Optimization of deep neural networks: a survey and unified taxonomy

El-Ghazali Talbi

Optimization of deep neural networks: a survey and unified taxonomy

EL-GHAZALI TALBI, University of Lille and INRIA

During the last years, research in applying optimization approaches in the automatic design of deep neural networks (DNNs) becomes increasingly popular. Although various appproaches have been proposed, there is a lack of a comprehensive survey and taxonomy on this hot research topic. In this paper, we propose a unified way to describe the various optimization algorithms which focus on common and important search components of optimization algorithms: representation, objective function, constraints, initial solution(s) and variation operators. In addition to large scale search space, the problem is characterized by its variable mixed design space, very expensive and multiple blackbox objective functions. Hence, this unified methodology has been extended to advanced optimization approaches such as surrogate-based, multi-objective and parallel optimization.

CCS Concepts: • **Computing methodologies** → **Search methodologies**.

Additional Key Words and Phrases: Metaheuristics, Machine learning, Optimization, Deep neural networks, Hyperparameter optimization, Network architecture search

ACM Reference Format:
El-Ghazali TALBI. 2020. Optimization of deep neural networks: a survey and unified taxonomy. *ACM Comput. Surv.* 00, 00, Article 00 (2020), 36 pages. https://doi.org/00

1 INTRODUCTION

Over the last years, deep neural networks (DNNs) have enabled significant progress in many application domains including computer vision and natural langage processing (NLP) [63]. The design of DNNs has proven to be critical. Currently employed DNN architectures have mostly been developed manually by human experts, which is a time-consuming, error-prone process, and prevent finding new architectures that go beyond the human domain knowledge. Consequently, there is a growing interest in automated neural architecture search and hyperparameters optimization (AutoDNN) [81]. It allows the design of more efficient and effective DNNs and more accessibility to non expert for solving diverse learning tasks. AutoDNN approaches outperformed handcrafted architectures for some learning tasks, such as image classification [141], object detection [204] and semantic segmentation [31].

In the last five years, a lot of effort has been dedicated to automate the design of DNNs. Among the crucial contributing aspects for this progress are the design of new deep neural architectures and tuning of their associated hyperparameters. Scaling up DNNs capacity has been known as an effective approach to improve model quality for several learning tasks. Exact optimization approaches cannot be applied to such NP-complete optimization problems. A wide variety of specific heuristics

Author's address: El-Ghazali TALBI, el-ghazali.talbi@univ-lille.fr, University of Lille and INRIA, Polytech'Lille, Cité scientifique, Villeneuve d'Ascq, France, 59655.

0360-0300/2020/00-ART00 $15.00
https://doi.org/00

and metaheuristics have been used for architecture and hyperparameter optimization: random search [148][100][15][102], grid search [167], MCST (Monte Carlo Tree Seach) [130][183], reinforcement learning (RL) [7], and many families of metaheuristics such as evolutionary algorithms (EAs) and particle swarm optimization (PSO).

Some survey papers related to AutoDNN exist in the literature. Some papers focus on specific optimization problems such as hyperparameter optimization [13][56][118] and neural network architecture (NAS) [51][188]. In [51], the paper is structured according to three high-level dimensions: search space, search strategy and performance estimation strategy. Other survey papers focus on some families of optimization algorithms. In [40], the authors provide a survey of swarm and evolutionary computing approaches for general deep learning problems. Other surveys deal with neuroevolution [162] and reinforcement learning [85]. In [58], the authors propose a survey of metaheuristics for the training problem.

In this paper, a survey of optimization algorithms for AutoDNN is presented. A unified way to describe the optimization algorithms allow to focus on common and important search components for all AutoDNN methodologies: representation of DNNs (i.e. search space), formulation of objective function(s), handling of constraints, initialization of solution(s), and the design of variation operators (i.e. greedy such as RL, unary operators such as neighborhoods, mutation in EAs and velocity update in PSO, binary operators such as crossover in EAs, and indirect search operators). We also extend this unifying view to important optimization methodologies for AutoDNN dealing with surrogate-based optimization (i.e. Bayesian optimization), multi-objective optimization and parallel optimization. A unified taxonomy is proposed in an attempt to provide a common terminology and classification mechanisms. The goal of the general taxonomy given here is to provide a mechanism to allow comparison between different optimization methodologies. In addition, it is hoped that the categories and their relationships to each other have been chosen carefully enough to indicate areas in need of future work as well as to help classify future work.

The paper is structured as follows. In section 2, the main concepts of DNN and metaheuristics are detailed in a general and unified way. Section 3 formulates the problem and describes its main characteristics. In section 4, we present in a unified way the design of the various search components of metaheuristics: DNN representation, objective function definition, constraint handling, solution(s) initialization and variation operators design (i.e. greedy, unary, N-ary and indirect operators). In section 5 (resp. section 6, section 7) we focus on important aspects in AutoDNN dealing with surrogate-based optimization (resp. multi-objective optimization, parallel optimization). Finally, the last section presents the main conclusions and identifies some research perspectives.

2 MAIN CONCEPTS

This section provides an overview of the basic components of popular DNNs. Then, it presents in a unified way the main common search concepts of metaheuristics.

2.1 Deep neural networks

DNNs are accurate and efficient learning approaches, which represent one of the hottest research area in machine learning. DNNs are widely applied in computer vision, NLP, and robotics [63]. They are based on neural networks architectures, which interconnect multiple processing layers [73]. DNNs automatically extract features from big unstructured data such as image, text and audio. They learn the mapping between the features and predicted classes, layer by layer, through a transformation of the data, from low-level features to high-level features. This deep feature hierarchy enables DNNs to perform high-performance accuracy in many learning tasks.

DNNs come in two major families: *feed-forward* and *recurrent*. In feed-forward DNNs all the operations are carried out as a sequence of operations on the outputs of previous layers. In such DNNs, there is no memory. Feed-forward neural networks process information layer by layer, while recurrent neural networks have feedback loops between layers allowing them to be used in time-dependent tasks, such as NLP. One of the most popular feed-forward DNN is convolutional neural network (CNN). CNNs are comprised of three main types of layers: convolutional layers, pooling layers and fully-connected (FC) layers. In general, the training is performed by gradient-based algorithms (e.g. stochastic gradient descent). CNNs shows impressive results in computer vision for image and video processing. Many handcrafted CNN architectures have been proposed such as AlexNet [99], VGG [155], GoogLeNet [170], ResNet [70], and DenseNet [79]. Such DNNs can be giant and include many layers of different types and millions of hyperparameters.

There are other feed-forward DNNs such as Deep Boltzmann machines (DBMs), Deep Belief networks (DBNs), Auto-Encoders (AEs) and Restricted Boltzmann Machines (RBMs). Various single-layer unsupervised learning models have been proposed and stacked to build DNNs (e.g. sparse-response RBM (SR-RBM), autoencoder (AE), denoising AE (DAE)). RBM is a two-layers undirected graph, composed of one visible layer and one hidden layer with no connections allowed between nodes of the same layer [74]. An AE is a three-step DNN composed of an input layer, a hidden layer, and an output layer. The number of units in the input layer is the same as the output layer. The encoder is defined by the transformation from the input layer to the hidden layer, and extracts the features from the input data. The decoder transforms the hidden layer to the output layer, and reconstructs the input data from the features. DBN is a generative model consisting of multiple stacked restricted Boltzmann machines (RBMs) trained by contrastive divergence in a unsupervised way [75]. DBM is a network of symmetrically coupled stochastic binary units, which contains a set of visible units. There are connections only between hidden units in adjacent layers, as well as between the visible units and the hidden units in the first hidden layer.

Recurrent neural networks (RNNs) are specifically designed for time-dependant problems. They have both feedback and feedforward connections. RNNs have internal memory to allow long-term dependencies which will affect the output. Some intermediate nodes compute values that are stored internally in the DNN. Those internal values are used as inputs to other operations in conjunction with the processing of a later input. Long Short-Term Memory networks (LSTMs) are the most popular variant of RNNs capable of capturing long-term time dependencies [77].

2.2 Metaheuristics

The AutoDNN problem consists in searching the optimal DNN a^* from a set of possible solutions A which maximizes an *objective function* $f(a)$ while satisfying a set of *constraints*. The search space A is derived from the representation used to encode DNNs. Metaheuristics represent a class of general-purpose heuristic algorithms that can be applied to any optimization problem [172]. Unlike exact methods, metaheuristics allow to tackle large scale problems by delivering satisfactory solutions in a reasonable time. In the design of a metaheuristic, two contradictory criteria must be taken into account: *exploration* of the search space (*diversification*) and *exploitation* of the best solutions found (*intensification*).

2.2.1 Single-solution based metaheuristics. *Single-solution based metaheuristics* (S-metaheuristics) improve a single DNN. They could be seen as "walks" through neighborhoods or search trajectories through the search space [172]. S-metaheuristics iteratively apply the generation and replacement procedures from the current DNN. In the generation phase, a set of candidate DNNs are generated from the current solution a. This set $C(a)$ is generally obtained by local transformations of the

solution. In the replacement phase[1], a selection is performed from the candidate solution set $C(s)$ to replace the current DNN, i.e. a solution $a^{'} \in C(a)$ is selected to be the new DNN. Popular examples of such S-metaheuristics are local search (i.e. gradient), simulated annealing and tabu search. In addition to the representation of DNNs, their common search concepts are the definition of the *neighborhood* structure and the generation of the *initial solution*.

2.2.2 Population based metaheuristics. *Population based metaheuristics* (P-metaheuristics) could be viewed as an iterative improvement of a population of DNNs. P-metaheuristics start from an initial population of DNNs [2]. Then, they iteratively generate a new population of DNNs using variation operators. Popular examples of P-metaheuristics are evolutionary algorithms (EAs), ant colony optimization (ACO), particle swarm optimization (PSO), and estimation of distribution algorithms (EDA).

P-metaheuristics may be classified into two main categories:

- **Evolutionary-based:** in this category of P-metaheuristics, the DNNs composing the population are selected and reproduced using variation operators (e.g. mutation, crossover) acting *directly* on their representations. A new DNN is constructed from the different features of solutions belonging to the current population. Evolutionary algorithms (e.g. Differential evolution (DE), evolution strategy (ES), genetic programming (GP)) represent well-known examples of this class of P-metaheuristics.
- **Blackboard-based[3]:** here, the solutions of the population participate in the construction of a shared knowledge. This shared knowledge will be the main input in generating the new population of DNNs. Ant colonies and estimation distribution algorithms belong to this class of P-metaheuristics. For the former, the shared knowledge is represented by the pheromone matrix, while in the latter strategy, it is represented by a probabilistic learning model. For instance, in ant colonies, the generated DNNs by past ants will affect the generation of DNNs by future ants via the pheromone. Then, the generated DNNs participate in updating the pheromone.

Many stopping criteria have been used for solving the AutoDNN problem. In *static procedures*, the end of the search is known a priori. For instance, one can use a fixed number of iterations (i.e. generations in EAs [189], PSO [168]), a limit on CPU resources (i.e. time budget) [204], or a maximum number of training [82]. In an *adaptive procedure*, the end of the search cannot be fixed a priori. A fixed number of iterations (generations) without improvement or when a satisfactory DNN (e.g. given accuracy) is reached.

3 PROBLEM FORMULATION

Three popular formulations of the target optimization problem have been widely investigated in the literature:

- **Neural architectures search (NAS):** the goal is to search the optimal network topology (e.g. number of layers, types of operations, connections between operations) [51]. The hyperparameters are supposed to be apriori fixed and/or optimized in an independent post-processing search process.
- **Hyperparameter optimization (HPO):** this formulation requires an apriori definition of the DNN architecture. It consists in fixing the various hyperparameters of the DNN [56].

[1] Also named transition rule, pivoting rule and selection strategy.
[2] Some P-metaheuristics such as ant colony optimization start from partial or empty solutions.
[3] A blackboard system is an artificial intelligence application based on the blackboard architectural model, where a shared knowledge, the "blackboard", is iteratively updated by a diverse group of agents [52].

There are two types of hyperparameters: (1) operations hyperparameters which characterize the features associated to operations. For instance, the features of a convolution operation can be the filter size (width, height) and the stride size (width, height); (2) global hyperparameters which characterize the global features of the DNN. An example of global features are the optimization parameters (e.g. learning rate schedules, momentum, batch size) and regularization parameters (e.g. weight decay, dropout rates).

- **Joint optimization (AutoDNN):** the NAS and HPO optimization problems interact in a way that can render this separation suboptimal. In the AutoDNN joint optimization formulation, the two problems are solved in a joint manner. Neuroevolution (e.g. NEAT [163]) was a popular approach to solve the AutoDNN problem, where both the architecture and the hyperparameters are optimized in a global way [127]. An important question is related to the level (i.e. architecture or hyperparameter) in which optimization is carried out at each iteration. Three strategies can be applied: (1) *Global optimization:* which consists in optimizing all levels at the same time [143][162]; (2) *Nested optimization:* which consists in optimizing the different levels in a hierarchical way. At each iteration, the architecture is optimized, then the hyperparameters for this given architecture are optimized [140]; (3) *Sequential optimization:* where the NAS problem is solved first. Then, the hyperparameters for the obtained final solution are optimized.

Let us formulate the general AutoDNN problem. A DNN a can be defined by the quadruplet $a = (V, E, \lambda_V, \lambda_a)$ where V is a set of nodes denoting the layers (i.e operations) of the DNN, E is a set of edges (i.e. connections) between operations, λ_V is the feature set of operations and λ_a is the global feature set of the DNN. The induced graph $G = (V, E)$ defines the topology of the DNN. Each node has one of L labels, representing the corresponding operations. The space grows exponentially in both $|V|$ and L. Given the space of all datasets $\mathbf{D}$, the space of all deep learning models $\mathbf{M}$, and the search space of architectures $\mathbf{A}$, the optimal DNN consists to optimize the following objective function: $\Theta : \mathbf{A} \times \mathbf{D} \longrightarrow \mathbf{M}$. Let d be a given input dataset, in which d_{train} represents the training set and d_{valid} represents the validation set. The deep learning algorithm Θ estimates the model $m_a \in \mathbf{M}_a$ by minimizing:

$$\Theta(a, d) = arg\ min_{m_a \in M_a} L(m_a, d_{train})$$

where L represents the loss function. The problem consists in finding the optimal DNN a^* maximizing the objective function f using the validation data:

$$a^* = arg\ max_{a \in \mathbf{A}} f(\Theta(a, d_{train}), d_{valid}) = arg\ max_{a \in \mathbf{A}} f(a)$$

where the objective function f can be defined as the negative loss function $\mathbf{L}$ which measures the accuracy. The most popular loss functions are RMSE (resp. cross-entropy) for regression (resp. multi-class classification) problems [17].

The NAS and HPO can be seen as a reduced AutoDNN problem. Given an DNN topology defined by the graph G, the hyperparameter optimization problem (HPO) consists to find its optimal hyperparameter configuration: $\lambda^* = (\lambda_V, \lambda_a)^* = arg\ max_{\lambda \in \Lambda} f(a, \lambda)$, where Λ represents the set of all possible values for the hyperparameters, and a is the DNN induced by G. The NAS problem can be formulated as finding an architecture x^* when all architectures are evaluated under apriori fixed hyperparameter choices: $x^* = arg\ max_{x \in \mathbf{G}} = f(x, \lambda^*)$.

The AutoDNN problem is characterized by the following important properties:

- **Large-scale optimization problem:** a DNN could be composed of millions of decision variables. State-of-the-art DNNs have more than 100 layers [79] and billions of hyperparameters [80]. Moreveor, the input dataset can be very large.

- **Mixed optimization problem:** three different types of decision variables arize in AutoDNN: continuous, discrete ordinal and discrete categorical. Continuous variables refer to real numbers defined within a given interval (e.g. learning rate, momentum). Discrete ordinal (i.e. quantitative) variables are related to measurable integer values. Typical examples are the size of the filter and the stride in CNN pooling operations. Categorical (i.e. qualitative) variables are non-relaxable variables defined within a finite set of choices (e.g. type of operations, training optimizer). It is important to notice that different types of variables will require different optimization approaches.
- **Variable-size design space:** the search space of the AutoDNN problem contains conditionality. A decision variable is relevant only if another variable (or some combinations) takes a certain value. For instance, the number of layers influences the number of per-layer hyperparameters; the type of operation will induce a different number and type of features variables. The search space of the problem and the definition of the objective and constraint functions vary dynamically during the optimization process as a function of some variables values [135].
- **Extremely expensive black-box objective function(s):** the problem has very expensive objective function(s) which consist in training the whole DNN and computing the quality of the network (e.g. loss function). When facing very large-scale datasets the learning might take several hours, days or even months. Morever, the black-box objective function do not give access to a gradient or the Hessian, and do not have properties such as convexity and smoothness which are used in classical optimization.
- **Multi-objective optimization problem:** the AutoDNN problem can be formulated as a multi-objective problem in which many different and conflicting objectives are optimized. Indeed, in addition to maximizing the accuracy, some objectives dealing with cost, size, energy consumption, inference time of a DNN may be taken into account.

4 SEARCH COMPONENTS

Our survey is based on a unifying view of optimization algorithms according to their main search components. The most important and common search components in all metaheuristics are the problem landscape and the search operators (Fig.1). The problem landscape is defined by the encodings of solutions which induces the search space, the definition of the objective function(s) and handling of the constraints. The search operators are mainly the initialization of solution(s) and the design of variation operators.

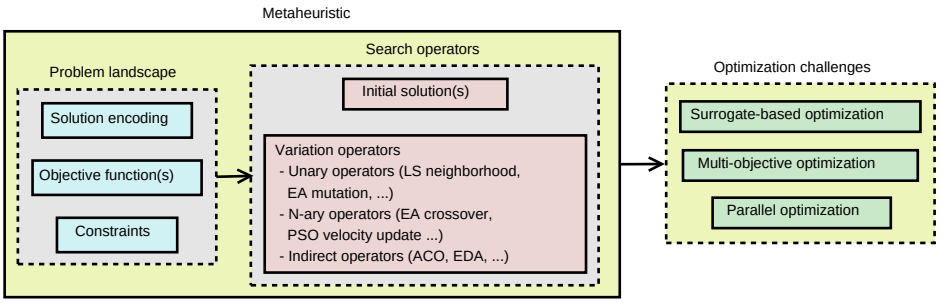

Fig. 1. A unified view of problem landscape and search components for AutoDNN metaheuristics, and challenging optimization issues.

4.1 Representation of DNNs

Designing any AutoDNN metaheuristic needs an encoding (i.e. representation) of a solution. The encoding plays a major role in the efficiency and effectiveness of any metaheuristic and then constitutes an essential step in designing an AutoDNN metaheuristic. This encoding defines the search space associated to the problem. It is preferable that an encoding has the following characteristics:

- **Completeness:** efficient DNNs can be represented. Indeed, many proposed encodings reduce the search space and might miss efficent DNNs.
- **Connexity:** a search path must exist between any two DNNs. Any solution of the search space, especially the global optimum solution, can be attained from any initial solution.
- **Efficiency:** the encoding must be easy to manipulate by the variation operators. The time and space complexities of the operators dealing with the encoding must be reduced.

Many alternative representations have been used in the literature (Fig.2):

- **Direct representations:** the encoding specifies a complete DNN. it describes completely the topology and the hyperparameters associated to the DNN.
- **Indirect representations:** the representation does not encode a complete DNN. A decoder (e.g. rules, greedy procedure) is required to generate the DNN given by the encoding. The decoding may be deterministic or non deterministic.

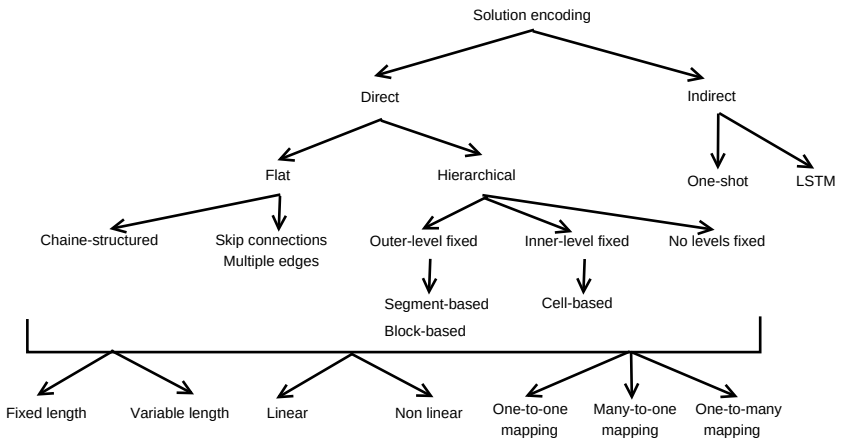

Fig. 2. Classification of the different encodings of DNNs.

4.1.1 Direct representations. Two main families of DNNs may be found in the literature: flat DNNs and hierarchical DNNs.

Flat DNNs: DNNs are generally defined as flat networks (e.g. DBN, some CNNs). The most simple and popular flat network is the *chain-structured* (Fig.3) [204]. Hence, the topology associated to DNNs can be represented by DAG (Directed Acyclic Graphs) $G = (V, E)$, where each node $v \in V$ represents an operation (i.e. layer), and each edge e represents a feature map connecting two operations. Let us notice I_i the set of input edges associated to an operation v_i. The computation of the output edge O_i is: $O_i = v_i(I_i)$. The network can be represented by a sequence of operations such that any operation v_i receives its input from operation v_{i-1}: $O_i = v_i(O_{i-1})$ [63]. An example of such popular DNNs are VGGNet [155] and AlexNet [99].

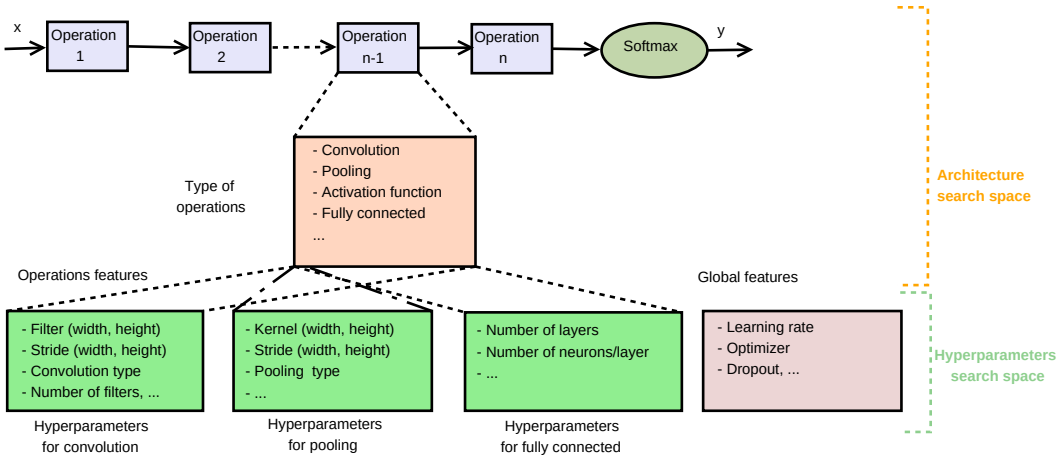

Fig. 3. Chain-structured DNNs. Different colors define different operations. For a CNN they represent unary operations such as convolutions, pooling, activation functions, or multivariate operations such as concatenation or summation.

Extended flat DNNs include skip connections, highway connections and multiple edges between operations (Fig.4) [203][20][50][142][25]. Hence, the incident edges of an operation v_i is the union of O_{i-1} and other ancestor edges: $O_{i-1} \cup O_j / j < i-1$. Those topologies enable more flexibility in designing DNNs. *Residual networks* (ResNet) [70] (resp. *DenseNets networks* (DenseNet) [79]) belongs to this family of architectures, in which the previous operations are summed (resp. concatenated).

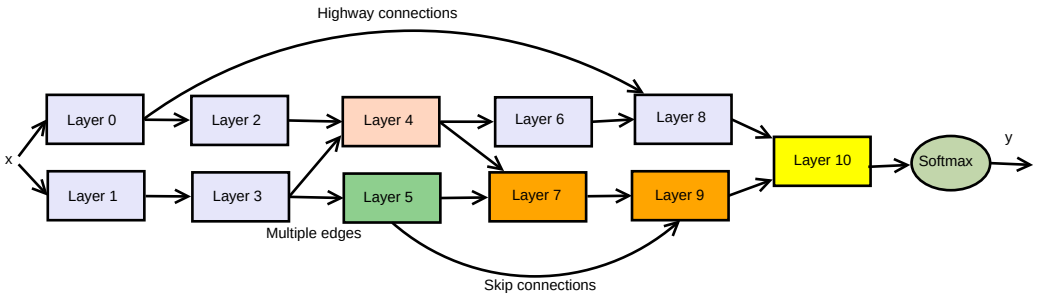

Fig. 4. Skip connected and multiple edges chain-structured DNNs.

A complete encoding must represent the whole information of a DNN a defined by $a = (V, E, \lambda_V, \lambda_a)$ (Fig.3). On one hand, the encoding must specify the architecture of the DNN which is defined by the graph $G = (V, E)$. Hence, the number of operations, type of operations (e.g. convolution for a CNN), and connections between operations must be given. A general graph G (e.g. RNNs) can be represented by its binary adjacency matrix, while a DAG (e.g. CNNs) can be represented by a lower triangular binary matrix. Indeed, a DAG can be encoded so that all the directed edges connects nodes from a lower number to a higher number [143][114]. On the other hand, the encoding must represent the features of all active operations (e.g. number of filters, size of filters and strides for a convolution), and the global features of the DNN (e.g. learning rate). The

main property of features encodings are their *variable and mixed nature*: continuous (e.g. learning rate), ordinal discrete (e.g. number of layers) and categorical discrete (e.g. type of operations).

Many different encodings have been used to represent flat DNNs. *Linear representations* encoded by string of symbols of a given alphabet are widely used. DBN networks are generally represented by linear encodings, which include topological parameters such as the number of hidden layers and neurons by hidden layer, and some global features for the contrastive divergence (e.g. weight cost) and back-propagation (e.g. learning rates for weights and biases). For CNNs, the presence of many conditioned variables makes that the encoding in intrinsically of *variable-length*. For instance, the topology (resp. hyperparameters) is conditioned by the number of layers (resp. type of operation). However, many authors use *fixed-length encodings* by assuming some restrictions. In HPO optimization, the architecture (i.e. graph $G = (V, E)$) is a priori fixed. Then, a fixed-length mixed linear encoding is mostly used to represent the operations features $lambda_v$ and global features $lambda_a$ of DNNs (e.g. chain-structured architectures [57][203]). In NAS and AutoDNN optimization, a fixed-length encoding still possible when the number of operations (i.e. layers) is fixed [180]. Compared to the previous encoding, it will include the set of operations and their connections [120]. In [7], the type of operations (e.g. convolution, pooling, fully connected, global average pooling), and hyperparameter settings (e.g. number of filters, kernel size, stride and pooling size) are considered in a linear fixed-length encoding. When the number of layers is bounded by a maximal value, the use a fixed-length encoding can also be an alternative. In [113], the proposed fixed-length mixed encoding includes the number of layers (ordinal discrete), learning rate (continuous), type of activation function (categorical discrete) and the gradient algorithm used in training (categorical discrete).

Variable-length encodings is another suited alternative to encode flat DNNs. In [168][92], a variable length encoding is used to solve the AutoCNN problem. The encoding represents different numbers of convolutional layers and pooling layers, and their hyperparameters. In [5], a variable-length sequence of layers and their respective hyperparameters is used to solve the AutoCNN problem. The encoding represents the general structure of the network (i.e. sequence of layers) and the hyperparameters associated to each layer using a human-readable *context-free grammar*. In [178], the encoding is inspired from IP address in computer networks to represent a variable length encoding of CNNs. An IP address is represented by sequence of decimal numbers delimited by full stops (e.g. 192.159.1.354). The network is encoded by k IP adresses where k is the maximum number of layers. Each layer is represented by an IP address, and non used layers are disabled.

Non linear encodings such as grammars, CGP (Cartesien Genetic Programming) [123][176][124], and tree structures [139][130] have also been used to encode flat DNNs.

Hierarchical DNNs: in the last years, a widely used network type to tackle the scalability issue in DNNs is hierarchical networks [108]. They allow to reduce the search space, integrate human knowledge in the definition of the building blocks, and can be more flexible to solve other learning tasks [204]. Compared to flat networks, they have smaller degree of freedom in the architecture design. In hierarchical DNNs, the main idea is to have several blocks[4] which are used as building blocks in the design of multi-level DNNs. Many popular hierarchical network have been handcrafted, including ResNet [70] and DenseNet [79]. Cell-based CNN architectures [204], inception and xception networks [171] represent the most popular hierarchical DNNs. Except a three-level model proposed in [12], most of the hierarchical DNNs are composed of two levels. The inner-level represents the set of primitive buiding blocks, while the outer-level contains the

[4]Also called patterns, modules, stages, segments, motifs, and cells.

full architecture which is a composition of the building blocks. Depending on the optimized level, different encodings have been proposed:

- **Inner-level optimization:** the topology of the DNN at the outer-level is apriori fixed. The problem consists in finding the optimal inner-level blocks. In [189][117][198][174], each block can be composed of a given number of layers n. Let k be the number of possible configurations for a layer. Then, the size of the search space will be $(k \times (n-1)!)^b$, where b is the number of blocks. In [187], *path encoding* is proposed in which they represent the set of directed paths of a cell. The total number of paths is exponential in n: $\sum_{i=0}^{n} k^i$ while the adjacency matrix scales quadratically.
  Many proposed encodings are *many-to-one mappings* [26], in which many encodings can represent the same DNN, and then duplicate evaluations are possible. In [189][117], a hierarchical chained structured DNN is proposed. The outer-level is considered as a sequence of a given number of S connected stages $B_s, s = 1, ..., S$ (Fig.5). The hyperparameters of the stages are fixed. The search space is related to the configuration of inner-level segments. Each segment is defined as a set of n maximal predefined operations $B_{s,i}, s = 1, ..., S \ \& \ i = 1, ..., n$ such as convolution, pooling layers and batch normalization. The proposed encoding is based on a *fixed-length binary vector* (i.e. size of $n \times n - 1 \div 2$) which represents the connections between the nodes. This encoding is flexible enough so that many well-known hand-crafted DNNs can be represented such as VGGnet, ResNet and DenseNet. This encoding is a *many-to-one* mapping, and induces a search space of size $\Lambda = S \times 2^{n(n-1)\div 2}$.
  In [198], a DNA-based encoding is proposed. A DNN is defined as a fixed-length sequence of blocks. Each block is composed of a set of convolution layers with a given maximal number of layers. For each convolution layer, there are three kinds of hyperparameters to be encoded: number of filters, kernel size, and input layer indices. In [174], the encoding is represented by connecting segments. Each segment has repeating patterns of operations, and is parameterized by the operation type and the number of repetitions of the patterns. Each pattern is a sequence of connected operations.

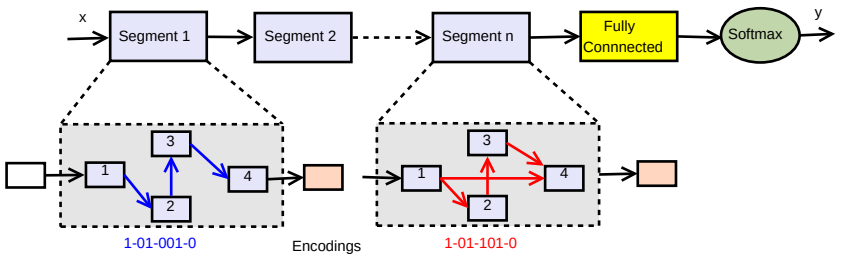

Fig. 5. Template-based hierarchical DNNs chained architectures using summation as merging operation. Only the dark orange network for each segment has to be designed. The other operations are fixed.

- **Outer-level optimization:** this methodology is widely used in *cell-based architectures* [204]. Cell-based CNNs are designed by a combination of repeated cells in a predefined arrangement. A cell can be defined as a small DAG which transforms a feature using an ordered sequence of N nodes [200][171][46][186] (Fig.6). A popular example of such DNN architecture is NASNet [204]. The cells can be stacked to form a CNN or recursively connected to form a RNN. Various macro-architectures are used such as a sequence where each cell receives the outputs of the two preceding cells as input [204], or combined in a multi-branch chained network

[25]. In general, the topology of the different types of cells is predefined. Then, the problem consists in finding the optimal architecture at the outer-level. In [204], the authors consider two different types of cells: normal cells (resp. reduced cells) which preserves (resp. reduces) the dimension of the input. The architecture is optimized by finding the optimal sequence of those two types of cells.

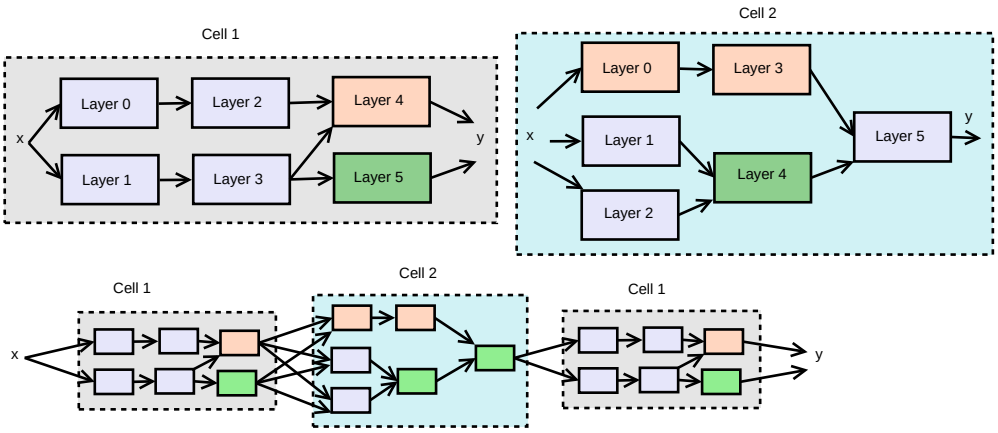

Fig. 6. Cell-based DNNs. Two different cells are illustrated: cell1 and cell2. The final architecture is built by the sequence (cell1, cell2, cell1). More sophisticated sequencing can be designed such as multi-branch spaces, by replacing layers with cells.

- **All levels optimization:** some approaches perform the search in both levels: the outer-level (i.e. macro-architecture of DNN) and the inner-level (i.e. micro-architecture of blocks) [106][201]. In [106], the authors propose a trellis-like network level search space that augments the block level search space to form a hierarchical architecture search space. To reduce the complexity of the search, continuous relaxation of discrete variables is performed to be optimized by a gradient algorithm.

This idea of relaxing discrete representations into continuous ones has been explored in many flat and hierarchical DNNs allowing the application of gradient-based optimization algorithms [147][109][2][153][177][105][196]. In [109][33], each operation is encoded by a mixture of candidate operations, where the operations mixing weights are parameterized by a continuous vector. Then, the categorical choice of a given operation is reduced to a Softmax over all possible operations.

4.1.2 Indirect representations. Direct encodings represent strong specification schemes that may have a problem with scalability. They require longer encodings as DNN size increases, and search space will be increased accordingly. Indirect encoding allows a more compact representation in which the DNN is not totally specified in the representation, although they can be derived from it. Instead, a decoding strategy (e.g. greedy algorithm, set of rules) is used to decode the generated DNNs. For the sake of efficiency, we need to be sure that indirect encodings do not restrict DNNs to some suboptimal class of DNNs [66]. The most popular indirect encodings are: one-shot architectures and LSTMs.

One-shot architectures: they represent the most popular indirect encodings of DNNs. The main motivation is that instead of training hundreds of different DNNs from scratch, one can train a single large network capable of generating any DNN architecture in the search space. All architectures are treated as different subgraphs of a supergraph and shares weights between

architectures that have edges of this supergraph in common [109] (Fig.7). First, the weights of a single one-shot model are trained. Then, architectures (i.e. subgraphs of the one-shot model) are generated and evaluated by weights sharing from the one-shot model. The drawback of one-shot arhitectures is that their associated supergraph restricts the search space to its subgraphs [149]. The one-shot architecture[5] search consists of four steps [12]: (1) Define a search space to encode a wide variety of DNNs using a single one-shot model. (2) Train the one-shot model to find the weights. (3) Evaluate generated architectures on the validation set using the pre-trained one shot model. (4) Re-train the best found DNNs from scratch and assess their performance on the test set. Decoding one-shot architectures are generally based on sampling independently from a fixed probability distribution. In [20], a random search is applied, but it can be replaced by metaheuristics.

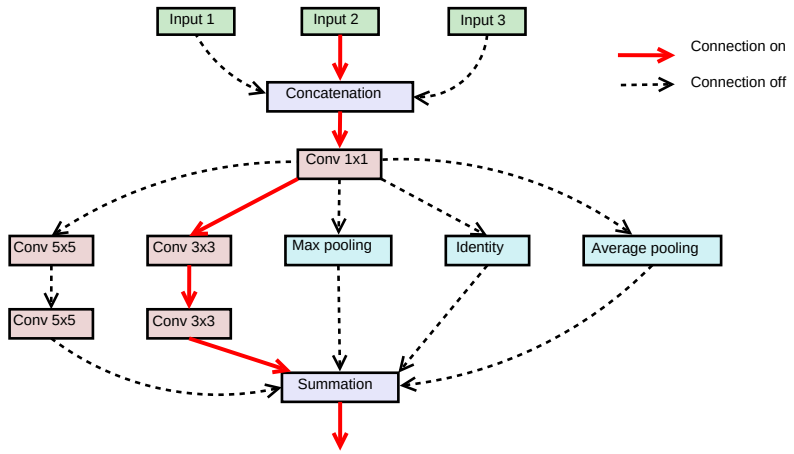

Fig. 7. Example of one-shot DNN cell architecture. It is composed of five separate operations. By sampling we can select the two conv 3x3 operations path. To be evaluated, the network will not retrain the weights.

LSTM encoding-decoding: the original DNN architecture a is mapped to continuous representation ϵ using the encoding function $E : \mathbf{A} \longrightarrow \epsilon$ [119]. A single layer vanilla LSTM is the basic model of encoder and the hidden states of the LSTM are used as the continuous representation of a. Then $E(a)$ is optimized into $E(a')$ via a gradient descent. Afterwards $E(a')$ is transformed into a new architecture a' using the decoder LSTM network. The decoder is responsible for decoding the string tokens in a', taking $E(a')$ as input and in an autoregressive manner. The encoder and decoder are trained by minimizing the combination of performance prediction loss and structure reconstruction loss.

4.2 Objective function

The objective function f formulates the goal to achieve. It associates to each DNN a real value which describes its quality $f : \mathbf{A} \longrightarrow \mathbb{R}$. The classical way to evaluate the performance of a DNN a is to train it on training data and compute its performance on validation data. Accuracy on unseen data is the most used metric to assess the performance of the learned model. The most time-consuming part of the optimization process is the training of the DNN. Speeding up the training process is widely used in order to reduce the computational cost. While these low-fidelity estimations reduce the computational cost, they also introduce bias in the estimate as performance will typically be

[5] Also called Hypernetworks.

under-estimated. An important property of low-fidelity procedures is that the relative ranking of architectures remain the same [149]. The main families of the approaches allowing to speedup the computation of the objective function can be classified as follows (Fig.8):

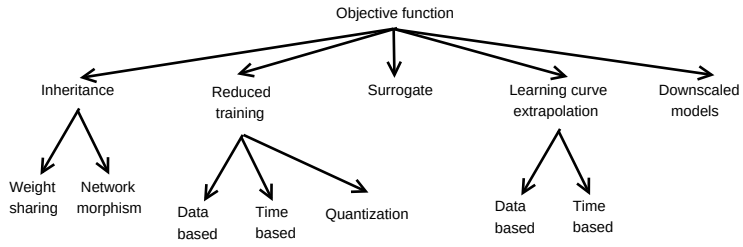

Fig. 8. Main approaches for speeding up the computation of the objective function.

- **Inheritance:** this approach avoid the training from scratch and thereby substantially reduces the required training time per DNN. It is based on knowledge transferring between DNNs. *Weight sharing* is a well-known approach in which we initialize the weights of the generated DNNs based on weights of already trained DNNs [137][24]. Hence, a DNN can be transformed while leaving some weights unchanged [82][184][88][49]. Instead of a random initialization of the weights, informed decisions (e.g. *Xavier* initialization) [60] have also been used. Pre-trained weights using *transfer learning* also allows to reduce the huge cost of training DNNs from scratch [185]. Another popular inheritance-based approach is *network morphisms* [184]. In the context of DNNs, network morphism refers to a parameter-transferring map from a given DNN to a generated DNN that preserves its function and outputs [50]. Morphing types are demonstrated including depth morphing [32], width morphing, kernel size morphing, and subnet morphing [184].
- **Reduced training:** this low-fidelity approach in training consists in reducing the training time [194], the number of epochs [168][202], or the input dataset [97]. For example, one can carry out search on CIFAR-10 and "transfer" the obtained DNN (with some changes, e.g. changing the number of filters) to ImageNet [104][190]. *Quantization approaches* represent weights using a small set of permitted values, reducing the number of bits required to store each weight. In [20][38], the weights takes binary values, and then the complexity of multiplications operations will be reduced during training. Existing quantization methods can be mainly divided into two categories. The first category of methods seeks to design more effective optimization algorithms to find better local minima for quantized weights. For instance, these works introduce *knowledge distillation* [144]. The second category focus on improving the quantization function (e.g. binarization).
- **Surrogate[6]:** an alternative to reduce the high-complexity of the objective function is the use of a surrogate. Surrogate models replace expensive objectives with models that provide an approximation. Section 5 details this important class of optimization approaches, named surrogate-based optimization[7]. In [59], the idea of *weight agnostic DNNs* has been proposed, where there is no use of any explicit weight training in the optimization process. They are supposed to have strong inductive biases that can already perform various tasks with random weights.

[6] Also known as meta-model and approximation.
[7] Also called Bayesian optimization.

- **Downscaled models:** many strategies consist in using downscaled models. Reduction can be applied to data and network. In data reduction, a partial dataset is used instead of the whole dataset [178]. Downsampling techniques (e.g. lanczos, nearest, bilinear, bicubic, hamming, box) have also been used to reduce the resolution of images [34]. In network reduction, downscaled models are using a subset of the network for training. In [204][194], reduced architectures with less filters per layer and less cells have been trained.
- **Learning curve extrapolation:** it describes two different strategies: time-based and data-based. In time-based learning curve extrapolation, the performance of the training procedure is learned function from its number of iterations or training time [198]. Different learning models have been investigated such as logistic regression [169], neural networks [97], support vector machines regression [8], linear regression [8], random forest [8], and recurrent neural network (e.g. LSTM) [104]. In [45], the learning curve model is used to terminate training of DNNs when it is unlikely to beat the performance of the best found DNN. In data-based learning curve extrapolation, the performance of the training procedure is learned function of the size of the available dataset for training. In [139], a training is carried out for a few epochs, and then meta-learner network (e.g. RNN) predicts the performance a fully trained network would have.

Low-fidelity approaches can also help to avoid *overfitting*. When using low-fidelity approaches, full training are generally applied at the end of the optimization for the best found DNNs [180]. Other adaptive approches wich gradual increase in fidelities during the search have been investigated [101][53].

4.3 Constraints

Many constraints characterize the AutoDNN problem, such as the number of layers, model complexity, computation cost, memory consumption cost, training time, prediction time, and energy usage [175]. The constraints may be of any kind: linear or non linear, equality or inequality constraints. The proposed constraint handling techniques can be classified as:

- **Reject:** reject strategies represent a simple approach, where only feasible solutions are kept during the optimization process and then infeasible solutions are automatically discarded [78][46]. This kind of strategies is conceivable if the portion of infeasible solutions of the search space is very small. Moreover, they do not exploit any information on infeasible solutions.
- **Penalizing:** in penalizing strategies, infeasible solutions are considered during the search process. The objective function is extended by a penalty function which will penalizes infeasible solutions using for instance linear penalization $f^{'}(a) = f(a) + \lambda c(a)$, where c represents the penalty function (e.g. number of violated contraints, amount of infeasibility) and λ the weighting factor (e.g. static, dynamic, adaptive). This is the most popular approach in which many alternatives have been used to define the penalties [174][203][201][177].
- **Repairing:** repairing strategies consist in heuristic algorithms transforming an infeasible solution into a feasible one. A repairing procedure is applied to infeasible solutions to generate feasible ones. Those strategies are applied in the case where the search operators may generate infeasible solutions.
- **Preserving:** in preserving strategies, the encoding and variation operators will insure the generation of feasible solutions. They incorporate problem-specific knowledge into the encoding and search operators to generate only feasible solutions and then preserve the feasibility of solutions. Incorporating prior knowledge about typical properties and allowable structures of DNNs can reduce the size of the search space and then simplify the search. One

can find the following constraints: maximum number of layers, possible types and number of operations [168], starting and ending layers [189][178][7], possible connections [192], and fixed sequence of operations [117].

4.4 Initial solution(s)

The generation of initial solutions(s) has a great impact on the effeciency and effectiveness of metaheuristics. For a single solution initialization, there is always a tradeoff between the use of random and "good" initial solutions in terms of the quality of solutions and computational time. In the initialization of a population of solutions, an additional criterion to deal with is diversification. If the initial population is not well diversified, a premature convergence can occur. Many approaches have been developed for the AutoDNN problem:

- **Random generation:** most iterative metaheuristics approaches initialize solution(s) in a random way (e.g. Gaussian distribution, uniform): EAs [189], PSO [168], DE [179], gradient [109][119].
- **Heuristic generation:** initial solutions can also generated by low-cost heuristics. In general, greedy algorithms (e.g. reinforcement learning) are used for their effectiveness. In [76], the authors suggest using a lower-dimensional search space to quickly identify promising areas (e.g. reducing the resolution of images). This information can then be used to initialize the metaheuristic for the original, higher-dimensional search space.
- **Partial architectures:** the optimization process starts with reduced small architectures [114][22][59], or well known skeleton architectures and tries to augment them. In [57] (resp. [180]), the VGGNet (resp. DenseNet) skeleton is used. Some metaheuristics start with poor trivial architectures and tries to improve them by fixing for instance the number of layers and connections [158][165][203], and reducing the type of operations [143]. This approach does not avoid the additional bias introduced by the skeletons.
- **Complete architectures:** some work propose initial solutions based on prior-knowledge hand-crafted architectures [117] and/or best known DNNs [91]. Other works start with giant DNNs to be compressed (i.e. dropout, swapout, subgraph search) for various learning tasks [55][177][196][137][30]. This approach adds an additional bias introduced by the used DNN.
- **Mixed initialization:** for a better compromise between diversification and quality, mixed strategies may be applied. In [108], a combination between random DNNs and trivial DNNs (e.g. chain of operations) is developed.
- **Diversified population:** to our knowledge there is no work dealing explicitely with diversifying an initial population of DNNs using for instance sequential or parallel diversification strategies [172].

4.5 Search operators

The role of variation operators is the generation of new DNNs during the search process. The main variation operators found in optimization approaches can be classified as greedy procedures, unary, n-ary and indirect.

4.5.1 Constructive procedures. Constructive (i.e. greedy) procedures start from an elementary or null DNNs and construct a complete DNN by adding operations and connections until a DNN that is capable of solving the task emerges [133]. Very few greedy algorithms have been proposed for the AutoDNN problem [114]. Sequential learning (i.e. Markov decision process) approaches such as RL can be considered belonging to this family of optimization algorithms. In RL approaches, an agent is trained to select the operation of a DNN in a particular order. The generation of a DNN architecture is carried out by the agent's action, in which the reward is based on an estimate of the performance

of the trained architecture on unseen data. The main questions in designing a RL are: how do they represent the agent's policy and how do they optimize it. In [7] the MetaQNN method use ϵ-*greedy Q-learning* to train a policy which sequentially chooses a layer's type (e.g convolution, pooling, and fully connected layers), connections between layers, and corresponding hyperparameters. This approach has been generalized to hierarchical DNNs, in which a block is repeated to construct a network [199]. In [203], a *policy gradient* is applied to approximate the reward function. The author uses *recurrent neural network (RNN)* policy to sequentially constructs the DNN architecture. This method has been extended in the state-of-the-art NASNet approach [204], which constructs repeated blocks composed of convolution and pooling operations. *Multi-armed bandits* approaches have also been investigated [101].

In contrast, pruning procedures start from a complete DNN and at each iteration reduce the complexity of the network by removing nodes or connections [129], in the hope to improve the generalization of the DNN. A large variety of DNN pruning approaches have been proposed using different pruning criteria [18]. In the "brain damage" approach [107], the authors remove the redundant parameters using derivate-related criteria. In [19], the weights are represented as Gaussian random variables and weights with lower mean value and larger uncertainty are pruned. In [166], the Hebbian rule is used as a pruning criterion, where more connections between weekly correlated neurons are skipped. The connecting weights can also be skipped by regularization terms such as the squared l_2 norm and $l_0 - norm$ [36].

4.5.2 Unary operators. Unary variation operators transform a single DNN into another one. In general, it represents a small change (i.e. perturbation) of a DNN. Some important properties must be taken into account in the design of unary operators: ergodicity, validity and locality [172]. The most popular unary operators in metaheuristics are neighborhood in S-metaheuristics (e.g. gradient) and mutation in EAs. The design of unary operators depends on the type of representations. For graph-based encodings, it consists in adding/deleting a node or a connection of the graph. In discrete representations, it generally consists in changing the value associated to an element by another value. For continuous variables, the most used class of unary operators has the form $x^{'} = x + M$, where M is a random variable which takes different forms (e.g. uniform random, Gaussian distribution). Unary operators have been applied to all levels of DNNs encodings (Fig. 9):

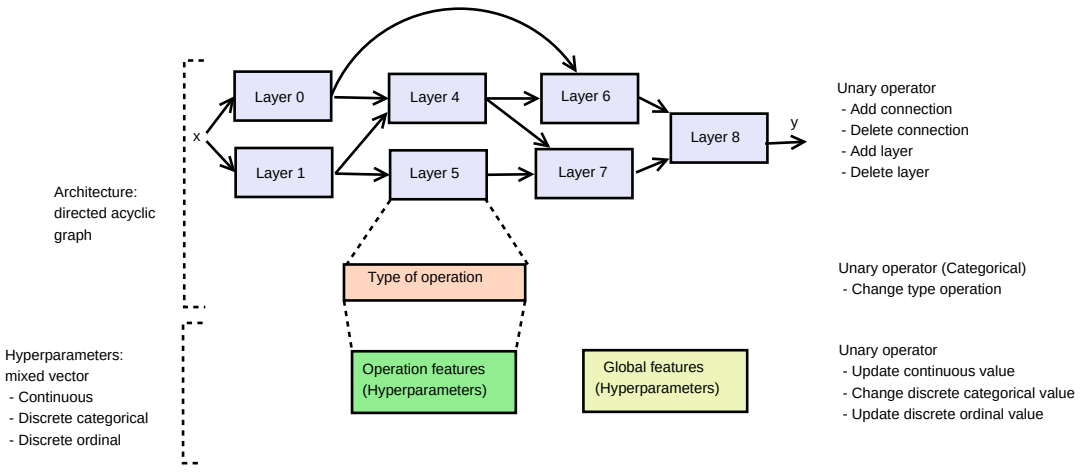

Fig. 9. Unary variation operators at different levels of a DNN.

- **Architecture:** unary operators at this level consists to update a DAG using for instance the following operations: add a layer, delete layer, change type of a layer, add a connection, and remove a connection. Those unary operators have been used in different optimization frameworks:
  - **Neighborhoods in S-metaheuristics:** in some papers, the authors have relaxed the discrete encoding of DNN into continuous encodings to enable gradient-based optimization [153][2][108][109][119]. Hence, gradient-based optimization is applied using classical neighborhoods of continuous variables.
  - **Mutation in EAs:** in flat networks, many mutation operators have been designed. Discrete mutations have been used in DAG representations of CNNs to connect or disconnect two operations [117][189], to add a layer, remove a layer [5][162][50][120][127], replicate a layer [4]. Continuous mutations have been applied in [154] into a CMA-ES algorithm by relaxing the binary adjacency matrix into continuous matrix and using rounding operations. In [139], tree-based mutations have been designed for LSTMs: (1) Mutation to randomly replace an operation with an operation of the same family, (2) Mutation to randomly inserts a new branch at a random position in the tree. (3) Mutation to shrink the tree by choosing a branch randomly. For hierarchical DNNs, the same unary operators can be applied at any level of the architecture hierarchy.
- **Hyperparameters:** global and operations features of a DNN are generally encoded by a mixed vector of continuous and discrete values:
  - **Neighborhood in S-Metaheuristics:** continuous neighborhoods [153][2], and mixed neighborhoods [50][157] have been designed to be used in local search algorithms.
  - **Mutation in EAs:** discrete mutations have been used in different EA frameworks. In [165], the $(1 + \lambda)$-ES is applied in which λ solutions are generated by discrete-based random uniform mutation on hyperparameters (e.g. number of filters, size of filters). In [91], discrete categorical mutations are applied in designing LSTMs, such as changing the element-wise operation and the activation function. Continuous mutations have been defined in a differential evolution (DE) algorithm [179] and a CMA-ES algorithm [116]. In [116], all discrete and continuous variables are scaled to be in $[0, 1]$, on which samples λ candidate DNNs are generated from a multivariate normal distribution. Mixed mutation operators have also been defined for global (e.g. learning rate, training optimizer) and operations hyperparameters (e.g. activation function, filter size) [113][143][50].

For hierarchical DNNs, the level in which unary opeators are applied can be sampled randomly. In [108], the authors sample the level k, the building block m at the level k, then a unary operator is applied to an element of this building block m.

4.5.3 N-ary operators. Unlike unary operators, n-ary variation operators recombine a set of n DNNs into another one: $\mathbf{A} \times \mathbf{A} \ldots \times \mathbf{A} \longrightarrow \mathbf{A}$. Their role is to inherit the building blocks of a set of DNNs to generate new DNNs. The most popular n-ary operators in metaheuristics are crossover in EAs and velocity update in PSO. The most used crossover operators are the *1-point crossover* and its generalization the *n-point crossover*, in which n crossover cuts are randomly selected and the solution is generated by interchanging the segment of each parent. In *uniform crossover*, each element of the solution is selected randomly from either parent, which is more flexible for mixed vectors. For continuous variables, one can add arithmetic and geometrical crossover operators. In PSO, the velocity updating of any particle x_i is a computed function of the global best DNN $gBest$ and local best DNN $pBest_i$. As for unary operators, the design of binary operators depend mainly on the variables to be inherited:

- **Architecture:** n-ary operators have been used in different optimization frameworks:

834 – **Crossover in EAS:** in flat DNNs using linear encodings, a 1-point crossover has been designed to recombine the layers [179][5][114]. Crossover operators specific to tree encodings of architectures have been also developed (e.g. Homologous crossover for LSTMs [139]). In hierarchical DNNs, a uniform crossover applied at the level-1 blocks has been used in [189]. Each pair of corresponding blocks are exchanged with a given probability. In a binary encoding of a DAG, a crossover operator preserves the common building blocks shared between both parents by inheriting the common bits from both parents [117]. Then, it maintains, relatively, the same complexity between the parents and their offsprings by restricting the number of "1" bits in the offspring's bit-string to lie between the number of "1" bits in both parents (Fig.10). In general, all the values at lower levels are inherited from the crossover operator involving higher levels [179].
– **Velocity update in PSO:** it needs to have a fixed length for all particles. Hence, new velocity updates have been designed for variable-length representations. In [168], the authors used truncation and padding to deal with variable-length encodings. In [178], a fixed-length bounded by the maximum length in which disabled layers are encoded in the representation and participate to the velocity update.

- **Hyperparameters:** any classical n-ary operators can be applied to mixed vectors characterizing the global and operations features of a DNN:
  – **Crossover in EAs:** unlike n-point crossovers, the uniform crossover is well adapted to fixed-length mixed encodings [113].
  – **Velocity update in PSO:** classical velocity updates are based on fixed-length continuous vectors. In [57], the discrete variables are relaxed to continuous variables, such that the classical velocity update is applied. Then, a cast (i.e. rounding) operation is carried out.

  Other n-ary operators have been applied in other metaheuristics. In [164] a tree growth algorithm (TGA) has been developed for AutoCNN. The n-ary operator consists in moving $N2$ solutions y_i to the distance between the closest best solutions x_1 and x_2, by producing linear combinations: $y_i = \lambda x1 + (1 - \lambda)x2$. For discrete variables, the obtained values are rounded to the closest integer value.

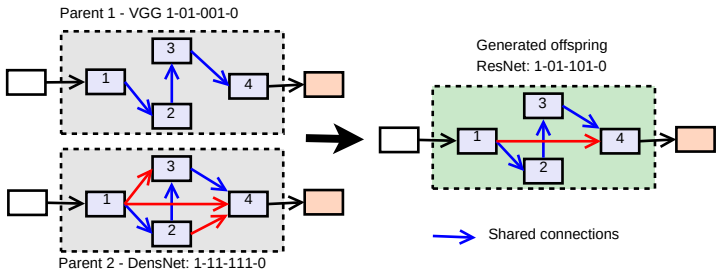

Fig. 10. An example of crossover operator inherting and recombining building blocks [117].

4.5.4 Indirect operators. The solutions of the population participate in the construction of a shared knowledge. This shared kowledge will be the main input in sampling the new population of DNNs. The recombination in this class of algorithm between solutions is indirect through this shared memory. ACO and EDA are the most popular algorithms belong to this class of Pmetaheuristics:

- **Ant colony optimization (ACO):** the shared knowledge is represented by the pheromone matrix. ACO have been developed to design the LSTM cell structure of the network. LSTMs

are generated by a given number of ants, and having them choose a path through the fully connected DNN biased by the amount of pheromone on each connection. The good quality generated DNNs are used to update the pheromone, reinforcing the features (i.e. connection between operations) that provide good solutions [43][48]. The same approach has been developed for CNN [22]. For a given depth of the CNN, each ants constructs a complete CNN by selecting the next operation by using the global pheromone.

- **Estimation of distribution algorithms (EDA):** the shared knowledge is represented by a probabilistic learning model. In [117], a Bayesian optimization algorithm (BOA) has been developed to find inherent correlations between the decision variables. In AutoDNN, this translates to correlations in the blocks and paths across the different segments. Exploitation uses past information across all networks evaluated to guide the final part of the search. More specifically, if we have a network with three segments s_1, s_2 and s_3, by using the history of generated solutions, the operator constructs a Bayesian Network relating those variables. It consists in modeling the probability of networks beginning with a particular segment s_1, the probability that s_2 follows s_1, and s_3 follows s_2. Those estimates are updated during the search, and new offsprings are generated by sampling from this Bayesian Network.

Other optimization approaches use indirect operators. A Bayes Monte Carlo procedure has been used [28]. A set of DNNs are sampled. Then, a probability distribution over high-performing DNNs is learned.

5 SURROGATE-BASED OPTIMIZATION

Surrogate-based optimization[8] (SBO) is a popular approach to deal with the AutoDNN problem. These algorithms are iterative sampling procedures relying on surrogate models (i.e. metamodels, approximation) of the considered objective function which are generally characterized by an expensive computational cost [9][152]. They iteratively determine and explore the most promising solutions of the design space, thus simultaneously refining the surrogate model and converging towards the problem optimum [89]. First, a set of diversified observations D_n are generated using for instance Design of Experiments (DoE) or Latin Hypercube. Using this set of observations D_n, a surrogate $s(f) : A \longrightarrow \mathbb{R}$ of the objective function f is constructed. Then, it consists in sampling iteratively, using the surrogate, the most promising solution $x_{n+1} \in arg\ max\ \ q_{s(f)}$ based on an *infill sampling criterion* (i.e. acquisition function) $q_{s(f)} : A \longrightarrow \mathbb{R}$. Usually the acquisition function uses exploiting and exploring sampling principles. The solution x_{n+1} is evaluated using the real objective function $y_{n+1} = f(x_{n+1})$ and is added to the set of observations $D_{n+1} = D_n \cup (x_{n+1}, y_{n+1})$. The surrogate is updated $s(f/D_{n+1})$ using the new acquisition function $q_{s(f/D_{n+1})}$, and a new solution is sampled, and so on, until a given budget of evaluated solutions is finished (Fig.11). Notice that the evaluation of the acquisition function q is much cheaper the the original function f which makes that the optimization effort is reduced.

The various surrogate-based metaheuristics for AutoDNN can be characterized by:

- **Surrogate model**: there are at least two desired properties for a surrogate: correlation with the true objective and sample efficiency. The most popular surrogate model in AutoDNN is the Gaussian process [94][82][13][158][61]. A Gaussian process $G = (\mu, \sigma)$, is defined by a mean $\mu(.)$ and a covariance function $\sigma^2(,)$. Gaussian processes are suited to continuous optimization problems and are characterized by poor scalability to high dimensions [160][159]. Then, other models have been investigated such as neural networks [157][160][159], radial basis functions [27][84], polynomial regression models [120], Tree Parzen Estimator (TPE)

[8] Also known as Bayesian Optimization

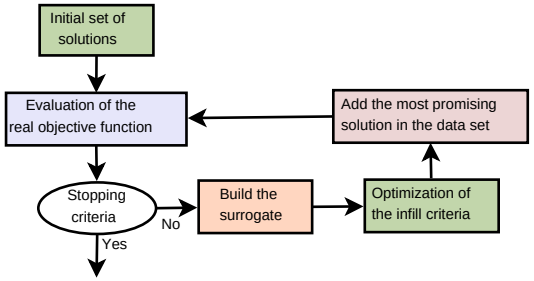

Fig. 11. The general framework of surrogate-based optimization algorithms.

[13], RNNs [27][46], graph neural networks [120] and random forest [82]. A recent trend consists in using multiple surrogates (i.e. ensembles of metamodels) to improve the accuracy of the surrogates [27].

- **Acquisition function**: the acquisition function determines the utility of different DNN candidates. They are based on a tradeoff between exploration by searching where predicted variance is high, and exploitation by searching where expected value is minimized. Different *infill criteria* have been used for updating the surrogate: lower confidence bound (LCB), upper confidence bound (UCB) [158][88], probability of improvement (PI), expected improvement (EI) [94][27][13][120], independent Thompson sampling [187], and predictive entropy search (PES) [72].
- **Target optimization problem:** several techniques exist in SBO of continuous functions. Hence, SBO has been widely used in solving the HPO problem. For instance, it has been applied to tune the number of layers and the size of hidden layers in DBNs [13] and deep neural networks [169], the size of the filter bank, and other hyperparameters (e.g. learning rate) in CNNs [86][194][14][125][158].
  Although SBO has seen great success in the HPO problem, several issues arise when it comes to solve the NAS and AutoDNN problems because of the discrete variables. Only few methods have been developed for mixed continuous/discrete problems [136]. Indeed, using SBO for AutoDNN requires so far specifying a distance function between DNN architectures, in order to define a surrogate model (i.e. kernel function). The kernel function, which measures the similarity between network architectures, is fundamental for selecting the architectures to evaluate during the search process [27][94][187]. As modern DNNs can have multiple layers, multiple branches and multiple skip connections, comparing two DNNs is non-trivial. In [27], the authors propose to map a diverse range of discrete architectures to a continuous embedding space through the use of RNNs and then define the kernel function based on the learned embedding space. In [94], the authors develop a distance metric in the space of CNN architectures which is computed via an optimal transport algorithm.
- **Optimization algorithms:** there are two different optimization algorithms to be defined: (1) the algorithm which optimizes the surrogate. Many optimization algorithms have been investigated such as EAs [94][120], gradient [27], and beam search [104]; (2) the algorithm which optimizes the acquisition function. Many global optimization algorithms have been applied such as EDA [13], CMA-ES [13], random procedure [187], and simulated annealing [88].

6 MULTI-OBJECTIVE OPTIMIZATION

Most of the work on AutoDNN formulate the problem as a single-objective problem based on the accuracy. However, many applications do not only require high accuracy on unseen data but also other objectives (e.g. inference time, model size, energy consumption). A multi-objective optimization problem (MOP) can be defined as [126]: $min_{a \in \mathbf{A}}(f_1(a), f_2(a), ..., f_k(a))$, where k ($k \geq 2$) is the number of objectives, and $\mathbf{A}$ denotes the set of feasible DNNs. Contrary to single-objective optimization, the solution of a MOP is not a single solution, but a set of solutions known as *Pareto optimal set*, which is called *Pareto front* when it is mapped in the objective space. Any solution of this set is optimal in the sense that no improvement can be made on one objective without worsening at least another objective. A solution a dominates a solution b if and only if:

$$\forall i \in [1..k] : f_i(a) \leq f_i(b) \quad \text{and} \quad \exists \in [1..k] : f_i(a) < f_i(b)$$

The Pareto optimal solutions are not dominated by any other solutions in the feasible space. A solution a is *Pareto optimal* iff: $\forall b \in \mathbf{A}, \forall i \in [1..k], f_i(a) \leq f_i(b) \quad \text{and} \quad f(a) \neq f(b)$.

6.1 Multi-objective single-task learning

In classical single-task learning problems, DNNs give high accuracy at the cost of high-computational complexity (e.g. billions of FLOPs). Recently, AutoDNN approaches have been applied to applications requiring light-weight models and fast run-time. It can be infeasible to run real-time applications on resource constrained platforms such as IoT, smartphones, robots, drones, autonomous vehicles and embedded systems. Indeed, those platforms are often constrained by hardware resources in terms of power consumption, available memory, available FLOPs, and latency constraints. Optimizing those multiple objectives will enable efficient processing of DNNs to improve energy efficiency and throughput without sacrificing application accuracy or increasing hardware cost. This is a critical aspect to the wide deployment of DNNs in AI systems. Many device-related and device-agnostic objectives have been investigated in the literature for the optimization and/or the inference steps:

- **Energy consumption:** in using DNN models in low-power mobile and embedded areas, there is a need to optimize the energy consumption (i.e. power) [134][82]. Power can be estimated via analytical models [23][145], simulation software [71] or measured on the target device (i.e. hardware-aware) [82]. It depends if the platform where the DNN is designed and the platform on which it is deployed are connected.
- **Inference speed:** the inference time is an important objective for real-time applications [96]. To measure this objective, it is necessary to deploy DNNs on the target hardware device [82][46].
- **Computational and memory cost:** this cost can be estimated by the number of floating-point operations (FLOPs) [117], and memory usage that a network performs during a forward phase [157][78][46]. This measure can concern both training and inference [180].
- **Hardware cost:** the cost of the hardware on which training and/or designing are carried out can also be taken into account [112].
- **Number of parameters:** minimizing the number of parameters of DNNs has been used in [113][174][87] as a second objective for an efficient deployment of DNNs on constrainted hardware (e.g. mobile devices [49]).
- **Size of the network:** it is mainly evaluated by the number of connection in the network [59]. The connecting sparsity has been considered in designing DBNs [110] and RNNs [156]. In [67], an objective consists in minimizing the number of non-zero weights. The resulting compressed networks will have lower bandwidth requirements and require fewer multiplications due to most weights being equal to zero.

- **Diversity:** ensemble models using diverse DNNs tends to achieve better generalization [21]. Diversity measures the discrepancy between the output of a DNN and the outputs of other DNNs. An example of such diversity wich measures the total correlation between the output of one DNN and the output of each of the other DNNs is [29]:

$$Min \;\; D_m = \sum_{i=1}^{N} (o_m^i - O^i) \sum_{j=1, i \neq j}^{M} (o_j^i - O^i)$$

where M is th number of DNN models, N the number of samples, o_m^i, o_j^i represents the output of the mth and the jth DNN for the ith training sample, and O^i denotes the average output for all DNNs. In [29], the Pareto DBNs networks are combined to form an *ensemble model*, where combination weights are optimized via a single-objective DE for a given learning task.

The aim of solving MOPs is to help a DNN designer to find a Pareto DNN which copes with his preferences. One of the fundamental questions in MOPs resolution is related to the interaction between the problem solver (e.g. metaheuristic) and the designer. Indeed, the Pareto DNNs cannot be ranked globally. The role of the designer is to specify some extra information to select his favorite solution. This interaction can take one of the three following forms: *a priori* [78], *a posteriori* [29][156], and *interactive*. To our knowledge there is no work dealing with interactive design of DNNs, where there is a progressive interaction between the designer and the optimizer. Different optimization approaches have been designed for multi-objective autoDNN:

- **Scalarization approaches:** those approaches transform the MOP problem into a single-objective one or a set of such problems. Among these methods one can find the aggregation methods, weighted metrics, Tchebycheff method, goal programming methods, achievement functions, goal attainment methods and the ϵ-constraint methods [126]. In [78], a weighted sum function $\alpha f_1 + (1-\alpha) f_2$ which agregates accuracy and energy consumption has been used to solve a bi-objective optimization problem. In [3], the authors provide a balance between the compression ratio and the accuracy using the function $f(x) = C(x)(2 - C(x)) \times \frac{A(x)}{A(ref)}$ where $C(x)$ is the compression ratio of the architecture x, $A(x)$ is the validation performance of x and $A(ref)$ is the validation performance of the reference network. The compression ratio $C(x)$ is defined as $C(x) = 1 - \frac{\#param(x)}{\#param(ref)}$.
- **Pareto approaches:** dominance-based approaches[9] use the concept of dominance and Pareto optimality to guide the search process. Population-based metaheuristics are particularly suitable to solve MOPs, because they deal simultaneously with a set of solutions which allows to find several Pareto DNNs in a single run of the algorithm. The main differences between the various proposed approaches arise in the following search components: fitness assignment, diversity management, and elitism [173]. Pareto EAs (e.g. NSGA-II: Non-Sorting Genetic Algorithm) have mostly been used in the literature for designing CNNs [96][117][113], RNNs [156] and LSTMs [10]. Other Pareto optimization algorithms have also been considered such as PSO (e.g. diversity based on crowding and dominance based on ϵ-Pareto dominance [180]), and local search [157].
- **Decomposition-based approaches:** most of decomposition-based algorithms in solving MOPs operate in the objective space. One of the well-known frameworks for MOEAs using decomposition is MOEOA/D [195]. It uses scalarization to decompose the MOP into multiple scalar optimization subproblems and solve them simultaneously by evolving a population of DNNs. Subproblems are solved using information from the neighbouring subproblems

[9] Also named Pareto approaches.

[121]. This approach has been developed using Tchebycheff scalarization in designing DBNs [110][29].

Most of the proposed MOP formulations are bi-objective. Very few many-objective models (i.e. more than 3 objectives) have been investigated. In [49], a 5-objective MOP has been formulated: accuracy on data set CIFAR-10, accuracy on data set CIFAR-100, number of parameters, number of add-multiply operations and inference time. Compared to accuracy, the proposed additional objectives (e.g. inference time, energy consumption) are generally cheap to evaluate. Hence, developing new MOP approaches which take into account this high heterogeneity in the computational cost of the objectives is essentiel. An approach based on decoupled objective evaluations has been proposed to enable independent evaluations across objectives [71]. In [82][49], a sequential approach is developed in handling cheap and expensive objective functions. First, cheap objectives are used to sample new solutions. Then, in a second phase, expensive objectives participate in the search process to generate Pareto DNNs for the whole MOP.

In surrogate-based MOP, new acquisition functions have to be developed. To identify Pareto-optimal DNNs, an acquisition function based on the hypervolume indicator has been proposed in [82]. In [82], the authors consider surrogate-based MOP with heterogeneous cost objectives. The acquisition function selects the objective across which the configuration will be evaluated in addition to selecting the next DNN to evaluate. A trade-off is made between the additional information obtained through an evaluation with the cost of obtaining it.

6.2 Multi-objective multi-task learning

Multi-task learning (MTL) allows to learn multiple different yet related tasks simultaneously. MTL has recently been explored in a variety of DNNs solving problems in computer vision [16] and NLP [41]. The number of parameters in a multi-task DNN would be less than in multiple DNNs optimized for their own single task. In addition, the trained DNNs for MTL should be able to synergize, enabling superior performance over learning each task independently using smaller datasets per task [93]. The MTL problem is inherently multi-objective, in which the various tasks may conflict. Hence, some trade-off models represented by Pareto solutions have to be found. In the literature MTL is mostly solved as a single objective optimization problem via hard or soft parameter sharing [146]. In hard parameter sharing, a subset of parameters is shared between tasks, while other parameters are task specific. In soft parameter sharing, all parameters are task specific, but they are jointly constrained via a Bayesian prior or a joint dictionary. In the design of a global model that shares some parameters across tasks, the parameters can be learned by solving a MOP that takes into account all uncertainties on the defined tasks.

Very few works in the literature investigate a multi-objective approach to solve the MTL problem. Various objectives (i.e. loss functions for different tasks) can be handled in the formulation of the problem: $Min_{\Theta}\ (\mathbf{L}_1(a, \theta_c, \theta_s^1), ..., \mathbf{L}_T(a, \theta_c, \theta_s^T)),\ t = 1, 2, \ldots, T$, where $\mathbf{L}_i$ is the loss function function of task i, T is the total number of tasks, θ_c are the shared parameters, and θ_s^t are the specific task parameters. Hence, Pareto solutions representing potential optimal architectures will be generated to solve the MTL problem. To our knowledge only scalarization approaches have been proposed. In [150][103], a weighted linear aggregation of the per-task losses has been applied and solved using gradient-based algorithms. In [62], many weighting approaches have been evaluated including uniform combination of losses, dynamic weight average (DWA) [111] and uncertainty weighting methods [95] with various sizes of datasets per-task.

7 PARALLEL OPTIMIZATION

On one hand, AutoDNN problems are more and more complex (e.g. dataset and network size) and their resource requirements in terms of computation and memory are ever increasing. Although the use of metaheuristics allows to significantly reduce the computational complexity of the search process, it remains time-consuming. On the other hand, the rapid development of technology in hardware design (e.g. GPU, TPU) makes the use of parallel computing increasingly popular. State-of-the-art DNNs required 3,150 and 2,000 GPU days [143][204]. Parallel optimization can be used for the following reasons: speedup the search, improve the quality of DNNs, reduce the energy, improve the robustness, and solve large scale and/or complex learning tasks. In this paper we make a clear distinction between the parallel design aspect and the parallel implementation aspect.

7.1 Parallel design

In terms of designing parallel metaheuristics for AutoDNN, three major parallel hierarchical models are identified (Fig.12):

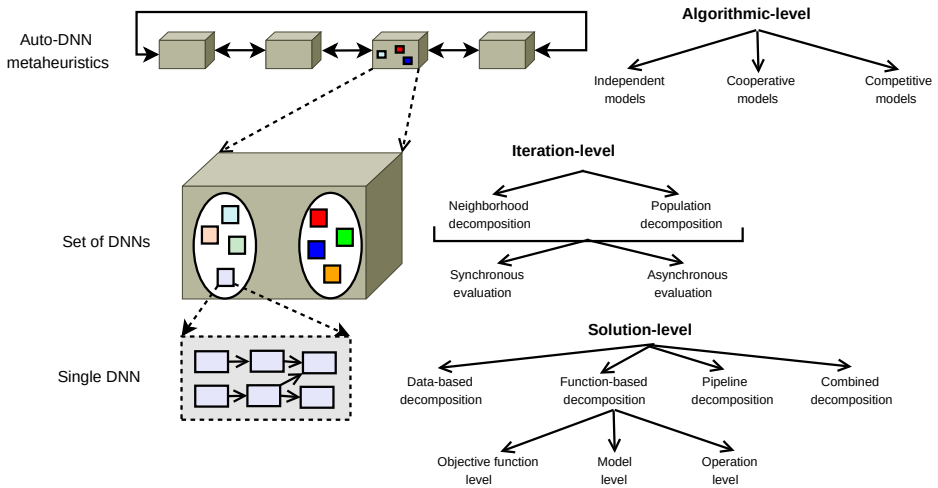

Fig. 12. Parallel models of metaheuristics for AutoDNN.

- **Algorithm-level:** in this parallel model, independent, cooperating or competitive self-contained metaheuristics are used. If the different metaheuristics are independent, the search will be equivalent to the sequential execution of the metaheuristics. However, cooperative and competitive models can alter the behavior of the metaheuristics and enable the improvement of the quality of DNNs by providing better convergence and diversity. Very few algorithm-level parallel models have been investigated in the literature. A parallel independent approach has been investigated for transfer learning [55]. A set of parallel asynchronous agents learn how to reuse the architecture of an existing DNN for new learning tasks. An agent based on an EA is used to evolve a subpart (i.e. PathNets) of a giant DNN. PathNet may be thought of as a form of "evolutionary dropout" in which instead of randomly dropping out nodes and their connections, "thinned" DNNs are evolved in parallel for various learning tasks.

Some parallel cooperative models has been developed for neural network design [44][127]. The algorithm is based on the popular "Island Parallel EA" in which a set of parallel EAs are co-operating to solve the problem [172]. In [127], two populations of DNN cells and topologies are evolving in parallel. During evaluation, the cells are combined into topologies to create a larger assembled DNNs. An example of algorithm-level competitive parallel model can be found in designing generative neural networks (GANs). GANs are composed of two adversarial DNNs: a generator and a descriminator [64]. The two networks are confronted in a zero-sum game. The generator creates fake noisy input data to deceive the discriminator, while the discriminator learns to distinguish between real and fake samples. In contrast to conventional GANs, which alternate the update of the generator and a discriminator, some algorithm-level parallel EA models have been proposed [37][182]. In [37], a co-evolutionary approach has been used, in which the discriminator and generator population of networks are trained simultaneously as adversaries. Two populations of generators and discriminators evolve in parallel following its own optimization process. The discriminator D (resp. generator G) networks optimize the following loss function: $L_D(D,G) = -\mathbb{E}_{x\ data}[logD(x)] - \mathbb{E}_{z\ noisy}[log(1 - D(G(z)))]$ (resp.) $-\mathbb{E}_{z\ noisy}[log(D(G(z)))]$ where $data$ represents the input dataset, z (resp. $noisy$) represents the noisy data (resp. noise distribution).

- **Iteration-level:** in this model, an iteration of a metaheuristic is parallelized. The behavior of the metaheuristic is not altered. The main goal is to speedup the algorithm by reducing the search time. Indeed, the iteration cycle of metaheuristics requires a large amount of computational resources for training. The most popular iteration-level parallel model consists in evaluating in parallel the generated DNNs. In the synchronous mode, a master manages the search process. At each iteration, the master distributes the set of new generated DNNs among the workers and waits for the results of all DNNs (e.g. EAs [143][122][193], $(1 + \lambda)$ES [123], PSO [115][180], multi-armed bandits [53]). While the results are collected, the search process is iterated. In the asynchronous mode, the evaluation phase is not synchronized with the other parts of the search process in EAs [108] and ACO [48]. The master does not wait for the return back of all DNNs evaluations to start the next iteration. The steady-state EA is a good example illustrating the asynchronous model [108].
- **Solution-level:** in this model, the parallelization process handles the training of a single DNN which is the most costly operation [11]. Training broadly comprises iterations over two dataflows steps: the forward step for training the sample data, and the backward step for updating weights (e.g. computing gradients). Four solution-level parallel models may be carried out for training:
  - **Data-based decomposition:** the same DNN model is duplicated among different workers with different portions of the training data [42]. The computations are carried out in parallel on different data partitions. In [138], each worker stores an identical copy of the model and computes gradients only on a partition of the training examples, and these gradients are aggregated to update the model.
  - **Function-based decomposition:** the DNN model is partitioned into different sub-functions. Each sub-function is evaluated in parallel using the same training data. Then, a reduction is performed on the results returned back by the computed sub-functions. By definition, this model is synchronous, so one has to wait the termination of all workers calculating the operations. Three different levels of function decomposition can be applied: (1) Objective level in which different objective functions are evaluated in parallel such as in multi-objective AutoDNN (2) Model level in which different sub-models (e.g. operations) are handled in parallel. For example, in [98], different workers train different parts of the model. A convolution with k filters can be splitted in n operations, each of which convolves

its input with $\frac{k}{n}$ filters. (3) Operation level in which a given operation (e.g. convolution) is handled in parallel. For instance, a FC layer can be modeled as matrix-matrix multiplication and is well suited to be performed in parallel [11].

- **Pipeline decomposition:** it consists in designing a pipeline of the layers composing a DNN, where one or more consecutive layers of a DNN form a chunk. The layers in a chunk are executed on one worker, and thus, different workers compute the DNN in a pipelined manner [80][69]. This parallel computing model is efficient for large DNNs and/or large datasets.
- **Combined decomposition:** the previous strategies can be combined. For instance, the function, data parallel and pipeline models can be used jointly. A combined parallelisation mixing functional and data parallelism has been proposed in [98][191]. In [98] the authors use data parallelism in the convolutional layers (i.e. compute intensive) and function parallelism in the FC layers (i.e. memory-intensive). Very few papers combine pipelining, function parallelism, and data parallelism [69].

7.2 Parallel implementation

Parallel implementation of AutoDNN metaheuristics deals with the efficient mapping of a parallel model of metaheuristics on a given parallel architecture. Computational throughput, power consumption and memory efficiency are three important indicators in parallel architectures. Parallel architectures are evolving quickly and are dominated by two types of architectures: *shared memory architectures* (e.g. multi-core CPU, accelerators such as GPU) and *distributed memory architectures* (e.g. clusters of CPUs).

Shared-memory architectures: accelerators and multi-core CPUs represent the most popular shared-memory architectures. Accelerators are often connected with a server through PCIe bus. They can be classified as *temporal* or *spatial architectures*. Popular temporal architectures are multi-cores and GPUs. They use SIMT (Single Instruction Multiple Threads) and SIMD (Single Instruction Multiple Data) as parallel computing models. They use a centralized control for a large number of ALUs, which can only fetch data from the memory hierarchy. Due to their high throughput support and an architecture designed specifically for data parallel workflows, GPUs are well adapted for DNN computational requirements. Using CUDA API, these frameworks boost their scale-up efficiency using threads to utilize multiple GPUs in a single node (single address space). However, conventional CPUs and GPUs are energy-inefficient due to their effort for flexibility, and then they are not preferred for power constrained applications [54]. Spatial architectures use dataflow processing, where a processing sequence is composed of ALUs transfer data from one to another. FPGA and ASICS (Application-Specific integrated Circuits) are the most widely used spatial architectures. FPGA allows to implement irregular parallelism, customized data type and application-specific hardware, offering great flexibility to accommodate new DNN models. However, their drawbacks are the on-chip memory limitation, and the lack of efficient high-level APIs. A major improvement in cost-energy performance comes from domain-specific hardware such as TPUs (Tensor Processing Units). They are AI-dedicated ASIC which targets a high volume of low-precision (e.g. 8-bit) arithmetic while maintaining low power consumption. However, they have less flexibility and longer development cycle than FPGAs [181].

Distributed-memory architectures: Clusters of CPU nodes represent the most popular distributed-memory architecture. The computer nodes are connected by high speed networks such as modern Ethernet and InfiniBand. The most important metrics for the interconnection network are latency and bandwidth. MPI (Message Passing Interface) is the omnipresent programming model for distributed memory architectures. The performance of single-node multi-GPU is nearing saturation

for large datasets and DNN models. Thus, scale-out efficiency with large clusters of heterogeneous nodes (e.g. CPU-GPU) is an emerging topic. Most of the top high-performance computing (HPC) systems[10] are composed of clusters of heterogeneous nodes (CPU and GPU) mixing shared-memory and distributed-memory models. Hence, traditional HPC applications have been successfully re-designed to scale-out using a hybrid programming model mixing MPI and CUDA.

Parallel implementation of AutoDNN metaheuristics on parallel hardware has to be consider maximizing accuracy and throughput, while minimizing energy and cost. The throughput of a parallel metaheuristic on a given parallel architecture depends mainly on its granularity. It computes the ratio between the computation cost and the communication cost. The three parallel models have a decreasing granularity from large-grained to fine-grained:

- **Algorithm-level:** this model has the largest granularity. There is a relatively low communication requirements. It is the most suited parallel model to conventional parallel architectures such as HPC systems and clusters of multi-cores. In terms of scalability, its degree of concurrency is limited by the number of metaheuristics involved in solving the problem. In [55], an implementation on a cluster of CPUs using 64 asychronous independent algorithms has been carried out.
- **Iteration-level:** a medium granularity is associated to the iteration-level parallel model. As the objective function is very expensive, this model has been widely implemented on multi-GPUs and clusters of multi-cores. It has been deployed efficiently for EAs [180][122][193], PSO [115] on GPUs powered clusters using MPI-CUDA, and ACO on clusters of multi-core CPUs using MPI [48]. The degree of concurrency of this model is limited by the size of the neighborhood for S-metaheuristics or the size of the population for P-metaheuristics. The use of very large neighborhoods and large populations will increase the scalability of this parallel model. Introducing asynchronous communications in the model will increase the efficiency of parallel metaheuristics [122].
- **Solution-level:** this model has the finer granularity and is therefore adapted for accelerators such as GPUs, FPGAs and TPUs [187]. The degree of concurrency of this parallel model is limited by the number of objective functions, data partitions and layers of DNNs. Most of the existing parallel implementations have been carried out on single GPUs. Few parallel models have been implemented on specific hardware such as FPGAs [128] (e.g. CNNs [181][68], LSTMs [197]), and Arm processors [112]. The size of DNNs raises some problems according to the GPU memory. The system would crash because of a shortage of GPU memory. Many approaches have been proposed to find memory-efficient DNNs [174]. Some parallel implementation have been developed on a single server with multiple GPUs having disjoint memory spaces [191][132]. The single server implementation scales only to 8 GPUs before the host server becomes overburdened by I/O, power, cooling, and CPU compute demands. Multiple servers where each server represents a cluster of multi-core and/or multiple GPU represent a more scalable implementation [35]. As communication is the major bottelneck in large clusters of GPUs, many techniques have been proposed to overlap communication and computation [47][83][151]. Many solution-based parallel models have been investigated:
  - **Data-based decomposition:** each node (e.g. GPU) trains on its own data partition while synchronizing weights with other nodes, using either collective communication primitives [65] or sharing memory with servers [39]. Data-based decomposition requires synchronous communication between nodes, since each node must communicate both gradients and parameter values on every update step [35]. Moreover, the mini-batch size gets multiplied by the number of used nodes.

[10] Top500.

- **Function-based decomposition:** it has been implemented on large clusters of CPUs [42], and HPC clusters of heterogenoeus nodes (i.e. multi-core CPU/GPU) using CUDA and MPI [131]. The operation level is always handled by single node accelerators. For instance, convolution in CNN and gate systems in RNNs (i.e. matrix-matrix multiplication) are generally implemented on fine grained architectures such as vector accelerators of CPUs or many-core architectures (e.g. GPU) [11]. The model level is generally implemented on clusters of GPUs and/or CPUs [35][11]. The objective level is generally implemented on heterogenous architectures. For multi-objective AutoDNN, one can decouple the evaluation of heterogeneous objectives on different hardware platforms. For instance, one can evaluate the accuracy on non-dedicated hardware and energy consumption on specific hardware [71].
- **Pipeline decomposition:** limited network bandwidth hardware induces high communication-to-computation ratios. Pipelining different micro-batches on sub-functions of layers allows to benefit memory utilization and thus make fitting giant models feasible. GPipe provides the flexibility of scaling a variety of different networks to gigantic sizes efficiently, and has been implemented on a single server with TPUv3s and NVIDIA P100 GPU [80]. Pipelining can also be applied between training different DNNs where the optimizer generates the next DNN to be trained and starts the training on GPU. Then, instead of waiting for the training to finish, it starts to generate the next DNN [88]. The idle time of nodes (e.g. GPU, CPU) is then reduced.

DNN librairies (e.g. cuDNN, Cuda-convnet) and frameworks (e.g. Tensorflow, Caffe, Torch, Thenao) have been developed to facilitate parallel implementation. Most DNN frameworks are limited to a single node (e.g. GPU) and have not been designed to be efficient of large clusters of heterogeneous nodes using MPI and CUDA [6]. TensorFlow maps the nodes of a dataflow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, general purpose GPUs, and custom-designed ASICs such as Tensor Processing Units (TPUs) and ARM-based platforms [1].

8 CONCLUSIONS AND PERSPECTIVES

In this paper, a survey and taxonomy for DNN optimization has been presented. A unified way to describe the optimization algorithms allowed to focus on common and important search components for all AutoDNN approaches. We have also extended this unifying view to important optimization methodologies dealing with surrogate-based, multi-objective and parallel optimization. Most of the proposed AutoDNN approaches have been applied to image classification. The proposed survey and taxonomy can help to extend the proposed taxonomy to other less explored applications in computer vision (e.g. image restoration, semantic segmentation), NLP (e.g. langage translation) and Industry 4.0 (e.g. predictive maintenance). It can also be reused for other types of deep learning architectures such as spiking neural networks (SNNs).

An important issue is the definition of efficient and effective encodings, objective function(s) and constraints. From a landscape analysis using measures such as FDC (i.e. fitness-distance correlation) [90], and autocorrelation (i.e. autocorrelation of the accuracies of visited DNNs in a random walk) [161], we can extract some knowledge for designing and understanding the behavior of optimization algorithms. Designing multi-fidelity surrogates for variable space mixed optimization problems represents an important research issue. The AutoDNN problem is intrinsically multi-objective. To our knowledge there is no work dealing with interactive multi-objective design of DNNs, in which there is a progressive interaction between the designer and the optimizer. Indeed, one can use his knowledge in helping the optimizer to converge towards interesting design subspaces.

HPC is evolving toward Exascale supercomputers composed of millions of cores provided in heterogeneous devices mainly multi-core processors with various architectures. To our knowledge there is no work using in conjunction the three hierarchical parallel models introduced in this paper. The massively parallel implementation of the three hierarchical parallel models on Exascale supercomputers is an interesting challenge. Moreover, highly energy-efficient hardware accelerators are required for a broad spectrum of challenging applications. Future works also need to assess the performance benefits according to the energy overheads.

The coupling of software frameworks dealing with optimization and deep learning is an important issue for the future. This enables to reduce the complexity of developing optimization approaches for new AutoDNN problems and makes them increasingly popular. Finally, some efforts must be done in the definition of performance evaluation methodologies for the comparison of different AutoDNN methodologies. Particularly, we notice the lack of information needed to exactly reproduce the published results.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium*, pages 265–283, 2016.

[2] K. Ahmed and L. Torresani. Maskconnect: Connectivity learning by gradient descent. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 349–365, 2018.

[3] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani. N2N learning: Network to network compression via policy gradient reinforcement learning. In *6th International Conference on Learning Representations, ICLR'2018, Canada*, 2018.

[4] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro. Evolving the topology of large scale deep neural networks. In *21st European Conference on Genetic Programming EuroGP'2018, Italy*, volume 10781, pages 19–34, 2018.

[5] F. Assunçao, N. Lourenço, P. Machado, and B. Ribeiro. Denser: deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines*, 20(1):5–35, 2019.

[6] A. Awan, K. Hamidouche, J. Hashmi, and D. Panda. S-caffe: Co-designing MPI runtimes and caffe for scalable deep learning on modern GPU clusters. In *ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 193–205, 2017.

[7] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *5th International Conference on Learning Representations ICLR'2017, France*. OpenReview.net, 2017.

[8] B. Baker, O. Gupta, R. Raskar, and N. Naik. Accelerating neural architecture search using performance prediction. In *6th International Conference on Learning Representations ICLR'2018, Canada*, 2018.

[9] T. Bartz-Beielstein, B. Filipic, P. Korosec, and E-G. Talbi, editors. *High-performance simulation-based optimization*, volume 833 of *Studies in Computational Intelligence*. Springer, 2020.

[10] J. Bayer, D. Wierstra, J. Togelius, and J. Schmidhuber. Evolving memory cell structures for sequence learning. In *19th International Conference Artificial Neural Networks ICANN'2009, Cyprus*, volume 5769, pages 755–764, 2009.

[11] T. Ben-Nun and T. Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Comput. Surv.*, 52(4):65:1–65:43, 2019.

[12] G. Bender, P-J Kindermans, W. Zoph, V. Vasudevan, and Q. V. Le. Understanding and simplifying one-shot architecture search. In *35th International Conference on Machine Learning ICML'2018, Sweden*, pages 549–558, 2018.

[13] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *25th Annual Conference on Neural Information Processing Systems 2011. Spain*, pages 2546–2554, 2011.

[14] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[15] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

[16] H. Bilen and A. Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *CoRR*, abs/1701.07275, 2017.

[17] B. Bischl, O. Mersmann, H. Trautmann, and C. Weihs. Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary Computation*, 20(2):249–275, 2012.

[18] D. Blalock, J. G. Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[19] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *Proceedings of the 32nd International Conference on Machine Learning ICML'2015, France*, volume 37, pages 1613–1622, 2015.

[20] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Smash: one-shot model architecture search through hypernetworks. In *6th International Conference on Learning Representations, ICLR'2018, Canada*, 2018.

[21] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.

[22] E. Byla and W. Pang. Deepswarm: optimising convolutional neural networks using swarm intelligence. In *Advances in Computational Intelligence Systems, UK*, pages 119–130, 2019.

[23] E. Cai, D-C. Juan, D. Stamoulis, and D. Marculescu. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. *arXiv preprint arXiv:1710.05420*, 2017.

[24] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), USA*, pages 2787–2794, 2018.

[25] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu. Path-level network transformation for efficient architecture search. In *Int. Conf. on Machine Learning ICML'2018, Sweden*, pages 677–686, 2018.

[26] A. Camero, H. Wang, E. Alba, and T. Bäck. Bayesian neural architecture search using a training-free performance metric. *arXiv preprint arXiv:2001.10726*, 2020.

[27] S. Cao, X. Wang, and K. M. Kitani. Learnable embedding space for efficient neural architecture compression. In *7th International Conference on Learning Representations ICLR'2019, USA*, 2019.

[28] F. P. Paolo Casale, J. Gordon, and N. Fusi. Probabilistic neural architecture search. *CoRR*, abs/1902.05116, 2019.

[29] A. Chandra and Xin Yao. Ensemble learning using multi-objective evolutionary algorithms. *J. Math. Model. Algorithms*, 5(4):417–445, 2006.

[30] C. Chen, F. Tung, N. Vedula, and G. Mori. Constraint-aware deep neural network compression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 400–415, 2018.

[31] L-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *NIPS'2018*, pages 8699–8710, 2018.

[32] T. Chen, I. J. Goodfellow, and J. Shlens. Net2net: accelerating learning via knowledge transfer. In *4th International Conference on Learning Representations, ICLR'2016, USA*, 2016.

[33] Y. Chen, K. Zhu, L. Zhu, X. He, P. Ghamisi, and J. A. Benediktsson. Automatic design of convolutional neural network for hyperspectral image classification. *IEEE Trans. Geoscience and Remote Sensing*, 57(9):7048–7066, 2019.

[34] P. Chrabaszcz, I. Loshchilov, and F. Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[35] A. Coates, B. Huval, T. Wang, D. J. Wu, B. Catanzaro, and A. Y. Ng. Deep learning with COTS HPC systems. In *Proceedings of the 30th International Conference on Machine Learning ICML'2013, USA*, pages 1337–1345, 2013.

[36] M. D. Collins and M. Kohli. Memory bounded deep convolutional networks. *CoRR*, abs/1412.1442, 2014.

[37] V. Costa, N. Lourenço, and P. Machado. Coevolution of generative adversarial networks. In *Evoapplications Int. Conf. on Applications of Evolutionary Computation, Germany*, volume 11454, pages 473–487.

[38] M. Courbariaux, Y. Bengio, and J-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems, Canada*, pages 3123–3131, 2015.

[39] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing. Geeps: scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *European Conf. on Computer Systems EuroSys'2016, UK*, pages 1–16, 2016.

[40] A. Darwish, A. E. Hassanien, and S. Das. A survey of swarm and evolutionary computing approaches for deep learning. *Artificial Intelligence Review*, 53(3):1767–1812, 2020.

[41] A. Das, M. Hasegawa-Johnson, and K. Veselý. Deep auto-encoder based multi-task learning using probabilistic transcriptions. In *18th Conf. of the International Speech Communication Association, Sweden*, pages 2073–2077, 2017.

[42] J. Dean et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[43] T. Desell, S. Clachar, J. Higgins, and B. Wild. Evolving deep recurrent neural networks using ant colony optimization. In *15th European Conf. Evolutionary Computation in Combinatorial Optimization EvoCOP'2015, Denmark*, pages 86–98, 2015.

[44] T. Dokeroglu and E. Sevinc. Evolutionary parallel extreme learning machines for the data classification problem. *Computers & Industrial Engineering*, 130:237–249, 2019.

[45] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Int. Joint Conf. on Artificial Intelligence*, 2015.

[46] J-D Dong, A-C Cheng, D-H. Juan, W. Wei, and M. Sun. Dpp-net: device-aware progressive search for pareto-optimal neural architectures. In *15th European Conf. on Computer Vision ECCV'2018, Germany*, pages 540–555, 2018.

[47] N. Dryden, N. Maruyama, T. Moon, T. Benson, A. Yoo, M. Snir, and B. Van Essen. Aluminum: An asynchronous, GPU-aware communication library optimized for large-scale training of deep neural networks on HPC systems. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2018.

[48] A. ElSaid, F. El Jamiy, J. Higgins, B. Wild, and T. Desell. Optimizing long short-term memory recurrent neural networks using ant colony optimization to predict turbine engine vibration. *Applied Soft Computing*, 73:969–991, 2018.

[49] T. Elsken, J. Hendrik, and F. Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.

[50] T. Elsken, J-H. Metzen, and F. Hutter. Simple and efficient architecture search for convolutional neural networks. In *6th International Conference on Learning Representations ICLR'2018, Canada*, 2018.

[51] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20:55:1–55:21, 2019.

[52] R. S. Engelmore and A. Morgan. *Blackboard systems*. Addison-Wesley, 1988.

[53] S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning ICML'2018, Sweden*, pages 1436–1445, 2018.

[54] Xin Feng, Youni Jiang, Xuejiao Yang, Ming Du, and Xin Li. Computer vision algorithms and hardware implementations: A survey. *Integration VLSI journal*, 69:309–320, 2019.

[55] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[56] M. Feurer and F. Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, 2019.

[57] Ben Fielding and Li Zhang. Evolving image classification architectures with enhanced particle swarm optimisation. *IEEE Access*, 6:68560–68575, 2018.

[58] S. Fong, S. Deb, and X-S. Yang. How meta-heuristic algorithms contribute to deep learning in the hype of big data analytics. In *Progress in Intelligent Computing Techniques*, pages 3–25. Springer, 2018.

[59] A. Gaier and D. Ha. Weight agnostic neural networks. In *Advances in Neural Information Processing Systems NeurIPS'2019*, pages 5365–5379, 2019.

[60] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[61] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: a service for black-box optimization. In *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 1487–1495, 2017.

[62] T. Gong, T. Lee, C. Stephenson, V. Renduchintala, S. Padhy, A. Ndirango, G. Keskin, and O. Elibol. A comparison of loss weighting strategies for multi task learning in deep neural networks. *IEEE Access*, 7:141627–141632, 2019.

[63] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[64] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems NIPS, Canada*, pages 2672–2680, 2014.

[65] P. Goyal and other. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[66] F. Gruau. Genetic synthesis of modular neural networks. In *Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign, IL, USA*, pages 318–325, 1993.

[67] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems, Canada*, pages 1135–1143, 2015.

[68] X. Han, D. Zhou, S. Wang, and S. Kimura. Cnn-merp: An FPGA-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks. In *34th Int. Conf. on Computer Design (ICCD'2016)*, pages 320–327, 2016.

[69] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

[70] 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*, pages 770–778, 2016.

[71] J. M. Hernández-Lobato, M. A. Gelbart, B. Reagen, R. Adolf, D. Hernández-Lobato, P. N. Whatmough, D. Brooks, G-Y. Wei, and R. P. Adams. Designing neural network hardware accelerators with decoupled objective evaluations. In *NIPS workshop on Bayesian Optimization*, 2016.

[72] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, pages 918–926, 2014.

[73] G. Hinton, S. Osindero, and Y-W Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[74] G. E. Hinton. A practical guide to training restricted boltzmann machines. In G. Montavon, G. B. Orr, and K-R Müller, editors, *Neural networks: Tricks of the trade*, volume 7700, pages 599–619. 2012.

[75] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[76] T. Hinz, N. Navarro-Guerrero, S. Magg, and S. Wermter. Speeding up the hyperparameter optimization of deep convolutional neural networks. *Int. Journal of Computational Intelligence and Applications*, 17(02), 2018.

[77] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[78] C. H. Hsu, S. H. Chang, J. H. Liang, H. P. Chou, C. H. Liu, S. C. Chang, J. Y. Pan, Y. T. Chen, W. Wei, and D. C. Juan. Monas: Multi-objective neural architecture search using reinforcement learning. *arXiv:1806.10332*, 2018.

[79] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[80] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, and Y. Wu. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, pages 103–112, 2019.

[81] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated Machine Learning*. Springer, 2019.

[82] Md I. M. Shahriar, J. Su, L. Kotthoff, and P. Jamshidi. Flexibo: Cost-aware multi-objective optimization of deep neural networks. *arXiv*, 2020.

[83] F. Iandola, M. Moskewicz, K. Ashraf, and K. Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 2592–2600, 2016.

[84] I. Ilievski, T. Akhtar, J. Feng, and C. A. Shoemaker. Efficient hyperparameter optimization for deep learning algorithms using deterministic RBF surrogates. In *AAAI Conf. on Artificial Intelligence*, 2017.

[85] Y. Jaafra, J-L. Laurent, A. Deruyver, and M. S. Naceur. Reinforcement learning for neural architecture search: A review. *Image and Vision Computing*, 89:57–66, 2019.

[86] R. Jenatton, C. Archambeau, J. González, and M. W. Seeger. Bayesian optimization with tree-structured dependencies. In *Proceedings of the 34th International Conference on Machine Learning ICML'2017, Australia*, pages 1655–1664, 2017.

[87] J. Jiang, F. Han, Q. Ling, J. Wang, T. Li, and H. Han. Efficient network architecture search via multiobjective particle swarm optimization based on decomposition. *Neural Networks*, 123:305–316, 2020.

[88] H. Jin, Q. Song, and X. Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956, 2019.

[89] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1:61–70, 06 2011.

[90] Terry Jones et al. *Evolutionary algorithms, fitness landscapes and search*. PhD thesis, Citeseer, 1995.

[91] R. Józefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning ICML'2015, France*, pages 2342–2350, 2015.

[92] F. E. Junior and G. Yen. Particle swarm optimization of deep neural networks architectures for image classification. *Swarm and Evolutionary Computation*, 49:62–74, 2019.

[93] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit. One model to learn them all. *CoRR*, abs/1706.05137, 2017.

[94] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2016–2025, 2018.

[95] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *IEEE Conf. on Computer Vision and Pattern Recognition CVPR'2018, USA*, pages 7482–7491, 2018.

[96] Y. H. Kim, B. Reddy, S. Yun, and C. Seo. Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy.

[97] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Int. Conf. on Artificial Intelligence and Statistics, USA*, pages 528–536, 2017.

[98] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[99] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems, USA*, pages 1106–1114, 2012.

[100] L. Li and T. Ameet. Random search and reproducibility for neural architecture search. *arXiv:1902.07638*, 2019.

[101] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[102] L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. In *Conference on Uncertainty in Artificial Intelligence UAI'2019, Israel*, page 129, 2019.

[103] X. Lin, H-L. Zhen, Z. Li, Q-F. Zhang, and S. Kwong. Pareto multi-task learning. In *Advances in Neural Information Processing Systems NeurIPS'2019, Canada*, pages 12037–12047, 2019.

[104] C. Liu, Z. Barret, N. Maxim, S. Jonathon, H. Wei, L. Li-Jia, F-F. Li, Y. Alan, H. Jonathan, and M. Kevin. Progressive neural architecture search.

[105] C. Liu, L-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei. Auto-deeplab: hierarchical neural architecture search for semantic image segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 82–92, 2019.

[106] C. Liu, L-C Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and F-F. Li. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition CVPR'2019, USA*, pages 82–92, 2019.

[107] C. Liu, Z. Zhang, and D. Wang. Pruning deep neural networks by optimal brain damage. In *INTERSPEECH 15th Conf. Int. Speech Communication, Singapore*, pages 1092–1095.
[108] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv:1711.00436*, 2017.
[109] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv:1806.09055*, 2018.
[110] J. Liu, M. Gong, Q. Miao, X. Wang, and H. Li. Structure learning for deep neural networks based on multiobjective optimization. *IEEE Trans. Neural Networks Learn. Syst.*, 29(6):2450–2463, 2018.
[111] S. Liu, E. Johns, and A. J. Davison. End-to-end multi-task learning with attention. In *IEEE Conf. on Computer Vision and Pattern Recognition CVPR'2019, USA*, pages 1871–1880, 2019.
[112] A. Lokhmotov, N. Chunosov, F. Vella, and G. Fursin. Multi-objective autotuning of mobilenets across the full software/hardware stack. In *Int. Conf. on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning, ReQuEST@ASPLOS'2018, USA*, pages 6–16, 2018.
[113] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjödin. Deepmaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocessors and Microsystems*, page 102989, 2020.
[114] P. R. Lorenzo and J. Nalepa. Memetic evolution of deep neural networks. In *Genetic and Evolutionary Computation Conference GECCO'2018, Japan*, pages 505–512, 2018.
[115] R. R. Lorenzo, J. Nalepa, L. S. Ramos, and J. R. Pastor. Hyperparameter selection in deep neural networks using parallel particle swarm optimization. In *Genetic and Evolutionary Computation Conference Companion*, pages 1864–1871, 2017.
[116] I. Loshchilov and F. Hutter. CMA-ES for hyperparameter optimization of deep neural networks. *CoRR*, abs/1604.07269, 2016.
[117] Z. Lu, I. Whalen, V. Boddeti, Y. D. Dhebar, K. Deb, E. D. Goodman, and W. Banzhaf. NSGA-NET: A multi-objective genetic algorithm for neural architecture search. *CoRR*, abs/1810.03522, 2018.
[118] G. Luo. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network modeling Analysis in Health Informatics and Bioinformatics*, 5(1):18, 2016.
[119] R. Luo, T. Fei, Q. Tao Qin, C. Enhong, and L. Tie-Yan. Neural architecture optimization. In *Advances in neural information processing systems*, pages 7816–7827, 2018.
[120] L. Ma, J. Cui, and B. Yang. Deep neural architecture search with deep graph bayesian optimization. In *Int. Conf. on Web Intelligence WI'2019, Greece*, pages 500–507, 2019.
[121] G. Marquet, B. Derbel, A. Liefooghe, and E-G. Talbi. Shake them all! - rethinking selection and replacement in MOEA/D. In *Int. Conf. on Parallel Problem Solving from Nature PPSN XIII , Slovenia*, pages 641–651. Springer, 2014.
[122] D. Martinez, W. Brewer, G. Behm, A. Strelzoff, A. Wilson, and D. Wade. Deep learning evolutionary optimization for regression of rotorcraft vibrational spectra. In *IEEE/ACM Machine Learning in HPC Environments*, pages 57–66, 2018.
[123] S. Masanori, S. Shinichi, and N. Tomoharu. A genetic programming approach to designing convolutional neural network architectures. In *IJCAI'2018*, 2018.
[124] N. M. Masood and G. M. Khan. Signal reconstruction using evolvable recurrent neural networks. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 594–602, 2018.
[125] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pages 58–65, 2016.
[126] K. Miettinen. *Nonlinear multiobjective optimization*. Springer, 1999.
[127] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. *CoRR*, abs/1703.00548, 2017.
[128] S. Mittal. A survey of fpga-based accelerators for convolutional neural networks. *Neural computing and applications*, pages 1–31, 2018.
[129] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440, 2016.
[130] R. Negrinho and G. Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv:1704.08792*, 2017.
[131] K. Ni, R. Pearce, K. Boakye, B. Van Essen, D. Borth, B. Chen, and E. Wang. Large-scale deep learning on the yfcc100m dataset. *arXiv:1502.03409*, 2015.
[132] T. Paine, H. Jin, J. Yang, Z. Lin, and T. S. Huang. GPU asynchronous stochastic gradient descent to speed up neural network training. In *2nd Int. Conf. on Learning Representations ICLR'2014, Canada*, 2014.
[133] R. Parekh, J. Yang, and V. G. Honavar. Constructive neural-network learning algorithms for pattern classification. *IEEE Trans. Neural Networks Learn. Syst.*, 11(2):436–451, 2000.
[134] E. Park, D. Kim, S. Kim, Y-D Kim, G. Kim, S. Yoon, and S. Yoo. Big/little deep neural network for ultra low power inference. In G. Nicolescu and A. Gerstlauer, editors, *International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS, Netherlands*, pages 124–132, 2015.

[135] J. Pelamatti, L. Brevault, M. Balesdent, E-G. Talbi, and Y. Guerin. How to deal with mixed-variable optimization problems: An overview of algorithms and formulations. In *World Congress of Structural and Multidisciplinary Optimisation*, pages 64–82, 2017.

[136] J. Pelamatti, L. Brevault, M. Balesdent, E-G. Talbi, and Y. Guerin. Bayesian optimization of variable-size design space problems. *Optimization & Engineering Journal*, 2020.

[137] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *Int. Conf. on Machine Learning ICML, Sweden*, volume 80, pages 4092–4101.

[138] H. Qi, E. R. Sparks, and A. Talwalkar. Paleo: A performance model for deep neural networks. In *5th Int. Conf. on Learning Representations ICLR'2017, France*, 2017.

[139] A. Rawal and R. Miikkulainen. From nodes to networks: Evolving recurrent neural networks. *arXiv preprint arXiv:1803.04439*, 2018.

[140] W. Rawat and Z. Wang. Hybrid stochastic GA-Bayesian search for deep convolutional neural network model selection. *Journal of Universal Computer Science*, 25(6):647–666, 2019.

[141] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Aging evolution for image classifier architecture search. In *AAAI Conf. on Artificial Intelligence*, 2019.

[142] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *AAAI conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[143] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Int. Conf. on Machine Learning*, pages 2902–2911, 2017.

[144] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: hints for thin deep nets. In *Int. Conf. on Learning Representations ICLR'2015, USA*, 2015.

[145] B. B. Rouhani, A. Mirhoseini, and F. Koushanfar. Delight: Adding energy dimension to deep neural networks. In *Int. Symp. on Low Power Electronics and Design ISLPED'2016, USA*, pages 112–117.

[146] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[147] S. Saxena and J. Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, pages 4053–4061, 2016.

[148] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann. Evaluating the search phase of neural architecture search. *arXiv:1902.08142*, 2019.

[149] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann. Evaluating the search phase of neural architecture search. *CoRR*, abs/1902.08142, 2019.

[150] A. Sener and V. Koltun. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, pages 527–538, 2018.

[151] A. Sergeev and M. Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv:1802.05799*, 2018.

[152] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

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

[154] T. Shinozaki and S. Watanabe. Structure discovery of deep neural network based on evolutionary algorithms. In *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4979–4983, 2015.

[155] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Int. Conf. on Learning Representations ICLR'2015*, 2015.

[156] C. Smith and Y. Jin. Evolutionary multi-objective generation of recurrent neural network ensembles for time series prediction. *Neurocomputing*, 143:302–311, 2014.

[157] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer. Neural networks designing neural networks: multi-objective hyper-parameter optimization. In *Int. Conf. on Computer-Aided Design, ICCAD'2016, USA*, page 104, 2016.

[158] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[159] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. A. P. Prabhat, and R. P. Adams. Scalable bayesian optimization using deep neural networks. In *Int. Conf. on Machine Learning, France*, pages 2171–2180, 2015.

[160] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust bayesian neural networks. In *Advances in neural information processing systems*, pages 4134–4142, 2016.

[161] P. F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical chemistry*, 20(1):1–45, 1996.

[162] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

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

[164] I. Strumberger, E. Tuba, N. Bacanin, R. Jovanovic, and M. Tuba. Convolutional neural network architecture design by the tree growth algorithm framework. In *Int. Joint Conf. on Neural Networks IJCNN'2019, Hungary*, pages 1–8, 2019.

[165] M. Suganuma, M. Ozay, and T. Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In *Int. Conf. on Machine Learning ICML'2018, Sweden*, pages 4778–4787, 2018.

[166] Y. Sun, X. Wang, and X. Tang. Sparsifying neural network connections for face recognition. In *Conf on Computer Vision and Pattern Recognition CVPR'2016, USA*, pages 4856–4864, 2016.

[167] Y. Sun, B. Xue, M. Zhang, and G. Yen. An experimental study on hyper-parameter optimization for stacked auto-encoders. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2018.

[168] Y. Sun, B. Xue, M. Zhang, and G. Yen. A particle swarm optimization-based flexible convolutional autoencoder for image classification. *IEEE Trans. on Neural Networks and Learning Systems*, 30(8):2295–2309, 2018.

[169] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. A. Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv:1409.4011*, 2014.

[170] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conf. on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[171] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Conf. on computer vision and pattern recognition*, pages 2818–2826, 2016.

[172] E-G. Talbi. Metaheuristics: from design to implementation. 2009.

[173] E-G. Talbi. A unified view of parallel multi-objective evolutionary algorithms. *Journal of Parallel Distributed Computing*, 133:349–358, 2019.

[174] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. pages 2820–2828, 2019.

[175] C. Thornton, F. Hutter, H. Hoos H, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Int. Conf. on Knowledge Discovery and Data Mining*, pages 847–855, 2013.

[176] A. Turner. *Evolving artificial neural networks using Cartesian genetic programming*. PhD thesis, University of York, 2015.

[177] T. Veniat and L. Denoyer. Learning time/memory-efficient deep architectures with budgeted supernetworks. In *Conf. on Computer Vision and Pattern Recognition CVPR'2018, USA*, pages 3492–3500, 2018.

[178] B. Wang, Y. Sun, B. Xue, and M. Zhang. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In *Congress on Evolutionary Computation (CEC)*, pages 1–8, 2018.

[179] B. Wang, Y. Sun, B. Xue, and M. Zhang. A hybrid differential evolution approach to designing deep convolutional neural networks for image classification. In *Australasian Joint Conf. on Artificial Intelligence*, pages 237–250, 2018.

[180] B. Wang, Y. Sun, B. Xue, and M. Zhang. Evolving deep neural networks by multi-objective particle swarm optimization for image classification. In *Genetic and Evolutionary Computation Conference*, pages 490–498, 2019.

[181] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, 2016.

[182] C. Wang, C. Xu, X. Yao, and D. Tao. Evolutionary generative adversarial networks. *IEEE Transactions on Evolutionary Computation*, 23(6):921–934, 2019.

[183] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *CoRR*, abs/1903.11059, 2019.

[184] T. Wei, C. Wang, Y. Rui, and C.W. Chen. Network morphism. In *Int. Conf. on Machine Learning*, pages 564–572, 2016.

[185] K. Weiss, T. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big data*, 3(1):9, 2016.

[186] Y. Weng, T. Zhou, Y. Li, and X. Qiu. Nas-unet: Neural architecture search for medical image segmentation. *IEEE Access*, 7:44247–44257, 2019.

[187] C. White, W. Neiswanger, and Y. Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *arXiv:1910.11858*, 2019.

[188] M. Wistuba, A. Rawat, and T. Pedapati. A survey on neural architecture search. *CoRR*, abs/1905.01392, 2019.

[189] L. Xie and A. Yuille. Genetic CNN. In *IEEE Int. Conf. on Computer Vision*, pages 1379–1388, 2017.

[190] S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: stochastic neural architecture search. In *Int. Conf. on Learning Representations, ICLR'2019, USA*, 2019.

[191] O. Yadan, K. Adams, Y. Taigman, and M. Ranzato. Multi-gpu training of convnets. In *Int. Conf. on Learning Representations ICLR'2014, Canada*, 2014.

[192] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. *arXiv:1902.09635*, 2019.

[193] S. R. Young, D. C. Rose, T. P. Karnowski, S-H Lim, and R. M. Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Workshop on Machine Learning in HPC Environments, USA*, pages 41–45, 2015.

[194] A. Zela, A. Klein, S. Falkner, and F. Hutter. Towards automated deep learning: efficient joint neural architecture and hyperparameter search. *arXiv:1807.06906*, 2018.

[195] Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[196] X. Zhang, Z. Huang, and N. Wang. You only search once: single shot neural architecture search via direct sparse optimization. *CoRR*, abs/1811.01567, 2018.
[197] Y. Zhang, C. Wang, L. Gong, Y. Lu, F. Sun, C. Xu, X. Li, and X. Zhou. A power-efficient accelerator based on FPGAs for LSTM network. In *IEEE Int. Conf. on Cluster Computing (CLUSTER)*, pages 629–630, 2017.
[198] G. Zhong, T. Li, W. Jiao, L-N. Wang, J. Dong, and C-L. Liu. DNA computing inspired deep networks design. *Neurocomputing*, 382:140–147, 2020.
[199] Z. Zhong, J. Yan, and C-L Liu. Practical network blocks design with q-learning. *CoRR*, abs/1708.05552, 2017.
[200] Z. Zhong, J. Yan, W. Wu, J. Shao, and C-H. Liu. Practical block-wise neural network architecture generation. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 2423–2432, 2018.
[201] Y. Zhou, S. Arik, H. Yu, H. Liu, and G. Diamos. Resource-efficient neural architect. *arXiv:1806.07912*, 2018.
[202] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. D. Reid. Structured binary neural networks for accurate image classification and semantic segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition, USA*, pages 413–422, 2019.
[203] B Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Int. Conf. on Learning Representations ICLR'2017, France*, 2017.
[204] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.