply, the agent explores a finite search space of a set of architectures and iteratively figures out architecture designs with improved performance on the new task to be learned.

Gradient-based optimization is another common way for neural network architecture search. Liu et al. [84] proposed an approach based on continuous relaxation of the neural architecture allowing using a gradient descent for architecture search. Experiments showed that this approach excels in finding high-performance convolutional architectures for image classification tasks on CIFAR-10, and ImageNet datasets. Shin et al. [85] proposed a gradient-based optimization approach for learning the network architecture and parameters simultaneously. Ahmed and Torresani [86] used gradient based approach to learn network architecture. Experimental results on two different networks architecture ResNet and ResNeXt show that this approach yields to better accuracy and a significant reduction in the number of parameters.

Another direction for architecture search is *evolutionary algorithms* which are well suited for optimizing arbitrary structure. Miller et al. [87] considered an evolutionary algorithm to propose the architecture of the neural network and

network weights as well. Many evolutionary approaches based on genetic algorithms are used to optimize the neural networks architecture and weights [88–90] while others rely on hierarchical evolution [78]. Some recent approaches consider using the multi-objective evolutionary architecture search to optimize training time, complexity and performance [91,92] of the network. `LEAF` [93] is an evolutionary AutoML framework that optimizes hyper-parameters, network architecture and the size of the network. LEAF uses `CoDeepNEAT` [94] which is a powerful evolutionary algorithm based on `NEAT` [95]. LEAF achieved the state-of-the-art performance results on medical image classification and natural language analysis. For supervised learning tasks, evolutionary based approaches tend to outperform reinforcement learning approaches especially when the neural network architecture is very complex due to having millions of parameters to be tuned. For example, the best performance achieved on ImageNet and CIFAR-10 has been obtained using evolutionary techniques [96].

Bayesian optimization based on Gaussian processes has been used by Kandasamy et al. [97] and Swersky et al. [98] for tackling the neural architecture search problem. In addition, lots of work focused on using tree based models such as random forests and tree Parzen estimators [44] to effectively optimize the network architecture as well as its hyper-parameters [45,52,99]. Bayesian optimization may outperform evolutionary algorithms in some problems as well [100].

3.2 AutoDL Frameworks

Recently, some frameworks (e.g., `Auto-Keras` [101], and `Auto-Net` [99]) have been proposed with the aim of automatically finding neural network architectures that are competitive with architectures designed by human experts. However, the results so far are not significant. For example, *Auto-Keras* [101] is an open source efficient neural architecture search framework based on Bayesian optimization to guide the network morphism. In order to explore the search space efficiently, Auto-Keras uses a neural network kernel and tree structured acquisition function with iterative Bayesian optimization. First, a Gaussian process model is trained on the currently existing network architectures and their performance is recorded. Then, the next neural network architecture obtained by the acquisition function is generated and evaluated. Moreover, Auto-Keras runs in a parallel mode on both CPU and GPU.

Auto-Net [99] is an efficient neural architecture search framework based on SMAC optimization and built on top of `PyTorch`. The first version of Auto-Net is implemented within the Auto-sklearn in order to leverage some of the existing components of the machine learning pipeline in Auto-sklearn such as preprocessing. The first version of Auto Net only considers fully-connected feed-forward neural networks as they are applied on a large number of different datasets. Auto-net accesses deep learning techniques from Lasagne Python deep learning library [102]. Auto Net includes a number of algorithms for tuning the neural network weights including vanilla stochastic gradient descent, stochastic gradient descent with momentum, Adadelta [103], Adam [104], Nesterov momentum [105] and Adagrad [106].

Neural Network Intelligence (NNI)[25] is an open source toolkit by Microsoft that is used for tuning neural networks architecture and hyper-parameters in different environments including local machine, cloud and remote servers. NNI accelerates and simplifies the huge search space using built-in super-parameter selection algorithms including random search, naive evolutionary algorithms, simulated annealing, network morphism, grid search, hyper-band, and a bunch of Bayesian optimizations like SMAC [43], and BOHB [47]. NNI supports a large number of deep leaning frameworks including `PyTorch`, `TensorFlow`, `Keras`, `Caffe2`, `CNTK`, `Chainer` and `Theano`.

DEvol [26] is an open source framework for neural network architecture search that is based on genetic programming to evolve the number of layers, kernels, and filters, the activation function and dropout rate. DEvol uses parallel training in which multiple members of the population are evaluated across multiple GPU machines in order to accelerate the process of finding the most promising network.

enas [107] has been introduced as an open source framework for neural architecture search in Tensorflow based on reinforcement learning [74] where a controller of a recurrent neural network architecture is trained to search for optimal subgraphs from large computational graphs using policy gradient. Moreover, *enas* showed a large speed up in terms of GPU hours thanks to the sharing of parameters across child subgraphs during the search process.

NAO [108], and *Darts* [84] are open source frameworks for neural architecture search which propose a new continuous optimization algorithm that deals with the network architecture as a continuous space instead of the discretization followed by other approaches. In *NAO*, the search process starts by encoding an initial architecture to a continuous space. Then, a performance predictor based on gradient based optimization searches for a better architecture that is decoded at the end by a complementary algorithm to the encoder in order to map the continuous space found back into its architecture. On the other hand, *DARTS* learns new architectures with complex graph topologies from the rich continuous search space using a novel bilevel optimization algorithm. In addition, it can be applied to any specific architecture family without restrictions to any of the convolutional and recurrent networks only. Both frameworks showed a competitive performance using limited computational resources compared with other neural architecture search frameworks.

Evolutionary Neural AutoML for Deep Learning (LEAF) [93] is an AutoML framework that optimizes neural network architecture and hyper-parameters using the state-of-the-art evolutionary algorithm and distributed computing framework. LEAF uses `CoDeepNEAT` [94] for optimizing deep neural network architecture and hyper-parameters. LEAF consists of three main layers which are algorithm layers, system layer and problem-domain layer. LEAF evolves deep neural networks architecture and hyper-parameters in the algorithm layer. The system layer is responsible for training the deep neural networks in a parallel

[25] https://github.com/Microsoft/nni.
[26] https://github.com/joeddav/devol.

mode on a cloud environment such as Microsoft Azure[27], Google Cloud[28] and Amazon AWS[29], which is essential in the evaluation of the fitness of the neural networks evolved in the algorithm layer. More specifically, the algorithm layer sends the neural network architecture to the system layer. Then, the system layer sends the evaluation of the fineness of this network back to the algorithm layer. Both the algorithm layer and the system layer work together to support the problem-domain layers where the problems of hyper-parameter tuning of network architecture search are solved. Table 3 shows a summary of the main features of the state-of-the-art neural architecture search frameworks.

4 Open Challenges and Future Directions

Although in the last years, there has been increasing research efforts to tackle the challenges of the automated machine learning domain, however, there are still several open challenges and research directions that needs to be tackled to achieve the ultimate goals and vision of the AutoML domain. In this section, we highlight some of these challenges that need to be tackled to improve the state-of-the-art.

Scalability: In practice, a main limitation of the centralized frameworks for automating the solutions for the CASH problem (e.g., `Auto-Weka`, `Auto-Sklearn`) is that they are tightly coupled with a machine learning library (e.g., `Weka`, `scikit-learn`, `R`) that can only work on a *single* node which makes them not applicable in the case of large data volumes. In practice, as the scale of data produced daily is increasing continuously at an exponential scale, several distributed machine learning platforms have been recently introduced. Examples include `Spark MLib` [70], `Mahout`[30] and `SystemML` [109]. Although there have been some initial efforts for distributed automated framework for the CASH problem. However, the proposed distributed solutions are still simple and limited in their capabilities. More research efforts and novel solutions are required to tackle the challenge of automatically building and tuning machine learning models over massive datasets.

Optimization Techniques: In practice, different AutoML frameworks use different techniques for hyper-parameter optimization of the machine learning algorithms. For instance, `Auto-Weka` and `Auto-Sklearn` use the SMAC technique with cross-fold validation during the hyper-parameter configuration optimization and evaluation. On the other hand, `ML-Plan` uses the hierarchical task network with Monte Carlo Cross-Validation. Other tools, including `Recipe` [61] and `TPOT`, use genetic programming, and pareto optimization for generating candidate pipelines. In practice, it is difficult to find a clear winner or one-size-fits-all

[27] https://azure.microsoft.com/en-us/.
[28] https://cloud.google.com/.
[29] https://aws.amazon.com/.
[30] https://mahout.apache.org/.

technique. In other words, there is no single method that will be able to outperform all other techniques on the different datasets with their various characteristics, types of search spaces and metrics (e.g., time and accuracy). Thus, there is a crucial need to understand the Pros and Cons of these optimization techniques so that AutoML systems can automatically tune their hyper-parameter optimization techniques or their strategy for exploring and traversing the search space. Such decision automation should provide improved performance over picking and relying on a fixed strategy. Similarly, for the various introduced meta-learning techniques, there is no clear systematic process or evaluation metrics to quantitatively assess and compare the impact of these techniques on reducing the search space. Recently, some competitions and challenges[31,32] have been introduced and organized to address this issue such as the DARPA D3M Automatic Machine Learning competition [67].

Time Budget: A common important parameter for AutoML systems is the user *time budget* to wait before getting the recommended pipeline. Clearly, the bigger the time budget, the more the chance for the AutoML system to explore various options in the search space and the higher probability to get a better recommendation. However, the bigger time budget used, the longer waiting time and the higher computing resource consumption, which could be translated into a higher monetary bill in the case of using cloud-based resources. On the other hand, a small-time budget means a shorter waiting time but a lower chance to get the best recommendation. However, it should be noted that increasing the time budget from X to $2X$ does not necessarily lead to a big increase on the quality of the results of the recommended pipeline, if any at all. In many scenarios, this extra time budget can be used for exploring more of the unpromising branches in the search space or exploring branches that have very little gain, if any. For example, the accuracy of the returned models from running the `AutoSklearn` framework over the `Abalone` dataset[33] with time budgets of 4 h and 8 h are almost the same (25%). Thus, accurately estimating or determining the adequate time budget to optimize this trade-off is another challenging decision that can not be done by non-expert end users. Therefore, it is crucial to tackle such challenge by automatically predicting/recommending the adequate time budget for the modeling process. The `VDS` [67] framework provided a first attempt to tackle this challenge by proposing an interactive approach that relies on meta learning to provide a quick first model recommendation that can achieve a reasonable quality while conducting an offline optimization process and providing the user with a *stream* of models with better accuracy. However, more research efforts to tackle this challenge are still required.

Composability. Nowadays, several machine learning solutions (e.g., `Weka`, `Scikit-Learn`, `R`, `MLib`, `Mahout`) have become popular. However, these ML solutions significantly vary in their available techniques (e.g., learning algorithms,

[31] https://www.4paradigm.com/competition/nips2018.
[32] http://automl.chalearn.org/.
[33] https://www.openml.org/d/183.

preprocessors, and feature selectors) to support each phase of the machine learning pipeline. Clearly, the quality of the machine learning pipelines that can be produced by any of these platforms depends on the availability of several techniques/algorithms that can be utilized in each step of the pipeline. In particular, the more available techniques/algorithms in a machine learning platform, the higher the ability and probability of producing a well-performing machine learning pipeline. In practice, it is very challenging to have optimized implementations for all of the algorithms/techniques of the different steps of the machine learning pipeline available in a single package, or library. The `ML-Plan` framework [62] has been attempting to tackle the composability challenge on building machine learning pipelines. In particular, it integrates a superset of both `Weka` and `Scikit-Learn` algorithms to construct a full pipeline. The initial results of this approach have shown that the composable pipelines over `Weka` and `Scikit-Learn` do not significantly outperform the outcomes from `Auto-Weka` and `Auto-Sklearn` frameworks especially with big datasets and small time budgets. However, we believe that there are several reasons behind these results. First, combining the algorithms/techniques of more than one machine learning platform causes a dramatic increase in the search space. Thus, to tackle this challenge, there is a crucial need for a smart and efficient search algorithm that can effectively reduce the search space and focus on the promising branches. Using meta-learning approaches can be an effective solution to tackle this challenge. Second, combining services from more than one framework can involve a significant overhead for the data and message communications between the different frameworks. Therefore, there is a crucial need for a smart *cost-based* optimizer that can accurately estimate the gain and cost of each recommended composed pipeline and be able to choose the composable recommendations when they are able to achieve a clear performance gain. Third, the `ML-Plan` has been combining the services of two single node machine learning services (`Weka` and `Scikit-Learn`). We believe that the best gain of the composability mechanism will be achieved by combining the performance power of distributed systems (e.g., `MLib`) with the rich functionality of many centralized systems.

User Friendliness: In general, most of the current tools and framework can not be considered to be user friendly. They still need sophisticated technical skills to be deployed and used. Such challenge limits its usability and wide acceptance among layman users and domain experts (e.g., physicians, accountants) who commonly have limited technical skills. Providing an interactive and light-weight web interfaces for such framework can be one of the approaches to tackle these challenges.

Continuous Delivery Pipeline: Continuous delivery is defined as creating a repeatable, reliable and incrementally improving process for taking software from concept to customer. Integrating machine learning models into continuous delivery pipelines for productive use has not recently drawn much attention, because usually the data scientists push them directly into the production environment with all the drawbacks this approach may have, such as no proper unit and integration testing.

5 Conclusion

Machine learning has become one of the main engines of the current era. The production pipeline of a machine learning models passe through different phases and stages that require a wide knowledge of several available tools, and algorithms. However, as the scale of data produced daily is increasing continuously at an exponential scale, it has become essential to automate this process. In this chapter, we provided an overview of the state-of-the-art research effort in the domain of AutoML frameworks. We have also highlighted research directions and open challenges that need to be addressed in order to achieve the vision and goals of the AutoML process. We hope that our overview serves as a useful resource for the community, for both researchers and practitioners, to understand the challenges of the domain and provide useful insight for further advancing the state-of-the-art in several directions.

Acknowledgment. This work of Sherif Sakr is funded by the European Regional Development Funds via the Mobilitas Plus programme (grant MOBTT75). The work of Radwa Elshawi is funded by the European Regional Development Funds via the Mobilitas Plus programme (MOBJD341). The authors would like to thank Mohamed Maher for his comments.

References

1. Zomaya, A.Y., Sakr, S. (eds.): Handbook of Big Data Technologies. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49340-4
2. Sakr, S., Zomaya, A.Y. (eds.): Encyclopedia of Big Data Technologies. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-77525-8
3. Fitzgerald, B.: Software crisis 2.0. Computer **45**(4), 89–91 (2012)
4. Vafeiadis, T., Diamantaras, K.I., Sarigiannidis, G., Chatzisavvas, K.C.: A comparison of machine learning techniques for customer churn prediction. Simul. Modell. Pract. Theory **55**, 1–9 (2015)
5. Probst, P., Boulesteix, A.-L.: To tune or not to tune the number of trees in random forest. J. Mach. Learn. Res. **18**, 181–1 (2017)
6. Pedregosa, F., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
7. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA 2.0: automatic model selection and hyperparameter optimization in WEKA. J. Mach. Learn. Res. **18**(1), 826–830 (2017)
8. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In Proceedings of the 28th International Conference on Neural Information Processing Systems, NIPS 2015, vol. 2, pp. 2755–2763 (2015). MIT Press, Cambridge
9. Maher, M., Sakr, S.: SmartML: a meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms. In EDBT: 22nd International Conference on Extending Database Technology (2019)
10. Brazdil, P., Carrier, C.G., Soares, C., Vilalta, R.: Metalearning: Applications to Data Mining. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-73263-1

11. Vanschoren, J.: Meta-learning: a survey. CoRR, abs/1810.03548 (2018)
12. Bilalli, B., Abelló, A., Aluja-Banet, T.: On the predictive power of meta-features in OpenML. Int. J. Appl. Math. Comput. Sci. **27**(4), 697–712 (2017)
13. Bardenet, R., Brendel, M., Kégl, B., Sebag, M.: Collaborative hyperparameter tuning. In: International Conference on Machine Learning, pp. 199–207 (2013)
14. Soares, C., Brazdil, P.B., Kuba, P.: A meta-learning method to select the kernel width in support vector regression. Mach. Learn. **54**(3), 195–209 (2004)
15. Nisioti, E., Chatzidimitriou, K., Symeonidis, A.: Predicting hyperparameters from meta-features in binary classification problems. In: AutoML Workshop at ICML (2018)
16. Köpf, C., Iglezakis, I.: Combination of task description strategies and case base properties for meta-learning. In: Proceedings of the 2nd International Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-learning, pp. 65–76 (2002)
17. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
18. Giraud-Carrier, C.: Metalearning-a tutorial. In: Tutorial at the 7th International Conference on Machine Learning and Applications (ICMLA), San Diego, California, USA (2008)
19. Brazdil, P.B., Soares, C., Da Costa, J.P.: Ranking learning algorithms: using IBL and meta-learning on accuracy and time results. Mach. Learn. **50**(3), 251–277 (2003)
20. dos Santos, P.M., Ludermir, T.B., Prudencio, R.B.C.: Selection of time series forecasting models based on performance information. In: Fourth International Conference on Hybrid Intelligent Systems (HIS 2004), pp. 366–371. IEEE (2004)
21. Reif, M., Shafait, F., Goldstein, M., Breuel, T., Dengel, A.: Automatic classifier selection for non-experts. Pattern Anal. Appl. **17**(1), 83–96 (2014)
22. Guerra, S.B., Prudêncio, R.B.C., Ludermir, T.B.: Predicting the performance of learning algorithms using support vector machines as meta-regressors. In: Kůrková, V., Neruda, R., Koutník, J. (eds.) ICANN 2008. LNCS, vol. 5163, pp. 523–532. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87536-9_54
23. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)
24. Bengio, Y.: Deep learning of representations for unsupervised and transfer learning. In: Proceedings of ICML Workshop on Unsupervised and Transfer Learning, pp. 17–36 (2012)
25. Baxter, J.: Learning internal representations. Flinders University of South Australia (1995)
26. Caruana, R.: Learning many related tasks at the same time with backpropagation. In: Advances in Neural Information Processing Systems, pp. 657–664 (1995)
27. Razavian, A.S., Azizpour, H., Sullivan, J., Carlsson, S.: CNN features off-the-shelf: an astounding baseline for recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 806–813 (2014)
28. Donahue, J., et al.: Decaf: a deep convolutional activation feature for generic visual recognition. In: International Conference on Machine Learning, pp. 647–655 (2014)
29. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Advances in Neural Information Processing Systems, pp. 3320–3328 (2014)

30. Davis, L.: Handbook of genetic algorithms. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science. Springer, Boston (1991)
31. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: Boa: the Bayesian optimization algorithm. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, vol. 1, pp. 525–532. Morgan Kaufmann Publishers Inc. (1999)
32. Polak, E.: Optimization: Algorithms and Consistent Approximations, vol. 124. Springer, New York (2012). https://doi.org/10.1007/978-1-4612-0663-7
33. Montgomery, D.C.: Design and Analysis of Experiments. Wiley, New York (2017)
34. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
35. Zhilinskas, A.G.: Single-step Bayesian search method for an extremum of functions of a single variable. Cybern. Syst. Anal. **11**(1), 160–166 (1975)
36. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Global Optim. **13**(4), 455–492 (1998)
37. Snoek, J., et al.: Scalable Bayesian optimization using deep neural networks. In: International Conference on Machine Learning, pp. 2171–2180 (2015)
38. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems, pp. 2951–2959 (2012)
39. Dahl, G.E., Sainath, T.N., Hinton, G.E.: Improving deep neural networks for LVCSR using rectified linear units and dropout. In: IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 8609–8613. IEEE (2013)
40. Melis, G., Dyer, C., Blunsom, P.: On the state of the art of evaluation in neural language models. arXiv preprint arXiv:1707.05589 (2017)
41. Martinez-Cantin, R.: BayesOpt: a Bayesian optimization library for nonlinear optimization, experimental design and bandits. J. Mach. Learn. Res. **15**(1), 3735–3739 (2014)
42. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
43. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_40
44. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, pp. 2546–2554 (2011)
45. Bergstra, J., Yamins, D., Cox, D.D.: Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures (2013)
46. Eggensperger, K., et al.: Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In: NIPS Workshop on Bayesian Optimization in Theory and Practice, vol. 10, p. 3 (2013)
47. Falkner, S., Klein, A., Hutter, F.: BOHB: robust and efficient hyperparameter optimization at scale. arXiv preprint arXiv:1807.01774 (2018)
48. Sparks, E.R., Talwalkar, A., Haas, D., Franklin, M.J., Jordan, M.I., Kraska, T.: Automating model search for large scale machine learning. In: Proceedings of the Sixth ACM Symposium on Cloud Computing, pp. 368–380. ACM (2015)
49. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
50. Holland, J.H., et al.: Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT press, Cambridge (1992)

51. Fernández-Godino, M.G., Park, C., Kim, N.-H., Haftka, R.T.: Review of multi-fidelity models. arXiv preprint arXiv:1609.07196 (2016)
52. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
53. Jamieson, K.G., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: AISTATS, pp. 240–248 (2016)
54. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: a novel bandit-based approach to hyperparameter optimization. arXiv preprint arXiv:1603.06560 (2016)
55. de Sá, A.G.C., Freitas, A.A., Pappa, G.L.: Automated selection and configuration of multi-label classification algorithms with grammar-based genetic programming. In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D. (eds.) PPSN 2018. LNCS, vol. 11102, pp. 308–320. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99259-4_25
56. Tsoumakas, G., Katakis, I., Vlahavas, I.: Mining multi-label data. In: Maimon, O., Rokach, L. (eds.) Data Mining and Knowledge Discovery Handbook, pp. 667–685. Springer, Boston (2010). https://doi.org/10.1007/978-0-387-09823-4_34
57. Read, J., Reutemann, P., Pfahringer, B., Holmes, G.: MEKA: a multi-label/multi-target extension to WEKA. J. Mach. Learn. Res. **17**(1), 667–671 (2016)
58. Komer, B., Bergstra, J., Eliasmith, C.: Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In: ICML Workshop on AutoML, pp. 2825–2830 (2014)
59. Bergstra, J.., Yamins, D., Cox, D.D.: Hyperopt: a python library for optimizing the hyperparameters of machine learning algorithms. In: Proceedings of the 12th Python in Science Conference, pp. 13–20 (2013)
60. Olson, R.S., Moore, J.H.: TPOT:: a tree-based pipeline optimization tool for automating machine learning. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Proceedings of the Workshop on Automatic Machine Learning, volume 64 of Proceedings of Machine Learning Research, pp. 66–74, New York, USA, 24 Jun 2016. PMLR
61. de Sá, A.G.C., Pinto, W.J.G.S., Oliveira, L.O.V.B., Pappa, G.L.: RECIPE: a grammar-based framework for automatically evolving classification pipelines. In: McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P. (eds.) EuroGP 2017. LNCS, vol. 10196, pp. 246–261. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55696-3_16
62. Mohr, F., Wever, M., Hüllermeier, E.: ML-plan: automated machine learning via hierarchical planning. Mach. Learn. **107**(8–10), 1495–1515 (2018)
63. Chen, B., Wu, H., Mo, W., Chattopadhyay, I., Lipson, H.: Autostacker: a compositional evolutionary learning system. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, pp. 402–409. ACM, New York (2018)
64. Drori, I., et al.: AlphaD3M: machine learning pipeline synthesis. In: AutoML Workshop at ICML (2018)
65. Yang, C., Akimoto, Y., Kim, D.W., Udell, M.: OBOE: collaborative filtering for AutoML initialization. arXiv preprint arXiv:1808.03233 (2019)
66. Fusi, N., Sheth, R., Elibol, H.M.: Probabilistic matrix factorization for automated machine learning. arXiv preprint arXiv:1705.05355 (2017)
67. Shang, Z., et al.: Democratizing data science through interactive curation of ml pipelines. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) (2019)

68. Kraska, T., Talwalkar, A., Duchi, J.C., Griffith, R., Franklin, M.J., Jordan, M.I.: MLbase: a distributed machine-learning system. In: CIDR, vol. 1, pp. 1–2 (2013)
69. Wang, Q., et al.: ATMseer: increasing transparency and controllability in automated machine learning. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, p. 681. ACM (2019)
70. Meng, X., et al.: MLlib: machine learning in apache spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
71. Swearingen, T., Drevo, W., Cyphers, B., Cuesta-Infante, A., Ross, A., Veeramachaneni, K.: ATM: a distributed, collaborative, scalable system for automated machine learning, pp. 151–162, December 2017
72. Wei Wang, et al.: Rafiki: machine learning as an analytics service system. CoRR, abs/1804.06087 (2018)
73. Bengio, Y., et al.: Learning deep architectures for AI. Foundations Trends® Mach. Learn. **2**(1), 1–127 (2009)
74. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
75. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8697–8710 (2018)
76. Cai, H., Chen, T., Zhang, W., Yu, Y., Wang, J.: Efficient architecture search by network transformation. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
77. Liu, C., et al.: Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 19–34 (2018)
78. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436 (2017)
79. Hoffer, E., Hubara, I., Soudry, D.: Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS 2017, USA, pp. 1729–1739. Curran Associates Inc. (2017)
80. Li, L., Talwalkar, A.: Random search and reproducibility for neural architecture search (2019)
81. Sutton, R.S., Barto, A.G., et al.: Introduction to Reinforcement Learning, vol. 135. MIT Press, Cambridge (1998)
82. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**(3–4), 229–256 (1992)
83. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167 (2016)
84. Liu, H., Simonyan, K., Yang, Y.: Darts: differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
85. Shin, R., Packer, C., Song, D.: Differentiable neural network architecture search (2018)
86. Ahmed, K., Torresani, L.: MaskConnect: connectivity learning by gradient descent. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11209, pp. 362–378. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01228-1_22
87. Miller, G.F., Todd, P.M., Hegde, S.U.: Designing neural networks using genetic algorithms. In: ICGA, vol. 89, pages 379–384 (1989)
88. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)

89. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009)
90. Angeline, P.J., Saunders, G.M., Pollack, J.B.: An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. Neural Netw. **5**(1), 54–65 (1994)
91. Lu, Z., et al.: NSGA-Net: a multi-objective genetic algorithm for neural architecture search. arXiv preprint arXiv:1810.03522 (2018)
92. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via Lamarckian evolution (2018)
93. Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K., Miikkulainen, R.: Evolutionary neural AutoML for deep learning (2019)
94. Miikkulainen, R., et al.: Evolving deep neural networks. In: Artificial Intelligence in the Age of Neural Networks and Brain Computing, pp. 293–312. Elsevier (2019)
95. Real, E., et al.: Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 2902–2911. JMLR. org (2017)
96. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. arXiv preprint arXiv:1802.01548 (2018)
97. Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., Xing, E.: Neural architecture search with Bayesian optimisation and optimal transport (2018)
98. Swersky, K., Duvenaud, D., Snoek, J., Hutter, F., Osborne, M.A.: Raiders of the lost architecture: kernels for Bayesian optimization in conditional parameter spaces. arXiv preprint arXiv:1409.4011 (2014)
99. Mendoza, H., Klein, A., Feurer, M., Springenberg, J.T., Hutter, F.: Towards automatically-tuned neural networks. In: Workshop on Automatic Machine Learning, pp. 58–65 (2016)
100. Klein, A., Christiansen, E., Murphy, K., Hutter, F.: Towards reproducible neural architecture and hyperparameter search (2018)
101. Jin, H., Song, Q., Hu, X.: Efficient neural architecture search with network morphism. CoRR, abs/1806.10282 (2018)
102. Dieleman, S., et al.: Lasagne: first release, August 2015 (2016), 7878. https://doi.org/10.5281/zenodo
103. Zeiler, M.D.: ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)
104. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
105. Nesterov, Y.: A method of solving a convex programming problem with convergence rate $o(1/k^2)$ $o(1/k2)$. Sov. Math. Dokl. **27**
106. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)
107. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)
108. Luo, R., Tian, F., Qin, T., Chen, E., Liu, T.-Y.: Neural architecture optimization. In: Advances in Neural Information Processing Systems, pp. 7816–7827 (2018)
109. Boehm, M., et al.: SystemML: declarative machine learning on spark. Proc. VLDB Endowment **9**(13), 1425–1436 (2016)