Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges

Edgar Galván and Peter Mooney

***Abstract*—A variety of methods have been applied to the architectural configuration and learning or training of artificial deep neural networks (DNN). These methods play a crucial role in the success or failure of the DNN for most problems and applications. Evolutionary algorithms (EAs) are gaining momentum as a computationally feasible method for the automated optimization of DNNs. Neuroevolution is a term, which describes these processes of automated configuration and training of DNNs using EAs. While many works exist in the literature, no comprehensive surveys currently exist focusing exclusively on the strengths and limitations of using neuroevolution approaches in DNNs. Absence of such surveys can lead to a disjointed and fragmented field preventing DNNs researchers potentially adopting neuroevolutionary methods in their own research, resulting in lost opportunities for wider application within real-world deep learning problems. This article presents a comprehensive survey, discussion, and evaluation of the state-of-the-art in using EAs for architectural configuration and training of DNNs. This article highlights the most pertinent current issues and challenges in neuroevolution and identifies multiple promising future research directions.**

***Impact Statement*—The concept of deep learning originated from the study of artificial neural networks (ANNs). ANNs have achieved extraordinary results in a variety of diverse application areas. Numerous methods have been applied to the architectural configuration and learning or training of artificial DNN and these methods play a crucial role in the success or failure of the DNN for most problems and applications. Recently, EAs have been gaining momentum as a computationally feasible method (called neuroevolution) for the automated configuration and learning or training of DNNs. This article reviews over 170 recent scientific papers describing how major EAs paradigms are being applied by researchers to the configuration and optimization of multiple DNNs. By articulating a clear understanding of the context, state-of-the-art, and feasibility of Neuroevolution, researchers in AI, EAs, and DNN will benefit from this article. The impact of this article comes from contributing toward enhancing research capacity, knowledge, and skills for researchers currently working in neuroevolution and actively engaging those considering becoming involved in this area.**

***Index Terms*—Deep learning (DL), deep neural networks (DNNs), evolutionary algorithms (EAs), machine learning, neuroevolution.**

Manuscript received July 20, 2020; revised December 9, 2020 and January 30, 2021; accepted March 16, 2021. Date of publication March 22, 2021; date of current version December 13, 2021. This work was supported by the Department of Computer Science at MU. This article was recommended for publication by Associate Editor Sanaz Mostaghim upon evaluation of the reviewers' comments. *(Corresponding author: Edgar Galván.)*

The authors are with Naturally Inspired Computation Research Group, Department of Computer Science, Maynooth University, W23 F2H6 Maynooth, Ireland (e-mail: edgar.galvan@mu.ie; Peter.mooney@mu.ie).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TAI.2021.3067574.

Digital Object Identifier 10.1109/TAI.2021.3067574

I. Introduction

DEEP learning (DL) algorithms [57], [65], [94], a subset of machine learning algorithms, are inspired by deep hierarchical structures of human perception as well as production systems. These algorithms have achieved extraordinary results in diverse areas including computer vision [159], speech recognition [58], [115], board games [145], and video games [114], to mention a few. The design of deep neural networks (DNNs) architectures (along with the optimization of their hyperparameters) and their training plays a crucial part in their success or failure [105]. Architecture search is an area of growing interest as demonstrated by the large number of scientific works published in recent years. These works can be classified into one of the following two broad categories: evolution-based methods [6], [34], sometimes referred as neuroevolution [42], [170], and reinforcement learning (RL) methods [158]. Methods falling outside these two categories have also been proposed in the specialized literature including Monte Carlo based simulations [119], random search [11] and random search with weight prediction [14], hill-climbing [37], grid search [174], Bayesian optimization [12], [76], gradient-based [103], [168], and mutual information [161], [162], [173]. RL architecture-search methods started gaining momentum thanks to their impressive results [7], [16], [101], [179], [181], [182], and more recently, EA architecture-search methods began yielding impressive results in the automatic configuration of DNNs architectures [39], [102], [150]. It has been reported that neuroevolution requires less computational time compared to RL methods [114], [130], [150], [155]. Basically, a DNN is a feedforward artificial neural network (ANN) with many hidden layers with each layer constituting a nonlinear information processing unit. Usually having two or more hidden layers in an ANN signifies a DNN. By adding more layers and more units within a layer a DNN can represent functions of increasing complexity [57].

Evolutionary algorithms (EAs) [6], [34], also known as evolutionary computation systems, are nature-inspired stochastic techniques that mimic basic principles of life. These automatic algorithms are very popular and have proven competitive in the face of challenging problems' features such as discontinuities, multiple local optima, and nonlinear interactions between variables [33]. EAs have also proven to yield competitive results in many real-world problems against other AI approaches even comparing well against results achieved by human experts [86], [88]. Finding a well-performing architecture is often a very tedious and error-prone process. Indeed, Lindauer and

Hutter [100] remark that there are over 300 works published in the area of neural architecture search (NAS) with almost a third of these studies corresponding to neuroevolution in DNNs. We have witnessed an increased number of publications from 2017 up to the date of writing when considered over a period from 2009 to present (see Fig. 1 in the Supplementary Material along with details of the systematic literature search).

We focus exclusively on architecture EAs-based search methods in DNNs as well as EAs-based approaches in training DNNs. Particularly, this work considers both landmark EAs, such as genetic algorithms (GAs) [66], evolution strategies (ESs) [13], [132], and genetic programming (GP) [87] as well as more recent EA variants, such as differential evolution (DE) [127], neuroevolution of augmenting topologies (NEAT) [149], and grammatical evolution (GE) [135]. Furthermore, we consider the main DL architectures, as classified by Liu *et al.* [105] that have been used in neuroevolution, including autoencoders (AEs) [24], convolutional neural networks (CNNs) [90], deep belief networks (DBNs) [176], [177], and restricted Boltzmann machines (RBMs) [105], [118]. Other DL architectures considered in this study include recurrent neural networks (RNNs) [75] and long short-term memory (LSTM) [52]. The previous literature reviews in the area include those conducted by Floreano *et al.* [42] and Yao [170] with more recent reviews provided by Stanley *et al.* [148], Darwish *et al.* [23], and Baldominos *et al.* [9]. The former work explains the influence of modern computational power at scale in allowing the grand ambitions of neuroevolution and DL from many years ago to be achieved and fulfilled. Darwish *et al.* [23] deliver a broader and high-level introduction and overview of swarm intelligence and EAs in the optimization of the hyperparameters and architecture for neural networks in data analytics. Baldominos *et al.* [9] work discusses a few EAs methods employed in DNNs, with particular emphasis on CNNs. In contrast to these works, this article provides a new contribution by concentrating on configuration and design of neuroevolution approaches in DL. We consider how EAs approaches are applied in DL and in particular their specific configuration for this purpose. This article delivers our estimation of the state-of-the-art works in neuroevolution in DNNs.

The rest of this article is organized as follows. Section II provides some background to DL and EAs. Section III discusses how architectures of DNNs can be evolved efficiently using EAs. Section IV discusses training of DNNs with EAs while Section V sets out major challenges and fertile avenues for future work. Finally, Section VI concludes this article.

II. Background

A. Deep Neural Networks

Deep learning (DL) emerged from works, such as Hinton *et al.* [65], studying ANNs to become a very active research area [105]. An ANN consists of multiple, simple, connected units (neurons), each producing a sequence of real-valued activations where the process of training an ANN may require "long casual chains of computational stages" [138]. A DL algorithm is a class of machine learning algorithm using multiple layers to progressively extract higher level features from the raw data input where "deep" refers to the number of transformation layers for raw data. In DL, each subsequent level attempts to learn in order to transform input data into a progressively more abstract and composite representation. In the following section, we summarise where neuroevolution in DNNs has been applied to the development of a wide range of ANNs including, but not limited to, convolutional neural networks, autoencoders, deep belief networks and recurrent neural networks. Taxonomies for such architectures can be found in [80] (CNNs) and [108] (memory networks).

1) Deep Learning Architecture: Convolutional Neural Networks (CNNs): CNNs have shown impressive performance in processing data with a grid-like topology. The deep network consists of a set of layers each containing one or more planes. Each unit in a plane receives input from a neighborhood in the planes of the previous layer. This idea of connecting units to receptive fields dates back to the 1960s with the perceptron and the animal visual cortex organization discovered by Hubel and Wiesel [70]. The input, such as an image, is convolved with trainable kernels or filters at all offsets to produce feature maps. These filters include a layer of connection weights. Usually, four pixels in a feature map form a group and this is passed through a function, such as sigmoid function or hyperbolic tangent function. These pixels produce additional feature maps in a layer. n planes are normally used in each layer so that n features can be detected. These layers are called convolutional layers. Once a feature is detected, its exact location is less important and convolutional layers are followed by another layer in charge of performing local averaging and subsampling operation. Due to the high dimensionality of the NNs' inputs' weights, a CNN classifier may cause overfitting. This problem is addressed by using a pooling process, also called subsampling or down-sampling, reducing the overall size of the signal. Normally, the CNN is trained with the usual backpropagation gradient-descent procedure proposed by Lecun *et al.* [95]. The learning process of a CNN is determined by the following three key elements: 1) sparse interaction that reduces the computational processing with kernels that are smaller than the inputs; 2) parameter sharing refers to learning one set of parameters instead of learning one set at each location, and finally; 3) equivariance representation that means that whenever the input changes, the output changes in the same manner [57]. CNNs were the first successful DL architectures applied to face detection, handwriting recognition, image classification, speech recognition, natural language processing, and recommender systems [31], [90], [172].

Early evolution of CNN architectures has been slow but remarkable. LeNet [95] proposed in the late 1990s and AlexNet [90], proposed a decade later, are very similar with two and five convolutional layers, respectively. Moreover, they also used kernels with large receptive fields in the layer close to the input and smaller filters closer to the output. A major difference is that the latter used rectified linear units as an activation function, which became a standard in designing CNNs. Since AlexNet, the use of novel and deeper models took off. Simonyan and Zisserman [146] won the ImageNet challenge with their proposed 19-layer model known as VGG19. Other networks

have been proposed that are not only deeper but use more complex building blocks. Szegedy *et al.* [159] proposed GoogLeNet, also known as Inception, which is a 22-layer network that used inception blocks. Also in 2015, the residual network (ResNet) architecture, consisting of the so-called ResNet blocks, proposed by He *et al.* [64] won the ImageNet challenge. Moreover, multiple CNNs variants have been proposed such as combining convolutions with an AE [72], RBMs [29]. A description of the variants of this network can be found in [105].

2) Deep Learning Architecture: Autoencoders (AEs): AEs are simple learning circuits designed to transform inputs into outputs with the minimum amount of distortion. An AE consists of a combination of an encoder and a decoder function. The encoder function converts the input data into a different representation and then the decoder function converts the new representation back to the original form. AEs attempt to preserve information and provide range-bounded outputs, which make them suitable for data preprocessing and iterative architectures such as DNNs [91]. Despite this work appearing in 2020, the authors suggest that there is "still relatively little work exploring the application (of EAs to neural architecture search) to autoencoders." Baldi [8] argued that while AEs "had taken center stage in the deep architecture approach" there was still very little theoretical understanding of AEs with deep architectures to date. Interesting theoretical works have started filling this important gap by studying and using mutual information [161], [173]. Choosing an appropriate AE architecture in order to process a specific dataset will mean that the AE is capable of learning the optimal representation of the data [17]. Encoding AEs within a chromosome representation means that such an approach could be broad enough to consider most AE variations [17]. As an unsupervised feature learning approach, AEs attempt to learn a compact representation of the input data whilst retaining the most important information of the representation. This representation is expected to completely reconstruct the original input. This makes initialization of the AE critical [67]. Whilst AEs can induce very helpful and useful representations of the input data they are only capable of handling a single sample and are not capable of modeling the relationship between pairs of samples in the input data.

3) Deep Learning Architecture: Deep Belief Networks (DBNs): DBNs are generative models that can be implemented in a number of ways including RBMs (see Section II-A4) and AEs (see Section II-A2). DBNs are suited to the problem of feature extraction and have drawn "tremendous attention recently" [177]. DBNs, such as other traditional classifiers, have a very large number of parameters and require a great deal of training time [18]. When RBMs are stacked together they are considered to be a DBN. The fundamental building blocks of a DBN are RBMs consisting of one visible layer and one hidden layer. When DBNs are applied to classification problems the feature vectors from data samples are used to set the values of the states of the visible variables of the lower layer of the DBN. Then, the DBN is trained to generate a probability distribution over all possible labels of the input data. They offer a good solution to learn hierarchical feature representations from data.

4) DL Architecture: Other Network Types: We introduce other well-studied network architectures namely: recurrent neural networks (RNNs), restricted boltzmann machines (RBMs), and long short term memory (LSTM). RNNs: In CNNs, input is a fixed-length vector eventually producing a fixed-length vector as output. The number of layers in the CNN determine the amount of computational steps required. RNNs are more flexible and allow operation across a sequence of vectors. The connections between the units in the network form a directed cycle and this creates an internal state of the network allowing us to exhibit dynamic temporal behavior. This internal hidden state allows the RNN to store information about the past efficiently. RNNs are well suited to sequential data prediction and this has seen them being applied to areas such as statistical language modeling and time-series prediction. However, the computational power of RNNs make them very difficult to train. The main reasons for this difficulty are due to the exploding and the vanishing gradient problems [75], although vanishing gradient has been addressed with LSTM and Gated RNNs. In theory, RNNs can make use of information in arbitrarily long sequences, but realistically they are limited to considering look-back at only a few steps.

RBMs: A Restricted Boltzmann Machine (RBM) is a network of symmetrically connected neuron-like units, which are designed to make stochastic decisions about whether to be ON or OFF. They are an energy-based neural network. In an RBM, there are no connections between the hidden units and multiple hidden layers. Learning occurs by considering the hidden activities of a single RBM as the data for training a higher level RBM [136]. There is no communication or connection between layers and this is where the restriction is introduced to a Boltzmann machine. The RBMs are probabilistic models using a layer of hidden binary variables or units to model the distribution of a visible layer of variables. RBMs have been successfully applied to problems involving high dimensional data such as images and text [93]. As outlined by Fischer and Igel [41], RBMs have been the subject of recent research after being proposed as building blocks of multilayer learning architectures or DBNs. The concept is that hidden neurons extract relevant features from the data observations. These features can then serve as input to another RBM. This stacking of RBMs allows a network to learn features from features with the goal of arriving at a high-level representation [115].

LSTM: Long-short-term memory (LSTM) networks are a special type of RNNs capable of learning long-term dependencies. They work incredibly well on a large variety of problems and are currently widely used. The basic unit within the hidden layer of an LSTM network is called a memory block containing one or more memory cells and a pair of adaptive, multiplicative gating units, which gate input and output to all cells in the block [52]. In LSTM networks, it is possible to circumvent the problem of the vanishing error gradients in the network training process by the method of error back propagation. An LSTM network is usually controlled by recurrent gates. Errors are propagated back in time through a potentially unlimited number of virtual layers. In this way, learning takes place in LSTM, while preserving the memory of thousands and even millions of

time intervals in the past. Network topologies such as LSTM can be developed in accordance with the specifics of the problem. Recurrent neural networks (RNNs) with long short-term memory (LSTM) have emerged as an effective and scalable model for several learning problems related to sequential data [59]. Gers and Schmidhuber [53] showed that standard RNNs fail to learn in the presence of time lags exceeding as few as five to ten discrete-time steps between relevant input events and target signals. LSTM is not affected by this problem and are capable of dealing with minimal time lags in excess of 1000 discrete-time steps. LSTM clearly outperforms previous RNNs not only on regular language benchmarks (according to previous research) but also on context-free languages benchmarks [53].

B. Evolutionary Algorithms

Evolutionary algorithms (EAs) [6], [34], also known as evolutionary computation systems, refer to a set of stochastic optimization bioinspired algorithms that use evolutionary principles to build robust adaptive systems. The field has its origins in four landmark evolutionary methods: genetic algorithms [66], [55], evolution strategies [133], [139], evolutionary programming [43], and genetic programming [87]. The key element to these algorithms is undoubtedly flexibility in allowing the practitioner to use elements from two or more different EAs techniques. Consequently, the boundaries between these approaches are no longer distinct allowing a more holistic EA framework to emerge. EAs work with a population of μ-*encoded* (representation of the) potential solutions to a particular problem. Each potential solution, commonly known as an individual, represents a point in the search space, where the optimal solution lies. The population is evolved by means of genetic operators, over a number of generations, to produce better results to the problem. Each individual is evaluated using a fitness function to determine how good or bad the individual is for the problem at hand. The fitness value assigned to each individual in the population probabilistically determines how successful the individual will be at propagating (part of) its code to future generations.

The evolutionary process is carried out by using genetic operators. Selection, crossover, and mutation are the key operators used in most EAs. The selection operator is in charge of choosing one or more individuals from the population based on their fitness values. Multiple selection operators have been proposed. One of the most popular selection operators is tournament selection where the best individual is selected from a pool, normally of size $= [2–7]$, from the population. The stochastic crossover, also known as recombination, operator exchanges material normally from two selected individuals. This operator is in charge of exploiting the search space. The stochastic mutation operator makes random changes to the genes of the individual and is in charge of exploring the search space. The mutation operator is important to guarantee diversity in the population as well as recovering genetic material lost during evolution. This evolutionary process is repeated until a stopping condition is reached such as until a maximum number of generations has been executed. The population, at this stage, contains the best evolved potential solutions to the problem and may also represent the global optimal solution. Algorithm 1 shows the typical steps considered in EAs. The main EAs employed in DNNs are genetic algorithms, genetic programming, evolution strategies. Others are differential evolution, grammatical evolution and neuroevolution of augmenting topologies. Fig. 2, in the supplementary material, shows how GAs is the only EA method used for the training of DNNs and the rest of these for the configuration of DNNs. Taxonomies of these algorithms can be found in [44] and [160].

1) Evolutionary Algorithms: Genetic Algorithms (GAs): This EA was introduced by Holland [66] in the 1970s and highly

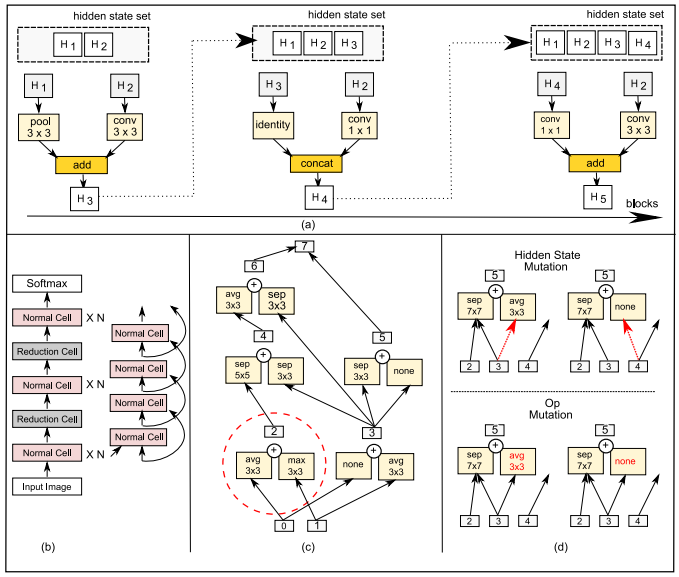

Fig. 1. (a) NASNet Search Space [182]. (b) Scalable architecture for image classification consisting of two repeated motifs termed normal cell and reduction cell. Left: The full outer structure (omitting skip inputs for clarity) and Right: Detailed view with the skip inputs. (c) Example of a cell: dotted red circle demarcates a pairwise combination. (d) Examples of how mutations are used. (a) and (b)–(d) redrawn from Zoph *et al.* [182] and Real *et al.* [130].

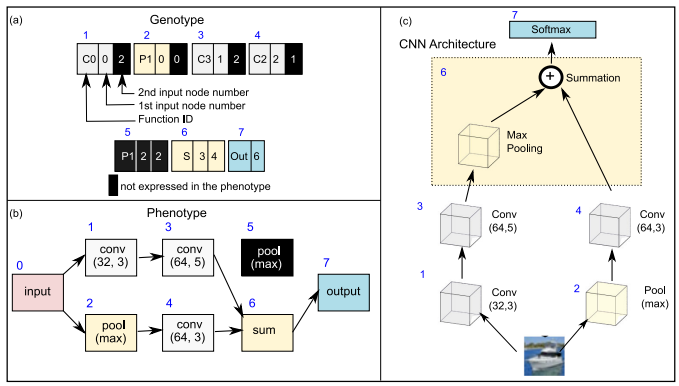

Fig. 2. (a) Genetic representation of a CGP individual encoding a CNN architecture. (b) Phenotypic representation. (c) CNN architecture defined by (a). Gene No. 5, coloured with a black background in the genotype (a) is not expressed in the phenotype. The summation node in (c), with light yellow background, performs max pooling to the LHS of the input (Node no. 3) to get the same input tensor sizes. Redrawn from Suganuma *et al.* [153].

Algorithm 1: A Common EA Process for Network Design. Adapted From [167].

Input: the reference dataset D, the number of generations T, the number of individuals in each generation N, the mutation and crossover probabilities P_m and P_c;
Initialisation: generating a set of randomised individuals $\{\mathbb{M}_0, n\}_{n=1}^{N}$, and computing their recognition accuracies;
for $t = 1, 2, \ldots, T$ **do**
  Selection: producing a new generation $\{\mathbb{M}'_{t,n}\}_{n=1}^{N}$ with a Russian roulette process on $\{\mathbb{M}_{t-1,n}\}_{n=1}^{N}$;
  Crossover: for each pair $(\{\mathbb{M}_{t,2n-1}, \mathbb{M}_{t,2n})\}_{n=1}^{[N/2]}$, performing crossover with probability P_c;
  Mutation: for each non-crossover individual $\{\mathbb{M}_{t,n}\}_{n=1}^{N}$, performing mutation with probability P_m;
  Fitness evaluation: computing the fitness (e.g., recognition accuracy) for each new individual $\{\mathbb{M}_{t,n}\}_{n=1}^{N}$;
end for
Output: a set of individuals in the final generation $\{\mathbb{M}_T, n\}_{n=1}^{N}$ with their fitness values.

popularised by Goldberg [55] subsequently achieving extraordinary results as well as reaching multiple research communities, including machine learning and neural networks. GAs were frequently described as function optimizers, but now the tendency is to consider GAs as search algorithms able to find near-optimal solutions. Multiple forms of GAs have been proposed in the specialized literature. The bitstring fixed-length representation is one of the most predominant encodings (representation) used in GAs. Crossover, as the main genetic operator, and mutation as the secondary operator, reproduce offspring over evolutionary search.

2) Evolutionary Algorithm: Genetic Programming (GP): This EA is a subclass of GAs popularised by Koza [87]. GP is a form of automated programming where individuals are randomly created by using functional and terminal sets required to solve a given problem. Multiple types of GP have been proposed in the literature with the typical tree-like structure being the predominant form of GP in EAs. Cartesian GP (CGP) [113] is another form of GP and has been used in neuroevolution in DNNs [152], [153].

3) Evolutionary Algorithm: Evolution Strategies (ES): These EAs were introduced in the 1960s by Rechenberg [133]. ESs are generally applied to real-valued representations of optimization problems. In ES, mutation is the main operator whereas crossover is the secondary, optional, operator. Historically, there were two basic forms of ES, known as the (μ, λ)-ES and the $(\mu + \lambda)$-ES. μ refers to the size of the parent population, whereas λ refers to the number of offspring that are produced in the following generation before selection is applied. In the former ES, the offspring replace the parents whereas in the latter form of ES, selection is applied to both offspring and parents to form the population in the following generation. The covariance matrix adaptation-ES, proposed by Hansen [61]–[63], is state-of-the-art and adapts the full covariance matrix of a normal search distribution.

4) Evolutionary Algorithm: Evolutionary Programming: These EAs were proposed in the 1960s by Fogel [43] but there is little difference between ES and EP. The difference lies in the lack of use of crossover in EP whereas this genetic operator rarely used in ES. In EP normally M parents produce M offspring whereas in ES the number of offspring produced by genetic operators is higher than their parents.

5) EA: Others: Multiple evolutionary-based algorithms have been used in DNNs, most notably: differential evolution (DE), grammatical evolution (GE) and neuroevolution of augmenting topologies (NEAT). DE: differential evolution was proposed by Price and Storn [127] in the 1990s. DE has proven to be highly efficient in continuous search spaces and it is often reported to be more robust as well as achieving a faster convergence speed compared to other optimization methods [128]. DE-variants perturb the population members with the scaled differences of randomly selected and distinct population members. GE: grammatical evolution is a grammar-based EA proposed by Ryan *et al.* [135] in the 1990s. A genotype-phenotype mapping process is used to generate (genetic) programs by using a binary string to select production rules in a Backus–Naur form grammar definition. GE can be seen as a special form of GP, where one of the main differences is that unlike GP, GE does not perform the evolutionary process on the programs themselves. NEAT: neuroevolution of augmenting topologies is a form of EA proposed by Stanley and Miikkulainen [149]. NEAT is a technique for evolving neural networks. In the following three elements are crucial for NEAT to work: (1) historical marking that allows solutions to be crossed over; (2) speciation that allows for defining niches; and (3) starting from minimal structure allowing us to incrementally find better solutions.

III. Evolving DNNs Architectures With EAs

A. Motivation

Recently, EAs have started gaining momentum for designing DNNs architectures [39], [42], [84], [102], [112], [130], [131], [149], [167]. The popularity of these algorithms is due to the fact that they are gradient-free, population-based methods offering a parallelized mechanism to simultaneously explore multiple areas of the search space while offering a mechanism to escape from local optima. These algorithms are inherently suited to parallelization meaning more potential solutions can be simultaneously computed within acceptable wall-clock time. Steady increases in computing power, including graphics processing units, are contributing to faster computational calculations in population-based EAs.

B. Critique

Despite the popularity of EAs for designing DNN architectures they have also been criticized for being slow learners as well as being computationally expensive to evaluate [33]. For

example, when using a small population-based EA of 20 individuals (potential solutions) and a training set of 50 000 samples, one generation alone (of hundreds, thousands, or millions of generations) will require one million evaluations through the use of a fitness function.

C. DL Architecture: CNNs

Dufourq and Bassett [32] used a GA to evolve CNN architectures. They used different operations and sizes of filters including one and two-dimension convolution, one and two-dimension max pooling, dropout, among others. The authors reported competitive results compared to state-of-the-art algorithms on the balanced-based and digit-based EMNIST dataset as well as in the fashion dataset. Desell [28] proposed an algorithm based on NEAT [149] to evolve CNN architectures. Desell carried out some modifications to the NEAT algorithm to evolve CNN architectures through selection, crossover, and mutation. Whereas all operators played an important role to produce well-behaved CNNs, the mutation operator, involving seven types of operations, seemed to be crucial to the results reported on the MNIST dataset.

Zoph *et al.* [182] proposed NASNet search space defined by a predetermined outer structure, depicted in Fig. 1, with the ultimate goal of enabling transferability. This structure is inspired by previous non-NAS work (e.g., ResNet [64] and DenseNet [68]). This outer structure is composed of convolutional cells, called normal cells [coloured in pink in Fig. 1(a)] and reduction cells (coloured in grey), repeated many times. The former type of cells returns a feature map of the same dimensions whereas the latter returns a feature map where its height and width is reduced by a factor of two. All cells of the same type are constrained to have the same architecture so that architectures of normal cells were different to the architectures of reduced cells. The goal of their architecture search process was to discover the architectures of these two types of cells. An example of this is shown in Fig. 1(b). Real *et al.* [130] proposed regularized evolution to evolve an image classifier achieving superior accuracy over hand-designed methods for the first time. The authors used a population-based EA with each fixed length member encoding the architecture of CNNs. They used the NASNet search space [182]. The goal was to discover the architectures of the normal cells and the reductions cells as depicted in Fig. 1(a). Real *et al.* [130] used a modified version of tournament selection and two types of mutation to drive evolution. Tournament selection (see Section II) was modified so that the newest genotypes were chosen over older genotypes. The mutation operator involved one of two operations taking place once for each individual: the hidden state mutation and the op mutation. To execute any of these types of mutation, first a random cell is chosen, then a pairwise combination is stochastically selected [see Fig. 1(c)], and finally, one of these two pairs is selected randomly. This hidden state is replaced with another hidden state with the constraint that no loop is formed. The op mutation differs only in modifying the operation used within the selected hidden state. Fig. 1(d) shows how these two mutation operations work. The authors used the CIFAR-10 dataset to test their proposed evolution and compared it against an RL-based method and random search. They achieved better accuracy in results and reducing the computational time required by their algorithm compared to the other two methods. The authors also used the fittest chromosome found by their algorithm and retrained it using the ImageNet dataset.

Xie *et al.* [167] proposed genetic CNN to automatically learn the structures of CNNs with a limited number of layers as well as limited sizes and operations of convolutional filters. Xie *et al.* [167] adopted a GA with binary fixed-length representation to represent parts of the evolved network with each network composed by various stages. Each of these stages is composed of nodes that represent convolutional operations. The binary encoding adopted by Xie *et al.* [167] represents the connection between a number of ordered nodes and this representation allowed the use of crossover, along with roulette selection and mutation. They defined a stage as the minimal unit to apply crossover and mutation. Even with these restrictions, the authors achieved competitive accuracy results using the CIFAR-10 and MNIST datasets while also demonstrating how their approach can be generalized using the learned architecture on the ILSVRC2012 large-scale dataset. This was achieved because their approach was able to produce chain-shaped networks such as AlexNet [90], VGGNet [146], multiple-path networks such as GoogLeNet [159] and highway networks such as deep ResNet [64], which have been reported to be beneficial when applied to computer vision problems.

Real *et al.* [131] used an EA to automatically optimize CNN architectures with individual architectures encoded as graphs with vertices representing rank-3 tensors (two of these represent the spatial coordinates of the image and the third is the number of channels). Activation functions, such as batch-normalization [71], are applied to the vertices. Eleven types of mutations involving inserting layers, removing layers, and modifying layers parameters are used. Real *et al.* [131] indicated that crossover did not improve the results yielded by mutation operators and reported competitive average accuracy results over five independent runs in the CIFAR-10 and CIFAR-100 datasets compared to state-of-the-art algorithms including ResNet [64] and DenseNet [68].

Suganuma *et al.* [153] used Cartesian GP [113] to automatically design CNN architectures. The genotype encodes information on the type and connections of the nodes. Fig. 2(a) depicts this idea. These types include ConvBlock, ResBlock, max pooling, average pooling, summation, and concatenation. ConvBlock consists of standard convolution processing followed by batch normalization and ReLU [118] whereas ResBlock is a ConvBlock followed by tensor summation. The CGP encoding scheme represents the program as a directed acyclic graph in a 2-D grid of N_r rows by N_c columns. Fig. 2(b) provides an example of the phenotype, obtained from Fig. 2(a), in the case of a grid defined by $N_r = 2$ by $N_c = 3$. The corresponding CNN architecture is depicted in Fig. 2(c). The authors used the CIFAR-10 dataset. As evaluating each of the CGP individuals is expensive, they adopted a simple (1+λ) ES (see Section II). The authors' approach achieved competitive results compared with well-known methods including ResNet [64]. The authors

reported encouraging results using CGP to automatically configure CNN architectures regardless of the sample size used in their work. For example, the CNN architecture produced by CGP for the small dataset scenario is wider compared to the architecture yielded by CGP for the standard scenario. Recently, Suganuma *et al.* [151] extended this work by proposing rich initialization and early termination. The former uses the ResNet and the densely connected CNN (DenseNet) to build the CGP individuals. The latter refers to terminating individual evaluation if accuracy is poor when the reference curve built is compared to previous accuracy curves.

Assunção *et al.* [3], [4] proposed DENSER, a hybrid mechanism of GAs and (dynamic structured) GE [135], to evolve DNNs architectures. The outer layer of their proposed approach is in charge of encoding the macro structure of the DNNs evolved by means of GAs. Dynamic structured GE is in charge of the inner layer that encodes the parameters of the DNNs in a backusnaur form. The authors used the typical genetic operators, including selection, two forms of crossovers (one-point and bit-mask) and three types of mutations (add, replicate, and remove unit) in the outer GA-based layer. They used multiple datasets including CIFAR-10, MNIST, and Rectangles. Similarly to other works [112], [153], Assunção *et al.* [3], [4] performed only ten epochs to train the DNNs and reported competitive results compared to state-of-the-art algorithms. Interestingly, they observed that as the fitness increases over time the number of hidden layers decreased suggesting that these two metrics are in conflict when optimising CNNs architectures. Sun *et al.* [157] proposed the use of a population-based GA, of a fixed-length encoding, to evolve, by means of selection, crossover, and mutation, unsupervised DNNs for learning meaningful representations for computer vision tasks. Their approach included the following two main parts (i) finding the optimal architectures in DNNs, the desirable initialization of weights and activation functions, and (ii) fine-tuning all the parameter values in connection weights from the desirable initialization. The first was primarily achieved by using an encoding, which was inspired by the work conducted by Zhao *et al.* [178], who captured all of the elements described in (i). As one gene represents on average 1000 parameters in this encoding, the exploitation achieved by crossover is reduced. To overcome this problem, Sun *et al.* used backpropagation in Part (ii). By hand-crafting the various parts of their approach, the authors demonstrated how the local search adopted in Part (i) was necessary in order to achieve promising results.

Recently, Sun *et al.* [156] proposed a GA, named evolving deep CNNs, to automatically discover CNN architectures. Inspired by Real *et al.* [131], Sun *et al.* [157] proposed a cost-effective method to evaluate the fitness of the individuals in the population for 30 independent runs. They also used selection, mutation, and crossover while crossover was not used in the studies carried out by Real *et al.* [131] (a limitation in Real's work). Sun *et al.* [157] use variable-length encoding for the convolutional, pooling, and full connection layer. Using the standard deviation and the average value of the connection weights they were able to efficiently evaluate each chromosome. Classification error as well the number of connection weights was used. To evaluate the chromosomes along with the normal CNN deep architectures the authors restricted the training to ten epochs. In the last epoch fitness is computed for each of the chromosomes. The authors reported highly encouraging results with many cases achieving better results compared to state-of-the-art algorithms on benchmark datasets.

In [163], van Wyk and Bosman described a neural architecture search (NAS) method to automate the process of finding an optimal CNN architecture for arbitrary image restoration. Their work demonstrates the feasibility for performing NAS under significant memory and computational time constraints. The ImageNet64x64 dataset was chosen for evaluation. The authors found that the human-based configured architecture was heavily overparameterized while this was not the case with the evolved NN, which performed the tasks with a significantly lower number of total parameters. Sun *et al.* [155] proposed an encoding strategy built on the state-of-the-art blocks namely ResNet and DenseNet and used a variable length GA allowing them to automatically evolve CNNs architectures of unrestricted depth. Sun *et al.* [155] used selection, crossover, and mutation to evolve candidate solutions and employed a repair mechanism to produce valid CNNs. The authors used the CIFAR-10 and the CIFAR-100 datasets and compared their results against nine manually designed methods, four semiautomatic methods, and five automatic methods. Interestingly, their results outperformed all hand-crafted methods as well as all semiautomatic methods in terms of the classification error rate.

Evolutionary multiobjective optimization (EMO) [21], [25], explained in Section V-D, has been little used in the automatic configuration of DNNs networks as well as in the optimization of their hyperparameters. Works on the latter include the recent approach proposed by Kim *et al.* [81], where the authors used speed and accuracy as two conflicting objectives to be optimized by means of EMO through the use of the nondominated sorting genetic algorithm II [26]. The authors reported interesting results using three classification tasks, including the use of the MNIST, CIFAR-10, and the drowsy behaviour recognition datasets. Inspired by the Kim *et al.* [81] study, Lu *et al.* [106] used the same EMO with the same conflicting objectives. Lu *et al.* [106] empirically tested multiple computational complexity metrics to measure speed including number of active nodes, number of active connections between nodes and floating-point operations, to mention a few. Lu *et al.* [106] indicated that the latter metric was more accurate and was used as a second conflicting objective for optimization. Moreover, the authors used an ingenious bitstring encoding, which allowed them to use homogeneous crossover and bit-flip mutation (at most one change for each mutation operation) as normally adopted in GAs. The authors used the CIFAR-10 and CIFAR-100 datasets achieving competitive results against RL-based approaches and human expert configurations.

Wang *et al.* [164] explored the ability of differential evolution (DE) to automatically evolve the architecture and hyperparameters of deep CNNs. The method called DECNN uses DE where control of the evolution rate is managed by a differential value. The DECNN evolves variable-length architectures for CNNs.

An IP-based encoding strategy is implemented here to use a single IP address to represent one layer of a DNN. This IP address is then pushed into a sequence of interfaces corresponding to the same order as the layers in DNNs. Six of the MNIST datasets are used for benchmark testing and the DECNN performed very competitively with 12 state-of-the-art competitors over the six benchmarks. Martín *et al.* [110] proposed EvoDeep, which is an EA designed to find the best architecture and optimize the necessary parameters to train a DNN. It uses a finite-state machine model in order to determine the possible transitions between different kinds of layers, allowing EvoDeep to generate valid sequences of layers where the output of one layer fits the input requirements of the next layer. It is tested on MNIST datasets. Elsken *et al.* [38] proposed a multiobjective optimization evolutionary algorithm (MOEA), named LEMONADE employing a Lamarckian-based inheritance mechanism based on approximate network morphism (mutation) operators for speeding up training in DNNs architectures with better results reported for test errors and number of parameters when compared with NASNet search space and mobile-based architectures.

Yang *et al.* [169] propose a continuous evolution strategy utilizing the knowledge learned in the last evolution generation for architecture search. A nondominated sort strategy is adopted to select several excellent architectures. This continuous evolutionary architecture search (CARS) provides a series of architecture models on the Pareto front with high efficiency. Results indicate that their CARS method gives superior results on benchmark datasets against other state-of-the-art models. Shirakawa *et al.* [144] propose a general framework for the dynamic optimization of both the network structure and connection weights. A parametric distribution is used to generate the network structure and then the distribution parameters are understood as the network hyperparameters. This method is shown to be more computationally efficient than static optimization approaches and more flexible than other conventional optimization approaches. The methodology is applied to the selection of layers, selection of the activation functions, adaptation of the stochastic network, and finally the selection of the connections for DenseNets. The authors conclude that their proposed method is capable of learning layer size and an appropriate mix rate for the activation functions within a reasonable computational time.

Optimizing the weights and architectures of an ANN within a single training run by considering all possible architectures as subgraphs of a supergraph is called one-shot architecture search or one-shot NAS. One-shot NAS is used by Akimoto *et al.* [1] to develop a generic optimization framework based on stochastic relaxation for architecture search. This framework can handle practically any type of architecture variation provided it is possible to define a parametric family of probability distributions upon it. Using a step-size adaptation mechanism for the stochastic natural gradient ascent improves the optimization speed and adds robustness against hyperparameter tuning. Experimental analysis indicates that the adaptive stochastic natural gradient method for one-shot NAS achieves significant speedup over evolutionary convolutional autoencoders (CAEs) without compromising performance. Awad *et al.* [5] use DE for NAS as DE has been shown to achieve an excellent performance on a range of NAS benchmarks. They found the best approach for applying DE when the parameters are discrete or categorical is to maintain the population in continuous space and then perform canonical DE while only using discretization of copies of individuals in order to evaluate them. DE is shown to perform very well against four recent NAS methods including one-shot NAS and baseline algorithms such as random search. The authors conclude DE has good ability to handle mixed data types and high-dimensional spaces.

D. DL Architecture: Autoencoders

Suganuma *et al.* [152] used cartesian GP [113] by adopting an ES (1+λ) technique and using selection and mutation operators only to optimize DNS architectures for image restoration. The authors used convolutional autoencoders (CAEs) built upon convolutional layers and skip connections. By optimizing the network in a confined search space of symmetric CAEs the authors achieved competitive results against other methods without the need of using adversarial training and sophisticated loss functions, normally employed for image restoration tasks. Luo *et al.* [107] propose a novel semisupervised AE called a discriminant AE for application in fault diagnosis. The proposed method has a different training process and loss function from traditional AEs. In the case of the discriminant AE, it is capable of extracting better representations from the raw data provided. A completely different loss function is used and the representations extracted by the discriminant AE can generate bigger differences between the sample classes. The discriminant AE makes full use of labels and feature variables to obtain the optimal representations. The centers of the groups of samples should be separated as much as possible. Ashfahani *et al.* [2] propose DEVDAN as a deep evolving denoising AE for application in data stream analytics. DEVDAN demonstrates a proposal of a denoising AE, which is a variant of the traditional AE but focused on retrieving the original input information from a noise perturbation. DEVDAN features an open structure where it is capable of initiating its own structure from the beginning without the presence of a preconfigured network structure. DEVDAN can find competitive network architecture compared with state-of-the-art methods on ten classification datasets.

E. DL Architecture: Deep Belief Networks

DBNs offer a promising solution as they can learn powerful hierarchical feature representations from the data provided. Chen *et al.* [18] use DBNs to automatically extract features from images by proposing the evolutionary function array classifier voter (EFACV), which classifies features from images extracted by a DBN (composed of stacked RBMs). An ES is used to train the EFACV and is mainly used for binary classification problems. For multiclass classification problems, it is necessary to have multiple EFACV. The EFACV shows fast computational speed and a reduction in overall training time. Experiments are performed on the MNIST dataset. Liu *et al.* [104] describe structured learning for DNNs based on multiobjective optimization. They propose a multiobjective optimisation evolutionary algorithm (MOEA). The DBN and its learning procedure use

an RBM to train the DN layer by layer. It is necessary to remove unimportant or unnecessary connections in the DNN and move toward discovering optimal DNN connection structure, which is as sparse as possible without lost of representation. Experiments based on the MNIST and CIFAR-10 datasets with different training samples indicate that the MOEA approach is effective. Zhang *et al.* [176] use DBNs for a prognostic health management system in aircraft and aerospace design by proposing MODBNE (multiobjective DBNs ensemble), which is a powerful multiobjective EA based on decomposition. This is integrated into the training of DBNs to evolve multiple DBNs simultaneously with accuracy and diversity as two conflicting objectives. The DBN is composed of stacked RBMs, which are trained in an unsupervised manner. MODBNE is evaluated and compared against a prominent diagnostics benchmarking problem with the NASA turbofan engine degradation problem. In the proposed approach, the structural parameters of the DBN are strongly dependent on the complexity of the problem and the number of training samples available. The approach worked outstandingly well in comparison to other existing approaches. Zhang *et al.* [177] considered the problem of cost-sensitive learning methods. This idea is to assign misclassification costs for each class appropriately. While the authors report that there are very few studies on cost-sensitive DBNs, these networks have drawn a lot of attention from researchers recently. Imbalances in the classes in input data are a problem. If there is a disproportionate number of class instances this can affect the quality of the applied learning algorithms. Zhang *et al.* [177] argue that DBNs are very well placed to handle these types of imbalanced data problems. The evolutionary cost-sensitive deep belief network (ECS-DBN) is proposed to deal with such problems by assigning differential misclassification costs to the data classes. The ECS-DBN is evaluated on 58 popular knowledge extraction-based on evolutionary learning (KEEL) benchmark datasets.

F. Other Networks: LSTM, RRN, RBM

Shinozaki and Watanabe [143] propose an optimization strategy for DNN structure and parameters using an EA and a GA. The DNN structure is parameterized by a directed acyclic graph. Experiments are carried out on phoneme recognition and spoken digit detection and were conducted upon a massively parallel computing platform using 62 general-purpose computing on graphics processing units. RBMs are used in the training phase. Ororbia *et al.* [121] develop evolutionay exploration of augmenting models (EXAMM), which is designed to devolve RNNs using a selection of memory structures. RNNs are well suited to the task of performing prediction of large-scale real-world time series data. EXAMM was designed to select from a large number of memory cell structures and this allowed the evolutionary approach to yield the best performing RNN architecture. Peng *et al.* [123] use the LSTM NN, which is capable of analyzing time series over long time spans in order to make predictions and tackle the vanishing gradient problem. Their study uses DE to identify the hyperparameters of the LSTM. The authors claim that this is the first time that DE has been used to choose hyperparameters for LSTM for forecasting applications. As forecasting involves complex continuous nonlinear functions, the DE approach is well suited to these types of problems. Gonçalvez *et al.* [56] introduced semantic learning machine (SLM). This shown to outperform other similar methods in a wide range of supervised learning problems. SLM is described as a geometric semantic hill climber approach for NNs following a $1 + \lambda$ strategy. In the search for the best NN architecture configuration this allows the SLM to concentrate on the current best NN without drawing any penalties for this. The crucial aspect of the SLM approach is the geometric semantic mutation. ElSaid *et al.* [36] proposed an approach based on [35], called network-aware adaptive structure transfer learning strategy with the goal of improving training time for deep RNNs. The authors used statistical information about the topology of the "source RNN" topology and the weight distributions. They reported better performing RNNs using half the number of genomes compared to a nontransfer learning-based method.

G. Summary: Evolving DNN Architectures Using EAs

EAs methods with different representations have been used for designing DNNs, ranging from methods including GAs, GP, and ES up to using hybrids combining, for example, the use of GA and GE. Ingenious representations and interesting approaches achieving extraordinary results against human-expert configured networks [130] are commonplace. Approaches in some cases employing hundreds of computers [131] to using a few GPUs [156] are described. Most neuroevolution studies have focused on designing deep CNNs but AE, RBM, RNN, LSTM, and DBM have also been considered despite commending less research attention.

Table I contains extracted information from almost 30 selected papers in neuroevolution. We selected these papers in our own ad-hoc way in order to find a selection of papers, which succinctly demonstrated the use of neuroevolution in DNNs. The table is ordered in alphabetical order of the lead-author surname and summarises: the EA representation used, the representation of individuals, genetic operators used, and the EA parameters. The table outlines the computational resources used in the corresponding study by attempting to outline the number of GPUs used. A calculation of the GPU days per run is approximated as in Sun *et al.* [155]. We indicate benchmark datasets used in the experimental analysis. Finally, the table indicates if the neural network architecture has been evolved automatically or by using a semiautomated approach whilst also indicating the target DNN architecture. Every selected paper does not report the same information. Some papers omit details about computational resources while others omit information about the number of runs. A very interesting output from this table is that there are numerous differences between the approaches used by all of the papers listed. Crossover is omitted from several studies mostly due to encoding adopted by various researchers. Population size and selection strategies for the EAs change between studies. While MNIST and CIFAR are clearly the most popular benchmark datasets we can see many examples of studies using benchmark datasets from specific application domains.

TABLE I
EA REPRESENTATIONS, GENETIC OPERATORS, PARAMETERS, AND VALUES USED IN NEUROEVOLUTION IN DESIGN OF DNNS ARCHITECTURES. INCLUDES DATASETS USED AND CORRESPONDING COMPUTATIONAL EFFORT IN GPU DAYS. AUTOMATIC AND SEMI-AUTO(MATIC) REFER TO WHERE ARCHITECTURE EVOLVED AUTOMATICALLY OR USING SEMI-AUTOMATIC APPROACHES. A DASH (–) INDICATES THE INFORMATION WAS NOT REPORTED.

Study	EA Method	Var./Fixed Length	Genetic Operators			EAs Parameters' values			Computational Resources	Datasets	GPU days per run	Automatic/ Semi-auto	Baselines	DNNs
			Cross	Mut	Selec	Pop	Gens	Runs						
Assunção et al. [3], [4]	GAs, GE	Fixed and variable	✓	✓	✓	100	100	10	Titan X GPUs	CIFAR-10, 3 MNIST variants, Fashion, SVHN, Rectangles, CIFAR-100	–	Automatic	Manual, RL, EAs, Bayesian	CNN
Charte et al. [17]	GAs, ES, DE	Variable	✓	✓	✓	150	20	–	1 NVidia RTX 619 2080 GPU	CIFAR10, Delicious, Fashion, Glass, Ionosphere, MNIST, Semeion, Sonar, Spect	Limited to 24 hours	Automatic	Random Search, Exhaustive Search	AE
Chen et al. [18]	EAs	Fixed	×	✓	✓	1+λ,λ=4	15000	30	–	MNIST	–	Automatic	RBM, RBM (SVM), RBM (ES)	DBN
Desell [28]	NEAT-based	Variable	✓	✓	✓	100		10	4,500 PCs	MNIST	–	Semi-auto	Manual	CNN
ElSaid [35]	EAs	Variable	✓	✓	✓	40 (max.)	–	10	–	National General Aviation Flight Information, Coal-fired Power Plant	–	Automatic	None	RNNs
ElSaid [36]	EAs	Variable	✓	✓	✓	40 (max.)	–	10	–	National General Aviation Flight Information	–	Automatic	None	RNN
Elsken et al. [38]	EAs	–	×	✓	✓	–	100	–	16 Titan X GPUs	CIFAR10, CIFAR100, ImageNet, ImageNet64x64	80	Automatic	NASNet, Mobile-based architecture	CNN
Gonçalvez et al. [56]	GAs	Fixed	×	✓	✓	1+λ, λ=4	gens	runs	No GPUs	4 Binary datasets: Cancer, Diabetes, Sonar, Credit	–	Automatic	Two MLP variants (both SGD)	CNN
Hajewski et al. [60]	EAs	Variable	✓	✓	✓	μ +λ = 10 (λ,μ not specified)	20	20-40	AWS (Nvidia K80 GPU)	STL10	–	Automatic	Random Search, Image Denoising, Manifold Learning	AE
Kim et al. [81]	EAs	–	–	–	–	50, 40, 60	–	–	60 Tesla M40 GPUs	MNIST, CIFAR-10, Drowsiness Recognition	–	Semi-auto	None	CNN
Lander et al. [91]	GAs	Variable	✓	✓	✓	30	50	5	No GPU	MINST	–	Automatic	Manual, Rnd AEs	AE
Liu et al. [104]	EAs	Variable	✓	✓	✓	100	5000	30	–	MINST, CIFAR-10	–	Automatic	–	AE,RBM
Lu et al. [106]	GAs	Fixed	✓	✓	✓	40	30	–	1 NVIDIA 1080Ti	CIFAR-10, CIFAR-100	8 in both	Automatic	Manual, RL, RL + Weight sharing	CNN
Martín et al. [110]	EAs	Fixed	✓	✓	✓	λ+μ=10 (λ,μ=5)	20	–	–	MNIST	–	Automatic	None	CNN
Peng et al. [123]	EAs	Fixed	✓	✓	✓	10	20	–	–	Electricity price	–	Automatic	Various forecasting models incl ANNs, BPNN	LSTM
Real et al. [130]	GAs	Variable	×	✓	✓	100	–	5	450 K40 GPUs	CIFAR-10, ImageNet	3150, 3150	Semi-auto	Rnd Search, RL	CNN
Real et al. [131]	GAs	Variable	×	✓	✓	1000	–	5	250 PCs	CIFAR-10, CIFAR-100	2750, 2750	Automatic	None	CNN
Shinozaki and Watanabe [143]	GAs, ES	Fixed	✓	✓	✓	62	30	–	62 GPUs	AURORA2 Spoken Digits	2.58	Automatic	Manual, GAs	RBM
Suganuma et al. [151]	CGP	Variable	×	✓	✓	1+λ, λ=2	500, 300	10	2 GPUs (GTX 1080 and 1080 Ti)	CIFAR-10 (2 variants), CIFAR-100	See paper	Automatic	Manual, Evolution, RL	CNN
Suganuma et al. [152]	ES	Fixed	×	✓	✓	1+λ, $\lambda = \{1, 2, 4, 8, 16\}$	250	–	4 P100 GPUs	Cars, CelebA, SVNH	12 (inpainting), 16 (denoising)	Automatic	Manual	AE
Suganuma et al. [153]	GP, ES	Variable	×	✓	✓	1+λ,λ = 2	300, 500, 1500	3	Multiple PCs, 2 GPUs: GTX 1080, Titan X	CIFAR-10 (2 variants)	27, 27	Automatic	Manual, RL	CNN
Sun et al. [155]	GAs	Variable	✓	✓	✓	20	20	5	3 GTX 1080 Ti GPUs	CIFAR-10, CIFAR-100	27, 36	Automatic	Manual, EAs, RL	CNN
Sun et al. [156]	GAs	Variable	✓	✓	✓	100	100	30	2 GTX1080 GPUs	Fashion, Rectangle, Convex Set, MNIST	8 (Fashion), 5 (others)	Automatic	Manual, EAs	CNN
Sun et al. [157]	GAs	Fixed	✓	✓	✓	50	–	30	–	Fashion, Rectangle, Convex Set, MNIST , CIFAR-10-BW	–	Automatic	Manual	CNN
van Wyk and Bosman [163]	EAs	Fixed	✓	✓	✓	20	20000	1	1 GPU (GTX 1080)	ImageNet64x64	Halted after 2 hrs	Automatic	Manual CNN	CNN
Wang et al. [164]	EAs	Variable	✓	✓	✓	30	20	30	–	MNIST and Convex Set	–	Automatic	–	CNN
Xie et al. [167]	GAs	Fixed	✓	✓	✓	20	50	–	10 GPUs (type not specified)	CIFAR-10, MNIST, ILSVRC2012, SHVN	17 (CIFAR-10), 2 (MNIST), 20 (ILSVRC2012)	Semi-auto	Manual, Stochastic	CNN
Zhang et al. [176]	EAs	Variable	×	✓	✓	20	500	10	No GPUs	NASA C-MAPSS (Aircraft Engine Simulator Datasets)	–	Automatic	DBN, SVM, Random Forest and others	DBN
Zhang et al. [177]	EAs	Fixed	✓	✓	✓	–	30	10	1 NVIDIA GTX 980 GPU	58 Knowledge Extraction based on Evolutionary Learning (KEEL) datasets	–	Automatic	Resampling Methods DBNs	DBN

IV. TRAINING DNNS THROUGH EAS

A. Motivation

Backpropagation has been one of the most successful and dominant methods used in the training of ANNs over the past number of decades [134]. This simple, effective, and elegant method applies stochastic gradient descent (SGD) to the weights of the ANN where the goal is to to minimize the overall error. However, as remarked by Morse and Stanley [117], the widely held belief, up to around 2006, was that backpropagation would suffer loss of its gradient within DNNs. This turned out to be false and it has subsequently been shown that backpropagation and SGD are effective at optimizing DNNs even when there are millions of connections [69]. Both backpropagation and SGD benefit from the availability of sufficient training data and the availability of computational power. In a problem space with so many dimensions the success of using SGD in DNNs is still surprising. Practically speaking, SGD should be highly susceptible to local optima [117]. Jin *et al.* [73] and Kleinberg and Li [85] argue, independently that the noise helps SGN escape saddle points due to the randomness in the estimator. Choromanska *et al.* [20] and Kawaguchi and Huang [77] hypothesise, independently that the presence of multiple local optima is not a problem as they are very similar to the best solution. EAs perform very well in the presence of saddle points as was discussed in Section I.

B. Critique

As there are no guarantees of convergence, the solutions computed using EAs are usually classified as "near optimal". Population-based EAs are in effect an approximation of the gradient as this is estimated from the individuals in a population and their corresponding objectives. On the other hand, SGD

computes the exact gradient. Subsequently, some researchers consider EAs unsuitable for DL tasks for this reason. However, it has been demonstrated that the exact approximation obtained by SGD is not absolutely critical in the overall success of DNNs using this approach. Lillicrap *et al.* [99] demonstrated that breaking the precision of the gradient calculation has no negative or detrimental effect on learning. Morse and Stanley [117] speculate that the reason for the lack of research focus on using evolutionary computation in DNNs was not entirely related to concerns around the gradient but rather from the belief that new approaches to DNN could actually emerge from outside of SGD.

C. DL Architecture: Convolutional Neural Networks

Such *et al.* [150] proposed a gradient-free method to evolve the weights of convolutional DNNs by using a simple GA, with a population of chromosomes of fixed length. The proposed mechanism successfully evolved networks with over four million free parameters. Some key elements in the study conducted by Such *et al.* [150] to successfully evolve these large neural networks include the following: (1) the use of the selection and mutation genetic operators only; (2) the use of a novel method to store large parameter vectors compactly by representing each of these as an initialization seed plus the list of the random seeds that produces the series of mutations that produced each parameter vector; and (3) the use of a state-of-the-art computational setting, including one modern computer with 4 GPUs and 48 CPU cores as well as 720 CPU cores across dozens of computers.

Instead of using reward-based optimization techniques by means of a fitness function, Such *et al.* [150] used novelty search [97] rewarding new behaviors of individuals. The authors used RL benchmark problems including atari 2600 [10], [114], hard maze [98], and humanoid locomotion [15]. They demonstrated how their proposed approach is competitive with state-of-the-art algorithms in these problems including DQN [114], policy-gradient methods [140], and ES [137]. Pawelczyk *et al.* [122] focused on encoding CNNs with random weights using a GA where the main goal was to let the EA learn the number of gradient learning iterations necessary to achieve a high-accuracy error using the MNIST dataset. Their EA-based approach reported the best results with around 450 gradient learnt iterations compared to 400 constant iterations, which yielded the best overall results.

D. DL Architecture: Autoencoders

David and Greental [24] used a GA of fixed length to evolve the weight values of an AE DNN. Each chromosome was evaluated by using the RMS error for the training samples. The authors used only ten individuals with a 50% elitism-policy. The weights of these individuals were updated using backpropagation and the other half of the population were randomly generated in each generation. They tested their approach with the CIFAR-10 dataset. They compared their approach against the traditional AE using SVM, reporting a better classification error when using their proposed GA-assisted method for the autocoder DNN (1.44% versus 1.85%). The authors indicated the reason why their method produced better results was because gradient descent methods such as backpropagation are highly susceptible to being trapped at local optima and their GA method helped to prevent this. Fernando *et al.* [40] introduced a differentiable version of the compositional pattern producing network called the differentiable pattern producing network (DPPN). The DPPN approach attempts to combine the advantages and results of gradient-based learning in NN with the optimization capabilities of evolutionary approaches. The DPPN has demonstrated superior results for the benchmark dataset MNIST. A generic EA is used in the optimization algorithm of DPPN. The results indicate that the DPPNs and their associated learning algorithms have the ability to dramatically reduce the number of parameters of larger NN. The authors argue that this integration of evolutionary and gradient-based learning allows the optimization to avoid becoming stuck in local optima points or saddle points.

E. Other Relevant Works

Morse and Stanley [117] proposed limited evaluation evolutionary algorithm (LEEA) using a population-based GA of fixed length representation to evolve, by means of crossover and mutation, 1000 weights of a fixed-architecture network. Inspired by SGD, LEEA can compute an error gradient from a single instance of the training set with fitness computed using a small fraction of the training set. As LEAA does not generalize to the whole training sample the authors proposed the following two approaches: (i) using small batches of instances and (ii) using a fitness function that considered both performance on the current minibatch and the performance of individuals' ancestors against their minibatches. In testing on tasks such as function approximation and time series prediction task the authors declared LEEA competitive against the other approaches. Even when the authors do not use DNNs, but a small artificial NN, it is interesting to note how this can be used in a DNN setting. Khadka and Tumer [79] remark that deep RL methods are "notoriously sensitive to the choice of their hyperparamaters and often have brittle convergence properties". These methods are also challenged by long time horizons where there are sparse rewards. EAs can respond very positively to these challenges where the use of fitness metrics allows EAs to tolerate the sparse reward distribution and endure long time horizons. However, EAs can struggle to perform well when optimization of a large number of parameters is required. The authors introduce their evolutionary RL (ERL) algorithm. The EA is used to evolve diverse experiences to train an RL agent. These agents are then subjected to mutation and crossover operators to create the next generation of agents. This ERL can be described as a "population-driven guide" that guides or biases exploration toward states with higher and better long-term returns, promoting diversity of explored policies, and introduces redundancies for stability.

Recurrent neural networks (RNNs) (see Section II-A4) incorporate memory into an NN by storing information from the past within the hidden states network. Kahdka *et al.* [78] introduced a new memory-augmented network architecture called the modular memory unit (MMU). This MMU disconnects the memory

TABLE II
EA REPRESENTATIONS, OPERATORS, AND PARAMETERS VALUES FOR NEUROEVOLUTION IN DNN TRAINING WITH DATASETS USED AND COMPUTATIONAL EFFORT (GPU DAYS). A (-) INDICATES INFORMATION WAS NOT REPORTED.

Study	EA Method	Representation	Genetic Operators			EAs Parameters' values			Computational Resources	Datasets	GPU days per run	DNN
			Cross	Mut	Selec	Pop	Gens	Runs				
Cui et al. [22]	ES	Fixed	✓	✓	✓	–	–	–	GPUs used but not specified	BN50, SWB300, CIFAR-10, PTB	–	–
David and Greental [24]	GAs	Fixed	✓	✓	✓	10		–	–	MNIST	–	AE
Dufourq and Bassett [32]	GAs	Variable	×	✓	✓	100	10	5	1 GTX1070 GPU	CIFAR-10, MNIST, EMNIST (Balanced & Digits), Fashion, IMDB	–	CNN
Fernando et al. [40]	GAs	–	✓	✓	✓	50	–	–	–	MNIST, Omniglot	–	AE
Khadka and Tumer [79]	EAs	Variable	✓	✓	✓	10	∞	5	–	6 Mujoco (continuous control) datasets	–	Read text
Khadka et al. [78]	EAs	Fixed	×	✓	✓	100	1000 10000 15000	–	GPU used but not specified	Sequence Recall, Sequence Classification	–	Read text
Morse and Stanley [117]	GAs	Fixed	✓	✓	✓	1,000	–	10	–	Function Approximation, Time Series, Housing	–	Read text
Pawelczyk et al. [122]	GAs	Fixed	✓	✓	✓	10	–	–	1 GPU (Intel Core i7 7800X, 64GB RAM)	MNIST	–	CNN
Such et al. [150]	GAs	Fixed	×	✓	✓	1,000 (A), 12,500 (H), 20,000 (I)	–	5 (A), 10 (I)	1 PC (4 GPUs, 48 CPUs) and 720 CPUs across dozens of PCs	Atari 2600, Image Hard maze, Humanoid locomotion	0.6 (Atari, 1 PC), 0.16 (Atari, dozens of PCs)	CNN

and central computation operations without requiring costly memory management strategies. Neuroevolutionary methods are used to train the MMU architecture. The performance of the MMU approach with both gradient descent and neuroevolution is examined and the authors find that neuroevolution is more repeatable and generalizable across tasks. The MMU NN is designed to be highly configurable and this characteristic is exploited by the neuroevolutionary algorithm to evolve the network. Population size is set to 100 with a fraction of elites set at 0.1. In the fully differentiable version of the MMU, they find that gradient descent performs better for sequence recall tasks than neuroevolution. However, neuroevolution performs significantly better than gradient descent in sequence classification tasks. Cui *et al.* [22] proposed the use of EAs and SGD to speed up the training of ANNs. Interestingly, the authors use multiple SGD optimizers with certain hyperparameters and learning schedules, and these are used to build the EAs individuals. They showed for different tasks (see Table II) how their proposed approach achieved better results compared to traditional SGD-based methods to speed up the training of neural networks. Meier *et al.* [111] proposed a gradient estimator that considers surrogate gradient directions as well as random search directions. The authors demonstrated that their proposed approach considerably improves the gradient estimation capabilities when employed with ES, particularly, when used the MNIST dataset, and to a lesser degree, in RL tasks. Shahab and Grot [141] proposed the use of EAs along with SGD for training neural models. The main motivation to do so is due to the fact that EAs are inherently parallel and it is suitable for a large-scale distributed training setup compared to data-parallel minibatch SGD. This is important because it has been demonstrated that the latter fails to reduce the number of training iterations beyond a minibatch size [142].

F. Summary: Training DNNs Using EAs

In the early years of neuroevolution, it was thought that evolution-based methods might exceed the capabilities of backpropagation [170]. As ANNs, in general, and as DNNs, in particular, increasingly adopted the use of SGD and backpropagation, the idea of using EAs for training DNNs instead has been almost abandoned by the DNN research community. EAs are a "genuinely different paradigm for specifying a search problem" [117] and provide exciting opportunities for learning in DNNs. When comparing neuroevolutionary approaches to other approaches such as gradient descent, authors such as Khadka *et al.* [78] urge caution. A generation in neuroevolution is not readily comparable to a gradient descent epoch. Despite the fact that it has been argued that EAs can compete with gradient-based search in small problems as well as using NN with a nondifferentiable activation function [109], the encouraging results achieved in the 1990s [54], [116], [126] have inspired some researchers to carry out research in training DNNs. This includes the work conducted by David and Greental [24] and Fernando *et al.* [40] both of which using deep AE and Pawelczyk *et al.* [122] and Such *et al.* [150] who use deep CNNs. Table II is structured in a similar way to Table I. As with Table I, we selected these papers in our own ad-hoc way in order to find a selection of papers, which succinctly demonstrated the use of EAs in the training of DNNs. Again we see mutation and selection used by all of the selected works with crossover omitted in certain situations. We see greater diversity in the types of benchmark datasets used with a greater focus on domain-specific datasets and problems.

V. FUTURE WORK ON NEUROEVOLUTION IN DNNS

A. Surrogate-Assisted EAs in DNNs

EAs have successfully been used in automatically designing artificial DNNs, as described throughout this article, and multiple state-of-the-art algorithms have been proposed in recent years including genetic CNN [167], large-scale evolution [131], evolving deep CNN [156], hierarchical evolution [102]. Despite their success in automatically configuring DNNs architectures, a common limitation in all of these methods is the training time needed. This can range from days to weeks in order to achieve competitive results. Surrogate-assisted, or meta-model based, EC uses efficient models, also known as meta-models or surrogates, for estimating the fitness values in EAs [74]. Hence, a well-posed surrogate-assisted EC considerably speeds

up the evolutionary search by reducing the number of fitness evaluations while at the same time correctly estimating the fitness values of some potential solutions. The adoption of this surrogate-assisted EA is limited in the research discussed in this article and is dealt with in a few limited exceptions. Sun *et al.* [154] demonstrate how meta-models, using ensemble members, can be successfully used to correctly estimate CNN accuracy. This considerably reduced training time, from 33 to 10 GPU days, while still achieving competitive accuracy results compared to the state-of-the-art. A limitation here is the unknown number of training runs necessary to achieve a good prediction performance.

B. Combining SGD and EAs

As discussed in Section IV, backpropagation is one of the most successful methods currently used in the training of ANNs. Backpropagation normally involves the application of SGD to the weights of the ANNs with the goal to minimize the overall error. Before this era of DL began (perhaps we consider the early 1990s as a date in time), the use of EAs to do so was common, but it was abandoned thanks to the impressive results of deep models trained with SGD. As we discussed in Section IV, there has been a small number of works appearing recently, which have considered replacing SGD by EAs. Indeed, we have begun to observe a trend of combining both techniques yielding better results while at the same time speeding up the training process of ANNs [117], [129]. This is another exciting and promising area of research.

C. Mutations and Neutral Theory

We have seen that numerous studies used selection and mutation only to drive evolution in automatically finding a suitable DNN architecture (see Section III) or to train a DNN (see Section IV). Tables I and II present a summary of the genetic operators used by various researchers. Interestingly, many researchers have reported highly encouraging results when using these two genetic operators, including the works conducted by Real *et al.* [130], [131] using GAs and hundreds of GPUs as well as the work carried out by Suganuma *et al.* [153] employing Cartesian GP and using only a few GPUs.

Kimura's neutral theory of molecular evolution [82], [83] states that the majority of evolutionary changes at molecular level are the result of random fixation of selectively neutral mutations. A mutation from one gene to another is neutral if it does not affect the phenotype. Thus, most mutations that take place in natural evolution are neither advantageous nor disadvantageous for the survival of individuals. It is then reasonable to extrapolate that, if this is how evolution has managed to produce the amazing complexity and adaptations seen in nature, then neutrality should aid also EAs. However, whether neutrality helps or hinders the search in EAs is ill-posed and cannot be answered in general. One can only answer this question within the context of a specific class of problems, (neutral) representations, and set of operators [46]–[51], [124], [125].

We are not aware of any works in neuroevolution in DNNs on neutrality. This work [45] discusses some venues of research on neutrality in DNNs. There are some interesting encodings adopted by researchers including Suganuma's work [153] (see Fig. 2) that allow the measurement of the level of neutrality present in evolutionary search and indicates whether its presence is beneficial or not in certain problems and DNNs. If neutrality is beneficial, taking into consideration specific classes of problems, representations, and genetic operators, this can also have an immediately positive impact in the training time needed because the evaluation of potential EA candidate solutions will not be necessary.

D. Multiobjective Optimization

In most reported research results one objective or variable has been used for NN training such as classification error in CNNs. Two or more objectives are rarely considered as the task becomes much more difficult [21], [25] as these objectives may conflict with each other. MO is concerned with the simultaneous optimization of more than one objective function. When such functions are in conflict, a set of tradeoff solutions among the objectives are sought as no single global optimum exists. The optimal tradeoffs are those solutions for which no objective can be further improved without degrading one of the others. This idea is captured in the Pareto dominance relation: a solution x in the search space is said to Pareto-dominate another solution y if x is at least as good as y on all objectives and strictly better on at least one objective. This is an important aspect in EMO [21], [25] because it allows solutions to be ranked according to their performance on all objectives with respect to all solutions in the population. EMO is one of the most active research areas in EAs [38]. Yet it is surprising to see that EMO approaches have been scarcely used for the automatic configuration of artificial DNNs architectures or learning in DNNs. Often, the configuration of these artificial DNNs requires simultaneously satisfying multiple objectives such as reducing the computational calculation of these on the training dataset while attaining high accuracy. EMO offers an elegant and efficient framework to handle these conflicting objectives. We are aware of only a few works in the area, e.g., [38], [81], [104], [106], [176], as summarised in Section III.

E. DNNs Fitness Landscape Analysis and Genetic Operators

All of the works in neuroevolution in DNNs have used core genetic operators including selection and mutation. Crossover has also been used in most of these works. The use of these operators are summarised in Tables I and II. The use of crossover can sometimes be difficult to adopt depending on the encoding used and some variants have been proposed such as in [156]. Other studies have adopted standard crossover operators such as those discussed in [81]. There are, however, no works in the area of neuroevolution in DNNs that have focused their attention in explaining why the adoption of a particular genetic operator is well-suited for that particular problem. The notion of a fitness landscape [165] has been with us for several decades. It is a nonmathematical aid that has proven to be

TABLE III
COMMON DATASETS USED IN NEUROEVOLUTION IN DNNS

Data set	Number of examples Training	Testing	Input Size	RGB, B&W, Grayscale	No. of classes
MNIST [96]	60,000	10,000	28×28	Grayscale	10
MNIST variants [92]	12,000	50,000	28×28	Grayscale	10
CIFAR-10 [89]	50,000	10,000	32×32	RGB	10
CIFAR-100 [89]	50,000	10,000	32×32	RGB	100
Fashion [166]	60,000	10,000	28×28	Grayscale	10
SVHN [120]	73,257	26,032	32×32	RGB	10
Rectangle [92]	1,000	50,000	28 × 28	B&W	2
Rectangle images [92]	10,000	50,000	28 ×28	Grayscale	2
Convex set [92]	6,000	50,000	28 × 28	B&W	2
ILSVRC2012 [27]	1.3M	150,000	224 × 224	RGB	1,000
GERMAN Traffic Sign [147]	50,000	12,500	32 × 32	Grayscale	43

very powerful in understanding evolutionary search. Viewing the search space, defined by the set of all potential solutions, as a landscape, a heuristic algorithm such as an EA, can be thought of as navigating through it to find the best solution (essentially the highest peak in the landscape). The height of a point in this search space, represents in an abstract way, the fitness of the solution associated with that point. The landscape is, therefore, a knowledge interface between the problem and the heuristic-based EA. This can help researchers and practitioners to define well-behaved genetic operators, including mutation and crossover, over the connectivity structure of the landscape.

F. Standardised Neuroevolution Studies in DNNs

In Section II, multiple DNNs architectures have been proposed in the specialized literature. Many of the research works reviewed in this article have compared their results to those yielded by neuroevolution algorithms. However, it is unclear why some techniques are better than others. Is it because of the type of operators used? Is it because of the representation adopted in these studies or is it because of the type of learning employed during training? Due to the lack of standardised studies in neuroevolution on DNNs, it is difficult to draw final conclusions that help us to identify what elements are promising in DNNs. The lack of these standardised studies means we cannot indicate what EAs paradigm with associated genetic operators should be preferred for the automatic configuration of a particular DNN architecture as well as its training.

G. Diversifying the Use of Benchmark Problems and DNNs

New, large, datasets combined with increasingly powerful computational resources have allowed DNNs to solve hard problems in domains such as image classification and speech processing. Image classification is certainly considered as the primary benchmark against which to evaluate DNNs [180]. Benchmark datasets (many included in Table III) are used to compare computational results of experimental setups produced by different research groups. We believe the success of DNNs provides opportunities to expand DNNs to other domains. However, to do so successfully comprehensive domain-specific benchmark datasets will be required. Without such benchmarks, it may be difficult to make convincing arguments to support DNNs for problems beyond traditional domains such as image classification and machine translation. New benchmark problems must allow for the appropriate comparison performance testing of NAS algorithms. Some steps have already been taken. For example, Ying *et al.* [171] propose NAS-Bench-101 that provides a search space on which to test NAS algorithms without incurring much compute cost. NAS-Bench-1-shot-1 was proposed by Zela *et al.* [175], which adapts NAS-Bench-101 so it can be used to search for 1-shot methods, such as DARTS, Proxyless NAS, and so on. Dong and Yang [30] extended NAS-Bench-101 with a different search space and results on multiple datasets.

It is critical that benchmark datasets are available freely and as open-data. Stallkamp *et al.* [147] argue that in a niche area such as traffic sign recognition it can be difficult to compare published work because studies are based on different data or consider classification in different ways. The use of proprietary data in some cases, which is not publicly available, makes comparison of results difficult. Zhang *et al.* [176] access data from a prognostic benchmarking problem related to NASA and aero-propulsion systems. Specific problem domains outside of those of vision, speech recognition, and language also have benchmark datasets available but may be less well-known. Zhang *et al.* [177] use datasets from KEEL but also use a real-world dataset from a manufacturing drilling machine in order to obtain a practical evaluation. Chen and Li [19] argue that as big data continues to play a vital role in areas such as predictive analytics new ways of thinking and novel algorithmic approaches will be needed due to the difficulty of defining big data benchmark datasets.

VI. CONCLUSION

A comprehensive survey of neuroevolution approaches in DNNs and key aspects of EAs in DL has been presented with important issues and challenges discussed in this area. Readers with a background in EAs will find this article useful in determining the state-of-the-art in NAS methods. DL community readers will be encouraged to use EAs approaches in their DNN work. Configuration of DNNs is not a trivial problem often becoming a tedious and error-prone process. EAs are competitive in automatic creation and configuration of such networks in situations where poorly or incorrectly configured networks can lead to the failure or underutilization of DNNs.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for providing very insightful comments for this article. Additional valuable comments were provided by many public mailing lists including genetic_programming@yahoogroups.com, UAI, and Connectionists. Numerous NN / DL / Neuroevolution experts provided additional valuable comments.

REFERENCES

[1] Y. Akimoto, S. Shirakawa, N. Yoshinari, K. Uchida, S. Saito, and K. Nishida, "Adaptive stochastic natural gradient method for one-shot neural architecture search," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 171–180.

[2] A. Ashfahani, M. Pratama, E. Lughofer, and Y. Ong, "Devdan: Deep evolving denoising autoencoder," *Neurocomputing*, vol. 390, pp. 297–314, 2020.

[3] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, in *Evolving the Topology of Large Scale Deep Neural Networks*, M. Castelli, L. Sekanina, M. Zhang, S. Cagnoni, and P. García-Sánchez, Eds. Berlin, Germany: Springer, 2018, pp. 19–34.

[4] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "DENSER: Deep evolutionary network structured representation," *Genet. Program. Evolvable Mach.*, vol. 20, no. 1, pp. 5–35, 2019.

[5] N. Awad, N. Mallik, and F. Hutter, "Differential evolution for neural architecture search," in *Proc. 1st Workshop Neural Architecture Search Int. Conf. Learn. Representations*, 2020.

[6] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford, U.K.: Oxford Univ. Press, 1996.

[7] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. 5th Int. Conf. Learn. Representations*, Conference Track Proceedings, Toulon, France, Apr. 24-26, 2017.

[8] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proc. Workshop Unsupervised Transfer Learning*, 2012, pp. 37–49.

[9] A. Baldominos, Y. Saez, and P. Isasi, "On the automated, evolutionary design of neural networks: Past, present, and future," *Neural Comput. Appl.*, vol. 32, no. 2, pp. 519–545, 2020.

[10] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Proc. 24th Int. Joint Conf. Artif. Intell.*, Buenos Aires, Argentina, Jul. 25-31, 2015, pp. 4148–4152.

[11] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[12] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. 30th Int. Conf. Int. Conf. Mach. Learn.*, 2013, pp. I-115–I-123.

[13] H. Beyer and H. Schwefel, "Evolution strategies -a comprehensive introduction," *Nat. Comput.: Int. J.*, vol. 1, no. 1, pp. 3–52, 2002.

[14] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through hypernetworks," in *Proc. 6th Int. Conf. Learn. Representations*, Conf. Track Proc., Vancouver, BC, Canada, Apr. 30 - May 3, 2018.

[15] G. Brockman *et al.*, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[16] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. 32nd AAAI Conf. Artif. Intell., 30th Innovative Appl. Artificial Intell., 8th AAAI Symp. Educational Adv. Artificial Intell.*, 2018, pp. 2787–2794.

[17] F. Charte, A. J. Rivera, F. Martnez, and M. J. delJesus, "EvoAAA: An evolutionary methodology for automated neural autoencoder architecture search," *Integr. Comput.-Aided Eng.*, vol. 27, pp. 211–231, 2020.

[18] S. Chen, G. Liu, C. Wu, Z. Jiang, and J. Chen, "Image classification with stacked restricted Boltzmann machines and evolutionary function array classification voter," in *Proc. IEEE Congr. Evol. Comput.*, 2016, pp. 4599–4606.

[19] X. Chen and X. Lin, "Big data deep learning: Challenges and perspectives," *IEEE Access*, vol. 2, pp. 1 514–525, 2014.

[20] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2015, pp. 192–204.

[21] C. A. C. Coello, "Evolutionary multi-objective optimization: A historical view of the field," *IEEE Comput. Intell. Mag.*, vol. 1, no. 1, pp. 28–36, Feb. 2006.

[22] X. Cui, W. Zhang, Z. Tüske, and M. Picheny, "Evolutionary stochastic gradient descent for optimization of deep neural networks," in *Adv. Neural Inf. Process. Syst.*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, Inc., 2018, pp. 6048–6058.

[23] A. Darwish, A. E. Hassanien, and S. Das, "A survey of swarm and evolutionary computing approaches for deep learning," *Artif. Intell. Rev.*, vol. 53, no. 3, pp. 1767–1812, 2019.

[24] O. E. David and I. Greental, "Genetic algorithms for evolving deep neural networks," in *Proc. Companion Publication Annu. Conf. Genet. Evol. Comput.*, 2014, pp. 1451–1452.

[25] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: Wiley, 2001.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[27] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[28] T. Desell, "Large scale evolution of convolutional neural networks using volunteer computing," in *Proc. Genet. Evol. Comput. Conf.*, 2017, pp. 127–128

[29] G. Desjardins and Y. Bengio, "Empirical evaluation of convolutional RBMs for vision," Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montreal, QC, Canada, Tech. Rep. 1327, 2008.

[30] X. Dong and Y. Yang, "Nas-Bench-201: Extending the scope of reproducible neural architecture search," in *Proc. 8th Int. Conf. Learn. Representations* Addis Ababa, Ethiopia, Apr. 26-30, 2020.

[31] C. dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *Proc. 25th Int. Conf. Comput. Linguistics: Tech. Papers*, Dublin, Ireland, 2014, pp. 69–78.

[32] E. Dufourq and B. A. Bassett, "Eden: Evolutionary deep networks for efficient machine learning," in *Proc. Pattern Recognit. Assoc. South Afr. Robot. Mechatronics*, 2017, pp. 110–115.

[33] A. E. Eiben and J. Smith, "From evolutionary computation to the evolution of things," *Nature*, vol. 521, pp. 476–482, May 28, 2015.

[34] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer-Verlag, 2003.

[35] A. ElSaid, J. Karnas, Z. Lyu, D. Krutz, A. Ororbia, and T. Desell, "Neuro-evolutionary transfer learning through structural adaptation," in *Proc. Int. Conf. Appl. Evolutionary Comput.*, 2020, pp. 610–625.

[36] A. ElSaid, J. Karns, Z. Lyu, D. Krutz, A. Ororbia, and T. Desell, "Improving neuroevolutionary transfer learning of deep recurrent neural networks through network-aware adaptation," in *Proc. Genet. Evol. Comput. Conf.*, 2020, pp. 315–323.

[37] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," in *Proc. 6th Int. Conf. Learn. Representations*, Vancouver, BC, Canada, Apr. 30 -May 3, 2018, 2017.

[38] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *Proc. 7th Int. Conf. Learn. Representations*, New Orleans, LA, USA, May 6-9, 2019.

[39] T. Elsken, J. H. Metzen, and F. Hutter, *Neural Architecture Search*. Cham, Switzerland: Springer, 2019, pp. 63–77.

[40] C. Fernando *et al.*, "Convolution by evolution: Differentiable pattern producing networks," in *Proc. Genet. Evol. Comput. Conf.*, 2016, pp. 109–116.

[41] A. Fischer and C. Igel, "An introduction to restricted Boltzmann machines," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, L. Alvarez *et al.*, Eds. Berlin, Germany: Springer, 2012, pp. 14–36.

[42] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, 2008.

[43] L. Fogel, A. Owens, and M. Walsh, *Artificial Intelligence Through Simulated Evolution*. Chichester, U.K.: Wiley, 1966.

[44] J. A. Foster, "Evolutionary computation," *Nat. Rev. Genet.*, vol. 2, pp. 428–436, 2001.

[45] E. Galván, "Neuroevolution in deep learning: The role of neutrality," *CoRR*, vol. abs/2102.08475, 2021. [Online]. Available: https://arxiv.org/abs/2102.08475

[46] E. Galván-López, "An analysis of the effects of neutrality on problem hardness for evolutionary algorithms," Ph.D. thesis, Dept. Sch. Comput. Sci. Elect. Eng., Univ. Essex, United Kingdo, Colchester, U.K., 2009.

[47] E. Galván-López, S. Dignum, and R. Poli, "The effects of constant neutrality on performance and problem hardness in GP," in *Proc. 11th Eur. Conf. Genet. Program.*, 2008, pp. 312–324

[48] E. Galván-López and R. Poli, "An empirical investigation of how and why neutrality affects evolutionary search," in *Proc. 8th Annu. Conf. Genet. Evol. Comput.*, 2006, pp. 1149–1156.

[49] E. Galván-López and R. Poli, "Some steps towards understanding how neutrality affects evolutionary search," in *Proc. 9th Int. Conf. Parallel Problem Solving Nat.*, Springer, 2006, pp. 778–787.

[50] E. Galván-López and R. Poli, "An empirical investigation of how degree neutrality affects GP search," in *Proc. 8th Mexican Int. Conf. Artif. Intell. Adv. Artif. Intell.*, 2009, pp. 728–739.

[51] E. Galván-López, R. Poli, A. Kattan, M. O'Neill, and A. Brabazon, "Neutrality in evolutionary algorithms... what do we know?," *Evolving Syst.*, vol. 2, no. 3, pp. 145–163, 2011.

[52] F. Gers, "Learning to forget: Continual prediction with LSTM," in *Proc. IET Conf.*, 1999, pp. 850–855.

[53] F. A. Gers and E. Schmidhuber, "LSTM recurrent networks learn simple context-free and context-sensitive languages," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1333–1340, Nov. 2001.

[54] C. Goerick and T. Rodemann, "Evolution strategies: An alternative to gradient based learning," in *Proc. Int. Conf. Eng. Appl. Neural Netw.*, 1996, pp. 479–482.

[55] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989.

[56] I. Gonçalves, M. Seca, and M. Castelli, "Explorations of the semantic learning machine neuroevolution algorithm: Dynamic training data use," in *Ensemble Construction Methods, and Deep Learning Perspectives*. Cham, Switzerland: Springer International Publishing, 2020, pp. 39–62.

[57] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[58] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2013, pp. 6645–6649.

[59] K. Greff, R.K. Srivastava, J. Koutnk, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[60] J. Hajewski, S. Oliveira, and X. Xing, "Distributed evolution of deep autoencoders," *CoRR*, vol. abs/2004.07607, 2020. [Online]. Available: https://arxiv.org/abs/2004.07607

[61] N. Hansen, S. D. Mller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol. Comput.*, vol. 11, no. 1, pp. 1–18, 2003.

[62] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1996, pp. 312–317.

[63] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput*., vol. 9, no. 2, pp. 159–195, Jun. 2001.

[64] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[65] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput*., vol. 18, no. 7, pp. 1527–1554, Jul. 2006.

[66] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[67] C. Hong, J. Yu, J. Wan, D. Tao, and M. Wang, "Multimodal deep autoencoder for human pose recovery," *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5659–5670, Dec. 2015.

[68] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.

[69] Y. Huang *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, pp. 103–112, 2019.

[70] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 106–154, 1962.

[71] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 448–456

[72] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. IEEE 12th Int. Conf. Comput. Vis.*, 2009, pp. 2146–2153.

[73] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, "How to escape saddle points efficiently," in *Proc. 34th Int. Conf. Mach. Learn.*, D. Precup and Y. W. Teh, Eds., Proc. Mach. Learn. Res., vol. 70, Aug. 6–11, 2017, pp. 1724–1732.

[74] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm Evol. Comput.*, vol. 1, no. 2, pp. 61–70, 2011.

[75] R. Józefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 2342–2350.

[76] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing, "Neural architecture search with Bayesian optimisation and optimal transport," in *Proc. Adv. Neural Inf. Process. Syst. 31: Annu. Conf. Neural Inf. Process. Syst.*, 2018, pp. 2020–2029.

[77] K. Kawaguchi and J. Huang, "Gradient descent finds global minima for generalizable deep neural networks of practical sizes," in *Proc. 57th Annu. Allerton Conf. Commun., Control, Comput.*, 2019, pp. 92–99.

[78] S. Khadka, J. J. Chung, and K. Tumer, "Neuroevolution of a modular memory-augmented neural network for deep memory problems," *Evol. Comput*., vol. 27, no. 4, pp. 639–664, 2019.

[79] S. Khadka and K. Tumer, "Evolution-guided policy gradient in reinforcement learning," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 1196–1208.

[80] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev*., vol. 53, no. 8, pp. 5455–5516, 2020.

[81] Y.-H. Kim, B. Reddy, S. Yun, and C. Seo, "Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy," *ICML 2017 AutoML Workshop*, 2017, pp. 1–8.

[82] M. Kimura, "Evolutionary rate at the molecular level," *Nature*, vol. 217, pp. 624–626, 1968.

[83] M. Kimura, *The Neutral Theory of Molecular Evolution*. Cambridge, U.K.: Cambridge Univ. Press, 1983.

[84] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Syst.*, vol. 4, pp. 461–476, 1990.

[85] R. Kleinberg, Y. Li, and Y. Yuan, "An alternative view: When does SGD escape local minima?," *Int. Conf. Mach. Learn.*, 2018, pp. 2698–2707.

[86] J. Koza, M. Keane, M. Streeter, W. Mydlowec, J. Yu, and G. Lanza, *Genetic programming IV: Routine Human-Competitive Machine Intelligence*, vol. 5. Berlin, Germany: Springer, 2006.

[87] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[88] J. R. Koza, "Human-competitive results produced by genetic programming," *Genet. Program. Evolvable Mach.*, vol. 11, no. 3/4, pp. 251–284, Sep. 2010

[89] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (Canadian Institute for Advanced Research)," 2021, Accessed: 2021-06-10. [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html

[90] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[91] S. Lander and Y. Shang, "Evoae—A new evolutionary method for training autoencoders for deep learning networks," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, 2015, pp. 790–795.

[92] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 473–480.

[93] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio, "Learning algorithms for the classification restricted Boltzmann machine," *J. Mach. Learn. Res.*, vol. 13, no. 22, pp. 643–669, 2012.

[94] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[95] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proc. IEEE*, 1998, pp. 2278–2324.

[96] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[97] J. Lehman and K. Stanley, "Novelty search and the problem with objectives," *Genet. Programm. Theory Pract.*, vol. 11, pp. 37–56, 2011.

[98] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evol. Comput*., vol. 19, no. 2, pp. 189–223, Jun. 2011.

[99] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random feedback weights support learning in deep neural networks," *Nature Commun.*, vol. 7, no. 13276, 2016.

[100] M. Lindauer and F. Hutter, "Best practices for scientific research on neural architecture search," *J. Mach. Learn. Res.*, vol. 21, no. 243, pp. 1–8, 2020.

[101] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vision*, Sep. 2018.

[102] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. 6th Int. Conf. Learn. Representations*, 2018, pp. 461–476.

[103] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2018.

[104] J. Liu, M. Gong, Q. Miao, X. Wang, and H. Li, "Structure learning for deep neural networks based on multiobjective optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2450–2463, Jun. 2018.

[105] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[106] Z. Lu *et al.*, "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 419–427.

[107] X. Luo, X. Li, Z. Wang, and J. Liang, "Discriminant autoencoder for feature extraction in fault diagnosis," *Chemometrics Intell. Lab. Syst.*, vol. 192, 2019, Art. no. 103814.

[108] Y. Ma and J. C. Principe, "A taxonomy for neural memory networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 6, pp. 1780–1793, Jun. 2020.

[109] M. Mandischer, "A comparison of evolution strategies and backpropagation for neural network training," *Neurocomputing*, vol. 42, no. 1, pp. 87–117, 2002.

[110] A. Martin, R. Lara-Cabrera, F. Fuentes-Hurtado, V. Naranjo, and D. Camacho, "Evodeep: A new evolutionary approach for automatic deep neural networks parametrisation," *J. Parallel Distrib. Comput.*, vol. 117, pp. 180–191, 2018.

[111] F. Meier, A. Mujika, M. M. Gauy, and A. Steger, "Improving gradient estimation in evolutionary strategies with past descent directions," *CoRR*, vol. abs/1910.05268, 2019. [Online]. Available: http://arxiv.org/abs/1910.05268

[112] R. Miikkulainen *et al.*, "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, 2017. [Online]. Available: http://arxiv.org/abs/1703.00548

[113] J. F. Miller, *Cartesian Genetic Programming*. Berlin, Germany: Springer, 2011, pp. 17–34.

[114] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[115] A. Mohamed, T. N. Sainath, G. Dahl, B. Ramabhadran, G. E. Hinton, and M. A. Picheny, "Deep belief networks using discriminative features for phone recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2011, pp. 5060–5063.

[116] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. 11th Int. Joint Conf. Artif. Intell.*, 1989, pp. 762–767.

[117] G. Morse and K. O. Stanley, "Simple evolutionary optimization can rival stochastic gradient descent in neural networks," in *Proc. Genet. Evol. Comput. Conf.*, 2016, pp. 477–484.

[118] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Int. Conf. Mach. Learn.*, 2010, pp. 807–814.

[119] R. Negrinho and G. Gordon, "Deeparchitect: Automatically designing and training deep architectures," *CoRR*, vol. abs/1704.08792, 2017. [Online]. Available: http://arxiv.org/abs/1704.08792

[120] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. Conf. Neural Inf. Process. Syst.*, Jan. 2011. [Online]. Available: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/37648.pdf

[121] A. Ororbia, A. ElSaid, and T. Desell, "Investigating recurrent neural network memory structures using neuro-evolution," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 446–455.

[122] K. Pawełczyk, M. Kawulok, and J. Nalepa, "Genetically-trained deep neural networks," in *Proc. Genet. Evol. Comput. Conf. Companion*, 2018, pp. 63–64.

[123] L. Peng, S. Liu, R. Liu, and L. Wang, "Effective long short-term memory with differential evolution algorithm for electricity price prediction," *Energy*, vol. 162, pp. 1301–1314, 2018.

[124] R. Poli and E. Galván-López, "On the effects of bit-wise neutrality on fitness distance correlation, phenotypic mutation rates and problem hardness," in *Foundations of Genetic Algorithms*, C. R. Stephens *et al.*, Eds. Berlin, Germany: Springer, 2007, pp. 138–164.

[125] R. Poli and E. Galván-López, "The effects of constant and bit-wise neutrality on problem hardness, fitness distance correlation and phenotypic mutation rates," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 279–300, Apr. 2012.

[126] V. W. Porto, D. B. Fogel, and L. J. Fogel, "Alternative neural network training methods," *IEEE Expert: Intell. Syst. Appl.*, vol. 10, no. 3, pp. 16–22, Jun. 1995.

[127] K. Price, R. Storn, and J. Lampinen, *Differential Evolution—A Practical Approach to Global Optimization*. Berlin, Germany: Springer, Jan. 2005.

[128] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition-based differential evolution," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 64–79, Feb. 2008.

[129] S. Rajbhandari, Y. He, O. Ruwase, M. Carbin, and T. Chilimbi, "Optimizing CNNs on multicores for scalability, performance and goodput," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2017, pp. 267–280.

[130] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. 33rd AAAI Conf. Artif. Intell., AAAI, 31st Innovative Appl. Artif. Intell. Conf., IAAI, 9th AAAI Symp. Edu. Adv. Artif. Intell.*, 2019, pp. 4780–4789.

[131] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[132] I. Rechenberg, "Evolutionsstrategien," in *Simulationsmethoden in Der Medizin Und Biologie*, B. Schneider and U. Ranft, Eds. Berlin, Germany: Springer, 1978, pp. 83–114.

[133] I. Rechenberg, "Evolution strategy: Nature's way of optimization," in *Optimization: Methods and Applications, Possibilities and Limitations*, H. W. Bergmann, Ed. Berlin, Germany: Springer, 1989, pp. 106–126

[134] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[135] J. Ryan Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Genetic Programming,* W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, Eds., Berlin, Germany: Springer, 1998, pp. 83–96

[136] R. Salakhutdinov and G. Hinton, "Deep Boltzmann machines," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, Proc. Mach. Learn. Res., Apr. 16–18, 2009, pp. 448–455.

[137] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017, *arXiv:1703.03864*.

[138] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.

[139] H.-P. Schwefel, *Numerical Optimization of Computer Models*. Hoboken, NJ, USA: Wiley, 1981.

[140] F. Sehnke, C. Osendorfer, T. Rckstie, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-exploring policy gradients," *Neural Netw.*, vol. 23, no. 4, pp. 551–559, 2010.

[141] A. Shahab and B. Grot, "Population-based evolutionary distributed SGD," in *Proc. Genetics Evol. Comput. Conf. Companion*, 2020, pp. 153–154.

[142] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *J. Mach. Learn. Res.*, vol. 20, pp. 112:1–112:49, 2019.

[143] T. Shinozaki and S. Watanabe, "Structure discovery of deep neural network based on evolutionary algorithms," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2015, pp. 4979–4983.

[144] S. Shirakawa, Y. Iwata, and Y. Akimoto, "Dynamic optimization of neural network structures using probabilistic modeling," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no, 1, 2018.

[145] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[146] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Representations*, San Diego, CA, USA, May 7-9, 2015.

[147] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Netw.*, vol. 32, pp. 323–332, 2012, 2011.

[148] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nat. Mach. Intell.*, vol. 1, pp. 24–35, 2019.

[149] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput*., vol. 10, no. 2, pp. 99–127, Jun. 2002.

[150] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *CoRR*, vol. abs/1712.06567, 2017. [Online]. Available: http://arxiv.org/abs/1712.06567

[151] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," *Evol. Comput.*, vol. 28, no. 1, pp. 141–163, 2020.

[152] M. Suganuma, M. Ozay, and T. Okatani, "Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search," in *Proc. 35th Int. Conf. Mach. Learn.*, Stockholmsmässan, Stockholm, Sweden, vol. 80, Jul. 10-15, 2018 pp. 4778–4787
[153] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genet. Evol. Comput. Conf.*, 2017, pp. 497–504.
[154] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, Apr. 2020.
[155] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.
[156] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Apr. 2020.
[157] Y. Sun, G. G. Yen, and Z. Yi, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Trans. Evol. Computation*, vol. 23, no. 1, pp. 89–103, Feb. 2019.
[158] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (A Bradford Book). Cambridge, MA, USA: MIT Press, 2018.
[159] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[160] E. Talbi, "A taxonomy of hybrid metaheuristics," *J. Heuristics*, vol. 8, no. 5, pp. 541–564, 2002.
[161] N. I. Tapia and P. A. Estevez, "On the information plane of autoencoders," in *Proc. Int. Joint Conf. Neural Netw.*, Jul 2020.
[162] N. Tishby and N. Zaslavsky, "Deep Learning and the Information Bottleneck Principle," *IEEE Inf. Theory Workshop*, Jerusalem, Israel, Apr. 26 - May 1, 2015, pp. 1–5, doi: 10.1109/ITW.2015.7133169.
[163] G. J. Van Wyk and A. S. Bosman, "Evolutionary neural architecture search for image restoration," in *Proc. Int. Joint Conf. Neural Netw.*, 2019, pp. 1–8.
[164] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," In *Proc. Adv. Artif. Intell. 31st Australasian Joint Conf.*, Lecture Notes Comput. Sci., Springer, vol. 11320, 2018, pp. 237–250.
[165] S. Wright, "The role of mutation, inbreeding, crossbreeding and selection in evolution," in *Proc. 6th Int. Congr. Genet.*, 1932, pp. 356–366.
[166] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-Mnist: A novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017. [Online]. Available: http://arxiv.org/abs/1708.07747
[167] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1388–1397.
[168] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," *Proc. 7th Int. Conf. Learn. Representations*, New Orleans, LA, USA, May 6-9, 2019.
[169] Z. Yang *et al.*, "Cars: Continuous evolution for efficient neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognition*, 2020, pp. 1829–1838.
[170] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[171] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, "Nas-Bench-101: Towards reproducible neural architecture search," in *Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.
[172] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 974–983.
[173] S. Yu and J. C. Principe, "Understanding autoencoders with information theoretic concepts," *Neural Networks*, Elsevier, vol. 117, pp. 104–123, 2019.
[174] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Brit. Mach. Vis. Conf.*, R. C. Wilson, E. R. Hancock, and W. A. P. Smith, New York, UK: BMVA Press, Sep. 19-22, 2016.
[175] A. Zela, J. Siems, and F. Hutter, "Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search," in *Proc. Int. Conf. Learn. Representations*, Addis Ababa, Ethiopia: OpenReview.net, Apr. 26-30, 2020.
[176] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2306–2318, Oct. 2017.
[177] C. Zhang, K. C. Tan, H. Li, and G. S. Hong, "A cost-sensitive deep belief network for imbalanced classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 1, pp. 109–122, Jan. 2019.
[178] Q. Zhao, D. Zhang, and H. Lu, "A direct evolutionary feature extraction algorithm for classifying high dimensional data," in *Proc. 21st Nat. Conf. Artif. Intell.*, 2006, pp. 561–566.
[179] Z. Zhong, J. Yan, and C. Liu, "Practical network blocks design with Q-learning," *CoRR*, vol. abs/1708.05552, 2017. [Online]. Available: http://arxiv.org/abs/1708.05552
[180] H. Zhu *et al.*, "Benchmarking and analyzing deep neural network training," in *Proc. IEEE Int. Symp. Workload Characterization*, 2018, pp. 88–100.
[181] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning." in *Proc. 5th Int. Conf. Learn. Representations*, Conf. Track Proc., Toulon, France, Apr. 24-26, 2017. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg
[182] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *IEEE Conf. Comput. Vision Pattern Recognition*, Salt Lake City, UT, USA: IEEE Comput. Soci., Jun. 18-22, 2018, pp. 8697–8710.