AutoML: An Automated Machine Learning Method

Wang Xin
HPCA-AI Group

Abstract

Machine learning techniques have deeply rooted in our everyday life. However, since it is knowledge and labor-intensive to pursue good learning performance, humans are heavily involved in every aspect of machine learning. To make machine learning techniques easier to apply and reduce the demand for experienced human experts, automated machine learning (AutoML) has emerged as a hot topic with both industrial and academic interest. This article provides the latest research on AutoML. Introduced the development history and latest development of AutoML, and summarized the Workflow and Framework of AutoML. On this basis, we give the definition of AutoML.

1 Introduction

Ensemble[1] and Stacking[2] technologies in machine learning and deep learning have achieved great success in many areas, such as: image recognition, recommendation systems, etc. Manually designing machine learning algorithms and DNN models requires a lot of debugging to get high-performance models, taking up a lot of energy and time for developers, and bringing various problems. As a technology that attempts to solve the above problems, AutoML is becoming a hot trend in the field of machine learning.

1.1 Application background

With the development of machine learning technology, machine learning has now succeeded in a wide range of applications. In the past few years, the demand for machine learning systems has soared. Designing a machine learning model that effectively meets the needs of an application requires more and more professionals to spend energy and invest a lot of time in development. This means higher costs and time.

In the traditional mode, the application of AI first needs to build a team of experienced data scientists, and it takes a lot of money to maintain the operation of the entire team. Second, even having a good team often requires more experience to decide which model is best for the current application problem, not knowledge.

From a theoretical perspective, AI technology supported by machine learning is dedicated to replacing part of the work of people. If we go deeper, many of the work in the design process of machine learning

algorithms, such as feature engineering, parameter tuning, and model selection, can also be implemented by algorithms. In other words, in the process of building a machine learning model, developers can use more computing power instead of human resources through Bayesian optimization, NAS, and Genetic programming.

As a result, the Auto Machine Learning system came into being, and AutoML tends to automate as many steps as possible in the ML pipeline, keeping the model's performance with minimal manpower. These systems can be used by non-machine learning experts[3] and achieve good results.

1.2 AutoML's History

The idea of AutoML emerged long before it was formally proposed as a new concept in Auto-weka[4] and ICML-2014s AutoML workshop. Besides, automation of different learning processes in AutoML, i.e., feature engineering, model selection and algorithm selection, as shown in Figure 1, has also been individually visited by many researchers in elds of machine learning, data mining and articial intelligence. However, AutoML only becomes practical and a big focus recently, due to the big data, the increasing computation of modern computers, and of course, the great demand of machine learning application.

Fig. 1. To use machine learning techniques and obtain good performance, humans usually need to be involved in data collection, feature engineering, model and algorithm selection. This picture shows a typical pipeline of machine learning application, and how AutoML can get involved in the pipeline and minimize participation of humans.

On the feature level, feature selection is a traditional topic in machine learning, it tries to automatically remove unnecessary features making learning models simpler and more interpretable[5]. Many methods have been proposed and Lass[6] is a landmark of them. Dimension reduction methods, such as PCA[7] and LDA[8], have also been popularly used to deal with high-dimensional features. These methods tries to nd a better representation for the input data. However, the size of the reduced dimension needs human specications. During 1990s, many attempts were done to automatically construct better features based on genetic algorithms[9, 10, 11]. However, only recently did automated learning feature representations become possible for some structured data, such as CNN for images and RNN for sequential data[12].

Model selection is the task of selecting a proper model from a set of candidate models. Theoretical foundation, i.e., statistical learning theory[13, 14], is rstly paved for model selection, which shows how a model can generalize from the given to unseen data. For a specic model, its performance is most inuenced by its hyper-parameters. The grid search is the most commonly used method to determine proper values for hyper-parameters. However, it can only deal with a very small number of hyperparameters. Later, optimization based methods, such as derivative-free optimization[15, 16] and gradient-based optimization[17], have been

considered for nding hyperparameters. These methods have strong assumptions on the optimization model, which limits their application in practice. These motivate many tailormade methods for the selection of various machine learning models, such as the kernel selection for SVM[18], learning K for K-means[19] and genetic programming for neural networks architectures[20, 21]. Finally, as a single model may not be strong enough, conguring an ensemble of models is considered in [22].

Finally, once the model is xed, optimization algorithms are required to nd good parameters. Algorithm selection originally dates back to 1970s, where many researcher tried to design better algorithms for hard combinatorial problems[23, 23]. As these problems are mostly NP-hard, it is not possible to nd their optimal solutions. Each algorithm has its own heuristics and strength for solve a certain type of problems. The algorithm selection attempts to identify the best algorithm for the given combinatorial problem. In the past, convex and simple models, such as logistic regression and SVM, are widely used in machine learning, and the data sets are also not large. At that time, algorithm selection is not an important problem, and it is easy to nd good parameters for a given model[24]. Recently, as the data gets larger, and complex models, e.g., deep networks, become popular, the optimization algorithms can have an important impact on the generalization performance[25, 26, 27]. Beside, more hyper-parameters are also involved in these algorithms, e.g., Adagrad[28] has more hyper-parameters than plain SGD[25], which make them harder to tune.

1.3 Research Status

At present, AutoML has become the most active research area in machine learning research. Since deep learning technology is currently the most widely used machine learning technology, deep learning based NAS is the most mainstream sub-area in AutoML research. There are many new opportunities and problems in AutoML that are not visited in the above history.

Lots of papers focusing on AutoML appear on various conferences and journals, such as ICML, NIPS, KDD, AAAI, IJCAI, JMLR and some of their open-source projects are extremely popular on GitHub (Table 15); many workshops are organized, such as AutoML workshop at ICML from 2014 to 2018; some competitions such as, AutoML Challenge at PAKDD[29] and NIPS[30], are hold as well.

Table 15

There are many AutoML libraries available, the first of which is Auto-WEKA[4], which was first released in 2013 and can automatically select models and hyperparameters. Other notable AutoML libraries include auto-sklearn[3, 31] (which extends Auto-WEKA to the python environment), H2O AutoML and TPOT[32]. AutoML.org (formerly known as ML4AAD, Machine Learning for Automated Algorithm Design) has been organizing AutoML seminars since 2014 at the ICML Machine Learning Conference.

In the industry, there are also a large number of NAS-based AutoML projects:

1) Google Cloud AutoML

In January 2018, Google released Cloud AutoML[33]. Later, Google opened up a lightweight framework based on TensorFlow, AdaNet[34], which uses a small amount of expert intervention to automatically learn high-quality models. According to reports, AdaNet is built on Google's recent intensive learning and evolution-based AutoML, which is fast and flexible and provides a learning guarantee. AdaNet adaptively grows the number of integrated neural networks. In each iteration, AdaNet measures the integration loss of each candidate neural network and then selects the best neural architecture into the next iteration.

2) Microsoft Neural Network Intelligence

Neural Network Intelligence is a toolkit to help users run automated machine learning (AutoML) experiments. The tool dispatches and runs trial jobs generated by tuning algorithms to search the best neural architecture and/or hyper-parameters in different environments like local machine, remote servers and cloud.

3) Baidu Easy DL

EasyDL is a graphical deep learning system launched by Baidu based on the PaddlePaddle framework. At present, EasyDL adopts the Auto Model Search algorithm, that is, the system will initiate multiple trainings of different model structures and hyperparameters at the same time, and use the corresponding algorithm to filter the final model to ensure better model effect. Auto Model Search is similar in function to AutoDL described later, but Baidu's AutoDL is a neural architecture search method that focuses on building neural networks from scratch using reinforcement learning.

EasyDL uses a large number of migration learning technologies. Various basic models are pre-trained on Baidu's large-scale datasets, and the knowledge learned from them is applied to the small-scale training dataset submitted by users. Model effects and fast model training.

4) Alibaba Cloud PAI

Alibaba Cloud Machine Learning and Deep Learning Platform PAI (Platform of Artificial Intelligence) is a machine learning platform built on the Alibaba Cloud MaxCompute (formerly ODPS) computing platform, which integrates data processing, modeling, offline prediction and online prediction. Alibaba Cloud Machine Learning encapsulates the mature algorithms in the Alibaba Group and provides a simpler operating experience for machine learning users. AutoML technology is currently used.

2 Related Works

In AutoML, the three most important parts are Auto Feature Engineering, Model selection, and NAS. In this section, we will focus on the important results of these core parts of AutoML.

2.1 Auto Feature Engineering

The quality of features, perhaps, is the most important perspective for the performance of subsequent learning models[35, 13]. Such importance is further veried by the success of deep learning models, which can directly learn a representation of features from the original data[12]. The problem of AutoML for feature engineering is to automatically construct features from the data so that subsequent learning tools can have good performance. The above goal can be further divided into two sub-problems, i.e., creating features from the data and enhance features discriminative ability.

One of the most representative works in automatic feature construction is ExploreKit[36]. It aims to generate new features to improve the performance of the learning tools. In this setting, a candidate configuration is a set of generated features. Figure 21 shows the system architecture of ExploreKit. Its main body of the is an iterative process, where each iteration comprises three steps, candidate feature generation, candidate feature ranking, candidate feature evaluation and selection. It is by instinct a greedy search strategy since the search space is extremely large. Additionally, metalearning techniques are employed in the ranking step to fast estimate the usefulness of candidate features and accelerate the subsequent evaluation step.

Fig. 21. The system architecture of ExploreKit (the image is from [36]).

The optimizer of ExploreKit employs a greed rule-based strategy to explore the search space. At the candidate feature generation step, new features are constructed by applying operators on features that are already selected. Employed operators include: 1) unary ones (e.g., discretization, normalization), 2) binary ones (e.g., +, , Œ, œ), and 3) higherorder ones (e.g., GroupByThenMax, GroupByThenAvg). A predetermined enumerating procedure is invoked to apply these operators on all selected features that are applicable to generate candidates. In order to limit the size of candidate feature set, generated features will not be reused to further generate new candidates.

Since ExploreKit generates candidates exhaustively, evaluating all these features may be computational intractable. To address this issue, ahead of the evaluation and selection step, ExploreKit uses meta-learning roughly rank all candidate features. At this step, a ranking classifier, trained with historical knowledge on feature engineering, is used to fast identify promising candidates. In the evaluation step that follows, features predicted more useful will be considered first.

Finally, ExploreKit conducts more expensive and accurate evaluations on the candidate features. Error reduction on the validation set is used as the metric for feature importance. Candidate features are evaluated successively according to their ranking, and selected if their usefulness surpass a threshold. This procedure terminates if enough improvement is achieved. The selected features will be used to generate new candidates in the following iterations.

TABLE 13 Experimental results of ExploreKit[36] on ten representative data sets. This table reports the percentage of error reduction brought by the features generated with ExploreKit. Three base machine learning models are considered, namely decision tree (DT), support vector machine (SVM), and random forest (RF).

Table 13 presents the improvement ExploreKit achieves on ten representative data sets. The performance of the generated features depends on the data set and the base classifier, and some encouraging results have been observed.

2.2 AutoML (Based on traditional ML model)

Once features have been obtained, we need to nd a model to predict the labels. Models selection contains two components, i.e., picking up some classiers and setting their corresponding hyper-parameters. In this AutoML setup , the task is to automatically select classiers and set their hyperparameters so that good learning performance can be obtained.

An representative example of automated model selection approach is Auto-sklearn[3], which is built on Scikit-Learn[37] package. Some widely used classiers and their hyperparameters are listed in Table 4. In Auto-sklearn, model selection is formulated as a CASH problem (Example 1), which aims to minimize the validation loss with respect to the model as well as its hyper-parameters and parameters.

Example 1 (CASH Problem[38, 39]]). Let F = F1, ... ,FR be a set of learning models, and each model has hyper-parameter j with domain j, Dtrain = (x1,y1), ... , (xn,yn) be a training set which is split into K cross-validation folds D1train, ... , DK train and D1valid, ... , DKvalid with Ditrain Divalid = Dtrain for i = 1, ... , K. Then, the Combined Algorithm Selection and Hyper-parameter (CASH) optimization problem is dened as ABCD13.

where L(Fj(wj; j), Ditrain, Divalid) denotes the loss that Fj achieves on Divalid with parameter wj, hyper-parameter j and training dataDitrain.

As discussed in Section 3.2.2, in the model selection problem, a candidate configuration comprises a classifier in Scikit-Learn and its hyper-parameters, and the search space is spanned by configurations of this kind. However, (2) is very hard to optimize. First, since there is no explicit expression for the objective,

we do not know its properties, e.g., the degree of smoothness, which can be very helpful for traditional optimization problems. In addition, the decision variables, i.e., and Fj , may not even be continuous, e.g., for kNN includes the number of nearest neighbors that is a discrete variable. Finally, in order to estimate the validation loss, we need to train the model Fj and update its parameter wj , which is usually very expensive.

In [4], sequential model-based algorithm configuration (SMAC)[39], a tree-based Bayesian optimization method, was used as the optimizer to solve (2). Then, for the evaluator, the basic method, i.e., direct evaluation was used. Besides, meta-learning was employed as the experienced technique to get better initialization (i.e., warm-starting configuration generation in Section 6.1.3). Finally, rather than discarding models searched in the configuration space, Auto-sklearn stores them and to use a post-processing method to construct an ensemble of them. This automatic ensemble construction makes the final result more robust against overfitting.

TABLE 10 Illustration of how examples in Table 1 fall into the proposed framework in Figure 6. The naive means there is no special design in the evaluator, the evaluation is directly done by optimizing parameters of learning tools on the training data.

Table 11 (from Table 3 in [3]) shows the performance of Auto-sklearn (denoted as ASK), where classifiers C1-15 denotes: Adaboost, Bernoulli naive Bayes, decision tree, extreme random trees, Gaussian naive Bayes, gradient boosting, KNN, LDA, linear SVM, kernel SVM, multinomial naive Bayes, passive aggressive, QDA, random forest, and linear classifier, respectively. As we can see, Auto-sklearn consistently finds the best configuration in its search space, which demonstrates the success of AutoML in model selection.

TABLE 11 Performance comparison between Auto-sklearn (ASK) and other 15 classiers (C1-C15) within its conguration space. OpenML data sets (their IDs are in the rst column) are used and median balanced error (BER) are reported. BER of Auto-sklearn is obtained based on an ensemble of top classiers it searched in conguration space; and BERs of C1-C15 are obtained by ne-tuning each classier separately. Bold number indicates the best BER as well not those statistical signicant ones.

2.3 Neural Architecture Search

As mentioned in Section 1, since the success of AlexNet on image classification of ImageNet data set [40], the design of new neural architectures has become the main means to get better predicting performance in the deep learning domain. This raises the research interests in automatic searching network architectures for given tasks [41, 42, 43].

Taking CNN as an example, the design choices for each convolution layer are listed in Figure 11. A configuration (architecture design) contains designs of all convolution layers in a CNN, which leads to a very

large search space. The common approach is to build the architecture layer-bylayer. As shown in Figure 20 one configuration in the NAS problem is a sequence of design decisions. However, in a CNN architecture, the effect of lower layer design depends on those of higher ones[44, 45]. This makes the final performance of the whole architecture a delayed reward. Motived by such facts, RL is employed in NAS to search for a optimal sequence of design decisions[43, 46]]. Besides, the direct evaluation is used in these works as the evaluator.

Fig. 11. Some common design choices for one convolutional layer in a CNN.

Fig. 20. Generating hyper-parameters for one convolutional layer using RNN (the image is from [47])

As RL is slow to converge, to make the search faster, transfer learning, which is firstly used to cut the search space into several blocks, was developed in [48, 46]; then, a parameter sharing scheming is proposed in[49] to further narrow down the search space; and greedy search has also been considered as a replacement for RL[50, 51].

3 The Worlflow of AutoML

In this section, we'll cover the overall workflow of AutoML.

3.1 General WorkFlow

Fig. n.

The traditional machine learning WorkFlow is shown in Figure n, including Data collection and processing, feature engineering, Model Selection, Optimization and Evaluation Model.

Congurations are tuned by human is shown in Figure n+1. Once a learning problem is dened, we need to nd some learning tools to solve it. These tools, which are placed in the right part of Figure n+1, can target at different parts of the pipeline, i.e., feature, model or optimization in Figure 1. To obtain a good learning performance, we will try to set a conguration using our personal experience or intuition about the underneath data and tools. Then, based on the feedback about how the learning tools perform, we will adjust the conguration wishing the performance can be improved. Such a trial-and-error process terminates once a desired performance is achieved or the computational budgets are run out.

Fig. n+1. The process of congurations tuned by humans.

In AutoML, most of these steps will be replaced by programs. As shown in Figure n+2, AutoML program

will minimize participation of humans.

Fig. n+2. AutoML

3.2 Steps of AutoML (Based on general ML)

The overall process of AutoML based on traditional machine learning is similar to ML. Includes feature engineering, model selection, and Optimization Algorithm Selection.

3.2.1 Feature Engineering

Feature engineering is the most important part of machine learning and the first step in all machine learning processes. The same is true for AutoML based on traditional machine learning techniques.

1) Feature Enhancing Methods

In many cases, the original features from the data may not be good enough, e.g., their dimensionality may be too high or samples may not be discriminable in the feature space[12]. Consequently, we may want to perform some post-processing on these features to improve the learning performance. Fortunately, while human assistance is still required, there are common methods and principled ways to enhance features. They are listed as follows:

Dimension reduction: It is the process of reducing the number of random variables under consideration by obtaining a set of principal variables.

Feature generation: Unexplored interactions among original features, once discovered, may signicantly improve the learning performance.

Feature encoding: The last category is feature encoding, which re-interprets original features based on some dictionaries learned from the data.

2) Search Space

There are two types of search space for above feature enhancing tools. The rst one is made up by hyper-parameters of these tools, and conguration exactly refers to these hyper-parameters[3, 31, 4]. It covers dimension reduction and feature encoding methods. For example, we need to determine the dimension of features when employing PCA, and the level of sparsity if sparse coding is used.

The second type of search space contains feature to be generated and selected. It commonly considered in feature generation, e.g., [50, 36, 52, 53, 54, 55, 56]. Basically, the search space is spanned by operations

on original features. One example of new feature generated from plus, minus and times operations is shown in Figure 8. For these methods, a conguration is a newly generated feature in the search space.

Fig. 8. An example of a newly generated feature (A Œ B (C + D)), which is based on plus (+), minus () and times (Œ) operations.

3.2.2 Model Selection

Once features have been obtained, we need to find a model to predict the labels. Models selection contains two components, i.e., picking up some classifiers and setting their corresponding hyper-parameters. In this AutoML setup , the task is to automatically select classifiers and set their hyperparameters so that good learning performance can be obtained.

1) Classication Tools

Many classication tools have been proposed in the literature, e.g., tree classiers, linear classiers, kernel machines and, more recently, deep networks. Each classier has its own strength and weakness in modeling underneath data[35, 57]. Some out-of-the-box classiers implemented in scikit-learn are listed in Table 4. As can be seen, different hyper-parameters are associated with each classier. Traditionally, the choice among different classiers and their hyper-parameters are usually determined by human with his/her experience in a trial-and-error manner.

TABLE 4 Example classiers in Scikit-Learn and their hyper-parameters. Generally, hyper-parameters can be (a) discrete, e.g., number of neighbors in kNN, or (b) continuous. e.g., the value of penalty in logistic regression.

2) Search Space

In the context of model selection, the candidate classiers and their corresponding hyper-parameters make up the search space. Figure 9 shows a hierarchical structure that is commonly used to represent the search space[3, 31, 4, 58, 59, 60]. The rationale behind this structure is that we need to determine the hyper-parameters only if the corresponding classier is considered.

Fig. 9. An illustration of the search space for model selection, where KNN, linear SVM and Bernoulli naive Bayes classiers are considered. Hyper-parameters are derived based on Scikit-Learn, c: indicates that the hyper-parameter is continuous while d: means that it is discrete. In this gure, a conguration is made up by the selection of KNN classier and its values in corresponding hyper-parameters. KNNSVMBernoulli Scikit-Learn"c" "d" KNN

3.2.3 Optimization Algorithm Selection

1) Optimization Algorithms

The last and the most time consuming step of machine learning is the model training, where optimization is usually involved. For classical learning models, optimization is not a concern, since they usually employs convex loss functions and their performance obtained from various optimization algorithms are nearly the same[25]. Hence, efciency is the main focus on the choice of optimization algorithm.

However, as the learning tools get increasingly more complex, e.g. from SVM to deep networks, optimization is not only the main consumer of computational budgets but also has a great impact on the learning performance[61, 26]. Consequently, the goal of algorithm selection is to automatically nd an optimization algorithm so that efciency and performance can be balanced.

2) Search Space

Traditionally, both the choices of optimization algorithms and their hyper-parameters are made by humans based on their understanding of the learning tools and observations of the training data. To automate algorithm selection, the search space is determined by configurations of optimization algorithms, which contains the choice of optimization algorithms and the values of their hyper-parameters, e.g, [62, 23, 63, 64]. There is also naturally a hierarchy in such search space, which is similar to that shown in Figure 9, as hyper-parameters of an algorithm will be considered only when the corresponding algorithm is selected.

3.3 The Workflow of NAS

The targets of NAS is searching good deep network architectures that suit the learning problem. There are three main reasons why we discuss it in parallel with the full scope. First, NAS itself is currently an extremely hot research topic under which many papers have been published, i.e., [41, 61, 65, 66, 67, 68, 69, 70, 71, 72] and etc. The second reason is that the application domain for deep networks is relative clear, i.e., the domain of learning from low-semantic-level data such as image pixels. Finally, since the application domain is clear, domainspecic network architectures can fulll the learning purpose, where feature engineering and model selection are both done by NAS.

Figure 2n. AutoML NAS Workflow.

In General, the workflow of NAS can be described by Figure 2n, and we will cover the working process of NAS in section 4.

4 Neural Structure Search

Deep learning can automatically learn useful features, break away from the dependence on feature engineering, and achieve results beyond other algorithms in tasks such as images and speech. This success is largely due to the emergence of new neural network structures such as ResNet, Inception, and DenseNet. However, designing a high-performance neural network requires a lot of expertise and trial and error, and the cost is extremely high, which limits the application of neural networks on many problems. Neural Structure Search (NAS) is a technology that automatically designs neural networks. It can automatically design high-performance network structures based on sample sets. It can even meet the standards of human experts in some tasks, and even discover some humans. The network structure that has not been proposed can effectively reduce the use and implementation cost of the neural network. The principle of NAS is to give a set of candidate neural network structures called search spaces, and use some strategy to search for the optimal network structure. The strengths and weaknesses of neural network structure, that is, performance is measured by certain indicators such as accuracy and speed, called performance evaluation.

4.1 Search Space

The search space defines the type of neural network that the NAS algorithm can search for, and also defines how the neural network structure is described. The computations implemented by the neural network can be abstracted into a directed acyclic graph (DAG) without isolated nodes. The nodes of the graph represent the layers of the neural network, and the edges represent the flow of data. Each node receives data from its precursor node (with edge injection), and after calculation, outputs the data to subsequent nodes (with edge injection). In theory, as long as it is a DAG without isolated nodes, it is a legal neural network structure. According to different scales, the structural definition of a neural network contains the following levels of information:

4.1.1 Network Topology

How many layers are there in the network, and the connection relationships between these layers. From simple graph structure to arbitrary DAG, it also reflects the development of the entire neural network structure. The simplest neural network is a linear chain structure, where each node of the corresponding graph has at most one precursor and one subsequent, similar to the linked list in the data structure. Early fully connected neural networks, convolutional neural networks, are all such topologies. Nodes in Inception, ResNet, and DenseNet allow multiple predecessors, multiple subsequents, to form a multi-branch, cross-layer connection structure, which are more complex diagrams. These typical topologies are shown below.

When describing the topology of the network, the predecessor node is generally defined, that is, the predecessor node of each node is defined, and once the information is determined, the network topology is determined.

4.1.2 Type of each layer

The type of the middle layer is optional except that the first layer must be the input layer and the last layer must be the output. They represent the different types of operations, ie layer types. Typical connections include full join, convolution, deconvolution, hole convolution, pooling, and activation functions. However, the combined use of these layers generally meets certain rules.

4.1.3 Hyperparameters of each layer

The hyperparameters of the convolutional layer have the number of convolution kernels, the number of channels of the convolution kernel, the height, the width, the step size in the horizontal direction, and the step size in the vertical direction. The hyperparameters of the fully connected layer have the number of neurons. The hyperparameters of the activation function layer have the type of the activation function, the parameters of the function (if any), and so on. The super parameters of various typical layers are shown in the table below.

If there is only one precursor node of a node, the output value of the previous driver node is directly used as the input of the node. If there are multiple predecessor nodes, you need to summarize the values of the predecessor nodes and input them into the node. There are two strategies: add and splicing. The typical representative of the former is ResNet[73], and the typical representative of the latter is DenseNet[74]. Since the number of layers of the neural network is not fixed, the number of hyperparameters of each layer is not fixed, so the parameters describing the network structure are variable length.

In order to improve search efficiency, sometimes the search space is limited or simplified. In some NAS implementations, the network is divided into basic units (cells, or blocks), and the stacking of these units forms a more complex network. The basic unit consists of multiple nodes (layers of neural networks) that repeat multiple times throughout the network but have different weighting parameters. Another approach is to define the overall topology of the neural network, drawing on the experience of human design neural networks. Although these practices reduce the amount of computation of the NAS algorithm, they also limit the types of neural networks that the algorithm can look for.

Since the parameters describing the structure of the neural network contain discrete data (such as the definition of the topology, the type of the layer, and the discrete hyperparameters within the layer),

the network structure search is a discrete optimization problem. The number of parameters defining the structure is generally large, so it is a high-dimensional optimization problem. In addition, for this problem, the algorithm does not know the specific form of the optimization objective function (the functional relationship between each network structure and the performance of the network), and therefore belongs to the black box optimization problem. These features pose a huge challenge for NAS.

4.2 Search Approaches

The search strategy defines how to find the optimal network structure, usually an iterative optimization process, which is essentially a hyperparameter optimization problem. Currently known search methods include random search, Bayesian optimization, genetic algorithm, reinforcement learning, and gradient-based algorithms. Among them, reinforcement learning, genetic learning, and gradient-based optimization are the current mainstream algorithms and the focus of this chapter.

4.2.1 Simple search

Simple search is a naive search approach, they make no assumptions about the search space. Each conguration in the search space can be evaluated independently. Grid search and random search are two common approaches.

Grid search (brute-force): it is the most traditional way of hyper-parameters tuning. To get the optimal hyperparameter setting, grid search have to enumerate every possibleconfigurationsinthesearchspace.Discretization is necessary when the search space is continuous.

Random search: it randomly samples congurations in the search space. Random search empirically performs better than brute-force grid search[75]. As shown in Figure 12, random search can explore more on important dimensions than grid search.

Simple search approaches gather the feedbacks from the evaluator merely to keep track of the good congurations. Because simple search does not exploit the knowledge gained from the past evaluations, it is usually inefcient. However, due to its simplicity, it is still popularly used in AutoML.

TABLE 6 Comparison of various techniques for the optimizer. Multi-step means the configuration can be made up by a several decision steps.

4.2.2 Heuristic Search

Heuristic search methods are often inspired by biologic behaviors and phenomenons. They are widely used to solve optimization problems that are non-convex, non-smooth, or even non-continuous. The majority of them are populationbased optimization methods, and differences among them are how to generate and select populations. The framework of heuristic search is shown in Figure 13. The initialization step generates the rst population (a bunch of congurations in AutoML). At each iteration, a new population is generated based on the last one, and the tness (performances) of the individuals are evaluated. The core idea of heuristic search is how to update the population.

Fig. 13. Work ow of heuristic search. It is the population-based search approach, and starts with a initialization process.

Some popular heuristic search methods are listed as follow:

Particle swarm optimization (PSO)[59]: PSO is inspired by the behavior of biological communities that exhibit both individual and social behavior; examples of these communities are flocks of birds, schools of fishes and swarms of bees. Members of such societies share common goals (e.g., finding food) that are realized by exploring its environment while interacting among them. At each iteration, the population is updated by moving towards the best individuals. PSO optimizes by searching the neighborhoods of the best samples. It has a few hyperparameters itself and can be easily parallelized. In such way, PSO hopes to find the best position in search space.

Evolutionary algorithms[76]: Evolutionary algorithms are inspired by biological evolution. The generation step of evolutionary algorithms contains crossover and mutation. Crossover involves two different individuals (ancestors) from the last generation. It combines them in some way to generate an new individual. In principal, the more promising an individual is, the more likely is it to be chosen as an ancestor. Mutation, on the other hand, slightly changes an individual to generate a new one. With crossover mainly to exploit and mutation mainly to explore, the population is expected to evolve towards better performance.

The above methods have been widely applied in AutoML. For example, evolutionary algorithms has been applied in feature selection and generation[52, 9, 10, 11, 77] , and model selection[32]. PSO has been used for model selection[78, 78], feature selection for support vector machine (SVM)[78], and hyperparameter tuning for deep networks[79]. While evolutionary algorithms have already be enused in NAS one decade ago[20, 21, 80], it is only recently that better performance than human designed architecture are achieved[42, 81, 82, 83]. In these works, network structures are encoded with binary strings, on which the evolutionary operations are performed.

4.2.3 Reinforcement Learning

Reinforcement learning (RL)[84] is a very general and strong optimization framework, which can solve problems with delayed feedbacks. Figure 15 illustrates its general framework when used in AutoML. Basically, the policy in RL acts as the optimizer, and its actual performance in the environment is measured by the evaluator. However, unlike previous methods, the feedbacks (i.e., reward and state) do not need to be immediately returned once an action is taken. They can be returned after performing a sequence of actions.

Fig. 15. Workow of reinforcement learning. It is different from heuristic search and model-based derivative-free optimizationas feedbacks need not be immediately returned for receiving a conguration.

Resulting from the above-mentioned unique property, RL is recently used in NAS[41, 43]. The reason is that CNN can be built layer-by-layer, and the design of one layer can be seen as one action given by the optimizer. However, the performance of an architecture can only be evaluated after its whole structure is composed, which implies a delayed reward. Thus, the iterative architecture generation naturally follows the property of RL. However, due to the delayed feedbacks, AutoML with reinforcement learning is highly source-consuming, and more efficient methods needs to be explored. Some current endeavors addressing this problems are learning transferable architectures from smaller data sets[48], and cutting the search space by sharing parameter[49, 51].

Besides, a special case of RL, i.e., bandit-based approach, where rewards are returned for each action without a delay, is introduced to AutoML for hyper-parameter optimization[85, 86]. Finally, RL has also been used for optimization algorithms search[61], automated feature selection[87], and training data selection in active learning[88].

4.3 Performance evaluation

The goal of the search strategy is to find a neural network structure that maximizes certain performance metrics, such as precision on previously unseen data sets. To guide the search process, NAS algorithms need to estimate the performance of a given neural network structure, which is called a performance evaluation strategy.

For the neural network structure searched by the search strategy, first training on a training set, and then testing the accuracy value on the verification set. The training and verification process is time consuming, so it is necessary to take steps to reduce the cost of performance evaluation. A simple way to reduce training costs is to reduce training time (number of iterations), train on a subset of training samples, train on low-resolution images, or reduce the number of convolution kernels in certain layers during training. . These practices may result in deviations in performance evaluation values while reducing computational costs. Although the

search strategy only needs to sort the advantages and disadvantages of various network structures without knowing their accurate performance indicators, this approximation may still lead to deviations in the ranking results.

A more complicated approach is to predict the performance of the neural network (extrapolation), that is, to infer its final performance through the performance of the previous iterations during training, or to predict the entire network by the characteristics of the searched units (blocks). performance. Weight sharing is also a solution. Taking the weight of the previously trained sub-network as the initial weight of the sub-network to be evaluated can effectively improve the training speed, accelerate convergence, and avoid training from scratch. ENAS[89] and DARTS[90] directly let each subnet share the same set of weight parameters.

5 Generation Definition

In the previous sections, we systematically expounded the development of AutoML, the specific workflow and design points of AutoML and NAS. In this section, we will make a generation definition of AutoML.

5.1 The definition of AutoML

We can see that AutoML not only wants to have good learning performance (from machine learnings perspective) but also requires such performance being achieved without human assistance (from automations perspective). Thus, an informal and intuitive description of AutoML can be expressed as

ABCD1. (1)

Put it more formally, we describe what AutoML is in Denition 2. Such denition is inspired by Denition 1 and the fact that AutoML itself can also be seen as another machine learning approach (Figure 2).

Fig. 2. AutoML from machine learnings perspectives. AutoML

Denition 2 (AutoML). AutoML attempts to construct machine learning programs (specied by E, T and P in Denition 1), without human assistance and within limited computational budgets.

TABLE 2 Why we need to have AutoML: an overview comparison of classical machine learning and AutoML.

A comparison of classical machine learning and AutoML is in Table 2. Basically, in classical machine learning, human are heavily involved in configuring learning tools by operating feature engineering, model selection and algorithm selection. As a result, human take the most labor and knowledge-intensive job

in machine learning practices. However, in AutoML, all these can be done by computer programs. To understand Definition 2 better, let us look back at those three examples in Table 1:

Automated feature engineering: When original features are not informative enough, we may want to construct more features to enhance the learning performance. In this case, E is the raw feature, T is construction of features, and P is the performance of models which are learned with the constructed features. DSM[91] and ExploreKit[36] remove human assistance by automatically construct new features based on interaction among input features.

Automated model selection: Here, E denotes input training data, T is a classication task, and P is the performance on the given task. When features are given, Auto-sklearn can choose proper classiers and nd corresponding hyper-parameters without human assistance.

Neural architecture search (NAS): When we try to do some image classication problems with the help of NAS, E is the collection of images, T is the image classication problem, and P is the performance on testing images. NAS will automatically search for a neural architecture, i.e., a classier based on neural networks, that has good performance on the given task.

Finally, note that Denition 2 is general enough to cover most machine learning approaches that can be considered automatic. With this denition, a machine learning pipeline with xed congurations, that do not adapt according to different E, T, and P, is also automatic. Approaches of this kind,though require no human assistance, are rather limited in their default performance and application scopes. Thus, they will not be further pursuit in the sequel.

From above discussion, we can see that while good learning performance is always desired, AutoML requires such performance can be obtained in a more special manner, i.e. without human assistance and within limited computational budgets. These set up three main goals for AutoML (Remark 2n).

Remark 2n (Core goals). The three goals of AutoML:

(A). Good performance: good generalization performance across various input data and learning tasks can be achieved;

(B). No assistance from humans: congurations can be automatically done for machine learning tools; and

(C). High computational efciency: the program can return an reasonable output within a limited budget.

Since AutoML itself can be seen as a machine learning tool (Figure 2), here we remark that the goal (A) actually intends to escape the curse of the notorious No Free Lunch theorems stated in [38] and [92].

These theorems state that in a noise-free scenario of supervised learning, all learning algorithms have the same generalization performance (error rate) when averaged over all possible learning tasks. Although these theorems are mathematically proven, it is hard (and even impossible) to apply them to the reality and make empirical test. This is because that the average on the performance over all possible learning tasks (with equal weights) is very brutal. It is highly possible that learning tasks in reality take up only a very narrow spectrum in all theoretically possible tasks. And the information on the distribution of reality tasks can be utilized in a context of meta-learning or transfer learning, for which we will give further investigation in Section 6.1 and 6.2.

Once above three goals can be realized, we can fast deploy machine learning solutions across organizations, quickly validate and benchmark the performance of deployed solutions, and let human focus more on problems that really need humans engagements, i.e., problem denition, data collection and deployment in Figure 1. All these make machine learning easier to apply and more accessible for everyone.

5.2 AutoML Framework

In Section 5.1, we have dened the AutoML problem (Definition 2) and introduced its core goals (Remark 2n). In this section, we propose a basic framework for AutoML approaches.

5.2.1 Human Tuning

However, before that, let us learn how congurations are tuned by human. Such process is shown in Figure 4. Once a learning problem is dened, we need to nd some learning tools to solve it. These tools, which are placed in the right part of Figure 4, can target at different parts of the pipeline, i.e., feature, model or optimization in Figure 1. To obtain a good learning performance, we will try to set a conguration using our personal experience or intuition about the underneath data and tools. Then, based on the feedback about how the learning tools perform, we will adjust the conguration wishing the performance can be improved. Such a trial-and-error process terminates once a desired performance is achieved or the computational budgets are run out.

Fig. 4. The process of congurations tuned by humans.

5.2.2 Proposed AutoML Framework

Motivated by the human-involved process above and controlling with feedbacks in the automation[93], we summarize a framework for AutoML, as shown in Figure 6. Compared with Figure 4, in this figure, an

AutoML controller takes the place of human to find proper configurations for the learning tools. Basically, we have two key ingredients inside the controller, i.e., the optimizer and the evaluator.

Their interactions with other components in Figure 6 are as follows:

Evaluator: The duty of the evaluator is to measure the performance of the learning tools with configurations provided by the optimizer. After that, it generates feedbacks to the optimizer. Usually, to measure the performance of learning tools with given configuration, the evaluator needs to train a model based on the input data, which can be time-consuming. However, the evaluator can also directly estimate the performance based on external knowledge, which mimics humans experience. Such estimation is very fast but may be inaccurate. Thus, for the evaluator, it needs to be efficient but also accurate in measuring the performance of configurations.

Optimizer: Then, for the optimizer, its duty is to update or generate configurations for learning tools. The search space of the optimizer is determined by the targeted learning process, and new configurations are expected to have better performance than previous ones. However, feedbacks offered by the evaluator are not necessarily required or exploited by the optimizer. This depends on which type of the optimizer we are utilizing. Finally, the optimizer should be chosen based on the learning process and corresponding search space, as the latter determines the applicability of different optimization methods. We also wish the structure of the search space can be simple and compact so that more generic and efficient optimization methods can be employed.

As we will see, this framework is general enough to cover nearly all existing works. In Section 7, we provide more detailed examples demonstrating how the it can cover existing works. Furthermore, this framework is also precise enough to help us setup taxonomies for AutoML approaches (Section 2.3), and it gives insight to the future direction of AutoML (Section 8.3). future direction of AutoML (Section 8.3).

5.3 Taxonomies of AutoML Approaches

In this section, we give taxonomies of existing AutoML approaches based on what and how to automate.

5.3.1 "What to automate": by problem setup

The choice of learning tools inspires the taxonomy based on problem setup in Figure 5(a), this defines what we want to make automated by AutoML. Basically, for general learning problems, we need to do feature engineering, model selection and optimization algorithm selection. These three parts together make up the full scope of general machine learning applications (Figure 1). We also list NAS there as a very important and special case. The reason is that NAS targets at deep models, where features, models and algorithms are

configured simultaneously. The focus and challenges of AutoML problem under each setup are detailed in Section 3.

5.3.2 How to automate: by techniques

Figure 5(b) presents the taxonomy by AutoML techniques. These are the techniques used for the controller, and categorize how we solve an AutoML problem. In general, we divide existing techniques into basic and experienced ones:

Fig. 5. AutoML approaches taxonomies by problem setup and techniques, which is inspired by the proposed framework in Figure 6. Taxonomy by problem setup depends on which learning tools we used, it claries what we want to make automated; taxonomy by techniques depends on the how we want to solve AutoML problems. Specically, feature engineering, model selection and optimization algorithm selection together make up the full scope of general machine learning applications (Figure 1).

Basic techniques: As there are two ingredients, i.e., the optimizer and evaluator, in the controller, we categorize basic techniques based on which ingredient they operating on. The optimizer focus on the searching and optimizing congurations, and there are many methods can be used, from simple methods as grid search and random search[75] to more complex ones as reinforcement learning[41] and automatic differentiation[94]. However, for the evaluator, which mainly measures the performance of learning tools with current congurations by determine their parameters, there are not many methods can be taken as basic ones.

Experienced techniques: Experienced techniques learn and accumulate knowledge from past searches or external data. They usually need to be combined with basic techniques and enhance the optimizer and/or evaluator in various manners. Generally, there are two main methods popularly used in AutoML, i.e., meta-learning[95, 96] and transfer learning[97].

Fig. 6. Basic framework for how existing approaches solving AutoML problem. The dashed line (feedback) inside the controller, which depends on what techniques are used for the optimizer, is not a must.

Note that, as E, T and P are also involved in the AutoMLs denition (Denition 2), taxonomies of machine learning, e.g., supervised learning, semi-supervised learning and unsupervised learning, can also be applied for AutoML. However, they do not necessarily connect with removing human assistance in nding congurations (Figure 4). Thus, taxonomies here are done based on the proposed framework in Figure 6 instead.

6 Conclusion

Motivated by the academic dream and industrial needs, the automated machine learning (AutoML) has recently became a hot topic. In this survey, we give a systematical review of existing AutoML approaches, then summarize the Workflow and Framework of AutoML. At last, We also dene what is the AutoML problem.

References

[1] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.

[2] Barbora Micenková, Brian McWilliams, and Ira Assent. Learning representations for outlier detection on a budget. *arXiv preprint arXiv:1507.08104*, 2015.

[3] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.

[4] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.

[5] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[6] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[7] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[8] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[9] Haleh Vafaie and Kenneth De Jong. Genetic algorithms as a tool for feature selection in machine learning. In *Proceedings Fourth International Conference on Tools with Artificial Intelligence TAI'92*, pages 200–203. IEEE, 1992.

[10] Walter Alden Tackett. Genetic programming for feature discovery and image discrimination. In *ICGA*, pages 303–311, 1993.

[11] Wojciech Siedlecki and Jack Sklansky. A note on genetic algorithms for large-scale feature selection. In *Handbook of Pattern Recognition and Computer Vision*, pages 88–107. World Scientific, 1993.

[12] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[13] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[14] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[15] Michael JD Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162, 1964.

[16] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[17] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

[18] Nedjem-Eddine Ayat, Mohamed Cheriet, and Ching Y Suen. Automatic model selection for the optimization of svm kernels. *Pattern Recognition*, 38(10):1733–1745, 2005.

[19] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Advances in neural information processing systems*, pages 281–288, 2004.

[20] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[21] Chunkai Zhang, Huihe Shao, and Yu Li. Particle swarm optimisation for evolving artificial neural network. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 4, pages 2487–2490. IEEE, 2000.

[22] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18. ACM, 2004.

[23] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *International Conference on Principles and Practice of Constraint Programming*, pages 454–469. Springer, 2011.

[24] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*. Mit Press, 2012.

[25] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

[26] Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 265–272. Omnipress, 2011.

[27] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[28] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[29] I Guyon, WW Tu, et al. Automatic machine learning challenge 2018: Towards ai for everyone. *PAKDD 2018 Data Competition*, 2018.

[30] WW Tu. The 3rd automl challenge: Automl for lifelong machine learning. *NIPS 2018 Challenge*, 2018.

[31] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830, 2017.

[32] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Automated Machine Learning*, pages 151–160. Springer, 2019.

[33] FF Li and J Li. Cloud automl: Making ai accessible to every business. *Internet: https://www.blog.google/topics/google-cloud/cloud-automl-making-ai-accessible-everybusiness*, 2018.

[34] Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 874–883. JMLR. org, 2017.

[35] Tom M Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[36] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016.

[37] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[38] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[39] Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image processing*, 15(12):3736–3745, 2006.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[41] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[42] Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388, 2017.

[43] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[44] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[45] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[46] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2423–2432, 2018.

[47] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[48] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[49] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Faster discovery of neural architectures by searching for paths in a large model. 2018.

[50] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[51] Juan-Manuel Pérez-Rúa, Moez Baccouche, and Stephane Pateux. Efficient progressive neural architecture search. *arXiv preprint arXiv:1808.00391*, 2018.

[52] Matthew G Smith and Larry Bull. Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines*, 6(3):265–281, 2005.

[53] Binh Tran, Bing Xue, and Mengjie Zhang. Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Computing*, 8(1):3–15, 2016.

[54] Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B Khalil, and Deepak S Turaga. Learning feature engineering for classification. In *IJCAI*, pages 2529–2535, 2017.

[55] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327*, 2017.

[56] Ambika Kaul, Saket Maheshwary, and Vikram Pudi. Autolearnautomated feature generation and selection. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 217–226. IEEE, 2017.

[57] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning. springer series in statistics. In *∴*. Springer, 2001.

[58] Evan R Sparks, Ameet Talwalkar, Daniel Haas, Michael J Franklin, Michael I Jordan, and Tim Kraska. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 368–380. ACM, 2015.

[59] Hugo Jair Escalante, Manuel Montes, and Luis Enrique Sucar. Particle swarm model selection. *Journal of Machine Learning Research*, 10(Feb):405–440, 2009.

[60] Vincent Calcagno, Claire de Mazancourt, et al. glmulti: an r package for easy automated model selection with (generalized)

linear models. *Journal of statistical software*, 34(12):1–29, 2010.

[61] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 459–468. JMLR. org, 2017.

[62] Christopher J Merz. Dynamical selection of learning algorithms. In *Learning from Data*, pages 281–290. Springer, 1996.

[63] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[64] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, et al. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

[65] Haifeng Jin, Qingquan Song, and Xia Hu. Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 2018.

[66] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. *arXiv preprint arXiv:1806.02639*, 2018.

[67] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[68] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[69] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[70] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pages 58–65, 2016.

[71] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[72] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

[73] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[74] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

[75] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[76] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[77] Randal S Olson, Nathan Bartley, Ryan J Urbanowicz, and Jason H Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 485–492. ACM, 2016.

[78] Shih-Wei Lin, Kuo-Ching Ying, Shih-Chieh Chen, and Zne-Jung Lee. Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert systems with applications*, 35(4):1817–1824, 2008.

[79] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the genetic and evolutionary computation conference*, pages 481–488. ACM, 2017.

[80] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[81] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017.

[82] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[83] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

[84] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[85] András György and Levente Kocsis. Efficient multi-start strategies for local search algorithms. *Journal of Artificial Intelligence Research*, 41:407–444, 2011.

[86] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

[87] Romaric Gaudel and Michele Sebag. Feature selection as a one-player game. 2010.

[88] Meng Fang, Yuan Li, and Trevor Cohn. Learning how to active learn: A deep reinforcement learning approach. *arXiv preprint arXiv:1708.02383*, 2017.

[89] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter

sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[90] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[91] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015.

[92] David H Wolpert, William G Macready, et al. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[93] Charles L Phillips and Royce D Habor. *Feedback control systems*. Simon & Schuster, Inc., 1995.

[94] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153), 2018.

[95] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130, 2015.

[96] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

[97] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.