AutoML: A Survey of the State-of-the-Art

Xin He, Kaiyong Zhao, Xiaowen Chu*
Department of Computer Science, Hong Kong Baptist University
Email: {csxinhe, kyzhao, chxw}@comp.hkbu.edu.hk

***Abstract*—Deep learning has penetrated all aspects of our lives and brought us great convenience. However, the process of building a high-quality deep learning system for a specific task is not only time-consuming but also requires lots of resources and relies on human expertise, which hinders the development of deep learning in both industry and academia. To alleviate this problem, a growing number of research projects focus on automated machine learning (AutoML). In this paper, we provide a comprehensive and up-to-date study on the state-of-the-art AutoML. First, we introduce the AutoML techniques in details according to the machine learning pipeline. Then we summarize existing Neural Architecture Search (NAS) research, which is one of the most popular topics in AutoML. We also compare the models generated by NAS algorithms with those human-designed models. Finally, we present several open problems for future research.**

***Index Terms*—machine learning, AutoML, NAS, survey**

I. Introduction

Nowadays, deep learning has been applied in various fields and solved lots of challenging AI tasks, including object classification and detection, language modeling, recommendation system, etc. Specifically, after AlexNet [1] outperformed all other traditional manual methods in the 2012 ImageNet Challenge, more and more complex and deep neural networks were proposed. Taking VGG-16 [2] as an example, it has more than 130 million parameters, occupying nearly 500 MB of memory space, and needs 15.3 billion floating-point operations to complete an image recognition task. However, it has to be pointed out that these models were all manually designed by experts through trial and error, which means that even with considerable expertise, we still have to spend a significant amount of resources and time to design such well-performed models.

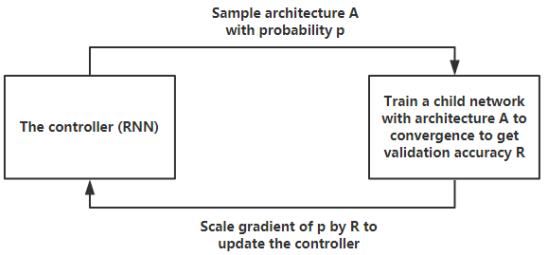


Figure 1: An overview of neural architecture search using reinforcement learning (from [3]).

In order to reduce such huge cost, a new idea of automating the whole pipeline process of machine learning, i.e., automated machine learning (AutoML), has emerged. There are various definitions of AutoML. For example, according to the Wikipedia [1], "AutoML is the process of automating the end-to-end process of applying appropriate data-preprocessing, feature engineering, model selection, and model evaluation to solve the certain task". In [4], AutoML is defined as the combination of automation and machine learning. In other words, AutoML can automatically build a machine learning pipeline with limited computational budgets.

No matter how to define AutoML, it must be pointed out that AutoML is not a completely new concept. The reason why it has been a hot topic recently both in industry and academia is the great improvement in computing power so that it is possible and practical to combine different techniques dynamically to form an end-to-end easy-to-use pipeline system (as shown in Fig. 2). Many AI companies have provided such systems (e.g. Cloud AutoML [2] by Google) to help people with little or even no machine learning knowledge to build a high-quality custom models. As matter of a fact, it is the work of Zoph et al. [5] that draws attention to AutoML. In [5], a recurrent network is trained by reinforcement learning to search for the best performing architecture automatically (Fig. 1). Since then, more and more works on AutoML have been proposed, and most of them mainly focus on neural architecture search (NAS) that aims to generate a robust and well-performing neural architecture by selecting and combining different basic components from a predefined search space according to some search strategies. We will introduce NAS from two perspectives. The first is the structures of the model. The common structures includes entire structure [5]–[7], cell-based structure [6], [8]–[11], hierarchical structure [12] and morphism-based structure [13], etc. The second is hyperparameter optimization (HPO) for designing the model structure. The widely used methods contains *reinforcement learning* (RL) [5], [6], [8], [9], [14], *evolutionary algorithm* (EA) [15]–[21], and *gradient descent* (GD) [10], [22], [23], Bayesian optimization [24]–[30] and so on. Besides NAS, AutoML also involves other techniques that have been studied for a long time, and we classify these techniques into the following categories based on the machine learning pipeline, shown in Fig. 2: data preparation, feature engineering, model generation and model evaluation.

It must be stated that many sub-topics of AutoML are large enough to have their own surveys. However, our goal is not to make a thorough investigation of all sub-topics, but to focus on

*Corresponding author.

[1] https://en.wikipedia.org/wiki/Automated_machine_learning
[2] https://cloud.google.com/automl/

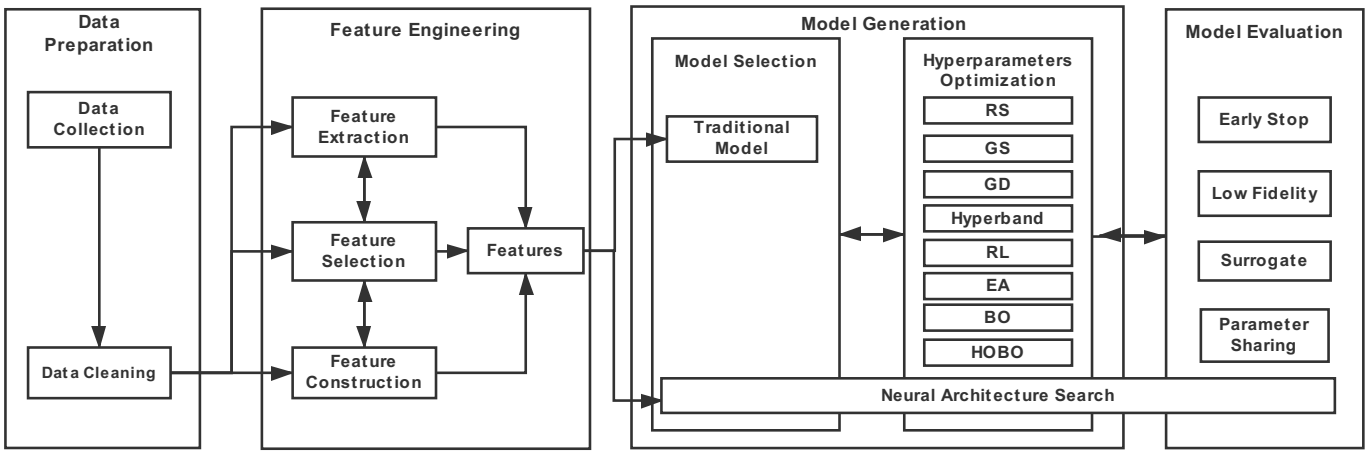


Figure 2: An overview of AutoML pipeline covering data preparation (Section II), feature engineering (Section III), model generation (Section IV) and model evaluation (Section V).

the breadth of research in the field of AutoML. Therefore, in the following sections, we will only select the most representative works for discussion and analysis. In addition, although there exist vague boundaries between different sub-topics, we don't think this will affect our understanding of the development of AutoML, because sometimes it is difficult to separate several issues clearly. For example, model selection can also be though as the problem of HPO, because its main purpose is to optimize the combination of primitive model components, but model selection contains some important techniques that HPO doesn't involve in, hence it is necessary to take model selection apart from HPO for discussion.

The contributions of this paper are as follows:

1) Although there are several surveys related to AutoML (e.g. [3], [4], [31]), we cover a wider range of existing AutoML techniques according to the pipeline (Fig. 2) of machine learning, which gives the beginners a comprehensive and clear understanding of AutoML. Specifically, We extend the process of data collection to the pipeline, which could boost the generality of AutoML framework.
2) Currently, NAS has been one of the most popular sub-topics in AutoML. Therefore, this paper provides a detailed comparison of various NAS algorithms from various aspects, including the performance on baseline datasets, and the time and resource cost for searching and the size of the best-performing model.
3) Besides summarizing the existing works, we discuss some open problems we are facing now and propose some promising future works.

The rest of this paper proceeds as follows: the process of data preparation, feature engineering, model generation and model evaluation are presented in Section II, III, IV, V, respectively. In Section VI, we make a detailed summary of NAS algorithms and compare the performance of the models generated by NAS and human-designed models. In Section VII, we propose several open problems on AutoML and discuss corresponding promising future works. Finally, we conclude the survey in Section VIII.

II. Data Preparation

As is known, the very first step in machine learning pipeline is to prepare data, but for many tasks, like medical image recognition, it is hard to obtain enough data or the quality of data is not good enough. Therefore, a robust AutoML system should be able to deal with this problem. In the following, we will divide it into two steps: data collection and data cleaning.

A. Data Collection

With the study of machine learning (ML) going deeper and deeper, people gradually realize that data is very important and that's why a large number of open datasets have emerged. In the early stage of the study of ML, a handwritten digit dataset, i.e. MNIST [32] was provided. After that, the larger dataset like CIFAR10 & CIFAR100 [33] and ImageNet [1] were also proposed. What's more, we can also search for a variety of datasets by typing the keywords in these web sites: Kaggle [34], Google Dataset Search (GOODS) [35], Elsevier Data Search [36].

However, when it comes to special tasks, especially like medical tasks or other privacy involved tasks, it is usually very difficult to find a proper dataset through the above approaches. There are two types of methods proposed to solve this problem: data synthesis and data searching.

1) Data Synthesis: In the light of the broad range of computer vision problems, here we will only discuss some representative approaches of generating synthetic data. One of the most commonly used methods is to augment the existing dataset. For image data, there are many augmentation operations, including cropping, flipping, padding, rotation, and resize, etc. The Python libraries like torchvision [37] and Augmentor [38] provide these augmentation operations to generate more images. Wong et al. [39] propose two approaches for creating additional training examples: data

warping and synthetic over-sampling. The former generates additional samples by applying transformation on data-space and the latter creates additional samples in feature-space. For text data, synonym insertion is a common way of augmenting. Another idea is to first translate the text into some foreign language, and then translate it back to the original language. Recently, Xie et al. [40] have proposed a non-domain-specific data augmentation strategy that uses noising in RNNs and works well in NLP tasks such as language modeling and machine translation. Yu et al. [41] propose to use back-translation for data augmentation for reading comprehension.

In terms of some special tasks, such as autonomous driving, it is not possible to test and adjust the model directly in the real world during the research phase, because there exists the problem of potential safety hazards. Thus, a practical method of creating data for such tasks is data simulator, which tries to match the real world as much as possible. For example, OpenAI Gym [42] is a popular toolkit that provides various simulation environment, where the developers can concentrate on designing their algorithms, instead of struggling for generating data. It is also used to assist the task of machine learning [43]. Furthermore, a reinforcement learning-based method is proposed in [44] for optimizing the parameters of data simulator to control the distribution of the synthesized data.

Figure 3: The Human Face Images Generated by GAN. (from [45])

Another novel technique is *Generative Adversarial Network* (GAN) [45], which can not only be used to generate images, but also the text data. Fig. 3 shows some human face images, which are generated by GAN in the work of Karras et al. [46]. Instead of generating images, Eno et al. [47] develop a synthetic data definition language (SDDL) to create new data for Iris dataset [48]. Moreover, natural scene text can also be generated in [49], [50]. Oh and Jaroensri et al. [51] build a synthetic dataset, which captures small motion for video motion magnification.

2) Data Searching: As there exists inexhaustible data on the Internet, an intuitive method of collecting a dataset is to search for web data [52]–[55]. But there exist some problems with using web data.

On the one hand, sometimes the search results do not exactly match the keywords. To solve this problem, one way is to filter unrelated data. For example, Krause et al. [56] divide inaccurate results into cross-domain and cross-category noise, and they remove the images that appear in search results for more than one category. Vo et al. [57] re-rank relevant results and provide search results linearly according to keywords. The other problem is that web data may have wrong labels or even no labels. The learning-based self-labeling method is often used to solve this problem. For example, active learning [58] is a method that selects the most "uncertain" unlabeled individual examples to ask human for labeling, then it will label the rest of the data iteratively. To take the human out of labeling and further accelerate the process of labeling, many semi-supervised learning self-labeling methods are proposed. Roh et al. [59] have summarized the self-labeling methods into the following categories: self-training [60], [61], co-training [62], [63], and co-learning [64]. Besides, due to the complexity of the content of web images, a single label can not describe the image well, so Yang et al. [65] assign multiple labels to a web image, and if these labels have very close confidence score or the label with the highest score is the same with the original label of the image, they select this image as a new training sample.

On the other hand, the distribution of web data can be extremely different from the target dataset, which would increase the difficulty of training the model. A common solution is to fine-tune these web data [66], [67]. Yang et al. [52] an iterative algorithm for model training and web data filtering. Additionally, dataset imbalance is also a common problem, because there probably are only a small number of web data for some special classes. To solve this problem, Synthetic Minority Over-Sampling Technique (SMOTE) [68] was proposed to synthesises new minority samples between existing real minority samples instead of up-sampling them or down-sampling the majority samples. Guo et al. [69] propose to combines boosting method with data generation to enhance the generalization and robustness of the model against imbalanced data sets.

B. Data Cleaning

Before stepping into feature generation, it is essential to preprocess the raw data, because there exist many types of data errors (e.g. redundant, incomplete, or incorrect data). Taking the tabular data as an example, the common error types are missing values, wrong data type. The widely used operations for data cleaning includes standardization, scaling, binarization of quantitative characteristic, one-hot encoding qualitative characteristic, and filling missing values with mean value, etc. In terms of image datasets, sometimes the image may be assigned with a wrong label, in which case the techniques like self-labeling that has been mentioned above, can be used to solve that problem. However, the data cleaning process usually needs to be defined manually in advance, because different methods may have different requirements even for the same dataset. For example, the neural network can only work on numerical data, while decision tree-based methods can deal with both numerical and categorical data. The works

including [70]–[73] are proposed to automate the process of data cleaning.

III. Feature Engineering

In industry, it is generally accepted that data and features determine the upper bound of machine learning, and models and algorithms only approximate it. The purpose of feature engineering is to maximize the extraction of features from raw data for use by algorithms and models. Feature engineering consists of three sub-topics: feature selection, extraction, and construction. Feature extraction and construction are the variants of feature transformation through which a new set of features is created [74]. Feature extraction is usually aimed to reduce the dimension of features by some functional mapping, while feature construction is used to expand original feature spaces. Additionally, the purpose of feature selection is to reduce feature redundancy by selecting important features. In some degrees, the essence of automatic feature engineering is a dynamical combination of these three processes.

A. Feature Selection

Feature selection is a process that builds a feature subset based on the original feature set by reducing irrelevant or redundant features, which is conducive to simplify the model, hence avoiding overfitting and improving model performance. The selected features are usually divergent and highly correlated with object values. According to the work proposed by [75], there are four basic steps in a typical process of feature selection (see Fig. 4):

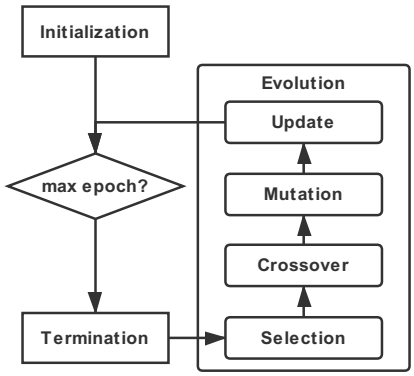


Figure 4: The iterative process of feature selection. First, a subset of features is selected based on a search strategy and then evaluated. After that, a validation procedure is implemented to check whether the subset is valid. If not, repeat the above steps. Otherwise, it ends.

The search strategy for feature selection can be divided into three categories: complete, heuristic, and random search algorithms. For complete search, it involves exhaustive and non-exhaustive searching, which can be further split into four methods, i.e., Breadth First Search, Branch and Bound, Beam Search, and Best First Search. For heuristic search, it includes Sequential Forward Selection (SFS), Sequential Backward Selection (SBS), and Bidirectional Search (BS). In the first two cases, they iteratively add features from an empty set or remove features from a full set, respectively, whereas, for BS, it uses both SFS and SBS to search until these two algorithms obtain the same subset. In terms of random search, methods like Simulated Annealing (SA) and Genetic Algorithms (GA), are usually used.

For subset evaluation, there are three different types of methods. The first is the filter method, which scores each feature according to divergence or correlation, and then select features by a threshold. To score each feature, the commonly used scoring criteria are variance, correlation coefficient, Chi-square test, and mutual information, etc. Wrapper method is another choice, which classifies the sample set with the selected feature subset, and the classification accuracy is used as the criterion to measure the quality of the feature subset. The third method is the embedded method which performs variable selection as part of the learning procedure. Regularization, decision tree, and deep learning are all embedded methods.

B. Feature Construction

Feature construction is a process that constructs new features from the basic feature space or raw data to help improve prediction accuracy and generalization of the model, so its essence is to increase the representative ability of original features. Traditionally, this process highly depends on human expertise and common methods are preprocessing transformations which are also applied in data preprocessing, such as standardization, normalization, feature discretization. As we all know, there are many kinds of features, and the operations applied to these features are also different. For example, the operations, including conjunctions, disjunctions, negation, are usually applied in boolean features, and for numerical features, we can use operations like min, max, addition, subtraction, mean, etc. For nominal features, the common operations are Cartesian product [76] and M-of-N [77].

It is impossible to explore all possibilities manually. Hence, to further improve efficiency, some automatic feature construction methods have been proposed and can also achieve the same results as human expertise or even better. These algorithms mainly attempt to automate the search and evaluation of the operation space. In terms of searching, the exemplar algorithms includes decision tree-based methods [77], [78] and genetic algorithm [79]. Unlike the two algorithms mentioned above, annotation-based approaches eliminate the requirement of predefining the operation space, as it allows human to provide domain knowledge in the form of annotation along with the training examples [80]. Such methods can be traced back to the work proposed by [81], in which the authors introduced the interactive feature space construction protocol where the learner identifies inadequate regions of the feature space and in coordination with a domain expert adds descriptiveness through existing semantic resources. After selecting possible operations and constructing a new feature, feature selection techniques will be applied to measure the new feature.

C. Feature Extraction

Feature extraction is a dimensionality reduction process through some mapping functions, which extracts informative and non-redundant features according to some specific metrics. Different from feature selection, feature extraction will alter the original features. The kernel of feature extraction is a mapping function, which can be implemented in many ways. The worth mentioning approaches are Principal Component Analysis (PCA), independent component analysis, isomap, nonlinear dimensionality reduction, Linear discriminant analysis (LDA). Recently, a widely used method is the feed-forward neural networks approach, which uses the hidden units of a pretrained model as extracted features. Many algorithms are proposed based on autoencoder. Meng et al. [82] propose a Relation Autoencoder model considering both data features and their relationships. An unsupervised feature extraction method using autoencoder trees is proposed by [83].

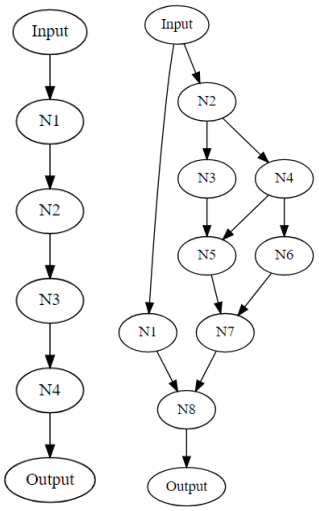


Figure 5: Examples of the chain-structure neural architectures which are generated as a whole. Each node in the graph indicates a layer with a specific operation, e.g. N1 represents the layer 1. The operation of each node is selected from the search space, which includes convolution or max pooling, etc. The edge indicates the flow of information. For example, the edge from N2 to N3 in the left graph represents N3 receives the output of N2 as its input.

IV. Model Generation

After generating the features, we need to select a model and set its corresponding hyperparameters. As shown in Fig. 2, there are two types of approaches in model selection: traditional model selection and NAS. The former is to select the best-performing model from the many traditional machine learning algorithms, e.g. *support vector machines* (SVM), *k-nearest neighbors* (KNN), *decision tree*, *KMeans* etc. In this paper, we will focus more on NAS, which is a very hot topic currently and aims to design a novel neural architecture without human assistance. To give readers a clear understanding of NAS, we will introduce them from two aspects: the types of model structures and the algorithms of optimizing the model parameters automatically.

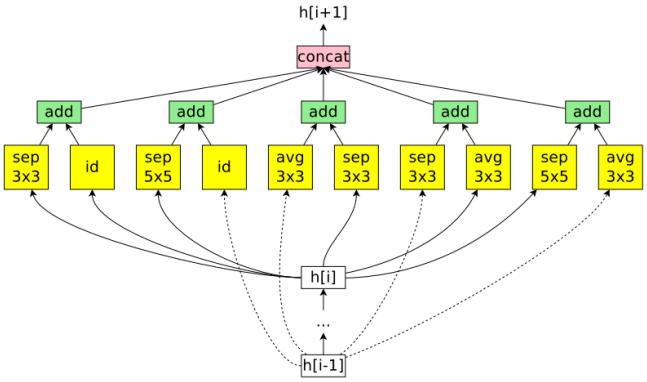


Figure 6: The cell structure discovered in [6]. A cell consists of B nodes. In this cell, there are 7 nodes, each of which has two child nodes with specified operations. The first two nodes (i.e. $h[i]$ and $h[i-1]$) are treated as the inputs, which are the outputs of the two previous cells. [6]

A. Model Structures

The basic components of the whole architecture are selected from a collection, namely search space, which determines all possible operations of each layer in the network. Search space is usually predefined by human because it is not practical to include all operations. The operations can be broadly classified into convolution, pooling, concatenation, elemental addition, skip connection, etc. The parameters of these operations are also usually predefined empirically. For example, the kernel size of convolution is usually set 3×3 and 5×5, and the widely used types of convolution operations are also designed by human, such as depthwise separable [84], dilation [85], and deformable convolution [86]. So it is a promising direction to use NAS to generate novel basic operations like convolution. By reviewing the literature related to NAS, we summary several representative structures as follows:

1) Entire structure: The first and intuitive way is to generate an entire chain-structure neural network [5], [6], which is similar to the traditional neural network structure. Fig. 5 gives two different examples of the generated network. The left is very simple, while the right is relatively complex as it uses some hand-crafted structure, like skip connections and multi-branches networks, which have been proven effective in practice. However, there are some obvious disadvantages to this type of architecture. The search space is usually very large, which requires a lot of time and computing resources to find the best architecture. Besides, the final generated architecture is short of transferability, which means that the architecture generated on a small dataset is hard to fit on a larger dataset, hence we can only regenerate a new deeper model on the larger dataset.

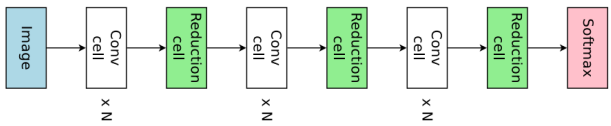
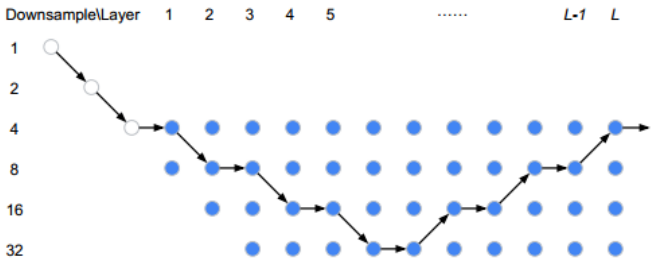
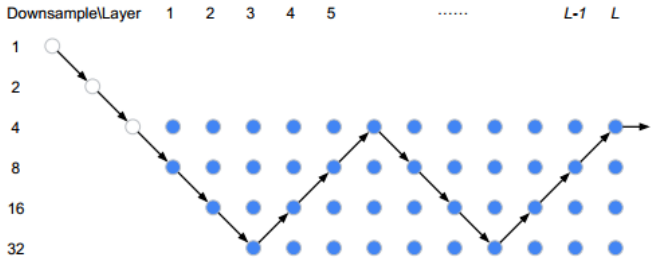


Figure 7: The example of connecting 3 blocks, each with N convolution cells and 1 reduction cell, to make the final network. Convolution cells keep the spatial resolution of the feature tensor, and reduction cells divide the spatial resolution by 2 and multiply the number of filter by 2. [6]



(a) Network level architecture used in Conv-Deconv



(b) Network level architecture used in Stacked Hourglass

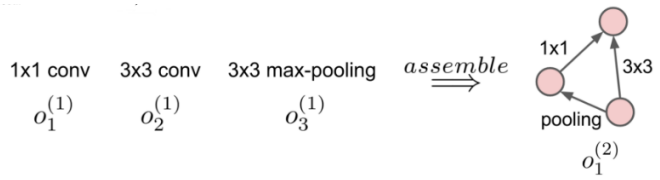
Figure 8: The network level search space. Three gray nodes at the beginning indicate the fixed "stem" structures, and each blue point is a cell structure as described above. The black arrows along the blue points indicate the final selected network level structure. Best viewed in color. (from [87])

2) Cell-based structure: To solve the shortcomings of the entire structure and inspired by some excellent network structures [88], [89], some works [6], [8], [9] propose to the first search for a cell structure and then stack a predefined number of discovered cells to generate a whole architecture in a way similar to chain-structure. Fig. 6 presents an example of cell structure discovered for convolutional neural network in [6]. Such a method of designing neural architecture can greatly reduce the search complexity. To illustrate this, let's make the following assumptions. Suppose there are 6 predefined operations, and for layer L_j, there are $j-1$ layers that can be connected to it, leading to $6 \times 2^{j-1}$ possible decisions for each layer. For entire structure, When $L = 12$, there are $6^L \times 2^{L(L-1)/2} = 6^8 \times 2^{12(12-1)/2} = 1.6 \times 10^{29}$ possible networks. For cell-based structure, there are B nodes in a cell, and each node consists of two child nodes with specified operations. The input of each child node of node $i (i \geq 2)$ is from previous $i-1$ nodes. As all decisions for each node is independent, there are $(6 \times (B-2)!)^2$ possible cell structure. In addition to the convolutional cell, there is usually another structure, called reduction cell, which is used to reduce the spatial dimensions of inputs by a factor of 2. Fig. 7 shows how to combine convolution cell and reduction cell into a complete network. Suppose there are 8 nodes in a cell, then the final size of search space for cell structure is $(6 \times (B-2)!)^4 = (6 \times (8-2)!)^4 = 3.5 \times 10^{14}$, which is much smaller than the search space of the entire structure. One thing worth mentioning is that for the entire structure, each layer only indicates a single operation, while for cell-based structure, each layer indicates a complex cell structure. What's more, the cell-based structure can be easier transferred from small dataset to large dataset by simply stacking more cells.

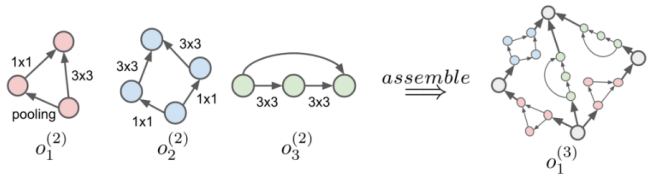
By reviewing the cell-based structures [6], [8], [9], [14], [16], [17], we can find that they all follow a two-level hierarchy: the inner is cell level, which selects the operation and connection for each node; the outer is network level, which controls the spatial resolution changes. However, they only focus on the cell level and ignore the network level. Because once a cell structure is designed, the full network is generated by stacking cells in a chain way. As indicated in Fig. 7, the network is constructed by combining a fixed number of convolution cells, which keep the spatial resolution of the feature tensor, and a reduction cell, which divides the spatial resolution by 2. To jointly learn a good combination of repeatable cell structure and network structure, Liu and Yuille et al. [87] define a general formulation of network-level structure as depicted in Fig. 8 , where many existing good network designs can be reproduced in this way.

3) Hierarchical Structure: The hierarchical structure is similar to a cell-based structure, but the main difference is the way of generating cells. As mentioned before, the number of cell nodes in cell-based structure is fixed. After generating the cell, the network is built by stacking a fixed number of cells in a chain way. However, for a hierarchical structure, there are many levels, each with a fixed number of cells. The higher-level cell is generated by incorporating lower-level cell iteratively. As shown in Fig. 9, the primitive operations, such as 1×1 and 3×3 convolution and 3×3 max pooling in level-1 are the basic components of level-2 cells. Then level-2 cells are used as primitive operations to generate level-3 cell. The highest-level cell is a single motif corresponding to the full architecture. Besides, a higher level cell is defined by a learnable adjacency upper triangular matrix G, i.e., $G_{ij} = k$ means that the k-th operation 0_k is implemented between nodes i and j. For example, the level-2 cell in Fig. 9(a) is defined by a matrix G, where $G_{01} = 2, G_{02} = 1, G_{12} = 0$ (the index starts from 0). Compared with cell-based structure, this method can discovery more types of network structure with complex and flexible topologies.

4) Network Morphism based structure: Network Morphism (NM) based structure [13], [18], [90] is the way of transferring the information stored in an existing neural network into a new



(a) The level-1 primitive operations are assembled into level-2 cells.



(b) The level-2 cells are viewed as primitive operations and assembled in to level-3 cells.

Figure 9: An example of a three-level hierarchical architecture representation. [12]

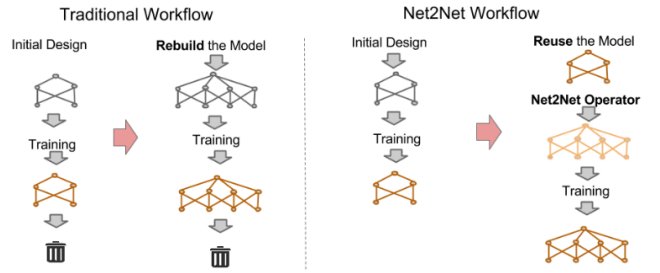


Figure 10: Comparison between a traditional way of generating structures and Net2Net [13], which reuses information from an existing network to generate a new structure.

neural network. Therefore, it can update the structure based on previous well-performing one, instead of regenerating a new structure from scratch, as shown in Fig. 10. At the same time, the new network is guaranteed to perform better than, or at least equivalent to the old network. For example, in [13], the authors generate a new network based on VGG16 by using network morphism, and final results on ImageNet are 69.14% top-1 and 89.00% top-5 accuracy, which is better than the original result of VGG16, i.e. 67.3% top-1 and 88.31% top-5 accuracy.

B. hyperparameter optimization

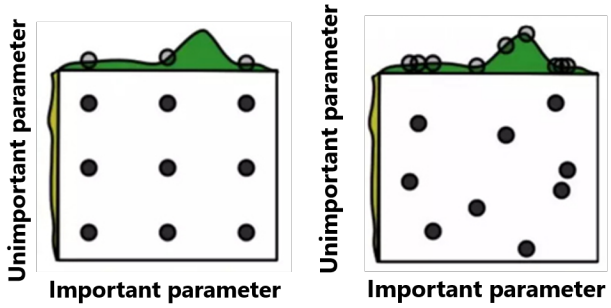
After defining the representation of the network structure, we need to find a best-performing architecture from a large search space. Such a process can be thought of as the optimization of the operations and connections of each node. Hence, in this paper, we consider the process of searching a network architecture as hyperparameter optimization (HPO), similar to optimizing learning rate, batch size, etc. We summarize the commonly used HPO algorithms as follows.

1) Grid & Random Search: Grid search and random search are the most widely used strategies of HPO. Grid search is the method that divides the space of possible hyperparameters into regular intervals (a grid), then train your model for all values on the grid, and choose the best performing one, whereas random search, as its name suggests, select a set of hyperparameters at random.

At the beginning of the study on hyperparameters search, grid search [91]–[93] is one of the most popular methods as it is simple to implement in parallel and tends to find as good as or better hyperparameters than manual search in the same amount of time, but it also has some drawbacks. As we can see from Fig. 11, grid search can only test three distinct configurations for nine trials, whereas random search can test more possible hyperparameters. In [94], it has been proved that not all hyperparameters are equally important to tune, but grid search allocates too many trials to the exploration of unimportant hyperparameters. To exploit well-performing region of hyperparameter space, Hsu et al. [95] recommend using a coarse grid first, and after finding a better region on the gird, they implement a finer grid search on that region. Similarly, Hesterman et al. [96] propose a contracting-grid search algorithm. It first computes the likelihood of each point in the grid, and a new grid is generated centered on the maximum likelihood value. The separation of points in the new grid is reduced to half of that of the old grid. This procedure is repeated for a fixed number of iterations to converge to a local minimum.

Although Bergstra and Bengio [94] show empirically and theoretically that random search is more practical and efficient for HPO than grid search, there exists a problem that it is difficult to determine whether the best set of hyperparameters is found, as it is generally accepted that the longer the search time is, the more likely it is to find the optimal hyperparameters. To alleviate this problem, Li and Jamieson et al. [97] introduce a novel search algorithm, namely *hyperband*, which trades off between resource budgets and performance. Hyperband only allocates the limited resources (like time or CPUs) to the most promising hyperparameters by successively discarding the worst half of configuration settings long before the training process has finished.

2) Reinforcement Learning: As described in Section 1, NAS is first introduced in [5], where they train a recurrent neural network (RNN) to generate network architectures automatically using reinforcement learning (RL) technique. After that, MetaQNN [14] provides a meta-modeling algorithm using Q-learning with ϵ-greedy exploration strategy and experience replay to sequentially search neural architectures. Generally, the RL-based algorithm consists of two parts (see Fig. 1): the controller, which is an RNN and used to generate different child networks at different epoch, and the reward network, which trains and evaluates the generated child networks and uses the reward (e.g. accuracy) to update RNN controller. Although these two works have achieved state-of-the-art results of datasets including CIFAR10 and Penn Treebank (PTB [98]), [5] took 28 days and 800 K40 GPUs to search for the best-performing architecture, which is unaffordable for individual researchers and even companies, MetaQNN [14] took 10 days



(a) Grid Search (b) Random Search

Figure 11: Comparison between grid and random search of nine trials for optimizing a two-dimensional space function expresses as $f(x, y) = g(x) + h(y) \approx g(x)$, which means that the parameter in $g(x)$ (above each square) is relatively important, but the parameter in $h(y)$ (left each square) is unimportant. For grid search, nine trials only cover three different important parameter values. However, random search can explore nine distinct values of g. Therefore, random search is more likely to find the optimal combination of parameters than grid search. (from [94])

and 10 GPUs to search for different architectures trained on different datasets, including CIFAR10, CIFAR100, SVHN and MNIST.

As mentioned above, designing the entire architecture is time-consuming and requires many computing resources, that's why the above two RL-based methods are not efficient. In order to improve the efficiency, many RL-based algorithms are proposed to construct cell-based structures, including NASNet [8], BlockQNN [9] and ENAS [6]. BlockQNN finished searching the network structure in three days. Besides, ENAS has made a greater breakthrough as it only took about 10 hours using 1 GPU to search for the best architecture, which is nearly 1000× faster than [5], meanwhile, it also maintain the accuracy. The novelty of ENAS is that all child architectures are regarded as sub-graph of the predefined search space so that they can share parameters to eschew training each child model from scratch to convergence.

3) Evolutionary Algorithm: Evolutionary algorithm (EA) is a generic population-based meta-heuristic optimization algorithm, which takes inspiration from biological evolution. Compared with traditional optimization algorithms such as calculus-based methods and exhaustive methods, an evolutionary algorithm is a mature global optimization method with high robustness and wide applicability. It can effectively deal with the complex problems that traditional optimization algorithms are difficult to solve without being limited by the nature of the problem.

Different EA-based NAS algorithm may use different types of encoding schemes for network representation, so the genetic operations vary from approaches. There are two types of encoding schemes: direct and indirect. Direct encoding is a widely used method, which specifies explicitly the phenotype.

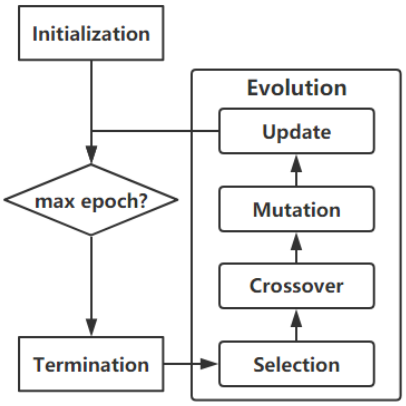


Figure 12: The overview of evolutionary algorithms.

For example, Genetic CNN [21] uses a binary encoding scheme to represent network structure, i.e. *1* means that two nodes are connected, vice verse. The advantage of binary encoding is that it can be performed easily, but its computational space is square about the number of nodes. What's worse, the number of nodes is usually limited and fixed. To represent variable-length network structures, Suganuma et al. [19] use Cartesian genetic programming (CGP) [99], [100] encoding scheme to represent the network as directed acyclic graphs, for the CNN architecture representation. In [16], individual architectures are also encoded as a graph, where vertices indicate rank-3 tensors or activations (batch normalization with ReLUs or plain linear units), and edges indicate identity connection or convolutions. Neuroevolution of augmenting topologies (NEAT) [15], [16] also uses a direct encoding scheme, where every node and every connection are stored in the DNA. Indirect encoding specifies a generation rule to build the network and allows for a more compact representation. Cellular Encoding (CE) [101] proposed by Gruau is an example of a system that utilizes indirect encoding of network structures. CE encodes a family of neural networks into a set of labeled trees, which is based on a simple graph grammar. Recently, some works [18], [102]–[104] also use indirect encoding scheme to represent the network. For example, the network in [18] is encoded by function. Each network can be modified using function-preserving network morphism operators, hence the capacity of child network is increasing and it is guaranteed the child network will behave at least as well as parent networks.

A typical evolutionary algorithm consists of the following steps: selection, crossover, mutation, and update (see Fig. 12):

- **Selection**: This step is to select a portion of the networks from all the generated networks for crossover. There are three strategies for selecting an individual network. The first is *fitness selection*, which means the probability of individual being selected is proportional to its fitness value, i.e. $P(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{N} Fitness(h_j)}$, where h_i indicates the i-th individual network. The second is *rank selection*, which is similar to fitness selection but the selection probability is proportional to relative fitness rather than absolute

fitness. *Tournament selection* [12], [16]–[18] is among the most widely used selection strategies in EA-based NAS algorithm. For each iteration, it first selects k (tournament size) individuals from the population at random. Then k individuals are sorted by their performance and the best individual is selected with probability p, while for the second-best individual, the probability is $p \times (1-p)$, and so on.

- **Crossover** After selection, every two individuals are selected to generate a new offspring, which inherit the partial genetic information of both parents. It is analogous to reproduction and biological crossover. The way of crossover varies from the encoding scheme. For binary encoding, networks are encoded as a linear string of bits so that two parent networks can be combined by one point or multiple point crossover. However, sometimes it may damage the useful information. So in [21], instead of using each bit as a unit, the basic unit in the crossover is a stage, which is a higher-level structure constructed by a binary string. For cellular encoding, a randomly selected sub-tree is cut from one parent tree and replaces a sub-tree from the other parent tree. NEAT performs artificial synapsis based on historical markings, allowing it to add new structure without losing track of which gene is which throughout a simulation.
- **Mutation**
  As the genetic information of the parents is copied and inherited to the next generation, gene mutation also occurs. A point mutation [19], [21] is one of the most widely used operations, i.e. flipping each bit independently and randomly. There are two types of mutations in [20]: one enables or disables a connection between two layers, and the other adds or removes skip connections between two nodes or layers. Real and Moore et al. [16] predefine a set of mutation operators, which includes altering learning rate and filter size, and remove skin connection between nodes, etc. Although the process of mutation may look like a mistake that causes damage to the network structure and leads to a loss of functionality, it is possible to explore more novel structures and ensure diversity.
- **Update**
  When the above steps are completed, many new networks will be generated. Generally, it is necessary to remove some networks due to limited computation resources. In [16], two individuals are selected at random, and the worst of the pair is immediately removed from the population, but in [17], the oldest individuals are removed. Some methods [19]–[21] discard all models at regular intervals. However, Liu et al. [12] do not remove any networks from the population, instead, allow it to grow with time.

4) Bayesian Optimization: In terms of the grid and random search and evolutionary algorithm, each trial of measuring the performance of one hyperparameter setting is independent. In other words, some poorly-performing regions of search space are repeatedly tested. Bayesian optimization (BO) is an algorithm that builds a probability model of the objective function, then uses this model to select the most promising hyperparameters and finally evaluates the selected hyperparameters on the true objective function. Therefore, BO can update the probability model iteratively by keeping track of past evaluation results. The probability model mathematically maps hyperparameters to a probability of a score on the objective function.

Algorithm 1 Sequential Model-Based Optimization

INPUT: $f, \mathcal{X}, S, \mathcal{M}$
$\mathcal{D} \leftarrow$ INITSAMPLES $(f, \mathcal{X})$
for $i \leftarrow |\mathcal{D}|$ to T **do**
  $p(y|\mathbf{x}, \mathcal{D}) \leftarrow$ FITMODEL $(\mathcal{M}, \mathcal{D})$
  $\mathbf{x}_i \leftarrow \arg\max_{\mathbf{x}\in\mathcal{X}} S(\mathbf{x}, p(y|\mathbf{x}, \mathcal{D}))$
  $y_i \leftarrow f(\mathbf{x}_i)$ ▷ Expensive step
  $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{x}_i, y_i)$
end for

Sequential model-based optimization (SMBO) is a succinct formalism of Bayesian optimization. The steps of SMBO is expressed in Algorithm 1 (from [105]). At first, a probability model is randomly initialized using a small portion of samples from the search space $\mathcal{X}$. D is a dataset containing sample pairs: (x_i, y_i), where $y_i = f(x_i)$ is an expensive step. The model $\mathcal{M}$ is tuned to fit the dataset D, hence a new set of hyperparameters, which obey the distribution of $\mathcal{M}$, are sequentially selected by a predefined *acquisition function* S. The acquisition function can be regarded as a cheap surrogate for the expensive objective function f. There are several types of acquisition functions: such as 1) improvement-based policies 2) optimistic policies; 3) information-based policies and 4) portfolios of acquisition functions. In terms of the type of probability model, BO algorithms can be divided into the following categories: *Gaussian Processes (GPs)* [106], [107], *Tree Parzen Estimators (TPE)* [30], and *Random Forests* [28]. Several popular open-source software libraries for Bayesian optimization are summarized in Table. I, from which one can see that the GP-based BO algorithm is the most popular.

The Bayesian optimization algorithm can be effective even if the objective function is s stochastic, non-convex, or even noncontinuous [105], but when it comes to optimizing the parameters for deep neural network, it is a different picture. Besides, although Hyperband, a bandit-based configuration optimization algorithm, can search for the promising hyperparameters within limited budgets, it lacks the guidance to ensure to converge to the best configuration as quickly as possible. To solve above problems, Falkner and Klein et al. [108] propose Bayesian Optimization-based Hyperband (BOHB), which combines the strengths of both Bayesian optimization and Hyperband and outperforms on a wide range of problem types, such as SVM, neural networks, and deep reinforcement learning. Furthermore, a faster BO procedure is introduced in [26], namely FABOLAS. It maps the validation loss and training time as a function of dataset size, i.e. a generative model is trained on a sub dataset whose size increases gradually.

Table I: List of Some Popular Open Source Software Libraries for Bayesian Optimization. GP, RF, TPE indicates Gaussian process, random forest and tree parzen estimator, respectively. ANCL indicates Academic non-commercial license.

Software	Model	License
Spearmint https://github.com/HIPS/Spearmint	GP	ANCL
Moe https://github.com/Yelp/MOE	GP	Apache 2.0
PyBO https://github.com/mwhoffman/pybo	GP	BSD
Bayesopt https://github.com/rmcantin/bayesopt	GP	GPL
SkGP https://scikit-optimize.github.io/	GP	BSD
GPyOpt http://sheffieldml.github.io/GPyOpt	GP	BSD
SMAC https://github.com/automl/SMAC3	RF	ANCL
Hyperopt http://hyperopt.github.io/hyperopt	TPE	BSD

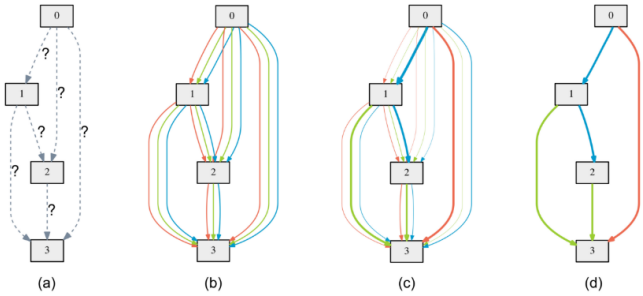


Figure 13: An overview of DARTS. (a) The direction of data can only flow from a lower-level to higher-level nodes, the operations on edges are initially unknown. (b) There is a mixture of candidate operations on each edge. (c) The weight of each operation on edges is learnable and range from 0 to 1, unlike discrete sampling method it can only be 0 or 1. (d) The final structure is constructed by preserving the operation with maximum weight value on each edge for efficiency.

As a result, FABOLAS is 10 to 100 times faster than other SOTA BO algorithms and Hyperband, meanwhile, it can also find the promising hyperparameters.

5) Gradient Descent: Although the search algorithms introduced above can all generate architectures automatically, meanwhile achieve state-of-the-art results, they search for hyperparameters in discrete way and the process of HPO is treated as a black-box optimization problem, which causes that many sets of parameters need to be evaluated and therefore requires a significant amount of time and computational resources. Many attempts ([10], [109]–[113]) have been made to solve the problems encountered in the discrete sample-based approach. Specifically, Liu et al. [10] propose to search for hyperparameters over continuous and differentiable search space by relaxing the discrete choice of hyperparameters using softmax function:

$$\overline{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\alpha_{o'}^{(i,j)}\right)} o(x) \tag{1}$$

where $o(x)$ indicates the operation for input x, such as convolution or pooling, $\alpha_o^{(i,j)}$ indicates the weight for the operation o between a pair of nodes (i,j), i.e $x_j = \alpha_o^{(i,j)} o(x_i)$, and $\mathcal{O}$ is a set of predefined candidate operations. After the relaxation, the task of searching for hyperparameters is simplified to the optimization of weight matrix $\alpha^{(i,j)}$, as illustrated in Fig. 13.

Gradient descent-based methods can reduce much time spent searching hyperparameters, but the requirements of GPU memory grows linearly to the number of candidate operations. In Fig. 13, there are three different candidate operations on each edge, hence for improving efficiency, only the operation with the highest weight value (greater than 0) of each edge can be retained. For saving memory consumption, In [114], a new path-level pruning method is introduced. At first, an over-parameterized network containing all possible operations is trained, and then the redundant parts are removed gradually, in which case only one single network needs training. However, a network containing all operations takes up lots of memory. Therefore, the architecture parameters are binarized and only one path is activated at training time. Hundt et al. [115] propose SharpDARTS, which use a general, balanced and consistent design, namely SharpSepConv Block. They also introduce *Differentiable Hyperparameters Grid Search* and *HyperCuboid search space*, leading to 50% faster than DARTS, meanwhile, the accuracy of the final model generated on CIFAR10 has been improved by 20% to 30%.

V. Model Estimation

Once a new network is generated, we have to evaluate the performance of this network. An intuitive method is to train the network to convergence and then judge whether it is good or bad according to the final result. However, this method requires a lot of time and computing resources. For example, in [5], a new network is generated in each epoch, then trained to converge before it can be evaluated, so finally, this method took 800 K40 GPUs and 28 days in total. Additionally, NASNet [8] and AmoebaNet [17] took 500 P100 GPUs and 450 K40 GPUs to discover architectures, respectively. To accelerate the process of model evaluation, several algorithms have been proposed, which are summarized as follows:

A. Low fidelity

Since the training time is highly related to the dataset and model size, we can accelerate model evaluation from different perspectives. On the one hand, we can reduce the number of images or the resolution of images (for image classification tasks). For example, FABOLAS [26] trains the model on a subset of the training set to accelerate model estimation. In [116], ImageNet64×64 and its variants 32×32, 16×16 are provided, at the same time these lower resolution dataset can remain the similar characteristics of the origin ImageNet dataset.

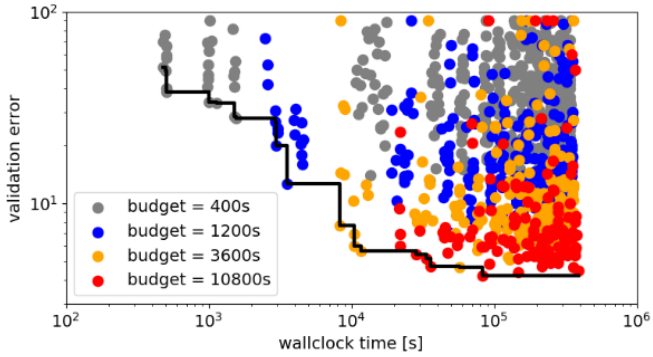


Figure 14: Validation error of each configuration generated on different budgets during the whole HPO procedure. For example, each grey point represents the validation error of the corresponding network whose configuration is obtained with a 400-second search. At last, the networks that take 3600 seconds (orange points) to search performs as well as one that takes 10800 seconds (red points).

On the other hand, low fidelity model evaluation can be realized by reducing the model size, such as training with less number of filters per layer [8], [17]. Similar to ensemble learning, [117] propose the *Transfer Series Expansion (TSE)* that constructs an ensemble estimator by linearly combining a series of base low fidelity estimators, hence it solves the problem that a single low fidelity estimator can be badly biased. Furthermore, Zela et al. [25] empirically demonstrate that there is a weak correlation between the performance after short and long training time, hence we do not need to spend too much time searching network configurations (see Fig. 14).

B. Transfer learning

At the beginning of the development of NAS algorithms, like [5], each network architecture, which is trained for a long time to converge, is dropped after evaluating its performance. Hence, the technique of transfer learning is used to accelerate the process of NAS. For example, Wong and Lu et al. [118] propose Transfer Neural AutoML that uses knowledge from prior tasks to speed up network design. ENAS [6] shares parameters among child networks, leading to 1000 times faster than [5]. The network morphism based algorithms [13], [18], [119] can also inherit the weights of previous architectures.

C. Surrogate

Surrogate-based method [120]–[122] is another powerful tool that approximates to the black box function. In general, once a good approximation is obtained, it is quite easier to find the best configurations than directly optimizing the original expensive objective. For example, PNAS [11] introduces a surrogate model to control the way of searching. Although ENAS [6] is very efficient, the number of models evaluated by PNAS is over 5 times than ENAS, and PNAS is 8 times faster when it comes to total compute. However, this method is not applicable when the optimization space is too large and hard to quantize, and the evaluation of each configuration is extremely expensive [123].

D. Early stopping

Early stopping is initially used to prevent overfitting in classical machine learning and is now being used to speed up model evaluation by stopping the evaluations which predicted to perform poorly on the validation set [124]–[126]. For example, [126] propose a learning curve model which is a weighted combination of a set of parametric curve models selected from the literature, which makes it possible to predict the performance of the network. Furthermore, [127] present a novel approach of early stopping based on fast-to-compute local statistics of the computed gradients, which no longer relies on the validation set as previous methods, but can allow the optimizer to make full use of all training data.

VI. NAS Performance Summary

Recently, NAS has become a very hot research topic, and various types of algorithms have been proposed, so it is necessary to summarize and compare different types algorithms to help readers have a better and more comprehensive understanding of NAS methods. We choose to summarize several popular types of NAS algorithms, including random search (RS), reinforcement learning (RL), evolutionary algorithm (EA) and Gradient Descent (GD) based algorithms. Besides, there are several model variants for each algorithm, and each variant may correspond to different dataset or model size. To measure the performance of these methods in a clear way, we consider several factors: result (like accuracy), time and resources for searching and the size of the generated model. Because the number of GPUs used by each algorithm is different, it is not fair to evaluate the efficiency of the algorithm only according to the searching time. Therefore, we use *GPU Days* to measure the efficiency of different algorithms, which is defined as:

$$\text{GPU Days} = N \times D \tag{2}$$

where N represents the number of GPU, D represents the practical number of days spent searching. The performance of different algorithms is presented in Table. II.

However, it is still difficult to make a fair comparison of these NAS methods because the version of GPU is not introduced in those literature. To get a general and intuitive understanding of the performance of these methods, we ignore the impact of GPU version and draw Fig. 15, from which one can find that the results of these approaches on CIFAR10 are very close, while GD-based methods are more efficient as they can not only take much less time and resources to find a best-performing architecture. Instead, EA tends to require a large amount of time and resource for searching, which might be attributed to the fact that it needs to search and evaluate lots of child networks at the same time. Another surprising finding is that random search based algorithms can also achieve comparable results.

So far, we have a certain understanding of NAS-based models and their performance, but we don't know if it is

Table II: The summary of the performance of different NAS algorithms. The *Metric(%)* for different tasks is different. For example, the metric of image classification task (like CIFAR10) is *accuracy*, which is the default metric in this table if without special declaration. For ImageNet dataset, the metric indicates the top-1 and top-5 accuracy. For other type of task, the metric is presented in the table. *GPU Days* (defined by Equation. 2) is used to evaluate the efficiency of the algorithm by considering both time and *Resource(GPUs)*, while *Resource(GPUs)* indicates the number of GPU used during the procedure of searching, and *Parameters(million)* indicates the size of the generated model. A dash in the table indicates data was either not available or not applicable.

Method	Model	Dataset	Metric(%)	GPU Days	Resource(GPUs)	Parameters(million)
Reinforcement Learning	RL NAS [5]	CIFAR10	96.35	22400	800 K40	37.4
			95.53			7.1
	MetaQNN [14]	CIFAR10	93.08	100	10	-
		CIFAR100	72.86			
		SVHN	97.94			
	NASNet [8]	CIFAR10	97.35	2000	500 P100	3.3
			97.60			27.6
		ImageNet	82.70/96.20	-	-	88.9
			74.00/91.60			5.3
	BlockQNN [8]	CIFAR10	97.65	96	32	33.3
		CIFAR100	85.17			
		CIFAR10	81.94	0.8	0.8	3.9
		CIFAR100	96.43			
		ImageNet	82.0/96.0	-	-	91
		COCO	70.5 AP	5	-	-
	ENAS [6]	CIFAR10	97.11	0.45	1	4.6
			96.13	0.32	1	38
		PTB	55.8 perplexity	0.42	1	24
Evolutionary Algorithm	CoDeepNEAT [20]	CIFAR10	92.70	-	-	-
		PTB	78 perplexity	-	1 GTX 980	
		COCO	BLEU-4 = 29.1 CIDEr = 88.0 METEOR = 23.8	154	100	
	GeNet [21]	MNIST	99.66	2	-	-
		CIFAR10	92.90	17		
		CIFAR100	70.97	-		
		SVHN	98.03	-		
		ImageNet	72.13/90.26	20	-	156
	Large-Scale [16]	CIFAR10	94.60	2500	250	5.4
		CIFAR100	77.00	-	-	40.4
	CGP(ConvSet)	CIFAR10	93.25	30.4	2	5.7
	CGP(ResSet)		94.02	27.4	2	6.4
	AmoebaNet-B [17]	CIFAR10	97.45	3150	450 K40	2.8
			97.87	-	-	34.9
	AmoebaNet-C	ImageNet	83.10/96.30	-	-	155.3
	AmoebaNet-C-mobile	ImageNet	75.10/92.40			5.1
	Lemonade [18]	CIFAR10	96.40	56	8 Titan	3.4
	Hierarchical [12]	CIFAR10	96.37	300	200	15.7
		ImageNet	79.7/94.8	-	-	64
Gradient Descent	DARTS [10]	CIFAR10	97.06	4	4 GTX 1080Ti	2.9 (first-order)
			97.27	4	4	3.4 (second-order)
		ImageNet	73.10/91.00	-	-	4.9
		PTB	60.5 perplexity	0.5	1	23 (first-order)
			56.1 perplexity	1	4	23 (second-order)
		WikiText-2	66.90 perplexity	-	-	33
	Proxyless [114]	CIFAR10	97.92	-	-	5.7
		ImageNet	74.6/92.2	8	-	-
	MaskConnect [22]	CIFAR100	73.15	-	8	32.1
			75.89	-	8	0.81
		ImageNet	79.80/94.80	-	8	-
	sharpDARTS [115]	CIFAR10	98.07	0.8	1 RTX 2080Ti	3.6
		ImageNet	74.9/92.2	0.8	-	4.9
Random Search	DARTS [10]	CIFAR10	96.51	-	-	3.1
		PTB	61.5 perplexity	-	-	23
	NAO [128]	CIFAR10	96.47	0.3	1 V100	2.5 (parameter sharing)
			97.89	200	200 V100	128
		CIFAR100	84.33	-	-	10.8
			85.25	-	-	128
		PTB	56.6 perplexity	0.4	200 V100	27 (parameter sharing)
			56.0 perplexity	300	200 V100	27
		WikiText-2	67 perplexity	-	-	36
	Hierarchical [12]	CIFAR10	96.09	8	200	15.7
		ImageNet	79.0/94.5	-	-	64
	RS NAS [129]	CIFAR10	97.15	2.7	-	4.3
		PTB	55.5 perplexity	0.25	-	-

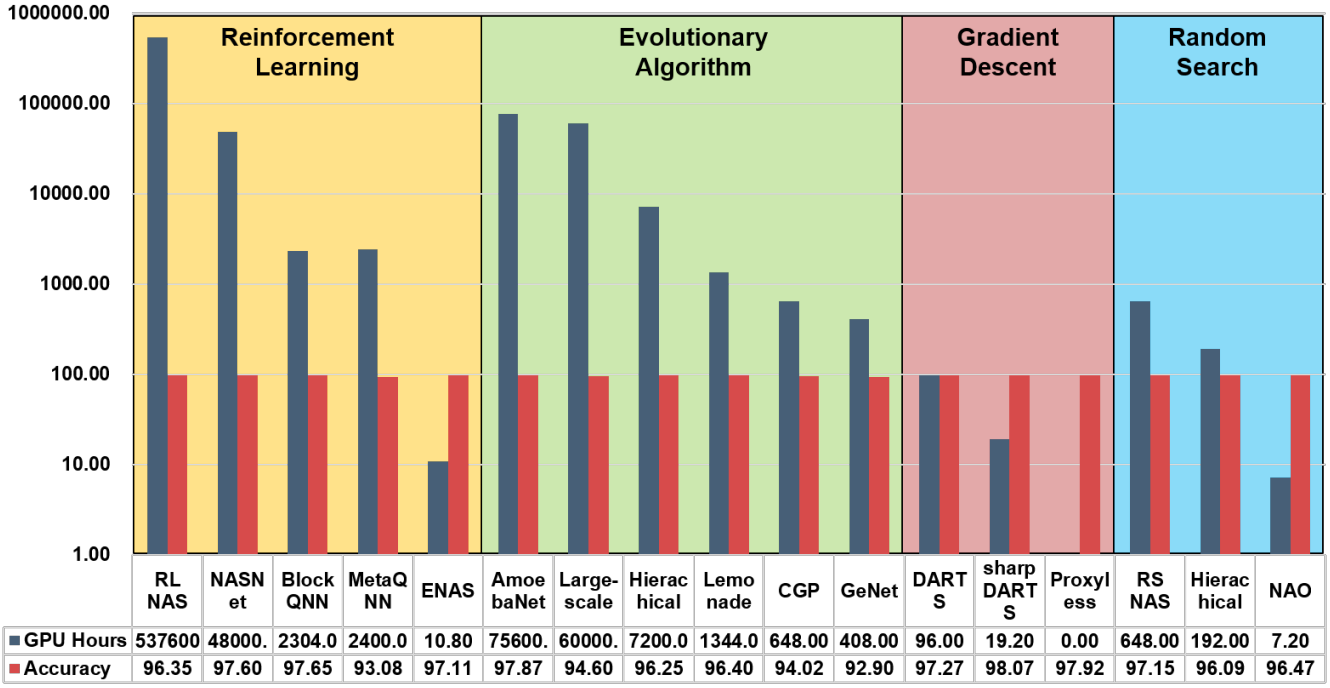


Figure 15: The performance comparison of different NAS algorithms on CIFAR10. There are four types of NAS algorithms, and for each NAS algorithm, we select the best result to compare. GPU Hours = GPU Days $\times$ 24 (the search time is not provided in Proxyless [114]).

better than manually designed models. To figure it out, we use CIFAR10 and PTB datasets to compare the models automatically generated with the models designed by human experts, because these two datasets are one of most commonly used baseline datasets for object classification and language modeling task, respectively. We refer to the data from the website *paperswithcode.com* [3] and draw the Fig. 16, where figure (a) presents currently top-performing models on CIFAR10 dataset. GPIPE [130] achieves the best result on CIFAR10 based on AmoebaNet-B [17] and one can easily see that the models generated automatically have already outperformed the handcrafted models (SENet [131] and WRN [132]). In terms of language modeling task, there is still a big gap between automatically generated models and the models designed by experts. As Fig. 16 (b) shows, the first four models that perform best on PTB dataset all manually designed, i.e. GPT-2 [133], FRAGE [134], AWD-LSTM-DOC [135] and Transformer-XL [136].

VII. Open Problems and Future Work

Currently, a growing number of researchers are focusing on AutoML and a lot of excellent works are proposed to solve different problems. But there are still many problems that need to be solved in theoretical and practical aspects. We summarize these problems as follows.

[3]https://paperswithcode.com/sota

A. Complete AutoML Pipeline

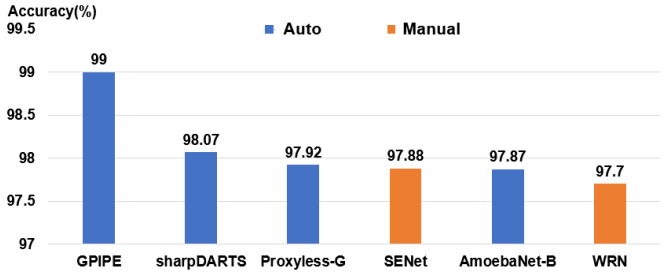
Although there are several pipeline libraries proposed, such as TPOT [137] and Auto-Sklearn [138], they all lack the procedure of data collection, a process that is usually finished manually and therefore time-consuming and tedious. Additionally, few AutoML pipelines incorporate automated feature engineering due to the complexity of combining each process dynamically. However, in the long term, the ultimate aim is to optimize every process in Fig. 2 and integrate then into a complete system.

B. Interpretability

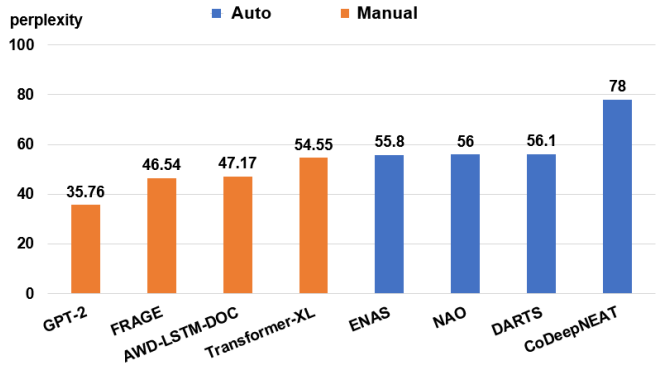
For the process of model generation, the algorithm itself can usually find better configuration settings than human. However, there is no scientific and formal evidence to prove why some specific operations perform better. For example, in BlockQNN [9], the block tends to choose "Concat" operation instead of elemental addition at the last layer, but elemental addition operations are common in the exploitation stage. Besides, the 5×5 convolutional filter is not often used, neither. All these results are usually explained abstractly and intuitively, but lack rigorous mathematical proof. Therefore, the interpretability of AutoML is also an important research direction.

C. Reproducibility

As we all know, a big problem with machine learning is its non-reproducibility. AutoML is no exception, especially for NAS research, because the source code of many algorithms is not available. Even if the source code is provided, it is still



(a) The leaderboard of models on CIFAR10



(b) The leaderboard of models on PTB

Figure 16: The state-of-the-art models for CIFAR10 and PTB dataset. Blue bar represents the model automatically generated, and orange bar represents the model designed by human experts. Best viewd in color.

hard to reproduce the results because some approaches require even months of searching time, like [5]. NASBench [139] goes a step further and is a pioneering work to alleviate the problem of non-reproducibility. It provides a tabular dataset that contains 423,624 unique neural networks. These networks are generated and evaluated from a fixed graph-based search space and mapped to their trained and evaluated performance on CIFAR10. NASBench allows us to perform reproducible NAS experiments within seconds. Hence, fostering reproducibility for all process of AutoML pipeline would be desirable.

D. Flexible Encoding Scheme

A big difference between NAS algorithms is the structure encoding scheme, which is all predefined by human. So one interesting question is that is there a more general way of representing a network architecture and primitive operation? Because by reviewing the existing NAS algorithms, we can find that all the primitive operations and encoding schemes rely on the human experience. In other words, currently, it is unlikely that a new primitive operation (like convolution or pooling) or a novel network architecture (like transformer [136]) can be generated automatically.

E. More Area

As described in Section VI, most of NAS algorithms only focus on generating CNN for image classification or RNN for language modeling. Some methods have outperformed handcraft models on CIFAR10 dataset, but when it comes to PTB, there is still a long way to go. Besides, some works are proposed to solve the task of object detection [140] and semantic segmentation [141], [142]. Therefore, the exploration of more uncovered areas is another crucial future research.

F. Lifelong Learn

Last but not least, one can find that the majority of AutoML algorithms only focus on solving a specific task on some fixed datasets, e.g. the task of image classification on CIFAR10 and ImageNet. However, in the long run, a high-quality AutoML system should be able to lifelong learning, which can be understood from two different perspectives. On the hand, the system should be capable of reusing prior knowledge to solve new tasks (i.e. learning to learn). For example, a child can quickly identify tigers, rabbits, and elephants after seeing several pictures of them, but for current machine learning algorithm, it is should be trained on a large number of images to do so. A hot topic in this aspect is meta-learning, which aims to design models for new tasks using previous experience and has been studied for a long time. Based on the work of [143] published in 1976, K. A. Smith-Miles [144](2009) presents a unified framework to generalize the meta-learning concept to cross-disciplinary studies, exposing the similarities and differences of the approaches. Recently, several studies have started to use meta-learning technique to faster the process of searching for the neural architectures and succeeded to find the network for only one shot [7], [145]. On the other hand, it is the ability to constantly learning new data, meanwhile preserving the information of old data, that the AutoML system should be equipped with. However, based on the observation of the current neural network model, we can see that once we use other datasets to train the model, the performance of the model on the previous data sets will be greatly reduced. Incremental learning is a useful method to alleviate this situation. Li and Hoiem [146] propose Learning without Forgetting (LwF) method, which trains the model only using new data, while preserving the original capabilities. iCaRL [147] is a new training strategy based on LwF, which only uses a small part of old data to pretrained the information and gradually increases the number of new class of data to train the model.

VIII. Conclusions

In this paper, we provide a detailed and systematical review of AutoML according to the pipeline of machine learning, ranging from data preparation to model estimation. Additionally, since NAS has been a really hot topic recently, we also summarize the existing NAS algorithms clearly according to several factors: the baseline dataset and corresponding result, the time and resource cost for searching and the size of the best-performing model. After that, we provide several interesting and important open problems to discuss some valuable research directions. Although the research on AutoML is still in its infancy, we believe that the above problems will be solved efficiently in the future and hope that this survey can give

a comprehensive and clear understanding of AutoML to the beginners and make contributions to the future research.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *International Conference on Neural Information Processing Systems*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," 2018.

[4] Y. Quanming, W. Mengshuo, J. E. Hugo, G. Isabelle, and Y. Yang, "Taking human out of learning applications: A survey on automated machine learning," 2018.

[5] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning." [Online]. Available: http://arxiv.org/abs/1611.01578

[6] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," vol. ICML. [Online]. Available: http://arxiv.org/abs/1802.03268

[7] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: ONE-SHOT MODEL ARCHITECTURE SEARCH THROUGH HYPERNETWORKS," p. 22.

[8] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition." [Online]. Available: http://arxiv.org/abs/1707.07012

[9] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation." [Online]. Available: http://arxiv.org/abs/1708.05552

[10] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search." [Online]. Available: http://arxiv.org/abs/1806.09055

[11] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search." [Online]. Available: http://arxiv.org/abs/1712.00559

[12] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *ICLR*, p. 13.

[13] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *arXiv preprint arXiv:1511.05641*, 2015.

[14] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," vol. ICLR. [Online]. Available: http://arxiv.org/abs/1611.02167

[15] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," vol. 10, no. 2, pp. 99–127. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/106365602320169811

[16] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers." [Online]. Available: http://arxiv.org/abs/1703.01041

[17] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search." [Online]. Available: http://arxiv.org/abs/1802.01548

[18] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution." [Online]. Available: http://arxiv.org/abs/1804.09081

[19] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures." [Online]. Available: http://arxiv.org/abs/1704.00764

[20] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks." [Online]. Available: http://arxiv.org/abs/1703.00548

[21] L. Xie and A. Yuille, "Genetic CNN," vol. ICCV. [Online]. Available: http://arxiv.org/abs/1703.01513

[22] K. Ahmed and L. Torresani, "MaskConnect: Connectivity learning by gradient descent." [Online]. Available: http://arxiv.org/abs/1807.11473

[23] R. Shin, C. Packer, and D. Song, "DIFFERENTIABLE NEURAL NETWORK ARCHITECTURE SEARCH," p. 4.

[24] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," p. 8.

[25] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search." [Online]. Available: http://arxiv.org/abs/1807.06906

[26] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets." [Online]. Available: http://arxiv.org/abs/1605.07079

[27] S. Falkner, A. Klein, and F. Hutter, "PRACTICAL HYPERPARAMETER OPTIMIZATION FOR DEEP LEARNING," p. 5.

[28] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*, C. A. C. Coello, Ed. Springer Berlin Heidelberg, vol. 6683, pp. 507–523. [Online]. Available: http://link.springer.com/10.1007/978-3-642-25566-3_40

[29] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," p. 10.

[30] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," p. 9.

[31] M.-A. Zöller and M. F. Huber, "Survey on automated machine learning," *arXiv preprint arXiv:1904.12054*, 2019.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[33] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[34] "Kaggle," 2019. [Online]. Available: https://www.kaggle.com/

[35] "Google dataset search," 2019. [Online]. Available: https://toolbox.google.com/datasetsearch

[36] "Elsevier datasearch," 2019. [Online]. Available: https://www.datasearch.elsevier.com/

[37] "torchvision," 2019. [Online]. Available: https://pytorch.org/docs/stable/torchvision/transforms.html#transforms-on-pil-image

[38] "Augmentor — augmentor 0.2.3 documentation," 2019. [Online]. Available: https://augmentor.readthedocs.io/en/master/

[39] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?" *arXiv preprint arXiv:1609.08764*, 2016.

[40] Z. Xie, S. I. Wang, J. Li, D. Lévy, A. Nie, D. Jurafsky, and A. Y. Ng, "Data noising as smoothing in neural network language models," *arXiv preprint arXiv:1703.02573*, 2017.

[41] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, "Qanet: Combining local convolution with global self-attention for reading comprehension," *arXiv preprint arXiv:1804.09541*, 2018.

[42] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[43] T. M. Deist, A. Patti, Z. Wang, D. Krane, T. Sorenson, and D. Craft, "Simulation-assisted machine learning," *Bioinformatics*, 03 2019. [Online]. Available: https://doi.org/10.1093/bioinformatics/btz199

[44] N. Ruiz, S. Schulter, and M. Chandraker, "Learning to simulate," *arXiv preprint arXiv:1810.02513*, 2018.

[45] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[46] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *arXiv preprint arXiv:1812.04948*, 2018.

[47] J. Eno and C. W. Thompson, "Generating synthetic data to match data mining patterns," *IEEE Internet Computing*, vol. 12, no. 3, pp. 78–82, 2008.

[48] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[49] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," *arXiv preprint arXiv:1406.2227*, 2014.

[50] A. Gupta, A. Vedaldi, and A. Zisserman, "Synthetic data for text localisation in natural images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2315–2324.

[51] T.-H. Oh, R. Jaroensri, C. Kim, M. Elgharib, F. Durand, W. T. Freeman, and W. Matusik, "Learning-based video motion magnification," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 633–648.

[52] J. Yang, X. Sun, Y. K. Lai, L. Zheng, and M. M. Cheng, "Recognition from Web Data: A Progressive Filtering Approach," pp. 5303–5315, 2018.

[53] X. Chen, A. Shrivastava, and A. Gupta, "Neil: Extracting visual knowledge from web data," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 1409–1416.

[54] Y. Xia, X. Cao, F. Wen, and J. Sun, "Well begun is half done: Generating high-quality seeds for automatic image dataset construction from web," in *European Conference on Computer Vision*. Springer, 2014, pp. 387–400.

[55] N. H. Do and K. Yanai, "Automatic construction of action datasets using web videos with density-based cluster analysis and outlier detection," in *Pacific-Rim Symposium on Image and Video Technology*. Springer, 2015, pp. 160–172.

[56] J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev, T. Duerig, J. Philbin, and L. Fei-Fei, "The unreasonable effectiveness of noisy data for fine-grained recognition," in *European Conference on Computer Vision*. Springer, 2016, pp. 301–320.

[57] P. D. Vo, A. Ginsca, H. Le Borgne, and A. Popescu, "Harnessing noisy web images for deep representation," *Computer Vision and Image Understanding*, vol. 164, pp. 68–81, 2017.

[58] B. Collins, J. Deng, K. Li, and L. Fei-Fei, "Towards scalable dataset construction: An active learning approach," in *European conference on computer vision*. Springer, 2008, pp. 86–98.

[59] Y. Roh, G. Heo, and S. E. Whang, "A survey on data collection for machine learning: a big data - AI integration perspective." [Online]. Available: http://arxiv.org/abs/1811.03402

[60] D. Yarowsky, "Unsupervised word sense disambiguation rivaling supervised methods," in *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1995, pp. 189–196.

[61] I. Triguero, J. A. Sáez, J. Luengo, S. García, and F. Herrera, "On the characterization of noise filters for self-training semi-supervised in nearest neighbor classification," *Neurocomputing*, vol. 132, pp. 30–41, 2014.

[62] M. F. A. Hady and F. Schwenker, "Combining committee-based semi-supervised learning and active learning," *Journal of Computer Science and Technology*, vol. 25, no. 4, pp. 681–698, 2010.

[63] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of the eleventh annual conference on Computational learning theory*. ACM, 1998, pp. 92–100.

[64] Y. Zhou and S. Goldman, "Democratic co-learning," in *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*. IEEE, 2004, pp. 594–602.

[65] J. Yang, X. Sun, Y.-K. Lai, L. Zheng, and M.-M. Cheng, "Recognition from web data: a progressive filtering approach," *IEEE Transactions on Image Processing*, vol. 27, no. 11, pp. 5303–5315, 2018.

[66] X. Chen and A. Gupta, "Webly supervised learning of convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1431–1439.

[67] Z. Xu, S. Huang, Y. Zhang, and D. Tao, "Augmenting strong supervision using web data for fine-grained categorization," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2524–2532.

[68] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[69] H. Guo and H. L. Viktor, "Learning from imbalanced data sets with boosting and data generation: the databoost-im approach," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 30–39, 2004.

[70] S. Krishnan and E. Wu, "Alphaclean: Automatic generation of data cleaning pipelines," *arXiv preprint arXiv:1904.11827*, 2019.

[71] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "Katara: A data cleaning system powered by knowledge bases and crowdsourcing," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1247–1261.

[72] S. Krishnan, M. J. Franklin, K. Goldberg, J. Wang, and E. Wu, "Activeclean: An interactive data cleaning framework for modern machine learning," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 2117–2120.

[73] S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, T. Kraska, T. Milo, and E. Wu, "Sampleclean: Fast and reliable analytics on dirty data." *IEEE Data Eng. Bull.*, vol. 38, no. 3, pp. 59–75, 2015.

[74] H. Motoda and H. Liu, "Feature selection, extraction and construction," *Communication of IICM (Institute of Information and Computing Machinery, Taiwan) Vol*, vol. 5, no. 67-72, p. 2, 2002.

[75] M. Dash and H. Liu, "Feature selection for classification," *Intelligent data analysis*, vol. 1, no. 1-4, pp. 131–156, 1997.

[76] M. J. Pazzani, "Constructive induction of cartesian product attributes," in *Feature Extraction, Construction and Selection*. Springer, 1998, pp. 341–354.

[77] Z. Zheng, "A comparison of constructing different types of new feature for decision tree learning," in *Feature Extraction, Construction and Selection*. Springer, 1998, pp. 239–255.

[78] J. Gama, "Functional trees," *Machine Learning*, vol. 55, no. 3, pp. 219–250, 2004.

[79] H. Vafaie and K. De Jong, "Evolutionary feature space transformation," in *Feature Extraction, Construction and Selection*. Springer, 1998, pp. 307–323.

[80] P. Sondhi, "Feature construction methods: a survey," *sifaka. cs. uiuc. edu*, vol. 69, pp. 70–71, 2009.

[81] D. Roth and K. Small, "Interactive feature space construction using semantic information," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 2009, pp. 66–74.

[82] Q. Meng, D. Catchpoole, D. Skillicom, and P. J. Kennedy, "Relational autoencoder for feature extraction," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 364–371.

[83] O. Irsoy and E. Alpaydın, "Unsupervised feature extraction with autoencoder trees," *Neurocomputing*, vol. 258, pp. 63–73, 2017.

[84] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[85] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[86] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 764–773.

[87] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," *arXiv preprint arXiv:1901.02985*, 2019.

[88] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[89] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[90] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[91] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 473–480.

[92] H. H. Hoos, *Automated Algorithm Configuration and Parameter Tuning*, 2011.

[93] I. Czogiel, K. Luebke, and C. Weihs, *Response surface methodology for optimizing hyper parameters*. Universitätsbibliothek Dortmund, 2006.

[94] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," p. 25.

[95] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[96] J. Y. Hesterman, L. Caucci, M. A. Kupinski, H. H. Barrett, and L. R. Furenlid, "Maximum-likelihood estimation with a contracting-grid search algorithm," *IEEE transactions on nuclear science*, vol. 57, no. 3, pp. 1077–1084, 2010.

[97] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization." [Online]. Available: http://arxiv.org/abs/1603.06560

[98] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, "The penn treebank: annotating predicate argument structure," in *Proceedings of the workshop on Human Language Technology*. Association for Computational Linguistics, 1994, pp. 114–119.

[99] J. F. Miller and S. L. Harding, "Cartesian genetic programming," in *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*. ACM, 2008, pp. 2701–2726.

[100] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in cartesian genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.

[101] F. Gruau, "Cellular encoding as a graph grammar," in *IEEE Colloquium on Grammatical Inference: Theory, Applications & Alternatives*, 1993.

[102] C. Fernando, D. Banarse, M. Reynolds, F. Besse, D. Pfau, M. Jaderberg, M. Lanctot, and D. Wierstra, "Convolution by evolution: Differentiable pattern producing networks," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, 2016, pp. 109–116.

[103] M. Kim and L. Rigazio, "Deep clustered convolutional kernels," in *Feature Extraction: Modern Questions and Challenges*, 2015, pp. 160–172.

[104] J. K. Pugh and K. O. Stanley, "Evolving multimodal controllers with hyperneat," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 735–742.

[105] M. M. Ian Dewancker and S. Clark, "Bayesian optimization primer." [Online]. Available: https://app.sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf

[106] J. González, "Gpyopt: A bayesian optimization framework in python," http://github.com/SheffieldML/GPyOpt, 2016.

[107] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[108] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," p. 10.

[109] S. Saxena and J. Verbeek, "Convolutional neural fabrics," in *Advances in Neural Information Processing Systems*, 2016, pp. 4053–4061.

[110] K. Ahmed and L. Torresani, "Connectivity learning in multi-branch networks," *arXiv preprint arXiv:1709.09582*, 2017.

[111] R. Shin, C. Packer, and D. Song, "Differentiable neural network architecture search," 2018.

[112] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International Conference on Machine Learning*, 2015, pp. 2113–2122.

[113] F. Pedregosa, "Hyperparameter optimization with approximate gradient," *arXiv preprint arXiv:1602.02355*, 2016.

[114] S. H. Han Cai, Ligeng Zhu, "PROXYLESSNAS: DIRECT NEURAL ARCHITECTURE SEARCH ON TARGET TASK AND HARDWARE," 2019.

[115] G. D. H. Andrew Hundt, Varun Jain, "sharpDARTS: Faster and More Accurate Differentiable Architecture Search," Tech. Rep. [Online]. Available: https://arxiv.org/pdf/1903.09900.pdf

[116] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the CIFAR datasets," *CoRR*, vol. abs/1707.08819, 2017. [Online]. Available: http://arxiv.org/abs/1707.08819

[117] Y.-q. Hu, Y. Yu, W.-w. Tu, Q. Yang, Y. Chen, and W. Dai, "Multi-Fidelity Automatic Hyper-Parameter Tuning via Transfer Series Expansion ," p. 8, 2019.

[118] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural automl," in *Advances in Neural Information Processing Systems*, 2018, pp. 8356–8365.

[119] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *International Conference on Machine Learning*, 2016, pp. 564–572.

[120] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Surrogate benchmarks for hyperparameter optimization." in *MetaSel@ ECAI*, 2014, pp. 24–31.

[121] C. Wang, Q. Duan, W. Gong, A. Ye, Z. Di, and C. Miao, "An evaluation of adaptive surrogate modeling based optimization with two benchmark problems," *Environmental Modelling & Software*, vol. 60, pp. 167–179, 2014.

[122] K. Eggensperger, F. Hutter, H. Hoos, and K. Leyton-Brown, "Efficient benchmarking of hyperparameter optimizers via surrogates," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[123] K. K. Vu, C. D'Ambrosio, Y. Hamadi, and L. Liberti, "Surrogate-based methods for black-box optimization," *International Transactions in Operational Research*, vol. 24, no. 3, pp. 393–424, 2017.

[124] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," 2016.

[125] B. Deng, J. Yan, and D. Lin, "Peephole: Predicting network performance before training," *arXiv preprint arXiv:1712.03351*, 2017.

[126] T. Domhan, J. T. Springenberg, and F. 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.

[127] M. Mahsereci, L. Balles, C. Lassner, and P. Hennig, "Early stopping without a validation set," *arXiv preprint arXiv:1703.09580*, 2017.

[128] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Advances in neural information processing systems*, 2018, pp. 7816–7827.

[129] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," *arXiv preprint arXiv:1902.07638*, 2019.

[130] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *arXiv preprint arXiv:1811.06965*, 2018.

[131] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[132] H. Zhang, Y. N. Dauphin, and T. Ma, "Fixup initialization: Residual learning without normalization," *arXiv preprint arXiv:1901.09321*, 2019.

[133] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, p. 8, 2019.

[134] C. Gong, D. He, X. Tan, T. Qin, L. Wang, and T.-Y. Liu, "Frage: frequency-agnostic word representation," in *Advances in Neural Information Processing Systems*, 2018, pp. 1334–1345.

[135] S. Takase, J. Suzuki, and M. Nagata, "Direct output connection for a high-rank language model," *arXiv preprint arXiv:1808.10143*, 2018.

[136] Z. Dai, Z. Yang, Y. Yang, W. W. Cohen, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[137] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore, *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*. Springer International Publishing, 2016, ch. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pp. 123–137. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-31204-0_9

[138] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

[139] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards Reproducible Neural Architecture Search," *arXiv e-prints*, Feb 2019.

[140] G. Ghiasi, T. Y. Lin, R. Pang, and Q. V. Le, "Nas-fpn: Learning scalable feature pyramid architecture for object detection," 2019.

[141] C. Liu, L. C. Chen, F. Schroff, H. Adam, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," 2019.

[142] Y. Weng, T. Zhou, Y. Li, and X. Qiu, "Nas-unet: Neural architecture search for medical image segmentation," *IEEE Access*, vol. 7, pp. 44 247–44 257, 2019.

[143] J. R. Rice, "The algorithm selection problem," in *Advances in computers*. Elsevier, 1976, vol. 15, pp. 65–118.

[144] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, p. 6, 2009.

[145] X. Zhang, Z. Huang, and N. Wang, "You only search once: Single shot neural architecture search via direct sparse optimization," *arXiv preprint arXiv:1811.01567*, 2018.

[146] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2018.

[147] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.