Hindawi
Journal of Electrical and Computer Engineering
Volume 2019, Article ID 3217542, 12 pages
https://doi.org/10.1155/2019/3217542

Research Article

An Experiment on the Use of Genetic Algorithms for Topology Selection in Deep Learning

Fernando Mattioli, Daniel Caetano, Alexandre Cardoso, Eduardo Naves, and Edgard Lamounier

Faculty of Electrical Engineering, Federal University of Uberlândia, Uberlândia, MG, Brazil

Correspondence should be addressed to Fernando Mattioli; mattioli.fernando@gmail.com

Received 1 June 2018; Revised 26 October 2018; Accepted 3 December 2018; Published 15 January 2019

Guest Editor: Francesco Camastra

The choice of a good topology for a deep neural network is a complex task, essential for any deep learning project. This task normally demands knowledge from previous experience, as the higher amount of required computational resources makes trial and error approaches prohibitive. Evolutionary computation algorithms have shown success in many domains, by guiding the exploration of complex solution spaces in the direction of the best solutions, with minimal human intervention. In this sense, this work presents the use of genetic algorithms in deep neural networks topology selection. The evaluated algorithms were able to find competitive topologies while spending less computational resources when compared to state-of-the-art methods.

1. Introduction

Many digital signals, including images, present an hierarchical nature, in which higher level features are obtained through composition of lower level ones [1]. Deep neural networks (DNN) are a particular class of artificial neural networks (ANN), composed by stacked layers which explore this hierarchical behavior through automatic feature extraction [2]. With the addition of more layers and more processing units in each layer, DNN are able to achieve significant performance in problems of increasing complexity [3]. DNN have presented impressive results, specially in classification and regression applications, including image recognition and computer vision [4].

The design of neural network topologies depends on previous domain knowledge and expertise. Recently, the development of methods which minimize human interference has been discussed by researchers and practitioners [5]. Currently used approaches include random search, grid search, and transfer learning. In random search, the solution space is randomly sampled, whereas in grid search, all possible combinations of a reduced solution subspace are evaluated [6]. In transfer learning, networks exhaustively trained with standard datasets are adapted to another application, by fine-tuning its weights with the target dataset [7].

These approaches present some limitations which narrow down their application. In pure random search, the effort of finding a good solution is not rewarded since other solutions will be equally sampled from the search space. Grid search can be impractical if the search space is too big and also subjected to local optimum convergence if a non-representative subspace is selected. In transfer learning, the complexity of existing and available topologies can make them unsuitable for the required application, for example, if severe real-time constraints are to be satisfied. In this sense, the development of methods to assist the design of new ANN topologies, with minimal intervention and limited computational resources, remains an important research topic.

The definition of a DNN topology for a given problem can be considered an optimization problem, given the high number of parameters to be chosen [8]. Some of these parameters, including number of layers, neurons per layer, activation functions, and learning algorithms, form a vast search space making exhaustive search practically impossible [9]. Therefore, the need for robust optimization algorithms arises.

Genetic algorithms (GA) provide direct search, inspired by the theory of evolution, being widely applied in complex optimization problems with lots of parameters [10, 11]. GA are a subclass of evolutionary computation, which has been successfully used to assist ANN design [12, 13]. Through genetic operators, GA are capable of exploring promising search subspaces (by combining good solutions) and also escape local optima, reducing the amount of computational resources needed when compared to exhaustive search [9, 13].

The context presented in the above paragraphs motivated the investigation presented in this work. Two main objectives were defined: (i) To assess the efficiency of a search method using GA, with minimal human intervention, in DNN topology selection. (ii) To evaluate the contribution of a fitness prediction method in the algorithm performance. We believe that the investigation of these research objectives should help filling two gaps, namely, the appreciation of good solutions (found in the search space) and the economy of computing resources by discarding (i.e., not evaluating) solutions predicted to be potentially bad.

2. Background and Related Work

Deep learning allows computational models with multiple computing layers to process data in different levels of abstraction [1]. In addition, deep learning models provide automatic feature extraction capabilities, by learning the representation (features) together with the input/output mapping [2, 3].

Training a deep learning model from scratch requires two main steps: selection of the model topology and presentation of input/output data. The fundamental building blocks of a network topology are the processing layers, composed by neurons, convolutional filters, activation functions, dropout rate, and spatial reductions such as pooling and stride [14]. These building blocks form the model hyperparameters, and their choice is known to directly affect the network learning speed and performance [15].

Genetic algorithms are a set of evolutionary computing algorithms in which the solution space is explored through recombination of previous explored solutions [16]. In an analogy to natural selection, better solutions have more probability to be chosen for recombination, while bad solutions tend to be discarded [10]. The result of this recombination process is an offspring of new solutions, in which individuals inherit different characteristics from their parents [9].

In standard GA, at first, a population of candidate solutions is randomly generated. Each solution is represented by a data structure (chromosome), containing the parameters to be tuned. These solutions are evaluated by a fitness function, representing the performance of the solution on the target problem with higher values being returned by the best solutions. In the next step, a selection algorithm is used to choose existing solutions for recombination. The chosen individuals are then subjected to the genetic operators of crossover and mutation to create an offspring of new solutions (generation). In crossover, elements of selected chromosomes are exchanged, creating new solutions inheriting characteristics of their parents. In mutation, an element of the chromosome is randomly changed, producing a new solution. The solutions in the new generation will be evaluated and will replace the previous population, optionally keeping some of its best individuals through elitism [11, 17].

Recently, the use of GA in deep learning hyperparameter optimization has been addressed by many researchers. In [4], a GA is used to evolve topologies of large-scale DNN. An encoding schema, based on the configuration of each DNN layer, is proposed. The adaptation of the genetic operators of crossover and mutation for the topology selection problem is also presented. Reference [9] presents the use of evolutionary optimization and convolutional neural networks in a cancer diagnosis application. A fixed topology of 3 layers is used, and the hyperparameters of these 3 layers (filters, kernel size, and pooling window) are encoded in the chromosome. A genetic programming method to evolve convolutional neural network architectures is also presented in [13]. In the proposed method, solutions are represented by a set of predetermined building blocks, together with their connections. Solutions are evaluated based on their error rates and number of parameters.

The results reported in these works confirm the benefits of the application of GA to deep learning hyperparameter optimization. However, the computational cost of these approaches is still high, given the number of solutions that must be evaluated by the GA. Therefore, techniques to optimize the use of computation resources in this class of applications remain an important area of inquiry. An example of such a technique is presented in [18]. In their work, a method based on design space exploration to optimize DNN hyperparameters is presented. The proposed method uses an ANN to predict the performance of a given topology. Together with Pareto efficiency, this prediction prevents the waste of computational resources by not evaluating potentially bad solutions. In [19], another approach to minimize the number of hyperparameters of a deep learning model is presented. In their work, the authors demonstrate how pyramidal-structured convolutional neural networks present competitive results when compared to state-of-the-art models, while scaling down the number of parameters and reducing the use of computational resources.

This work aims at contributing to the field of deep learning hyperparameter optimization by providing an evolutionary method for network topology selection with minimal human intervention. The proposed method might be used by researchers and practitioners to define an initial working topology, without requiring previous expertise on the field. The initial topology can be further tuned and optimized according to specific needs such as dataset size and available computational resources. In the next sections of this paper, details of the proposed selection method will be presented.

3. Materials and Methods

In the experiments presented in this paper, four topology search methods (described later in this section) were evaluated. The previously reported MNIST and CIFAR-10 datasets were used to support this study and are available at DOI 10.1109/MSP.2012.2211477 [20] and in [21]. For the MNIST dataset, 1024 samples were randomly extracted for training, while 256 samples (distinct from the training set) were chosen to test the neural networks. With the CIFAR-10 dataset, 2000 samples for training and 500 samples for test were used. To provide fair comparisons, all search methods used the same data. Model performance was measured as the test accuracy, i.e., the number of test patterns correctly classified divided by the total number of test patterns, as follows:

$$\mathrm{acc}\,(\%) = \frac{n_C}{n_C + n_W} \times 100, \tag{1}$$

where n_C is the number of test patterns correctly classified and n_W is the number of wrong classifications.

In order to represent model complexity, the number of connections in the model was used. These two metrics—test accuracy and number of weights—will be used to evaluate a solution with respect to the Pareto front, in a multiobjective optimization problem: maximize the test accuracy and minimize the number of weights.

The Caffe framework was used to implement the deep learning models evaluated in this work [22]. A customized genetic algorithm was developed using the C++ programming language and can be used as a reference implementation of the presented method. A personal computer without GPU acceleration was used to train the models.

In all evaluated search methods, each solution is represented by two stacks of layers, the first being convolutional layers and the second fully connected layers. Each layer is presented as a string descriptor, as shown in the examples of Figure 1. To facilitate parsing these descriptors, elements are separated by semicolon characters.

3.1. Random Search. To evaluate random search performance, a set of 4000 distinct solutions was randomly generated, following the search space restrictions presented in Table 1. These same restrictions were used by the GA-based methods.

In order to preserve the random nature of the search method, no guidance was set for model generation. When invalid models are created (for example, because of spacial reduction), they are replaced with valid ones. Each model is then trained for 100 epochs, using the standard Adam optimization method, with parameters $\alpha = 0.001$, $\beta1 = 0.9$, $\beta2 = 0.999$, and $\varepsilon = 10^{-8}$ [23]. Batch sizes of 64 for the MNIST dataset and 50 for the CIFAR-10 dataset were used. To prevent resource waste, an early stopping of 10 epochs was set up, so that training is interrupted when no improvement in the test accuracy observed for 10 consecutive epochs.

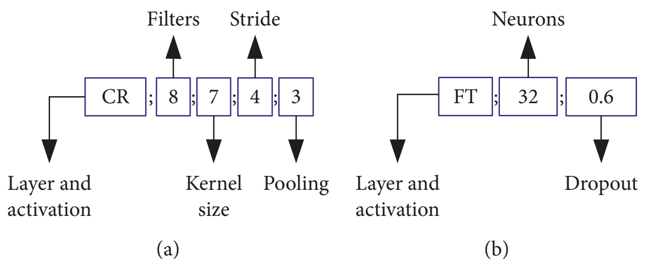

Figure 1: Layer descriptor examples. In (a), a convolutional layer, activated by ReLU, with 8 filters, a 7 × 7 kernel, stride of 4, and pooling of 3. In (b), a fully connected layer, activated by TanH, with 32 neurons and a dropout rate of 0.6.

Table 1: Search space restrictions.

Description	Value(s)
# convolutional layers	0–3
Convolutional filters	{2, 4, 8, 16, 32}
Kernel size	{3, 5, 7, 9, 11}
Stride	{1, 2, 3, 4}
Pooling	{2, 3, 4}
Activation functions	{Linear, Sigmoid, TanH, ReLU}
# fully connected layers	0–2
Neurons	{8, 16, 32, 64}
Dropout	{0.0, 0.2, 0.4, 0.6}
Activation functions	{Linear, Sigmoid, TanH, ReLU}

3.2. Grid Search. The objective of grid search is the exhaustive exploration of a solution space. However, this is impractical in deep learning topology optimization, given the high number of hyperparameters to be configured. Therefore, to provide a fair comparison setup, a more restrictive subset of the search space presented in Table 1 was chosen. This subset, presented in Table 2, contains a total of 3654 models, resulting from all combinations of its hyperparameter values. In grid search evaluation, discarded invalid solutions were not replaced, to preserve strict exploration of the reduced search space.

3.3. Genetic Algorithm. The GA-based search was conducted using the parameters presented in Table 3. By defining a population size of 200 and 20 generations, a maximum of 4000 solutions is to be evaluated. Crossover and mutation probabilities were set to 70% and 20%, respectively. An elitism parameter was configured so that the 10% fittest solutions are automatically propagated to the next generation. The remaining solutions will be obtained through recombination of the current population. The standard roulette wheel was used as the selection method so that each solution has a selection probability proportional to its fitness.

3.3.1. Chromosome. Each solution, s_i, $i = 1, \ldots, 200$, is represented by two stacks of layer descriptors, one for convolutional layers and the other for fully connected layers. Therefore, each chromosome is composed of two vectors of strings, each string being the descriptor of a layer as

Table 2: Search space restrictions for grid search.

Description	Value(s)
# convolutional layers	0–3
Convolutional filters	{16, 32}
Kernel size	{3, 5}
Stride	1
Pooling	2
Activation function	ReLU
# fully connected layers	0–2
Neurons	{32, 64}
Dropout	0.4
Activation functions	{Sigmoid, TanH, ReLU}

Table 3: GA parameters.

Description	Value(s)
Population size	200
Generations	20
Crossover probability	0.7
Mutation probability	0.2
Elitism	0.1
Selection method	Roulette

presented in Figure 1. Convolutional layers descriptors contain the activation function, number of filters, kernel size, stride, and pooling size. Fully connected layer descriptors contain the number of neurons and the dropout probability. These solutions compose the initial population P, presented as follows:

$$P = \{s_1, s_2, \ldots, s_{200}\}. \quad (2)$$

3.3.2. Evaluation. The fitness function, used to evaluate a given solution, is represented by the test accuracy of the solution in the corresponding classification problem. The solution's layer descriptors are used to build a deep learning model, which is trained using the same procedure described in Section 3.1. The best accuracy of the model when classifying test samples (samples which are not present in training), calculated using equation (1), is then used as the solution fitness. Once a model is trained, it is stored in a local database, to prevent it from being re-evaluated if it is present in further generations.

3.3.3. Selection. Solutions are selected for recombination based on their fitness by using the roulette wheel selection method. To provide better solutions with higher chances of being selected, a probability q_i is assigned to each solution s_i according to the following equation [11]:

$$q_i = \frac{f(s_i)}{\sum_{k=1}^{200} f(s_k)}, \quad i = 1, 2, \ldots, 200, \quad (3)$$

where $f(s_i)$ is the fitness (i.e., test accuracy) of the i_{th} solution. The cumulative probability $\hat{q}_i$ for solution s_i is defined as [11]

$$\hat{q}_i = \sum_{k=1}^{i} q_k, \quad i = 1, 2, \ldots, 200. \quad (4)$$

After randomly generating a floating point $p \in [0, 1]$, solution s_i is chosen if $\hat{q}_{i-1} < p \le \hat{q}_i$, given $\hat{q}_0 = 0$.

3.3.4. Crossover. Crossover is performed by swapping layers between 2 parent solutions, according to the randomly chosen cut points. Since each solution is composed by two stacks of layers (convolutional and fully connected), the recombination is conducted independently for each stack, using the standard single-point crossover [17]. For each layer stack, a random floating point $p \in [0, 1]$ is generated, and crossover is conducted if $p \le 0.7$.

In Figure 2(a), two parent solutions (S1 and S2) are presented. Solution S1 has 3 convolutional and 1 fully connected layers, while S2 has 2 convolutional and 3 fully connected layers. The cut-points of both stacks of each parent are indicated using arrows in Figure 2(b). Finally, in Figure 2(c), the two children solutions resulting from the crossover operation are presented: S3 has one convolutional layer and 3 fully connected layers, while S4 presents 4 convolutional and 1 fully connected layers.

3.3.5. Mutation. The mutation operation consists on adding or removing layers at random positions in the solution's chromosome. As with crossover, mutation is conducted independently, for convolutional and fully connected layers. For each layer stack, a random floating point $p \in [0, 1]$ is generated. If $p \le 0.2$, a second random floating point $p_{\text{add}} \in [0, 1]$ will be generated, indicating the type of mutation. If $p_{\text{add}} \le 0.5$, a random layer will be added at a random position, in the corresponding layer stack. Otherwise, one of the existing layers will be removed from the corresponding layer stack.

In Figure 3(a), the two solutions previously obtained after crossover are presented. The arrows in Figure 3(b) indicate mutation points: for solution S3, a new random layer will be added after the arrow; for S4, the layer pointed by the arrow will be removed. The two solutions presented in Figure 3(c) are the resulting solutions of these operations.

After crossover and mutation, valid produced solutions are evaluated and added to the population. Invalid solutions are discarded, and new recombinations take place until the population is complete with 200 individuals.

3.4. Genetic Algorithm + Fitness Predictor. Many of the solutions evaluated in the previous methods present a low test accuracy after deep learning model training. This is an undesirable effect since computational resources were allocated to train a model which will be further discarded. For this reason, a modified version of the GA-based search was experimented. In this version, new generated solutions are subjected to a performance prediction step. The predicted performance will then be used to determine if this solution is worth evaluation. To achieve this, an ANN-based predictor

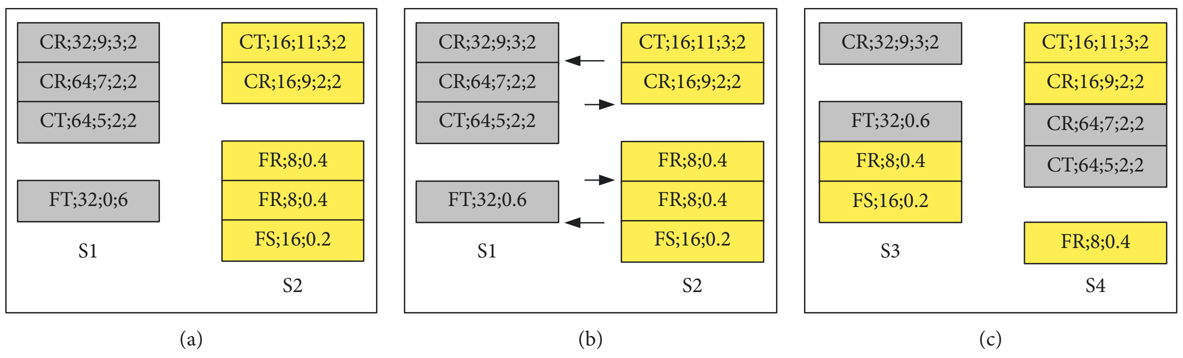

Figure 2: Crossover example: solutions S3 and S4 are produced from crossover of parents S1 and S2.

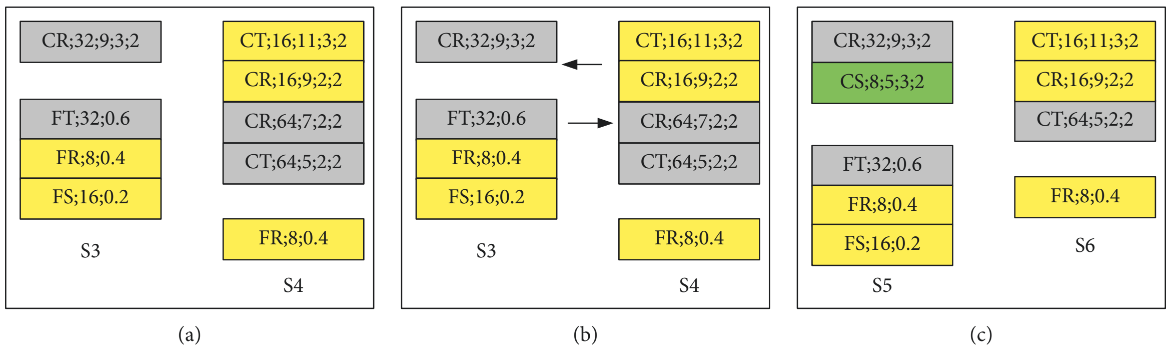

Figure 3: Mutation example: solutions S5 and S6 are produced after mutation of S3 and S4.

was set up, using knowledge gained from previous evaluations to estimate performance of new models.

The fitness predictor was setup with a fixed topology of 3 fully connected layers, with 64 neurons each, and a dropout rate of 0.4. The inputs of the predictor are formatted as a vector of integers containing the descriptors of the model layers. Each convolutional layer is represented by a 9-element vector, containing the number of filters, kernel size, stride, pooling, normalization, and activation function. Each fully connected layer is represented by a 6-element vector, containing the number of neurons, activation function, and the dropout rate. The final input vector contains 39 integers, composed of 3 convolutional descriptors and 2 fully connected descriptors. Empty layers are represented by a series of zeros in the corresponding positions. Figure 4 presents examples of the predictor inputs.

The fitness predictor was trained for 100 epochs, with an early stopping of 10 epochs, using the same optimizer as the MNIST and CIFAR-10 classifiers. The available training data (set of models previously trained and the corresponding fitness) is split into training (80% of the available data) and test (20% of the available data) patterns. When a solution is evaluated, it feeds the ANN predictor, which is retrained with the new pattern.

The decision on whether to evaluate a model or not is made upon the comparison of the predicted fitness with the already evaluated models. To this end, the Pareto efficiency was used, considering two dimensions: the solution fitness (performance of the classifier) and the number of weights (complexity of the model). Dominant solutions (i.e., solutions that are superior to others in all attributes) have a 90% probability of being evaluated, while dominated solutions have a 10% probability of being evaluated. The complete algorithm used in this search method is described in the next paragraphs.

In Figure 5, the main tasks for generating the initial population are presented.

(1) Initially, 10 solutions are randomly generated and evaluated.

(2) The Pareto front is created, containing the dominant solutions among these 10 solutions.

(3) The fitness predictor is trained, using the 10 solutions as input patterns. 8 solutions are randomly chosen to be used as training patterns, while the 2 remaining solutions are used as test patterns.

(4) While the number of solutions in the population is less than the desired population size (200), new solutions are randomly generated.

(5) The predictor described in task 3 is used to predict the fitness of the new generated solution. This predicted fitness, together with the number of weights of the solution, will be used to determine if its network model will be trained or not. The procedure for solution evaluation is presented in Figure 6.

(6) If the solution (with predicted fitness) belongs to the Pareto front (dominant solution), it has a 90%

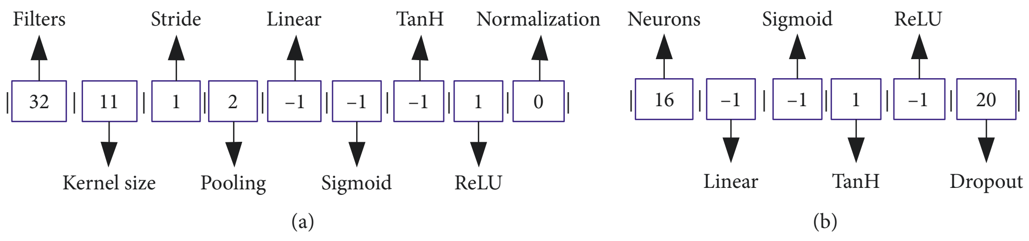

Figure 4: Predictor input examples. In (a), the descriptor CR;32;11;1;2 is represented. In (b), the descriptor FT;16;0.2 is represented.

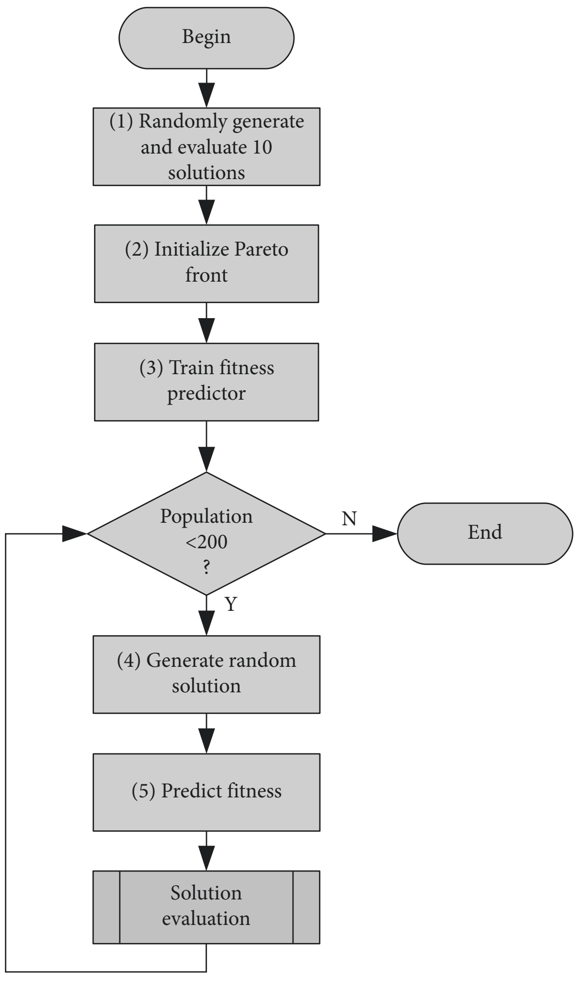
Begin
(1) Randomly generate and evaluate 10 solutions
(2) Initialize Pareto front
(3) Train fitness predictor
Population <200 ?
N
End
Y
(4) Generate random solution
(5) Predict fitness
Solution evaluation

Figure 5: Initial population generation.

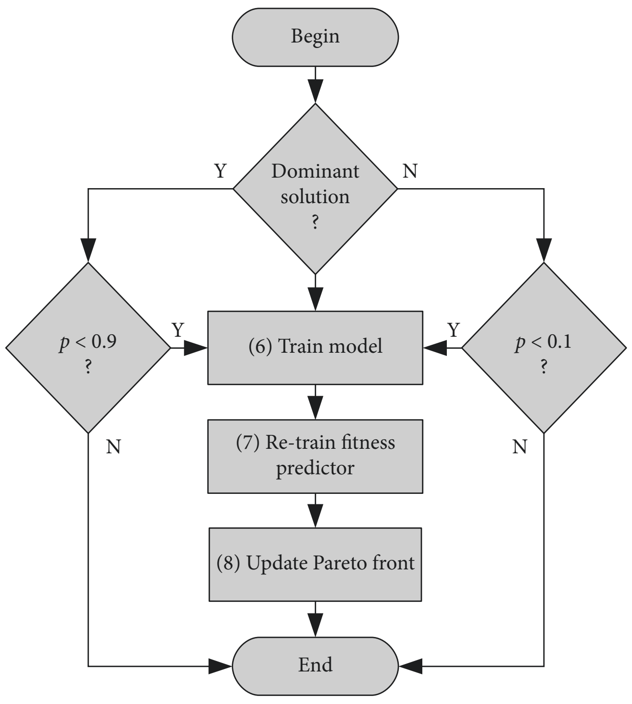

Figure 6: Solution evaluation.

probability of being trained. Otherwise (dominated solution), this probability decreases to 10%. This step has the objective of providing more resources to potentially good solutions, while preventing resource wasting by bad solutions. If a random generated number (p) is below the respective threshold, the solution's model is trained.

(7) After model training, the fitness predictor is updated, adding the new solution to the input patterns and being retrained from scratch. 80% of the available patterns are used for training, while the remaining patterns are used for the test.

(8) Finally, the Pareto front is updated with the new solution.

After the initial population is generated, it is evolved through the desired number of generations. In the results reported in this work, the population is evolved for 20 generations. The steps for population evolution are presented in Figure 7.

(9) If the number of generations is not achieved, a new generation is initialized with an elite of the 20 best solutions (10%) of the current population.

(10) Elite solutions which were not previously evaluated (predicted solutions) have their models trained.

(11) While the number of solutions in the current generation is less than the population size (200), two solutions are selected from the current population, using the roulette selection algorithm.

(12) The crossover operator (as described in Section 3.3.4) is applied, with a probability of 70%.

(13) The mutation operator (as described in Section 3.3.5) is applied, with a probability of 20%.

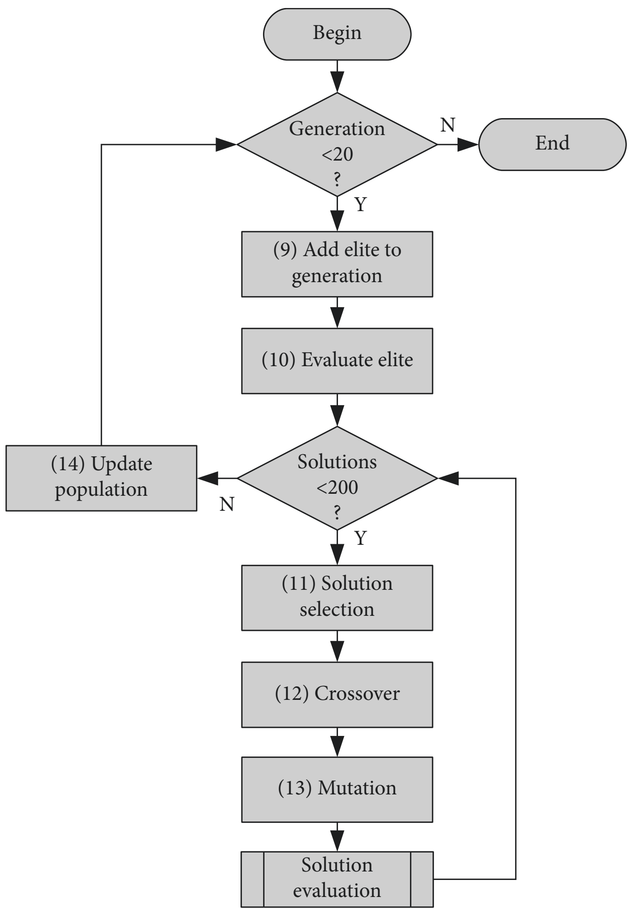

Figure 7: Population evolution.

(14) When the number of desired solutions is achieved, the population is updated with the solutions in the current generation.

4. Results and Discussion

After testing the four search methods described in the previous section, it was possible to identify some contributions of the proposed method, which will be discussed in this section. In Table 4, the correlation between the test accuracy and some hyperparameters of the neural networks is presented.

The results presented in Table 4 show that no strong correlation between the isolated hyperparameters and the test accuracy can be observed. This behavior indicates that the performance of the network in the target problem must be affected by the ensemble of hyperparameters, i.e., network layers and layer blocks, as the hyperparameters of adjacent layers can directly impact the contribution of a given layer in the network topology.

Figure 8 presents the evaluated solutions for the 4 search methods with the MNIST and CIFAR-10 datasets, highlighting the Pareto front obtained with each one. It is possible to notice the concentration of evaluated solutions near the Pareto front, for all experiments. In grid search, the exploration of a restricted solution subspace causes the prevalence of solutions with more weights, diverging from the other approaches which randomly explore the entire solution space. It is important to highlight that the Pareto front was determined using equal weights for both attributes (fitness and weights). Therefore, in these results, a small model with low accuracy is equally important as a big model with higher accuracy. To change this behavior, weights can be applied to each attribute, making, for example, accuracy more important than model complexity.

Table 4: Correlation between model hyperparameters and test accuracy.

Hyperparameters	Correlation coefficient (ρ)	
	MNIST	CIFAR-10
# convolutional layers	−0.08	−0.25
# fully connected layers	−0.23	−0.21
# convolutional filters	0.12	0.13
Kernel size	0.13	0.11
# fully connected neurons	0.24	0.20
Dropout rate	0.02	0.00

For the CIFAR-10 dataset, although some models achieved 100% accuracy on training samples, all evaluated models presented test accuracy below 60%. As the training and test samples were randomly chosen from the original dataset (preserving equal distribution of samples per class), we believe this results from the inherent complexity of the CIFAR-10 dataset, which would require more training samples to improve testing accuracy. To investigate this hypothesis, the best models were chosen to a second training stage, with more training samples from the original dataset. The results of this evaluation will be presented further in this section.

Figure 9 presents the evolution of both the population of solutions and its best individual across all generations for GA and GA + Predictor search methods. No significant difference was observed when comparing GA and GA + Predictor results, except for the fact that, with the MNIST dataset in the GA + Predictor experiment, the best solution was found earlier during evolution, which did not occur in the CIFAR-10 dataset. In all cases, the best solutions were kept in the next generations, given the use of elitism. Tables 5 and 6 summarize the results of the four search methods applied to both datasets.

The GA-based search methods were able to find competitive (and even superior) models compared to the random and grid search methods. However, the number of evaluated models was drastically reduced, as a result of the genetic operators which replicate good solutions in the further generations of the population, causing its convergence to the best evaluated solutions. In grid search, the reduction of the number of evaluated solutions is due to the selected solution space, which contained invalid solutions. This result would be different depending on the search space selection, but this would require prior domain knowledge, differently from the other methods which minimize human intervention.

Execution times were measured considering the time spent with optimization (population initialization, selection,

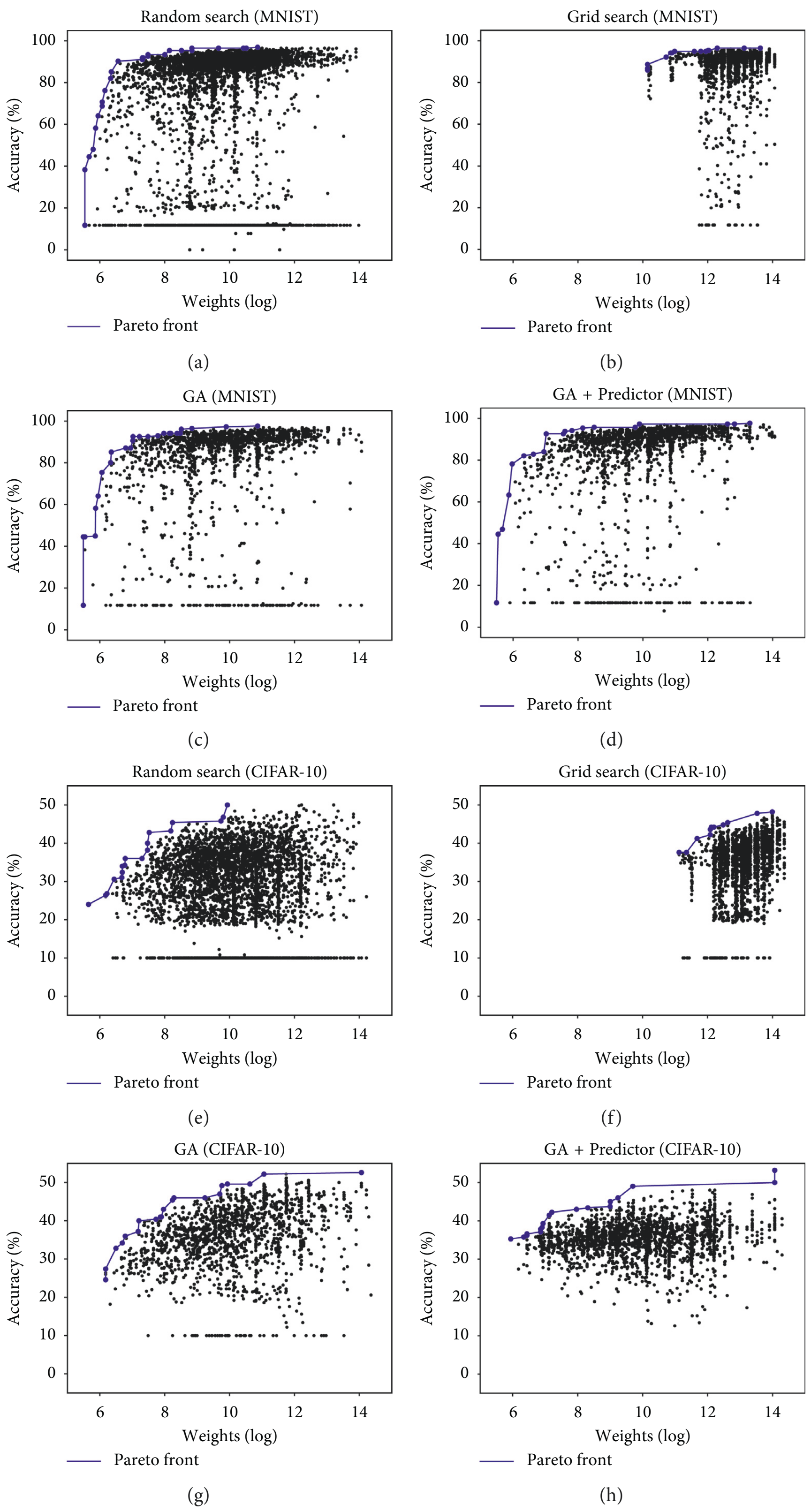

FIGURE 8: Pareto front obtained with each search method for the MNIST and CIFAR-10 datasets.

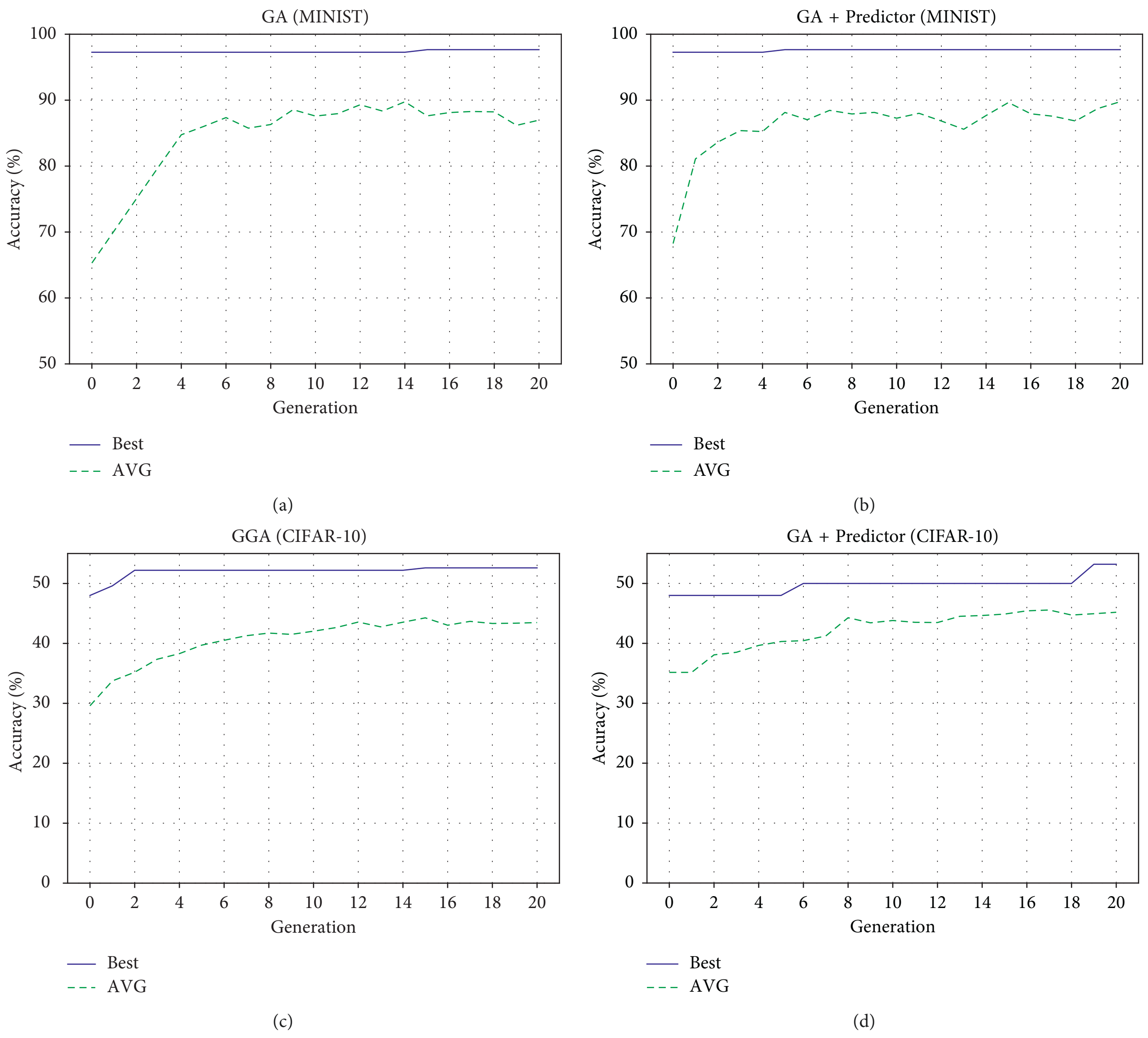

Figure 9: Population evolution for the MNIST and CIFAR-10 datasets.

Table 5: MNIST summary.

Method	Evaluated solutions	Best solution		
		Model	Accuracy (%)	Weights
Random search	4000	CTN;32;5;2;3 FL;16;0.6	97	52218
Grid search	2278	CR;32;3;1;2 CR;16;5;1;2 FS;32;0.4	96	218298
GA	1533	CT;32;5;2;3 FL;16;0.2	98	52218
GA + Predictor	1528	CRN;32;9;1;4 FL;64;0.4	98	595201

Table 6: CIFAR-10 summary.

Method	Evaluated solutions	Best solution		
		Model(s)	Accuracy (%)	Weights
Random search	4000	CT;16;5;2;4	50	20586
		CT;32;5;1;4	50	202442
		CT;16;7;1;4 FT;64;0.6	50	544778
Grid search	2754	CR;16;3;1;2 CR;32;5;1;2 FT;64;0.4	48	119364
GA	1733	CT;32;5;1;4 FT;64;0.6	53	1283146
GA + Predictor	975	CT;32;5;1;4 FS;64;0.4	53	1283146

genetic operators, etc.), fitness predictor training, and solution evaluation (model training). With the MNIST dataset, the GA search method was executed for 15 hours, with 12 minutes spent by the GA. The GA + Predictor method was executed for 14 hours, with 19 minutes spent by the GA and 9 minutes spent with predictor training. With the CIFAR-10 dataset, GA search was executed for 45 hours (8 minutes spent by the GA) and GA + Predictor search was executed for 24 hours (13 minutes spent by the GA and 8 minutes spent with predictor training). In all these cases, remaining time (total execution time minus optimization and predictor training) was spent with model training. These values show the minimal cost associated with optimization (GA and GA + Predictor search) and fitness prediction (GA + Predictor only), when compared to deep learning model training.

The use of the predictor did not result in significant reduction in the number of evaluated models and execution time for the MNIST dataset, when compared to the standard GA. Also, the best solution found by the GA + Predictor method is far more complex than the one found by the standard GA, which indicates that the prediction efficiency must be improved for this dataset. However, with the CIFAR-10 dataset, a reduction of 43% on the number of evaluated models was observed, in addition to a significant decrease in execution time. The best solution found by the GA + Predictor method is very similar to the solution found by the standard GA method, except for the activation function and dropout rate in the fully connected layer. These results demonstrate the economy of computational resources observed when prediction accuracy is good enough to discard potentially bad models and indicate that further investigation on fitness prediction must be conducted to extend this benefit to other datasets.

In order to compare the performance of the solutions optimized using the GA with existing models, a larger subset of the MNIST and CIFAR-10 datasets was used. For the MNIST dataset, 10048 training samples and 2048 test samples were randomly selected, while preserving the other configuration parameters. For the CIFAR-10 dataset, 10000 training samples and 2000 test samples were used. As in [19], the optimized models were compared to some of the state-of-the-art models, namely, LENET [20] and the Caffe Cifar-10 model (C10). In addition, two pyramidal structured models presented in [19]—SPyr_Rev_LENET and SPyr_Rev_C10—were also evaluated. To provide fair comparisons, all models were trained from scratch using the same training parameters, for a maximum of 100 epochs early stopping when no improvement on the test accuracy was observed for 10 consecutive epochs. Training and test accuracy, measured using equation (1), as well as the time required to train each model from scratch (Δt) were calculated. The results of this experiment are presented in Tables 7 and 8.

With the MNIST dataset samples, both standard GA (GANet) and GA + Predictor (GA + PNet) optimized models presented competitive performance when compared to state-of-the-art models, considering both accuracy and training time. In addition, GANet was the fastest model to train, presenting a decrease of 0.4% and 0.5% in training and test accuracy, when compared to the best model (LENET). These optimized solutions were obtained without human intervention, through the evolution of an initial population of randomized solutions. For CIFAR-10, the GANet model presented inferior performance, when compared to the benchmark results. The GA + PNet model showed slightly better results, with a decrease of 2-3% when compared to the benchmark average in the evaluated samples, despite requiring less training time than 2 of the benchmark models. However, as for the MNIST experiment, the proposed method's objective is to provide researchers and practitioners with competitive starting points, without requiring prior expertise or intervention. These starting models can and should be subjected to a fine-tuning process, in order to improve their performance for the target problem. Also, the classification improvement observed in this last experiment, when compared to the optimization stage, indicates that more challenging datasets might require larger training sets or bigger populations of solutions during optimization, at the cost of higher usage of computational resources. Further work will be conducted to investigate these hypotheses, as well as the improvement of the fitness predictor demonstrated in this work.

5. Conclusion

In this paper, a method for DNN topology selection using genetic algorithms was presented. The evolutionary-based

Table 7: Optimized and reference models (MNIST).

Model	Descriptor	Accuracy(%) Train	Accuracy(%) Test	Weights	Δt (s)
LENET	CL;20;5;1;2 CL;50;5;1;2 FR;500;0.0	100	98.8	8131080	373
C10	CR;32;5;1;3 CR;32;5;1;3 CR;64;5;1;3 FL;64;0.0	99.1	97.6	488042	4121
SPyr_Rev (LENET)	CL;50;5;1;2 CL;20;5;1;2 FR;500;0.0	100	98.6	3271830	652
SPyr_Rev (C10)	CR;64;5;1;3 CR;32;5;1;3 CR;32;5;1;3 FL;64;0.0	99.4	98.3	284042	3136
GANet (MNIST)	CT;32;5;2;3 FL;16;0.2	99.6	98.3	52218	242
GA + PNet (MNIST)	CRN;32;9;1;4 FL;64;0.4	99.7	98.8	595201	429

Table 8: Optimized and reference models (Cifar-10).

Model	Descriptor	Accuracy(%) Train	Accuracy(%) Test	Weights	Δt (s)
LENET	CL;20;5;1;2 CL;50;5;1;2 FR;500;0.0	100	100	12132080	1514
C10	CR;32;5;1;3 CR;32;5;1;3 CR;64;5;1;3 FL;64;0.0	97.8	98.6	882858	6953
SPyr_Rev (LENET)	CL;50;5;1;2 CL;20;5;1;2 FR;500;0.0	99.5	99.2	3271830	2005
SPyr_Rev (C10)	CR;64;5;1;3 CR;32;5;1;3 CR;32;5;1;3 FL;64;0.0	89.8	88.5	483850	13204
GANet (CIFAR-10)	CTN;32;5;1;4 FT;64;0.6	75.6	75.1	1283146	3794
GA + PNet (CIFAR-10)	CT;32;5;1;4 FS;64;0.4	94.1	93.9	1283146	4870

techniques were able to achieve competitive results with minimal human intervention and using less computational resources, when compared to random and grid search. The optimized solutions were compared to state-of-the-art models, offering promising starting points given the vast search space of DNN hyperparameters.

The use of a fitness predictor in the model evaluation stage resulted in a significant reduction on the number of evaluated models and execution time for one of the explored datasets. This fact demonstrates the benefits of fitness prediction but suggests that the use of other prediction methods, as well as the optimization of the predictor itself may lead to different results. Such methods should include accuracy prediction from early epochs and accuracy estimation from smaller subsets.

Future work will be conducted in order to evaluate the application of the proposed search methods in other datasets and applications, including image regression problems. Improvement of the fitness predictor and scalability of the search method to bigger datasets will also be investigated.

Data Availability

The previously reported MNIST and CIFAR-10 datasets were used to support this study and are available at https://doi.org/10.1109/MSP.2012.2211477 [20] and in [21].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by CAPES/Brazilian Ministry of Education & Culture, to whom the authors are deeply grateful.

Supplementary Materials

The source code used to support the findings of this study is included within the supplementary material of this paper. This material includes two C++ projects (ga-dnn-mnist and ga-dnn-cifar10), each one containing the source files used in dataset processing, deep neural network training, GA operations, and fitness prediction. A README file within each project provides details about individual source files and compilation instructions. Any additional request concerning the use of this material can be addressed to the corresponding author. (*Supplementary Materials*)

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] J. Patterson and A. Gibson, *Deep Learning: A Practitioner’s Approach*, O’Reilly Media, vol. 1, 1st edition, 2017.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, USA, 2016, http://www.deeplearningbook.org.

[4] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, “Evolving the topology of large scale deep neural networks,” in *Proceedings of European Conference on Genetic Programming*, pp. 19–34, Springer, Parma, Italy, April 2018.

[5] S. Huang, X. Li, Z. Cheng, Z. Zhang, and A. Hauptmann, “Gnas: a greedy neural architecture search method for multi-attribute learning,” 2018, http://arxiv.org/abs/1804.06964.

[6] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[7] J. Yang, Y.-Q. Zhao, and J. C.-W. Chan, “Learning and transferring deep joint spectral-spatial features for hyperspectral classification,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 8, pp. 4729–4742, 2017.

[8] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning,” 2016, http://arxiv.org/abs/1611.02167.

[9] A. L. Rincon, A. Tonda, M. Elati, O. Schwander, B. Piwowarski, and P. Gallinari, “Evolutionary optimization of convolutional neural networks for cancer mirna biomarkers classification,” *Applied Soft Computing*, vol. 65, pp. 91–100, 2018.

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

[11] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, “Tuning of the structure and parameters of a neural network using an improved genetic algorithm,” *IEEE Transactions on Neural Networks*, vol. 14, no. 1, pp. 79–88, 2003.

[12] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[13] M. Suganuma, S. Shirakawa, and T. Nagao, “A genetic programming approach to designing convolutional neural network architectures,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 497–504, ACM, Berlin, Germany, July 2017.

[14] S. Dutta and E. Gros, “Evaluation of the impact of deep learning architectural components selection and dataset size on a medical imaging task,” in *Proceedings of Medical Imaging 2018: Imaging Informatics for Healthcare, Research, and Applications*, vol. 10579, March 2018.

[15] J. Safarik, J. Jalowiczor, E. Gresak, and J. Rozhon, “Genetic algorithm for automatic tuning of neural network hyperparameters,” in *Proceedings of Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*, vol. 10643, article 106430Q. International Society for Optics and Photonics, Orlando, FL, USA, May 2018.

[16] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, Pearson, London, UK, 3rd edition, 2009.

[17] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, USA, 1998.

[18] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, “Neural networks designing neural networks: multi-objective hyper-parameter optimization,” in *Proceeding of 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, Austin, TX, USA, November 2016.

[19] I. Ullah and A. Petrosino, “About pyramid structure in convolutional neural networks,” in *Proceeding of 2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 1318–1324, IEEE, Vancouver, BC, USA, July 2016.

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

[21] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Tech. Rep., University of Toronto, Toronto, ON, Canada, 2009.

[22] Y. Jia, E. Shelhamer, J. Donahue et al., “Caffe: convolutional architecture for fast feature embedding,” 2014, http://arxiv.org/abs/1408.5093.

[23] D. P. Kingma and J. Ba, “Adam: a method for stochastic optimization,” 2014, http://arxiv.org/abs/1412.6980.