mathematics

MDPI

Article

Monarch Butterfly Optimization Based Convolutional Neural Network Design

Nebojsa Bacanin *, Timea Bezdan, Eva Tuba, Ivana Strumberger and Milan Tuba *

Faculty of Informatics and Computing, Singidunum University, Danijelova 32, 11010 Belgrade, Serbia; tbezdan@singidunum.ac.rs (T.B.); etuba@ieee.org (E.T.); istrumberger@singidunum.ac.rs (I.S.)

* Correspondence: nbacanin@singidunum.ac.rs (N.B.); tuba@ieee.org (M.T.); Tel.: +381-653093-224 (N.B.); +381-653093-223 (M.T.)

Received: 28 April 2020; Accepted: 4 June 2020; Published: 8 June 2020

Abstract: Convolutional neural networks have a broad spectrum of practical applications in computer vision. Currently, much of the data come from images, and it is crucial to have an efficient technique for processing these large amounts of data. Convolutional neural networks have proven to be very successful in tackling image processing tasks. However, the design of a network structure for a given problem entails a fine-tuning of the hyperparameters in order to achieve better accuracy. This process takes much time and requires effort and expertise from the domain. Designing convolutional neural networks' architecture represents a typical NP-hard optimization problem, and some frameworks for generating network structures for a specific image classification tasks have been proposed. To address this issue, in this paper, we propose the hybridized monarch butterfly optimization algorithm. Based on the observed deficiencies of the original monarch butterfly optimization approach, we performed hybridization with two other state-of-the-art swarm intelligence algorithms. The proposed hybrid algorithm was firstly tested on a set of standard unconstrained benchmark instances, and later on, it was adapted for a convolutional neural network design problem. Comparative analysis with other state-of-the-art methods and algorithms, as well as with the original monarch butterfly optimization implementation was performed for both groups of simulations. Experimental results proved that our proposed method managed to obtain higher classification accuracy than other approaches, the results of which were published in the modern computer science literature.

Keywords: swarm intelligence; monarch butterfly optimization; hybridized monarch butterfly optimization; convolutional neural networks; neuroevolution; network optimization

1. Introduction

Convolutional neural networks (CNNs) [1] are a special type of deep learning models that have demonstrated high performance on different types of digital image processing tasks. CNNs are used for image recognition, image classification, object detection, pose estimation, face recognition, eye movement analysis, scene labeling, action recognition, object tracking, etc. [2–8].

CNNs have become a fast-growing field in recent years, though their evolution started much earlier. In 1959, Hubel and Wiesel [9] published one of the most influential papers in this area. They conducted many different experiments with the aim to understand how neurons work in the visual cortex. They found that the primary visual cortex in the brain has a hierarchical organization with simple and complex neurons and the visual processing always starts with simple structures such as oriented edges, and the complex cells receive input from the lower-level simple cells.

The first example of an artificial neural network model, the Neocognitron, was introduced by Fukushima in 1980 [10]. The Neocognitron had the same logic with simple and complex cells that were discovered by Hubel and Wiesel. Fukushima built up a hierarchy with alternative layers of simple cells

(S-cells) and complex cells (C-cells), where the simple and complex cells show similar characteristics to the visual cortex's simple and complex cells. The simple cells contain parameters that are modifiable, and the complex cells, on top of them, performed pooling. At that time, backpropagation was not applied to the architecture. Later, in 1989, LeCun applied backpropagation on the Neocognitron artificial neural network, resulting in a 1% error rate and about a 9% reject rate on zip code digits [11]. LeCun further optimized the network in 1998 by utilizing an error gradient-based learning algorithm.

The first deep convolutional network was introduced in 2012 by Alex Krizhevsky, and the network is named AlexNet [12]. It achieved notable results, and this achievement has brought about a revolution in computer vision. The use of graphical processing units (GPUs) enabled great successful results, as well as the use of the ReLU activation function, data augmentation, and the dropout regularization technique.

Many studies focus on novel architecture development, which has achieved higher classification accuracy, and some of the most well-known modern networks are GoogleNet [13], ResNet [14], DenseNet [15], and SENet [16].

1.1. Convolutional Neural Networks and Hyperparameters' Optimization

CNNs consist of a sequence of different layers. The layers in the architecture mimic the visual cortex mechanism in the brain. The essential layers of the CNN are the convolutional layers, pooling layers, and dense layers. In every CCN, the first layer takes the input image and convolves it by filters, and upon the result, the activation function is applied. In this way, low-level features are extracted from the image, the resulting output becomes the input for the next layer, and each consequent layer extracts more and more complex, higher level features. The pooling layer is used for downsampling, and its kind is max or average pooling. At the end, the architecture has one or more flattened dense layers, and the final one classifies the image.

During the network training, in the weight learning process, the loss function should be optimized. For this purpose, in the modern literature, many optimizers have been proposed, such as stochastic gradient descent, momentum, Adam, rmsprop, adadelta, adagrad, and adamax [17–19]. Over-fitting occurs in the network, when the difference between the training accuracy and test accuracy is high; in other words, the network learns specific data, and the model is unable to predict new input data. To avoid this, different regularization techniques can be used. Some of the practical regularization methods are L_1 and L_2 regularization [20], dropout [21], drop connect [22], batch normalization [23], early stopping, and data augmentation.

The transfer function (activation function) that is applied to the convolved result is used for mapping the input to a non-linear output. Some of the most widely utilized transfer functions are sigmoid, tanh, and rectified linear unit (ReLU) [24]. ReLU represents the de facto standard in the transfer function choice, and its value is calculated as $f(x) = max(x, 0)$.

The operation in the convolutional layer and the activation function application can be defined as follows:

$$z_{i,j,k}^{[l]} = a(w_k^{[l]} x_{i,j}^{[l]} + b_k^{[l]}) \tag{1}$$

where the resulting feature map (activation map) is represented by $z_{i,j,k}^{[l]}$, w_k is the k^{th} filter, $x_{i,j}$ denotes the input at the i, j location, and b is the bias term. The superscript l denotes the l^{th} layer, and the activation function is represented as $a(\cdot)$.

For further reading about general CNNs' foundations and principles, please refer to [12,25].

The accuracy of any CNN depends on its structure, which in turn depends on the values of variables known as hyperparameters [26]. Some of the hyperparameters include the number of convolutional and fully-connected (dense) layers, the number of kernels and kernel size of each convolutional layer, weight regularization in the dense layers, the batch size, the learning rate, the dropout rate, the activation function, etc. With the goal of establishing satisfying classification accuracy for each particular problem instance, a CNN with specific structure (design) should be found.

The universal rule for finding the optimal network structure for a given problem does not exist, and the only way to create a CNN that will perform satisfactorily for a specific task is by utilizing the "trial and error" approach. Unfortunately, with each trial, the generated network should be trained and tested, and these processes are very time consuming and resource intensive.

In accordance with the above, the process of generating an optimal (in most cases, near optimal) CNN network structure for a specific task represents one of the most important challenges and issues from this domain, and it is known in the literature as the hyperparameter optimization problem [27]. Since there are many CNN hyperparameters, as more and more hyperparameters are used in optimization, the difficulty of the problem grows exponentially. That is why this challenge is categorized as an NP-hard task.

As in the case of any NP-hard challenge, the application of deterministic algorithms for tackling CNN hyperparameters' optimization tasks is not plausible, and to solve it, it is necessary to employ stochastic methods and algorithms. Researchers world-wide have recognized the need for developing a framework that will generate, train, and test different CNN architectures with the goal of finding one that is the most suitable for a specific task by using metaheuristic approaches [28–30]. The process of automatically discovering the most suitable CNN structures (CNN hyperparameters' optimization) for a specific tasks by using evolutionary, as well as other nature-inspired metaheuristics is known in the modern computer science literature as "neuroevolution" [31].

As part of the research that is presented in this paper, we also tried to develop such a framework by taking into account various CNN hyperparameters and by utilizing the enhanced version of one promising and recent swarm intelligence metaheuristic. The details of swarm intelligence methods, as well as the relevant literature review are given in Section 2.

1.2. Research Question, Objectives, and Scope

The research proposed in this paper is based on and represents an extension of our previously conducted experiments and simulations from this domain [32–34]. In our most current previous published research [34], we enhanced and adapted two swarm intelligence metaheuristics to generate CNN architectures automatically and tested them on an image classification task. In this research, during the optimization process, we utilized the following CNN hyperparameters: the number of convolutional layers with the number of filters and the kernel size for each layer, as well as number the size of the dense layers.

Inspired by the approaches presented in [31], in the research that is proposed in this paper, with the objective to generate network structures that will perform a given classification task with higher accuracy than other networks, we included more CNN hyperparameters for the optimization process than previously presented works from this domain [29,30,34]: the number of convolutional layers along with the number of kernels, the kernel size and activation function of each convolutional layer, the pooling size, the number of dense (fully-connected) layers with the number of neurons, the connectivity pattern, the activation function, the weight regularization, the dropout for each dense layer, as well as the batch size, the learning rate, and the rule as general CNN hyperparameters. We tried to generate state-of-the-art CNN structures by employing the hybridized monarch butterfly optimization (MBO) algorithm, since the original MBO [35] has proven to be a robust method for solving various NP-hard optimization challenges [36,37].

In our current, as well as in previous research with the MBO metaheuristics [38–40], we noticed some deficiencies, which were particularly emphasized in the exploration phase of the search process. However, by conducting empirical simulations with the basic MBO, we also noticed that the exploitation phase, as well as the balance between intensification and diversification could be further enhanced. For that reason, to tackle the CNN hyperparameter optimization problem, in this paper, we present newly developed hybridized MBO metaheuristics that obtains significantly better performance in terms of convergence and the results' quality than the original MBO approach. The developed hybrid MBO incorporates mechanisms from two well-known swarm algorithms, artificial bee colony (ABC) and the firefly algorithm (FA), to address the shortcomings of the original MBO.

In this paper, we present two sets of experiments. In the first group of simulations, the developed hybridized MBO is firstly tested on a set of standard unconstrained benchmarks, and comparative analysis with the original MBO, as well as one other state-of-the-art improved MBO approach is performed, the results of which were published in respected international journals [41]. We followed a good research practice in such a way that when a new method has been developed, before testing it on a practical problem, it should be tested on a wider benchmark set to evaluate its performance in more depth.

Afterwards, the proposed hybrid MBO was adapted and tested for the CNN design problem, and the results were compared with recent state-of-the-art approaches that were tested under the same experimental conditions and for the same image classification benchmark dataset [31]. Moreover, since the implementation of the original MBO for this problem has not been found in the modern literature, with the goal of more precise evaluation of the proposed hybridized MBO improvements over the original version, we also implemented basic MBO for this problem and performed a comparative analysis. The CNN structures generated were evaluated against the well-known image classification domain, hand-written recognition. For testing purposes, we utilized the MNIST database. The main reason for using this database was the fact that MNIST has been extensively reviewed and used for evaluating many methods, including the algorithms presented in [31], and that we have used them as a direct comparison with our proposed hybrid MBO approach.

According to the subject of research that was conducted for the purpose of this paper, the basic research question can be formulated as follows: Is it possible to generate state-of-the-art CNN architectures that will establish better classification accuracy than other known CNN structures by utilizing enhanced swarm algorithms and by taking into account more CNN hyperparameters in the optimization process?

The objective and scope of the proposed research is twofold. The primary objective is to develop an automated framework for "neuroevolution" based on swarm intelligence methods that will design and generate CNN architectures with superior performance (accuracy) for classification tasks. Besides our previously conducted research [32–34], other research has also addressed this NP-hard task [28–30,42,43]. However, as already noted above, contrary to all enumerated works, in the proposed research in this paper, we included more hyperparameters in the optimization process, as in [31]. Incorporating more hyperparameters makes the search space exponentially larger, as well as the optimization process itself.

The secondary objective of the proposed research is to enhance the basic MBO algorithm by overcoming its deficiencies. The basic motivation for utilizing this method is that the MBO, even in its original implementation, obtains outstanding performance, and our basic assumption is that the MBO in the enhanced version can potentially take its place as one of the best nature-inspired algorithms.

1.3. Structure of the Paper

The remainder of this paper is organized as follows. Section 2 provides insights into similar research from the CNN design domain that can be found in the modern computer science literature, followed by Section 3, in which we present the original, as well as the proposed hybrid MBO metaheuristics along with the detailed explanations regarding the observed drawbacks of the basic MBO approach.

In order to establish a better structure for the paper, the details of the experiments (simulations) conducted are organized into two sections, Sections 4 and 5. First, in Section 4, we present the details of the simulation environment and the datasets that are utilized in the simulations for the unconstrained benchmarks, as well as for the CNN hyperparameters' optimization. Afterwards, the obtained results along with the visual representation and comparative analysis with other state-of-the-art methods for both types of experiments (unconstrained benchmark and practical CNN design) are given in

Section 5. In the final Section 6, we provide a summary of the research conducted along with the scientific contributions and insights into the future work in this promising domain.

2. Swarm Intelligence Algorithms and Related Work

Swarm intelligence metaheuristics belong to a wider family of nature-inspired stochastic algorithms. Swarm algorithms' mechanisms that guide and direct the optimization process are inspired by natural systems, like colonies of ants, hives of bees, groups of bats, herds of elephants, etc. These methods start execution with a pseudo-random initial population, which is generated within the lower and upper boundaries of the search space. Afterwards, the initial population is improved in an iteration-based approach.

In each iteration, two major processes guide the search process: exploitation (intensification) and exploration (diversification). Intensification performs the search in the neighborhood of existing solutions, while exploration tries to investigate unknown areas of the search space. Exploitation and exploration equations are specific for each swarm algorithm, and they model an approximation of the real-world system. One of the major issues that has been addressed in many papers, in every swarm intelligence algorithm, is the exploitation-exploration trade-off [44,45].

Many successful implementations of swarm intelligence algorithms in original and improved/hybridized forms, which were validated against standard unconstrained (bound-constrained) and constrained benchmarks, as well as on many practical challenges, can be found in the modern literature sources. Some of the more well-known swarm algorithms include ant colony optimization (ACO) [46], particle swarm optimization (PSO) [47], artificial bee colony (ABC) [45,48], the firefly algorithm (FA) [49,50], cuckoo search (CS) [51,52], the bat algorithm (BA) [53], the whale optimization algorithm (WOA) [54], elephant herding optimization (EHO) [55–58], and many others [59–67]. Moreover, some of the practical problems and challenges for which swarm intelligence approaches managed to obtain state-of-the-art results include the following: path planning [68,69], portfolio optimization [70,71], wireless sensor network node localization [72,73], the radio frequency identification network (RFID) planning problem [74–76], cloud computing [39,77–79], image segmentation and threshold [80–82], as well as many others [83–88].

The other group of algorithms, which also belongs to the family of nature-inspired methods, are evolutionary algorithms (EA). The EA approaches model the process of biological evolution by applying selection, crossover, and mutation operators to the individuals from the population. In this way, only the fittest solutions manage to "survive" and to propagate into the next generation of the algorithm's execution. The most prominent subcategories of EA are genetic algorithms (GA) [89], evolutionary programming (EP) [90], and evolutionary strategies [91].

Based on the literature survey, swarm intelligence and EA methods have been applied to the domain of CNN hyperparameters' optimization and neuroevolution with the goal of developing an automatic framework that will generate an optimal or near-optimal CNN structure for solving a specific problem. However, due to the complexity of this task and computer resource requirements, many of them have tried to optimize only a few CNN hyperparameters, while the values of the remaining parameters were set to be fixed (static).

Two interesting PSO-based algorithms for CNNs design were presented in [27,92]. In [92], basic PSO was improved with gradient penalties for the generated optimal CNN structure. The proposed method was validated against three emotion states of subjects that were collected using EEG signals and managed to obtain significant results. An orthogonal learning particle swarm optimization (OLPSO), that was shown in [27], was used for optimizing the values of hyperparameters for VGG16 and VGG19 CNNs that were applied to the domain of plant disease diagnosis. The OLPSO outperformed other state-of-the-art methods that were validated for the same dataset in terms of the classification accuracy.

Four swarm intelligence algorithms, FA, BA, CS, and PSO, that were implemented and adapted for addressing the over-fitting problem, were presented in [43]. By using these approaches, this issue was overcome by establishing a proper selection of the regularization parameter dropout. All algorithms were tested against the well-known MNIST dataset for image classification, and satisfying accuracy was obtained. Another new PSO method for generating adequate CNN architecture was shown in [26]. The canonical PSO for CNN (cPSO-CNN) managed to adapt to the variable ranges of CNN hyperparameters by improving the canonical PSO exploration capability and redefining PSO's scalar acceleration coefficients as vectors. The cPSO-CNN was compared with seven outstanding methods for the same image classification task and obtained the best results in terms of classification accuracy, as well as computational costs. In [28], by applying a PSO-based method, the authors managed to generate CNNs with a better configuration than AlexNet for five image classification tasks. Hybrid statistically-driven coral reef optimization (HSCRO) for neuroevolution was proposed in 2020 [93]. This method was used for optimizing the VGG-16 model for two different datasets, CIFAR-10 and CINIC-10, and managed to generate CNNs with a lighter architecture along with an 88% reduction of the connection weights.

A method that successfully combines GA and CNNs for non-invasive classification of glioma using magnetic resonance imaging (MRI) was proposed in [94]. The authors developed an automatic framework for neuroevolution by evolving the architecture of a deep network using the GA method. Based on the results for the two test studies, the proposed method proved to be better than its competitors. In [95], the authors presented a method for generating a differentiable version of the compositional pattern producing network (CPP), called the DPPN. Microbial GAs are used to create DPNNs with the goal of replicating CNN structures. Recently, an interesting project for an automating deep neural network architecture, called DEvol, was established [96]. DEvol supports a variable number of deep, as well as convolutional layers. According to the presented documentation, the framework managed to obtain a test error rate of 0.6% on the MNIST dataset, which represents the state-of-the-art result.

3. Proposed Method

As was already stated in Section 1.2, for tackling the CNN hyperparameter optimization problem, in this paper, we propose a hybridized version of the MBO swarm intelligence metaheuristics. In this section, we first describe the original MBO algorithm and, afterwards, present the devised hybrid method. Moreover, in this section, we also give the details of the original MBO's drawbacks that our hybrid approach addresses.

3.1. Original Monarch Butterfly Optimization Algorithm

The first version of the MBO was proposed by Wang et al. in 2015 [35]. The algorithm was motivated by the monarch butterfly migration process. In the very first paper, where the MBO was presented for the first time, the authors compared the MBO to five other metaheuristic algorithms and evaluated it on thirty-eight benchmark functions [35]. The MBO achieved better results on all instances than the other five metaheuristics.

Monarch butterflies live in two different places, in the northern USA and southern Canada, and they migrate to Mexico. There are two behaviors in how they change their location: by the migration operator and the butterfly adjusting operator, respectively. That is to say, these two operators define the search direction of each individual in the population.

The basic rules of the algorithm, which perform approximation of the real system, include the following [35]:

1. The population of the individuals is in two different locations (Land 1 and Land 2);
2. The offspring are created on both places, by utilizing the migration operator;
3. If the new individual has better fitness than the parent monarch butterfly, it will replace the old solution;
4. The solutions with the best fitness value remain unchanged for the next iteration.

3.1.1. Migration Operator

The entire population, denoted as SN (solution number), of monarch butterfly individuals (solutions) is divided into two sub-populations, Sub-population 1 ($SP1$) and Sub-population 2 ($SP2$). $SP1$ and $SP2$ correspond to the two lands where they are located, Land 1 and Land 2, respectively.

$$SP1 = ceil(p \cdot SN) \tag{2}$$

$$SP2 = SP - SP1 \tag{3}$$

where p represents the individual's migration ratio in the Sub-population 1.

The process of migration is executed in the following way:

$$x_{i,j}^{t+1} = \begin{cases} x_{r_1,j}^t & \text{if } r \leq p, \\ x_{r_2,j}^t & \text{otherwise} \end{cases} \tag{4}$$

where the jth element of the ith individual at iteration $t+1$ is denoted by $x_{i,j}^{t+1}$. $x_{r_1,j}^t$ and $x_{r_2,j}^t$ represent the locations of r_1 and r_2 individuals, which are randomly selected from $SP1$ and $SP2$, respectively at iteration t, and j corresponds to the jth component.

The parameter r decides whether the jth element of the new solution will be selected from Sub-population 1 or Sub-population 2. The value of r is calculated as the product of a random number between zero and one ($rand$) and the period of migration $(peri)$.

$$r = rand \cdot peri \tag{5}$$

where the suggested $peri$ value is 1.2 [35].

3.1.2. Butterfly Adjusting Operator

The second mechanism to guide the individuals toward the optimum within the search space is the butterfly adjusting operator. In this process, if $rand \leq p$, the position will be updated according to the following formula:

$$x_{i,j}^{t+1} = x_{best,j}^t \tag{6}$$

where the jth parameter of the fittest solution at iteration t is denoted by $x_{best,j}^t$; and the monarch butterflies' updated position is indicated by $x_{i,j}^{t+1}$.

Contrariwise, if the random number is greater than the ratio of migration $(rand > p)$, the position update proceeds according to the following formula:

$$x_{i,j}^{t+1} = x_{r_3,j}^t \tag{7}$$

where $x_{r_3,j}^t$ indicates the jth element of a randomly selected solution from Sub-population 2 in the current iteration.

Furthermore, if the uniformly distributed random number is greater than the adjusting rate (butterfly adjusting rate, BAR) , the individual is updated based on the next equation:

$$x_{i,j}^{t+1} = x_{i,j}^{t+1} + \alpha \times (dx_j - 0.5) \tag{8}$$

where dx_j represents the walk step of the ith individual; dx is obtained by Lévy flight:

$$dx = Levy(x_i^t) \tag{9}$$

The scaling factor α is calculated as follows:

$$\alpha = S_{max}/t^2 \tag{10}$$

where the upper limit of the walk step that an individual can take at a time is represented by S_{max} and t points out the current iteration.

The parameter α is responsible for the right balance between intensification and diversification. If its value is larger, the exploration (diversification) is predominant; on the other hand, if the value of α is smaller, the search process is executed in favor of intensification (exploitation).

The pseudo-code of the basic MBO version is shown in Algorithm 1.

Algorithm 1 . Basic MBO pseudo-code.

```
Randomly initialize the population of $SP$ solutions (monarch butterflies)
Initialize the parameters: migration ratio ($p$), migration period ($peri$), adjusting rate ($BAR$),
and maximum step size ($S_{max}$)
Fitness evaluation
Set $t$, the iteration counter, to one, and define the maximum number of iterations ($MaxIter$)
while $t < MaxIter$ do
  Sort the solutions according to their fitness value
  Divide the whole population into two sub-populations ($SP1$ and $SP2$)
  for all $i = 1$ to $SP1$ (all individuals in Sub-population 1) do
    for all $j = 1$ to $D$ (all elements of the $i^{th}$ individual) do
      Generate $rand$ (a random number), and calculate the value of $r$ by using Equation (5)
      if $r \le p$ then
        Choose an individual from $SP1$, and create the $j^{th}$ element of the new solution by utilizing
        Equation (4)
      else
        Choose an individual from $SP2$, and create the $j^{th}$ element of the new solution by utilizing
        Equation (4)
      end if
    end for
  end for
  for all $i = 1$ to $SP2$ (all individuals in Sub-population 2) do
    for all $j = 1$ to $D$ (all elements of the $i^{th}$ individual) do
      Generate $rand$ (a random number), and calculate the value of $r$ by using Equation (5)
      if $r \le p$ then
        Create the $j^{th}$ element of the new solution by utilizing Equation (6)
      else
        Choose an individual from $SP2$, and create the $j^{th}$ element of the new solution by utilizing
        Equation (7)
        if $r > BAR$ then
          Apply Equation (8)
        end if
      end if
    end for
  end for
  Merge the two sub-populations into one population
  Evaluate the fitness of the new solutions
  $t = t + 1$
end while
Return the best solution
```

3.2. Hybridized Monarch Butterfly Optimization Algorithm

Recently, we developed improved and hybridized versions of MBO. All devised implementations showed better performance than the original MBO, and they were successfully applied to one practical problem [39] and also evaluated against global optimization benchmarks [38,40].

As also noted in our previous research that the major drawbacks of the basic MBO included the lack of exploration power and the inadequate balance (trade-off) between the intensification and diversification [38–40]. However, we have recently been conducting additional simulations with the original MBO and concluded that also the exploitation phase, as well as the balance between exploration and exploitation could be further improved.

For that reason, similar to our previous MBO's improvements, we incorporated the ABC exploration mechanism, as well as the ABC's control parameter, which adjusts the intensification [97]. Furthermore, with the goal of facilitating the exploitation phase of the original MBO, we adopted the search equation from the FA metaheuristics, which proved to be very efficient [49]. Based on the performed hybridization, we named the newly devised approach MBO-ABC firefly enhanced (MBO-ABCFE).

To incorporate all changes, MBO-ABCFE utilizes three more control parameters than the original MBO algorithm. The first parameter is the exhaustiveness parameter (*exh*). In every iteration, when a solution cannot be improved, its *trial* value is incremented. When a *trial* for a particular solution reaches the threshold value of *exh*, a new random solution is generated according to the following formula:

$$x_{i,j} = \phi \cdot (ub_j - lb_j) + lb_j, \tag{11}$$

where the upper and lower bound of the j^{th} element are denoted by ub_j and lb_j. ϕ is a random number from the uniform distribution.

The "exhausted" solution is then replaced with the newly generated random individual. In this way, the exploration capability of the original MBO is enhanced.

However, this approach can be dangerous. For example, if this exploration mechanism is triggered in late iterations of the algorithm's execution (with the assumption that the search has converged to the optimal region), then possibly good solutions may be replaced with random ones. To avoid this, we included the second control parameter, the discarding mechanism trigger (*dmt*). By utilizing the *dmt* parameter, we established the stop condition for ABC's exploration. In other words, if the current iteration number (t) is greater than the value of *dmt*, the ABC's exploration is not executed.

Moreover, it is also dangerous to perform "too aggressive" exploitation in early phases of algorithm's execution. In this way, if the algorithm has not converged to the optimal region, the search may be stuck in some of the suboptimal domains of the search space. To adjust the exploitation and to avoid the scenario of converging to suboptimal regions, we adopted one more parameter from the ABC metaheuristics, the modification rate (MR). This parameter is used within the search procedure of Sub-population 1, and it is applied only if the $\theta \leq MR$ condition is met. The value of θ is a randomly generated number between zero and one. We note that the best value for MR of 0.8 was established in previous research [48,97]. In our implementation, this value is hard coded and cannot be adjusted by the user.

Finally, the third control parameter, the firefly algorithm process (FAP), which is included in the proposed implementation, is used to control whether or not the FA's search equation will be triggered.

The exploitation equation for the FA search is formulated as follows [49]:

$$x_i^{t+1} = x_i^t + \beta_0 r^{-\gamma r_{i,j}^2}(x_j - x_i) + \alpha(\kappa - 0.5) \tag{12}$$

where the randomization parameter α, as well as parameters β_0 and γ represent basic FA search parameters. The notation κ denotes the random number between zero and one. According to the previous experiments, as well as the recommendations from the original FA paper [49], the values of α, γ, and β_0 were set to 0.5, 1.0, and 0.2, respectively, and they were hard coded, so they could not be changed by the user.

For more details on the firefly algorithm, refer to [49].

The parameter FAP is generated for each solutions' parameter, and its value is between zero and one. If its value is less than 0.5, the firefly search equation will be utilized, otherwise standard MBO's search equations will be used.

Finally, we note that the MR parameter and the FA search equations are only utilized in Sub-population 1 of the algorithm.

Taking into account all the presented details regarding the proposed MBO-ABCFE metaheuristics, the pseudo-code is given in Algorithm 2, while the flowchart diagram is depicted in Figure 1.

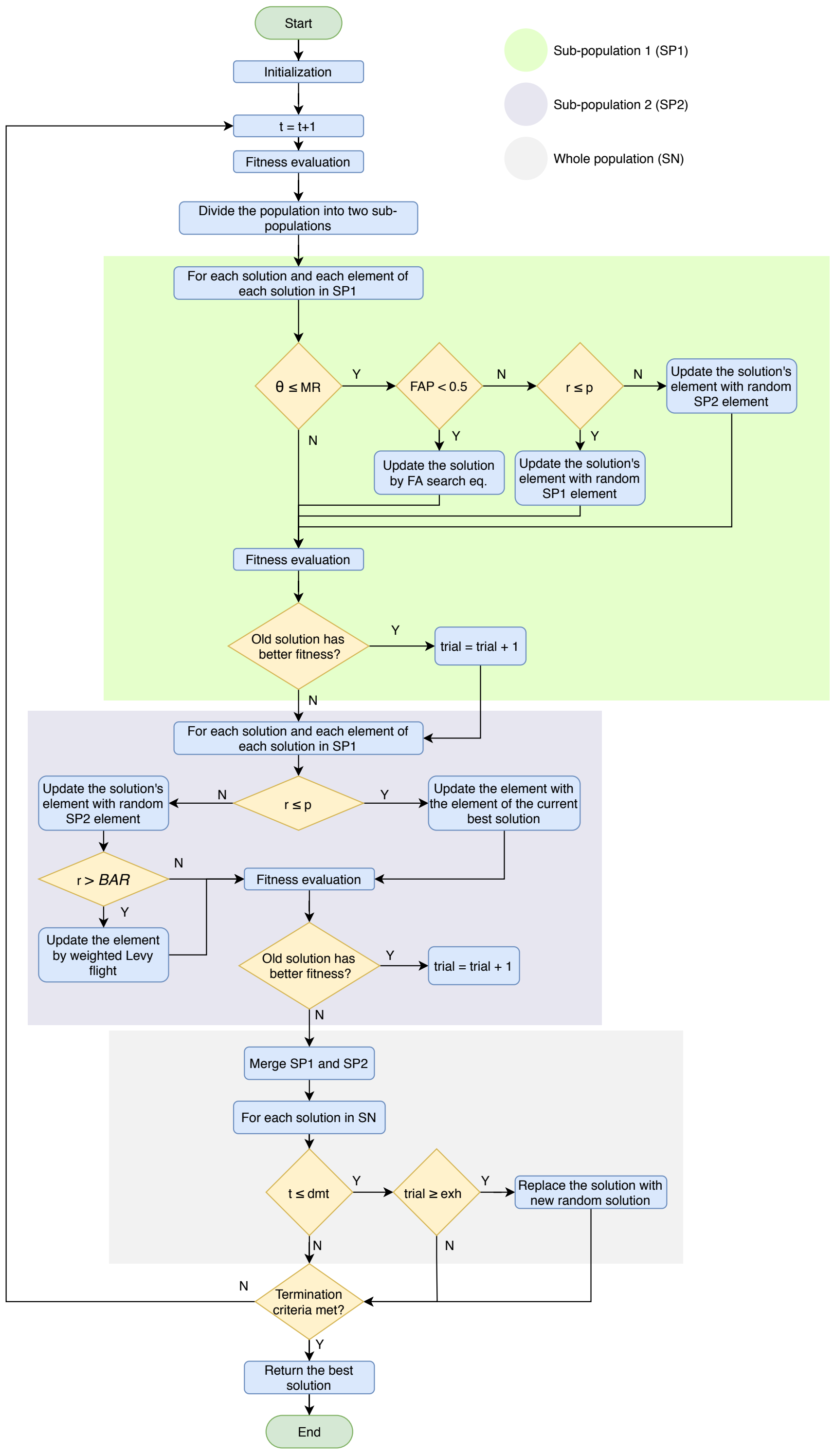

Start
Initialization
t = t+1
Fitness evaluation
Divide the population into two sub-populations
Sub-population 1 (SP1)
Sub-population 2 (SP2)
Whole population (SN)
For each solution and each element of each solution in SP1
θ ≤ MR
Y
N
FAP < 0.5
N
Y
r ≤ p
N
Y
Update the solution's element with random SP2 element
Update the solution by FA search eq.
Update the solution's element with random SP1 element
Fitness evaluation
Old solution has better fitness?
Y
N
trial = trial + 1
For each solution and each element of each solution in SP1
r ≤ p
N
Y
Update the solution's element with random SP2 element
Update the element with the element of the current best solution
r > BAR
N
Y
Update the element by weighted Levy flight
Fitness evaluation
Old solution has better fitness?
Y
N
trial = trial + 1
Merge SP1 and SP2
For each solution in SN
t ≤ dmt
Y
N
trial ≥ exh
Y
N
Replace the solution with new random solution
Termination criteria met?
N
Y
Return the best solution
End

Figure 1. Monarch butterfly optimization-ABC firefly enhanced (MBO-ABCFE) flowchart.

Algorithm 2 . MBO-ABCFE pseudocode.

```
Randomly initialize the population of $SN$ solutions (monarch butterflies); initialize the parameters: migration ratio ($p$), migration period ($peri$), adjusting rate ($BAR$), maximum step size ($S_{max}$), exhaustiveness parameter ($exh$), discarding mechanism trigger ($dmt$), and modification rate (MR);
Evaluate the fitness; set $t$, the iteration counter, to one and $trial$ to zero, and define the maximum number of iterations ($MaxIter$)
while $t < MaxIter$ do
  Sort the solutions according to their fitness value
  Divide the whole population into two sub-populations ($SP1$ and $SP2$)
  for all $i = 1$ to $SP1$ (all individuals in Sub-population 1) do
    for all $j = 1$ to $D$ (all elements of the $i^{th}$ individual) do
      Generate a random number between zero and one for $\theta$
      if $\theta \leq MR$ then
        if FAP<0.5 then
          Generate a new component by using Equation (12)
        else
          Generate $rand$ (a random number), and calculate the value of $r$ by using Equation (5)
          if $r \leq p$ then
            Choose a solution from $SP1$, and create the $j^{th}$ element of the new solution by Equation (4)
          else
            Choose a solution from $SP2$, and create the $j^{th}$ element of the new solution by Equation (4)
          end if
        end if
      end if
    end for
    Evaluate the fitness, and make a selection between the new and old solution based on the fitness value; if the old solution has better fitness, increment the parameter $trial$ by one
  end for
  for all $i = 1$ to $SP2$ (all individuals in Sub-population 2) do
    for all $j = 1$ to $D$ (all elements of the $i^{th}$ individual) do
      Generate $rand$ (a random number), and calculate the value of $r$ by using Equation (5)
      if $r \leq p$ then
        Create the $j^{th}$ element of the new solution by utilizing Equation (6)
      else
        Choose an individual from $SP2$, and create the $j^{th}$ element of the new solution by Equation (7)
        if $r > BAR$ then
          Apply Equation (8)
        end if
      end if
    end for
    If the old solution has better fitness, increment the parameter $trial$ by one
  end for
  Merge the two sub-populations into one population
  for all solutions in $SN$ do
    if $t \leq dmt$ then
      Discard the solutions for which the condition $trial \geq exh$ is satisfied, and replace them with randomly created solutions by utilizing Equation (11)
    end if
  end for
  Evaluate the fitness of the new solutions
  Adjust the value of the parameter $S_{max}$ by using Equation (10)
  $t = t + 1$
end while
Return the best solution
```

4. Simulation Setup

As noted in Section 1.3, two sections of this paper are devoted to experimental (practical) simulations. In this first experimental section, we show the control parameter setup of the proposed MBO-ABCFE and the dataset details utilized in unconstrained benchmark simulations and for the CNNs' neuroevolution (hyperparameter optimization) challenge.

4.1. Parameter Settings and Dataset for Unconstrained Simulations

The proposed hybridized swarm intelligence-based MBO-ABCFE algorithm was first applied to 25 unconstrained benchmark functions. In this way, we first wanted to establish the performance comparison and improvements of the original MBO approach on a wider set of functions.

Benchmark functions used in this paper were also used for testing purposes of the original MBO [35]. We also wanted to compare the proposed hybridized MBO with one other improved MBO implementation, the greedy strategy and self-adaptive crossover operator (GCMBO) [41], and we included this approach as well in the comparative analysis. In these simulations, the algorithm was executed in 50 independent runs.

Additionally, the proposed method was compared with other metaheuristics (hybrid ABC/MBO (HAM) [98], ABC, ACO, and PSO). In this way, we wanted to establish a better validation of the proposed method by comparing it with other state-of-the-art metaheuristics, the results of which were presented in the modern literature. In these simulations, the algorithm was tested on 12 benchmark functions over 100 runs.

For the purpose of making a fair comparison between the proposed MBO-ABCFE and the original MBO and GCMBO algorithms, as well as with other approaches, we performed the simulations under the same simulation condition and under the same dataset as in [35,41,98].

The population size (SN) was set to 50, the migration ratio (p) to 5/12, the migration period ($peri$) to 1.2, the adjusting rate (BAR) to 5/12, and the maximum step size (S_{max})to 1. The value of two sub-populations, Land 1 and Land 2, were set to 21 and 29, respectively. The newly introduced control parameters of the hybridized MBO-ABCFE algorithm were adjusted as follows: the exhaustiveness (exh) was initialized to the value of four the, and discarding mechanism trigger (dmt) was set to 33. As noted in Section 2, the value of the FAP parameter was randomly generated for each solution component. The values of ABC and FA specific parameters, the MR, α, γ, and β_0, were hard coded and set to 0.8, 0.5, 1.0, and 0.2, respectively.

Satisfying values of control parameters exh and dmt were determined by executing the algorithm many times for standard benchmarks. There was no universal rule for determining the optimal values of the metaheuristics' control parameters, and the "trial and error" approach was the only option.

However, in the case of the MBO-ABCFE algorithm, after many trials, we derived the expression for calculating the value of the exh parameter:

$$exh = round(\frac{maxIter}{N_p} \cdot 4), \tag{13}$$

where *round()* represents a simple function that rounds the argument to the closest integer value.

Moreover, in the case of MBO-ABCFE, we found that the value of the dmt parameter could be calculated by using the formula: $round(maxIter/1.5)$.

The values of ABC and FA specific parameters, the MR, α, γ, and β_0, were taken from the original papers [49,97]. In these articles, the authors suggested optimal values of these parameters that had also been determined empirically.

We also note that in order to obtain satisfying results, the control parameters' values should be adjusted for a particular problem or type of problem. For example, if with one set of the control parameters' values, the algorithm establishes a promising result when tackling Problem A, it does not necessarily mean that with the same parameter adjustments, satisfying results will be accomplished for Problem B as well.

The complete list of control parameters is displayed in Table 1.

Table 1. MBO-ABCFE parameters.

Parameter	Notation	Value
Population of the solutions	SN	50
Sub-population 1	$SP1$	21
Sub-population 2	$SP2$	29
Ratio of migration	p	5/12
Period of migration	$peri$	1.2
Max step size	S_{max}	1.0
Butterfly adjusting rate	BAR	5/12
Exhaustiveness	exh	4
Discarding mechanism trigger	dmt	33
Rate of modification	MR	0.8
Initial value for randomization parameter	α	0.5
Light absorption coefficient	γ	1.0
Attractiveness at r=0	β_0	0.2

In the original FA implementation [49], the trade-off between exploration and exploitation depended on the value of the α parameter, and it was dynamically adjusted during the run of an algorithm. In the proposed MBO-ABCFE metaheuristics, we used the same approach by utilizing the following equation [49,50,70,71,99]:

$$\alpha(t) = (1 - (1 - ((10^{-4}/9)^{1/t_{max}}))) \cdot \alpha(t-1), \tag{14}$$

where t_{max} denotes the maximum number of iterations in one run. At the early stages of a run, the value of α was greater (higher exploration power), and as the search progressed during iterations, the value of α decreased (the exploitation power increased, and the exploration decreased).

The formulations of benchmark functions (dataset) that were utilized in the simulations are given in Table 2.

Table 2. Benchmark function details.

ID	Function Name	Function Definition
F1	Ackley	$f(x) = -20e^{\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)} - -e^{\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)} + 20 + e^{(1)}$
F2	Alpine	$f(x) = \sum_{i=1}^{n} \lvert x_i \sin(x_i) + 0.1x_i \rvert$
F3	Brown	$f(x) = \sum_{i=1}^{n-1} (x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)}$
F4	Dixon and Price	$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} i\left(2x_i^2 - x_{i-1}\right)^2$
F5	Fletcher–Powell	$f(x) = 100\left\{[x_3 - 10\theta(x_1, x_2)]^2 + \left(\sqrt{x_1^2 + x_2^2} - 1\right)^2\right\} + x_3^2$
		where $2\pi\theta(x_1, x_2) = \begin{cases} \tan^{-1}\frac{x_2}{x_1} & \text{if } x_1 \geq 0 \\ \pi + \tan^{-1}\frac{x_2}{x_1} & \text{otherwise} \end{cases}$
F6	Griewank	$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$
F7	Holzman 2 function	$f(x) = \sum_{i=1}^{n} ix_i^4$
F8	Lévy 3 function	$f(x) = \sum_{i-1}^{5} i\cos((i-1)x_1 + i)\sum_{j=1}^{5} j\cos((j+1)x_2 + j)$
F9	Pathological function	$f(x) = \sum_{i=1}^{n-1}\left[0.5 + \frac{\sin^2\left(\sqrt{100x_i^2 + x_{i+1}^2}\right) - 0.5}{1 + 0.001\left(x_i^2 - 2x_i x_{i+1} + x_{i+1}^2\right)^2}\right]$

Table 2. *Cont.*

ID	Function Name	Function Definition
F10	Generalized Penalized Function 1	$f(x) = \frac{\pi}{n} \times \left\{ 10\sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 \left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2 \right\} + \sum_{i=1}^{n} u(x_i, a, k, m)$ where $y_i = 1 + \frac{1}{4}(x_i + 1)$, $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & \text{if } x_i > a \\ 0 & \text{if } -a \le x_i \le a \\ k(-x_i - a)^m & \text{if } x_i < -a \end{cases}$ $a = 10,\ k = 100,\ m = 4$
F11	Generalized Penalized Function 2	$f(x) = 0.1 \times \left\{ \sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 \left[1 + \sin^2(3\pi x_{i+1})\right] + (x_n - 1)^2 \left[1 + \sin^2(2\pi x_n)\right] \right\} + \sum_{i=1}^{n} u(x_i, a, k, m)$ where $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & \text{if } x_i > a \\ 0 & \text{if } -a \le x_i \le a \\ k(-x_i - a)^m & \text{if } x_i < -a \end{cases}$ $a = 5,\ k = 100,\ m = 4$
F12	Perm	$f(x) = \sum_{k=1}^{n} [\sum_{i=1}^{n} (i_k + \beta)((\frac{x_i}{i})^k - 1)]^2$
F13	Powel	$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$
F14	Quartic with noise	$f(x) = \sum_{i=1}^{n} ix_i^4 + random(0, 1)$
F15	Rastrigin	$f(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$
F16	Rosenbrock	$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
F17	Schwefel 2.26	$f(x) = \sum_{i=1}^{n} [-x_i \sin(\sqrt{\lvert x_i \rvert})]$
F18	Schwefel 1.2	$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$
F19	Schwefel 2.22	$f(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$
F20	Schwefel 2.21	$f(x) = \max\{\lvert x_i \rvert, 1 \le i \le n\}$
F21	Sphere	$f(x) = \sum_{i=1}^{n} x_i^2$
F22	Step	$f(x) = \sum_{i=1}^{n} (\lfloor x_i \rfloor + 0.5)^2$
F23	Sum function	$f(x) = \sum_{i=1}^{n} ix_i^2$
F24	Zakharov	$f(x) = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5ix_i)^2 + (\sum_{i=1}^{n} 0.5ix_i)^4$
F25	Wavy 1	$f(x) = \frac{1}{n} \sum_{i=1}^{n} 1 - \cos(10x_i) e^{-\frac{1}{2}x_i^2}$

4.2. Parameters' Settings and Simulation Setup for Cnns' Neuroevolution Simulations

Since the original MBO has not been previously tested on the CNN neuroevolution problem, with the aim of comparing the performance of our proposed MBO-ABCFE with the original algorithm, we also implemented and adapted the basic version for CNNs' simulations. The same values of the algorithms' control parameters that were used in unconstrained simulations were utilized in CNNs' neuroevolution experiments for both methods, MBO and MBO-ABCFE.

It was already mentioned in the previous subsection that with one set of control parameters' values, metaheuristics were not able to tackle every NP-hard challenge successfully, and parameter adjustments should be made for each particular problem. There is always a trade-off; for example, when tweaking some algorithm parameters, the performance could be enhanced for Problem A, but at the same time, the performance may be degraded for Problem B. That is the basic assumption behind testing the algorithm first on a wider set of benchmark functions, before applying it to the concrete NP-hard challenge.

In this case, the proposed MBO-ABCFE approach with the control parameters' settings that were shown in the previous subsection managed to obtain satisfying results on a wider set of tests. Guided by this, we assumed that with the same parameter adjustment, MBO-ABCFE would also be able to obtain promising results when tackling CNNs' neuroevolution. Since CNNs' neuroevolution is a very resource intensive and time consuming task, we would need a sophisticated hardware platform to perform simulations with different MBO-ABCFE control parameter settings for this NP-hard challenge.

Thus, we note that the proposed MBO-ABCFE potentially would be able to obtain even better results in tackling CNNs' neuroevolution with some other control parameters' values; however, we will investigate this in some of our future research from this domain.

In order to reduce the computation time, the values of all hyperparameters were discretized and defined within lower and upper bounds. To determine the set of values for each hyperparameter, we used the Gray code substring. The same strategy was utilized in [31].

Furthermore, since the approach presented in [31] was used for direct comparative analysis with our proposed MBO-ABCFE, we included the same hyperparameters in CNNs' neuroevolution and used the same experimental environment setup as in [31].

In the following two subsections, we describe in detail the calculation of the convolutional layer and dense (fully-connected) layer sets of parameters, as well as the setup of the general CNN hyperparameters.

4.2.1. Configuration of the Convolutional Layer

The convolutional layer (C_l) consists of a set of five hyperparameters:

$$C_l = \{n_c, n_f, f_s, a_c, p_s\} \tag{15}$$

The number of convolutional layers (n_c) is calculated as:

$$n_c = 1 + q \tag{16}$$

where $q = 0, 1, 2, 3$; hence, the set of the number of convolutional layers is $n_c = \{1, 2, 3, 4\}$.

The number of filters (n_f) is defined as follows:

$$n_f = 2^{q+1} \tag{17}$$

where $q = 0, 1, 2, 3, 4, 5, 6, 7$; consequently, the set is described as $n_f = \{2, 4, 8, 16, 32, 64, 128, 256\}$.

The filter size (f_s) is determined as:

$$f_s = 2 + q \tag{18}$$

where $q = 0, 1, 2, 3, 4, 5, 6, 7$; accordingly, the set of possible solutions of the filter size is defined as $f_s = \{2, 3, 4, 5, 6, 7, 8, 9\}$.

The type of the activation function (a_c) has two possible values, $a_c = q$, where q is zero or one. If the value is zero, the ReLU function is utilized; otherwise, if the value of q is equal to one, a linear activation function is selected.

Finally, the pooling layer size (p_s) is calculated as:

$$p_s = 2 + q \tag{19}$$

where $q = 0, 1, 2, 3, 4, 5, 6, 7$; accordingly the set of possible solutions of pooling layer size is defined as $p_s = \{2, 3, 4, 5, 6, 7, 8, 9\}$.

4.2.2. Configuration of the Fully-Connected Layer

The second category of hyperparameters is the fully-connected layer (FC_l), which contains six hyperparameters, and this set is defined as:

$$FC_l = \{fc_s, cp, n_u, a_f, wr, d\} \tag{20}$$

The number of fully-connected layers is denoted by fc_s, and its set is determined by the formula:

$$fc_s = 1 + q \tag{21}$$

where q is zero or one; accordingly, the set is $fc_s = \{1, 2\}$.

The connectivity pattern (cp) has three possible values, 0, 1, or 2. If the value is zero, rnn is used; if the value is one, $lstm$ is employed; in case the value is two, the dense layer is employed.

The number of hidden units is defined as:

$$n_u = 2^{3+q} \tag{22}$$

where n_u represents the hidden unit number and parameter q takes values between zero and seven, which results in a set $n_u = \{8, 16, 32, 64, 128, 256, 512, 1024\}$.

The type of activation function in the final layers (a_f) has two possible values, $a_c = q$, where q is zero or one. If the value is zero, the ReLU function is utilized; otherwise, if the value is equal to one, a linear activation function is selected.

Weight regularization (wr) is defined as:

$$wr = q \tag{23}$$

if q is equal to zero, no regularization technique used; if it is one, $L1$ regularization is applied; if q is two, $L2$ regularization is employed; and if it has a value of three, $L1L2$ is utilized.

In the case of the dropout (d) hyperparameter, the following formula is used: $d = q/2$. Further, we implemented two options: if the value of q is set to zero, the dropout method is not utilized, and if the value of q is one, the dropout is applied in the fully-connected layer with the dropout rate of 0.5, which is hard coded.

4.2.3. Configuration of the General Hyperparameters

The set of general hyperparameters contains the batch size, learning rule, and learning rate, which can be described as follows:

$$G_h = \{b_s, lr, \alpha\} \tag{24}$$

The size of the mini-batches are determined by the formula:

$$b_n = 25 \cdot 2^q \tag{25}$$

where $q = 0, 1, 2, 3.$; as a result, the elements of the batch size set are $b_s = \{25, 50, 100, 200\}$

The selection of the learning rule is made between eight different optimizers; it is defined as:

$$lr = q \tag{26}$$

where lr indicates the optimizer and q can have a value between zero and seven; if $q = 0$, sgd is used; if $q = 1$, momentum is used; if $q = 2$, Nesterov is used; if $q = 3$, adagard is used; if $q = 4$, adamax is used; if $q = 5$, Adam is used; if $q = 6$, adadelta is used; if $q = 7$, rmsprop is used.

In the case of the learning rate (α), we defined eight different rates. The set of learning rates is defined as $\alpha = \{1 \cdot 10^{-5}, 5 \cdot 10^{-5}, 1 \cdot 10^{-4}, 5 \cdot 10^{-4}, 1 \cdot 10^{-3}, 5 \cdot 10^{-3}, 1 \cdot 10^{-2}, 5 \cdot 10^{-2}\}$.

4.2.4. Benchmark Dataset

The well-known MNIST [100] benchmark dataset was used for the evaluation purposes of the proposed MBO-ABCFE and original MBO. The MNIST database is an extensive database that contains grayscale images of handwritten digits ranging from zero to nine. The database includes 70,000 labeled images with a size of 28 × 28 pixels. The MNIST database is a subset of an even larger dataset of the NIST Special Database. Initially, the images in the NIST dataset had a size of 20 × 20 pixels, while preserving their aspect ratio, by using anti-aliasing technique, the images were centered in a 28 $\times$ 28 pixel image. In our simulation, we split the dataset into training, validation, and test sets. For training purposes, fifty-thousands samples were used, while for validation and testing, 10,000 images. Examples of the digit images are shown in Figure 2.

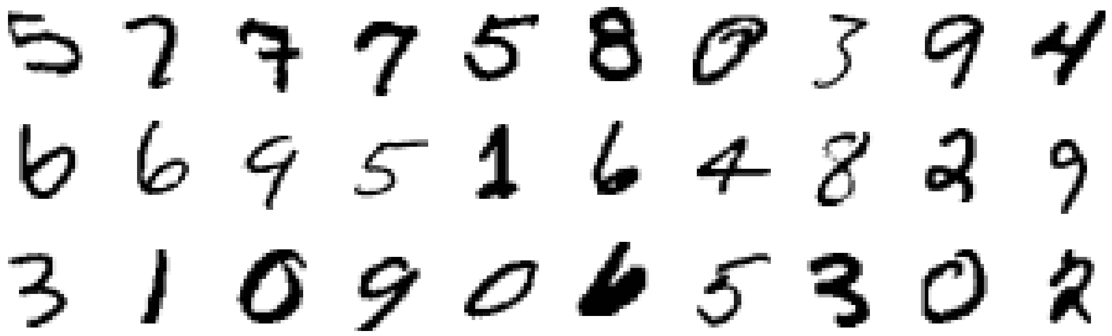


Figure 2. Example images of the MNIST database.

5. Experimental Results and Discussion

In this section, first we present the result of the proposed MBO-ABCFE algorithm on unconstrained benchmark function experiments and the comparison with other metaheuristics. Afterward, the CNN design's experimental results are presented.

Basic MBO and the proposed hybridized MBO-ABCFE were implemented and developed in Java Development Kit 11 (JDK 11) in the IntelliJ IDE (Integrated Development Environment). To evaluate the proposed method, the CNN framework was implemented in a deep learning programming library, the Deeplearning4j library. The simulation tests were performed on a 6 x NVIDIA GTX 1080 graphical processing unit (GPU) and machine with Intel® CoreTM i7-8700K CPU, 32GB RAM, and Windows 10 OS.

Since the CNNs' neuroevolution challenge is very resource intensive, to speed up the computations, we used $CUDA^{TM}$ technology and executed our code on six GPUs in parallel, instead of the CPU.

5.1. Experiments on Unconstrained Benchmark Functions

The unconstrained benchmark function experiment was conducted to test the performance of the proposed MBO-ABCFE algorithm on 25 standard benchmark problems with 20 and 40 dimensions. The obtained results were first compared to the original MBO [35] and one other variant of the MBO, the GCMBO [41] algorithm. Furthermore, the proposed method was compared with hybrid ABC/MBO (HAM), ABC, ACO, and PSO state-of-the-art approaches on 12 test functions with 20 dimensions.

For more details about the benchmark functions, please refer to Table 2.

Due to the stochastic behavior and random nature of swarm intelligence algorithms, their performance cannot be judged based on a single run. Thus, the experimental results were calculated on average for 50 independent runs, and the additional comparison with four metaheuristics was calculated on average for 100 runs.

The obtained experimental results of the objective function and the comparison of statistical results in the comparative analysis with basic MBO and GCMBO on 20-dimensional test instances are shown in Table 3, while the results on 40-dimensional benchmarks are presented in Table 4. The comparison with four other state-of-the-art metaheuristics is given in the Table 5. In both tables, the best solutions are shown in boldface. As can be seen from the presented tables, we took the best, mean, standard deviation, and worst metrics for performance evaluations, while in the third comparative analysis, we present the best and mean results for each of the comparative algorithms.

The results of the basic MBO and GCMBO were retrieved from [41]. In this paper, the authors used the number of fitness function evaluations (FFEs) as the termination condition (8000 FFEs for 20-dimensional and 16,000 FFEs for 40-dimensional tests) with 50 solutions in the population. Moreover, the authors executed 50 independent runs for each benchmark. We also show the averaged results in 50 independent runs of our proposed MBO-ABCFE in this comparative analysis (Tables 3 and 4).

Simulation results for HAM, ABC, ACO, and PSO were taken from [98]. Here, we used the number of iterations (generations), which was set to 50, as the termination condition. We also show results averaged over 100 independent runs of the algorithm's execution. To make a fair comparative analysis between HAM, ABC, ACO, PSO, and our proposed MBO-ABCFE in this comparative analysis (Table 5), we also show the average results over 100 runs. Here, we note that we could not perform a comparison for all 25 benchmarks as in the case of the comparative analysis with the basic MBO and GCMBO, since not all testing results were available in [98].

In all performed comparative analyses (Tables 3–5), we utilized 50 iterations as the termination condition, as in [98].

Table 3. Scientific results of MBO, GCMBO, and MBO-ABCFE on 20-dimensional problems averaged over 50 runs.

ID	Global Minimum	MBO				GCMBO				MBO-ABCFE			
		Best	*Mean*	*StdDev*	*Worst*	*Best*	*Mean*	*StdDev*	*Worst*	*Best*	*Mean*	*StdDev*	*Worst*
F1	0	0.01	11.43	6.43	18.83	6.7×10^{-16}	4.24	4.62	14.57	**0.00**	**2.21**	**2.96**	**11.47**
F2	0	7.7×10^{-5}	7.51	10.37	39.49	**2.2×10^{-16}**	0.03	0.20	1.44	**0.00**	**0.01**	**0.15**	**1.06**
F3	0	1.6×10^{-5}	48.58	102.82	494.07	2.2×10^{-16}	0.66	3.06	21.46	**0.00**	**0.32**	**2.28**	**8.39**
F4	0	155.44	1.2×10^{8}	1.0×10^{8}	3.2×10^{8}	0.18	1.0×10^{7}	1.0×10^{7}	4.4×10^{7}	**0.07**	**1.59×10^{3}**	**2.87×10^{3}**	**8.29×10^{3}**
F5	0	1.2×10^{5}	3.0×10^{5}	1.1×10^{5}	6.6×10^{5}	2.5×10^{4}	1.2×10^{5}	5.3×10^{4}	2.6×10^{5}	**1.19×10^{3}**	**2.81×10^{3}**	**4.22×10^{3}**	**9.85×10^{3}**
F6	0	1.01	93.72	94.71	342.68	**1.00**	**20.74**	**21.69**	**83.11**	**1.00**	28.52	30.74	97.51
F7	0	2.3×10^{-4}	6.2×10^{4}	5.9×10^{4}	1.9×10^{5}	1.2×10^{-6}	1.9×10^{3}	3.6×10^{3}	1.8×10^{4}	**7.21×10^{-10}**	**382.21**	**425.32**	**954.22**
F8	0	2.5×10^{-7}	20.58	33.27	113.39	2.2×10^{-16}	2.11	5.00	20.12	**0.00**	**1.18**	**3.85**	**9.53**
F9	0	0.04	1.62	0.94	3.51	2.2×10^{-16}	0.79	0.77	2.98	**0.00**	**0.47**	**0.53**	**1.52**
F10	0	2.5×10^{-9}	3.2×10^{7}	6.5×10^{7}	2.8×10^{8}	9.8×10^{-12}	3.1×10^{5}	1.2×10^{6}	7.5×10^{6}	**5.28×10^{-15}**	**5.97×10^{3}**	**1.84×10^{4}**	**5.74×10^{4}**
F11	0	1.6×10^{-7}	7.9×10^{7}	1.3×10^{8}	4.6×10^{8}	2.2×10^{-16}	1.1×10^{6}	5.6×10^{6}	3.9×10^{7}	**0.00**	**3.1×10^{4}**	**2.8×10^{4}**	**9.7×10^{5}**
F12	0	2.3×10^{46}	5.9×10^{50}	1.1×10^{51}	6.0×10^{51}	5.7×10^{46}	1.5×10^{51}	2.0×10^{51}	6.0×10^{51}	**5.87×10^{28}**	**4.58×10^{32}**	**8.54×10^{32}**	**9.57×10^{33}**
F13	0	0.04	2.1×10^{3}	1.9×10^{3}	6.4×10^{3}	2.2×10^{-16}	435.79	562.68	2.8×10^{3}	**0.00**	**211.20**	**429.32**	**895.25**
F14	0	1.2×10^{-14}	36.86	35.96	134.22	2.2×10^{-16}	0.09	0.31	1.90	**0.00**	**0.03**	**0.15**	**1.07**
F15	0	4.7×10^{-6}	41.18	36.19	119.22	2.2×10^{-16}	7.71	8.49	28.26	**0.00**	**3.52**	**5.69**	**18.28**
F16	0	1.9×10^{-3}	969.30	1.7×10^{3}	7.8×10^{3}	2.2×10^{-16}	69.97	116.50	414.22	**0.00**	**45.37**	**87.31**	**311.20**
F17	0	1.99	3.0×10^{3}	1.9×10^{3}	5.6×10^{3}	**2.5×10^{-4}**	**1.0×10^{3}**	**1.0×10^{3}**	**3.7×10^{3}**	4.68×10^{-3}	2.31×10^{5}	5.41×10^{5}	3.25×10^{6}
F18	0	1.4×10^{-4}	2.5×10^{4}	1.5×10^{4}	5.5×10^{4}	0.05	1.1×10^{4}	8.5×10^{3}	3.3×10^{4}	**0.02**	**8.32×10^{3}**	**8.42×10^{3}**	**9.35×10^{3}**
F19	0	8.87×10^{-4}	20.08	22.86	75.65	2.20×10^{-16}	2.46	4.61	20.45	**0.00**	**1.27**	**3.81**	**18.65**
F20	0	0.76	30.53	20.87	77.54	0.13	26.63	18.90	61.74	**0.08**	**15.21**	**16.78**	**58.32**
F21	0	9.72×10^{-7}	20.49	39.06	147.90	2.20×10^{-16}	0.12	0.43	2.55	**0.00**	**0.09**	**0.18**	**1.72**
F22	0	**1.00**	22.74	35.69	125.00	**1.00**	1.72	2.94	20.00	**1.00**	**0.89**	**1.31**	17.21
F23	0	4.30	1.81×10^{3}	1.39×10^{3}	4.32×10^{3}	2.20×10^{-16}	120.90	165.80	816.90	**0.00**	**79.25**	**112.52**	**297.52**
F24	0	6.36×10^{-3}	328.80	223.90	831.40	2.40×10^{-5}	137.20	128.70	452.90	**1.38×10^{-8}**	**52.68**	**85.36**	**584.32**
F25	0	0.04	320.00	281.20	1.07×10^{3}	3.43×10^{-3}	108.60	142.30	578.80	**1.68×10^{-7}**	**54.33**	**118.32**	**225.36**

Table 4. Scientific results of MBO, GCMBO, and MBO-ABCFE on 40-dimensional problems averaged over 50 runs.

ID	Global Minimum	MBO				GCMBO				MBO-ABCFE			
		Best	*Mean*	*StdDev*	*Worst*	*Best*	*Mean*	*StdDev*	*Worst*	*Best*	*Mean*	*StdDev*	*Worst*
F1	0	**0.01**	14.68	5.63	19.10	6.94×10^{-5}	8.19	5.25	16.33	8.52×10^{-8}	**3.21**	**4.92**	**12.28**
F2	0	4.18×10^{-3}	37.36	35.50	103.00	**2.20×10^{-16}**	1.76	4.53	22.85	0.00	**0.18**	**2.17**	**18.71**
F3	0	0.13	6.2×10^{8}	2.4×10^{9}	1.35×10^{10}	7.30×10^{-7}	23.87	53.71	294.60	**1.54×10^{-15}**	**19.38**	**24.71**	**117.32**
F4	0	1.58	1.0×10^{9}	8.5×10^{8}	2.65×10^{9}	1.32×10^{3}	1.2×10^{8}	1.3×10^{8}	5.22×10^{8}	**0.36**	**8.39×10^{5}**	**1.21×10^{8}**	**6.17×10^{6}**
F5	0	1.45×10^{6}	2.7×10^{6}	7.2×10^{5}	4.71×10^{6}	4.14×10^{5}	8.4×10^{5}	2.2×10^{5}	1.28×10^{6}	**0.83**	**7.12×10^{3}**	**3.56×10^{4}**	**2.83×10^{5}**
F6	0	**1.00**	341.06	287.07	827.20	**1.00**	**65.66**	**71.85**	**262.60**	**1.00**	73.16	132.58	285.36
F7	0	3.86	5.1×10^{5}	3.9×10^{5}	1.24×10^{6}	2.20×10^{-16}	4.7×10^{4}	5.1×10^{4}	2.07×10^{5}	**0.00**	**0.39**	**0.97**	**5.69×10^{3}**
F8	0	2.15×10^{-7}	103.80	118.35	392.30	3.28×10^{-9}	12.24	18.15	60.17	**2.65×10^{13}**	**0.69**	**8.63**	**58.24**
F9	0	0.08	5.73	3.08	10.69	**0.02**	**1.97**	**1.67**	**5.79**	0.06	2.05	2.98	7.51
F10	0	3.00×10^{-8}	2.9×10^{8}	3.3×10^{8}	1.04×10^{9}	3.97×10^{-11}	1.7×10^{6}	4.0×10^{6}	2.10×10^{7}	**8.67×10^{-15}**	**8.37×10^{4}**	**4.72×10^{4}**	**5.31×10^{6}**
F11	0	3.69×10^{-7}	5.3×10^{8}	6.3×10^{8}	1.89×10^{9}	2.01×10^{-8}	1.1×10^{7}	3.0×10^{7}	1.46×10^{8}	**9.75×10^{-11}**	**4.38×10^{5}**	**6.33×10^{5}**	**1.57×10^{7}**
F12	0	1.70×10^{124}	1.4×10^{127}	3.4×10^{127}	1.34×10^{128}	7.52×10^{116}	1.1×10^{127}	2.3×10^{127}	1.34×10^{128}	**8.15×10^{25}**	**5.74×10^{28}**	**8.91×10^{28}**	**1.35×10^{31}**
F13	0	3.04	7.3×10^{3}	7.9×10^{3}	2.56×10^{4}	2.20×10^{-16}	1.7×10^{3}	2.1×10^{3}	8.23×10^{3}	**0.00**	**0.09**	**1.12×10^{3}**	**1.97×10^{3}**
F14	0	3.81×10^{-9}	272.95	231.23	641.30	2.20×10^{-16}	11.60	19.85	97.64	**0.00**	**8.11**	**14.92**	**32.15**
F15	0	3.66×10^{-3}	162.21	113.05	317.92	2.20×10^{-16}	32.20	25.02	99.73	**0.00**	**71.98**	**115.32**	**458.21**
F16	0	0.02	7.3×10^{3}	8.0×10^{3}	2.3×10^{4}	2.20×10^{-16}	298.88	449.49	1.4×10^{3}	**0.00**	**115.21**	**389.27**	**536.25**
F17	0	0.21	8.4×10^{3}	3.8×10^{3}	1.4×10^{4}	**5.09×10^{-4}**	**3.9×10^{3}**	**2.5×10^{3}**	**8.1×10^{3}**	0.05	0.81	5.32×10^{3}	1.58×10^{4}
F18	0	18.87	1.2×10^{5}	6.9×10^{4}	3.0×10^{5}	0.35	4.9×10^{4}	3.5×10^{4}	1.2×10^{5}	**0.01**	**1.2×10^{3}**	**1.13×10^{3}**	**3.91×10^{4}**
F19	0	5.77×10^{-3}	88.24	67.55	177.00	2.20×10^{-16}	13.94	20.79	71.94	**0.00**	**8.21**	**15.32**	**38.27**
F20	0	0.48	40.07	27.64	91.91	0.36	36.74	25.71	83.00	**0.07**	**21.92**	**19.24**	**48.25**
F21	0	2.71×10^{-5}	123.60	115.50	307.50	2.20×10^{-16}	7.96	16.57	63.77	**0.00**	**4.85**	**7.36**	**41.28**
F22	0	**1.00**	117.90	120.80	326.00	**1.00**	13.02	19.16	80.00	**1.00**	**8.52**	**17.23**	**56.34**
F23	0	1.18	1.27×10^{4}	7.61×10^{3}	2.24×10^{4}	3.38×10^{-3}	2.26×10^{3}	1.84×10^{3}	7.56×10^{3}	**1.15×10^{-7}**	**23.54**	**45.89**	**98.54**
F24	0	0.03	2.45×10^{5}	1.45×10^{6}	1.06×10^{7}	2.31	600.70	314.90	1.14×10^{3}	**0.01**	**81.37**	**96.24**	**248.54**
F25	0	15.61	1.53×10^{3}	981.90	3.32×10^{3}	8.33×10^{-3}	507.40	368.30	1.48×10^{3}	**1.25×10^{-5}**	**123.11**	**251.25**	**652.54**

Table 5. Scientific results of HAM, ABC, ACO, PSO, and MBO-ABCFE on 20-dimensional problems averaged over 100 runs.

ID	Global Minimum	HAM		ABC		ACO		PSO		MBO-ABCFE	
		Best	*Mean*	*Best*	*Mean*	*Best*	*Mean*	*Best*	*Mean*	*Best*	*Mean*
F1	0	**2.46×10^{-2}**	**5.08×10^{-2}**	8.44	1.42×10^{1}	1.16×10^{1}	1.50×10^{1}	1.36×10^{1}	1.61×10^{1}	0.00	0.72
F6	0	**9.87×10^{-5}**	**2.15×10^{-3}**	1.43	1.33×10^{1}	4.49	1.34×10^{1}	4.31×10^{1}	8.21×10^{1}	1.00	7.35
F10	0	5.64×10^{-2}	**1.19×10^{-1}**	1.66×10^{-1}	1.72×10^{4}	**1.57×10^{-32}**	8.26×10^{7}	1.53×10^{5}	7.23×10^{6}	3.12×10^{-25}	4.32×10^{2}
F11	0	4.94×10^{-1}	7.50×10^{-1}	**4.13×10^{-1}**	1.54×10^{5}	1.35×10^{-32}	1.60×10^{8}	4.01×10^{6}	2.73×10^{7}	**0.00**	2.83
F14	0	4.69	5.79	6.85	1.10×10^{1}	1.22×10^{-1}	1.12	7.58×10^{-1}	3.39	**0.00**	**0.03**
F15	0	3.99	4.46×10^{1}	3.09×10^{1}	6.72×10^{1}	1.02×10^{2}	1.59×10^{2}	1.29×10^{2}	1.65×10^{2}	**0.00**	**3.52**
F16	0	7.54×10^{2}	1.79×10^{5}	7.54×10^{2}	1.79×10^{5}	8.56×10^{2}	1.91×10^{3}	2.22×10^{2}	6.13×10^{2}	**0.00**	**35.21**
F18	0	3.93	**4.21×10^{2}**	8.92×10^{3}	1.62×10^{4}	2.78×10^{3}	7.95×10^{3}	3.17×10^{3}	8.54×10^{3}	**0.02**	2.19×10^{2}
F19	0	7.66×10^{-2}	1.38×10^{-1}	9.01×10^{-1}	3.21	1.49×10^{1}	4.98×10^{1}	2.52×10^{1}	4.76×10^{1}	**0.00**	**1.27**
F20	0	2.67×10^{-2}	**5.58×10^{-2}**	4.12×10^{1}	6.07×10^{1}	1.91×10^{1}	3.97×10^{1}	3.20×10^{1}	5.30×10^{1}	**0.01**	9.21
F21	0	8.07×10^{-4}	2.26×10^{-3}	7.13×10^{-2}	3.66	1.33×10^{1}	3.32×10^{1}	1.46×10^{1}	2.45×10^{1}	**0.00**	**0.09**
F22	0	5.07	5.16	8.03×10^{1}	1.41×10^{3}	5.33×10^{2}	1.96×10^{3}	5.39×10^{3}	9.09×10^{3}	**0.79**	**0.98**

The obtained simulation results demonstrated that the proposed hybrid MBO-ABCFE improved the performance of the original MBO algorithm since MBO-ABCFE outperformed MBO on all 25 benchmark functions for both 20- and 40-dimensional problems. It was also observed from the results that MBO-ABCFE was significantly better than GCMBO. In the case of the 20-dimensional problem, MBO-ABCFE outperformed GCMBO in 23 out of 25 benchmark functions. In the case of a 40-dimensional problem, MBO-ABCFE established better performance indicators than GCMBO in 22 out of 25 test instances.

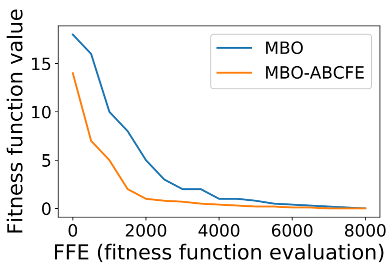
There were two test problems where all three algorithms showed similar performance. The GCMBO outperformed Griewank and Schwefel 2.26 functions on 20 dimensions, and Griewank, Pathological, and Schwefel 2.26 functions on 40 dimensions. For the Griewank function, on 20 dimensions, the best result was identical for MBO-ABCFE and GCMBO, and for the Schwefel 2.26 function, the best result was identical for all three metaheuristics. For Griewank and Schwefel 2.26 functions, on 40 dimensions, the best results were identical for MBO, GCMBO, as well as MBO-ABCFE. Finally, it is important to remark that there was no problem where MBO outperformed MBO-ABCFE.

The results presented in Table 5 proved the effectiveness of MBO-ABCFE. The algorithm outperformed all other compared metaheuristic algorithm for both best and mean indicators, in seven out of 12 test functions. In test instances with ID 11, 18, and 20, MBO-ABCFA was not able to establish the best values for the mean indicator; however, it showed the best performance compared to all other competitors for the best indicator. Only in the case of Griewank and Generalized Penalized Function 1 (benchmarks with ID 6 and 10, respectively), HAM and ACO obtained better results than our proposed MBO-ABCFE.

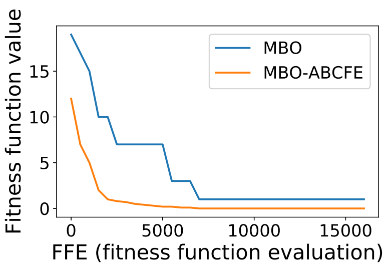
It should be noted that when the results were averaged over 100 runs (Table 5), for some test instances, MBO-ABCFE established better values for the best and/or mean indicators than in simulations with 50 runs (Table 3). This meant that when tested with more runs, MBO-ABCFE performed even better.

We performed an analysis of the obtained results in simulations with 100 runs and noticed that in tests instances when the global optimum was reached, in many runs, the algorithm managed to converge to the optimal solution, and that in turn also improved the mean values. The implications of this were that MBO-ABCFE had a strong exploration in early iterations (ABC's diversification mechanism), when it converged to the optimal domain of the search space, while in later iterations, due to FA's exploitation ability, MBO-ABCFE performed a fine-tuned search in and around the promising region of the search space.

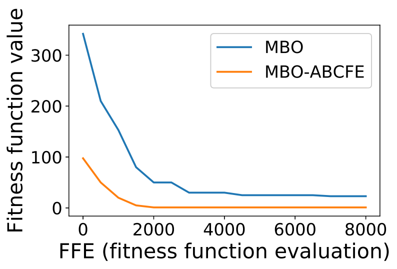
Finally, in order to visually represent the improvements of the proposed MBO-ABCFE over the original MBO, we tested MBO-ABCFE with 8000 and 16,000 FFEs for 20-dimensional and 40-dimensional benchmark instances, respectively, as was performed with the basic MBO in [41]. The convergence speed graphs for some test instances are shown in Figure 3. The convergence graphs were generated by using the same number of FFEs as in [41].



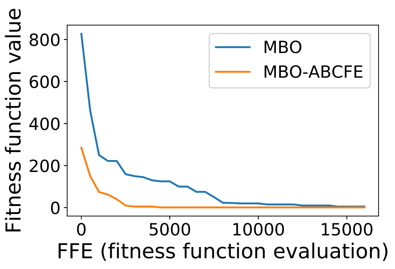
(**a**) F01 function with 8000 FFEs on 20D



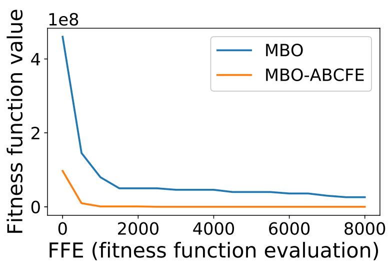
(**b**) F01 function with 16,000 FFEs on 40D



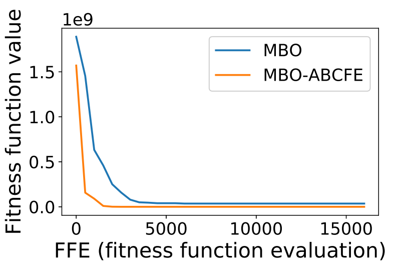
(**c**) F06 function with 8000 FFEs on 20D



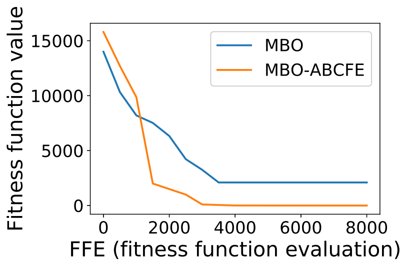
(**d**) F06 function with 16,000 FFEs on 40D



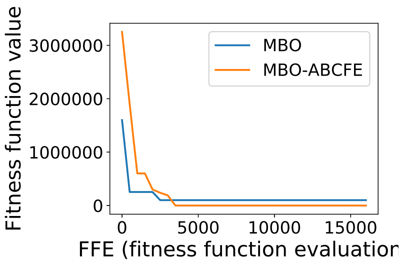
(**e**) F11 function with 8000 FFEs on 20D



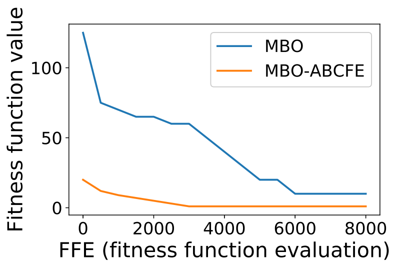
(**f**) F11 function with 16,000 FFEs on 40D



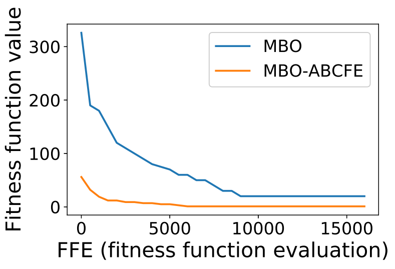
(**g**) F17 function with 8000 FFEs on 20D



(**h**) F17 function with 16,000 FFEs on 40D



(**i**) F22 function with 8000 FFEs on 20D



(**j**) F22 function with 16,000 FFEs on 40D

Figure 3. Convergence graphs. (FFE, fitness function evaluation): (**a**) F01 function with 8000 FFEs on 20D; (**b**) F01 function with 16,000 FFEs on 40D; (**c**) F06 function with 8000 FFEs on 20D; (**d**) F06 function with 16,000 FFEs on 40D; (**e**) F11 function with 8000 FFEs on 20D; (**f**) F11 function with 16,000 FFEs on 40D; (**g**) F17 function with 8000 FFEs on 20D; (**h**) F17 function with 16,000 FFEs on 40D; (**i**) F22 function with 8000 FFEs on 20D; (**j**) F22 function with 16,000 FFEs on 40D.

As a general conclusion for the unconstrained tests, it could be stated that the proposed MBO-ABCFE significantly improved the performance of the original MBO and also established better performance metrics than the improved GCMBO, as well as than the other state-of-the-art metaheuristic approaches, the results of which were retrieved from the most current computer science literature sources. Consequently, MBO-ABCFE is promising for real-world applications.

5.2. Convolutional Neural Network Design Experiment

In the second part of the simulation experiments, we optimized the CNN hyperparameter values by employing the original MBO algorithm and the proposed MBO-ABCFE metaheuristics. The MBO-ABCFE framework that was develop for tackling CNNs neuroevolution was named MBO-ABCFE-CNN.

The parameters that were subject to optimization were divided into three categories. The first category included the hyperparameters of the convolutional layer; the second category included the hyperparameters of the fully-connected (dense) layer; and the third category consisted of general CNN hyperparameters.

In Table 6, we summarize the hyperparameter values for each category. For more details regarding the hyperparameters' setup, please refer to Section 4.2.

Table 6. Convolutional neural network hyperparameters.

Category	**Hyperparameter**	**Notation**
Convolutional layer	Number of convolutional layers	n_c
	Number of filters	n_f
	Filter size	f_s
	Activation function	a_c
	Pooling layer size	p_s
Fully-connected layer	Number of fully-connected layers	fc_s
	Connectivity pattern	cp
	Number of units	n_u
	Activation function	a_f
	Weight regularization	wr
	Dropout	d
General hyperparameters	Batch size	b_s
	Learning rule	lr
	Learning rate	α

In order to reduce the computation time, the values of all hyperparameters were discretized and defined within lower and upper bounds, and their detailed formulation is described in Section 4.

The population size (SN) in the optimization algorithm was set to 50 solutions, which corresponded to the solution of the CNN structure (S), and it is defined as follows:

$$SN = \{S_1, S_2, \cdots, S_{50}\} \tag{27}$$

where each CNN structure S consists of the three hyperparameter categories.

Each CNN structure is encoded as the following set:

$$S = \{C_l, FC_l, G_h\} \tag{28}$$

where C_l, FC_l, and G_h denote sets of convolutional, fully-connected (dense), and general hyperparameters, respectively.

In the convolution layer category (C_l), each convolutional layer consisted of nested hyperparameters (number of filters, filter size, activation function, and pooling layer size).

Similarly, in the fully-connected hyperparameter category, each FC-layer further contained hyperparameters, such as connectivity pattern, number of hidden units, activation function, weight regularization, and dropout. The third category, the general hyperparameters, had only one level, with three hyperparameters: batch size, learning rule, and learning rate.

Each individual from the population (one CNN instance with specific hyperparameters) represented a data structure that contained all values (attributes) for each hyperparameter and for each CNN layer. In Figure 4, we show the proposed scheme for encoding the CNN structure.

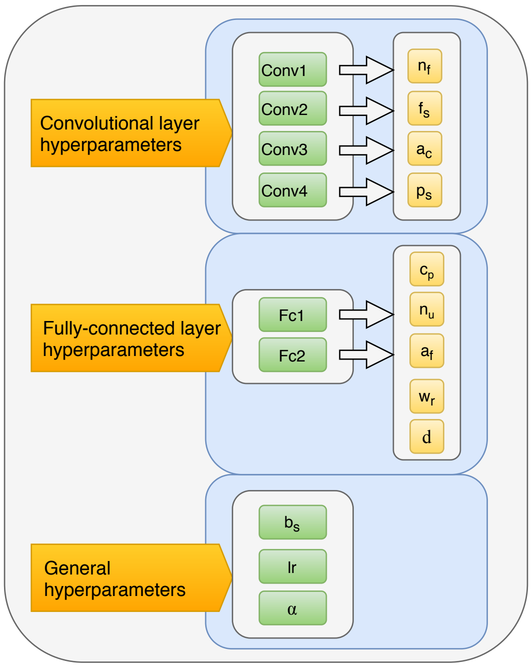


Figure 4. CNN structure encoding scheme.

In the initialization phase of the algorithm, first the initial population of CNN structures was generated randomly, according to the following equation:

$$S_{i,j} = min_j + rand \cdot (max_j - min_j) \tag{29}$$

where the jth parameter of the ith solution in the population is denoted by $S_{i,j}$, $rand$ is a random number between zero and one, and min_j, max_j are the lower and upper bounds of the jth parameter.

During the process of optimization, the algorithm searches for an optimal or near optimal solution, and after 100 iterations, the algorithm generates optimal and/or near optimal CNN structures. Similarly to [31], we implemented stop condition, and if there was no improvement, the algorithm stopped after 30 iterations. By using the stop condition, we avoided wasting expensive computational resources.

The reported CNN structure after each run was the solution with the best fitness value. The objective function that should be minimized was the classification error rate of the corresponding CNN architecture S. The fitness function was inversely proportional to the objective function, and it is defined as follows:

$$F(S) = \frac{1}{1 + |f(S)|} \tag{30}$$

where $F(S)$ denotes the fitness function of CNN structure S and $f(S)$ indicates the objective function of the corresponding structure.

We used the same values of the metaheuristic control parameters in the CNN hyperparameter optimization like in the unconstrained benchmark function experiment, and the summary of these parameters is presented in Table 1.

In order to speed up the computation, the structure was trained in five epochs with 50% of the data. After completion, the 20 best CNN architectures were fully trained in 30 epochs, and due to the stochastic behavior, the training process was repeated 20 times for each structure. The statistical results of the 20 best solutions (CNN architectures) of the MBO algorithm and MBO-ABCFE algorithm are presented in Tables 7 and 8, respectively. The boxplots of the error rate distribution of best 20 solutions generated by MBO and MBO-ABCFE algorithm are depicted in Figures 5 and 6, respectively.

Table 7. Classification error rate (in %) of the 20 best solutions (CNN architectures) of the MBO algorithm.

No. of Structures	Mean	StdDev	Median	Minimum	Maximum
1	0.5015	0.0609	0.505	0.41	0.63
2	0.5200	0.0219	0.520	0.47	0.55
3	0.4995	0.0387	0.515	0.42	0.56
4	0.4845	0.0562	0.485	0.37	0.57
5	0.4330	0.0320	0.440	0.39	0.48
6	0.5110	0.0319	0.515	0.45	0.57
7	0.4035	0.0282	0.410	0.36	0.45
8	0.5340	0.0348	0.545	0.46	0.59
9	0.4520	0.0314	0.450	0.39	0.5
10	0.5470	0.0435	0.555	0.46	0.6
11	0.4535	0.0348	0.450	0.41	0.52
12	0.5080	0.0595	0.505	0.42	0.61
13	0.5085	0.0395	0.490	0.46	0.59
14	0.5685	0.0385	0.575	0.49	0.62
15	0.4650	0.0380	0.475	0.40	0.52
16	0.4965	0.0320	0.500	0.44	0.54
17	0.5570	0.0762	0.540	0.44	0.71
18	0.5765	0.0385	0.595	0.50	0.62
19	0.5510	0.0465	0.535	0.49	0.65
20	0.5185	0.0283	0.520	0.47	0.58

Table 8. Classification error rate (in %) of the 20 best solutions (CNN architectures) of the MBO-ABCFE algorithm.

No. of Structures	Mean	StdDev	Median	Minimum	Maximum
1	0.5250	0.0326	0.525	0.47	0.59
2	0.4775	0.0497	0.475	0.40	0.58
3	0.5215	0.0217	0.520	0.47	0.56
4	0.4490	0.0460	0.450	0.36	0.53
5	0.5030	0.0332	0.500	0.44	0.57
6	0.5155	0.0277	0.520	0.45	0.57
7	0.4925	0.0288	0.485	0.45	0.55
8	0.4745	0.0401	0.475	0.41	0.54
9	0.4355	0.0353	0.430	0.38	0.48
10	0.4385	0.0307	0.450	0.39	0.49
11	0.4520	0.0506	0.445	0.35	0.54
12	0.3650	0.0132	0.370	0.34	0.39
13	0.5055	0.0229	0.500	0.45	0.55
14	0.4680	0.0294	0.460	0.42	0.51
15	0.4600	0.0446	0.460	0.40	0.58
16	0.4960	0.0559	0.510	0.42	0.59
17	0.4800	0.0288	0.485	0.43	0.54
18	0.4855	0.0319	0.480	0.44	0.56
19	0.4120	0.0220	0.415	0.37	0.45
20	0.4400	0.0383	0.435	0.38	0.49

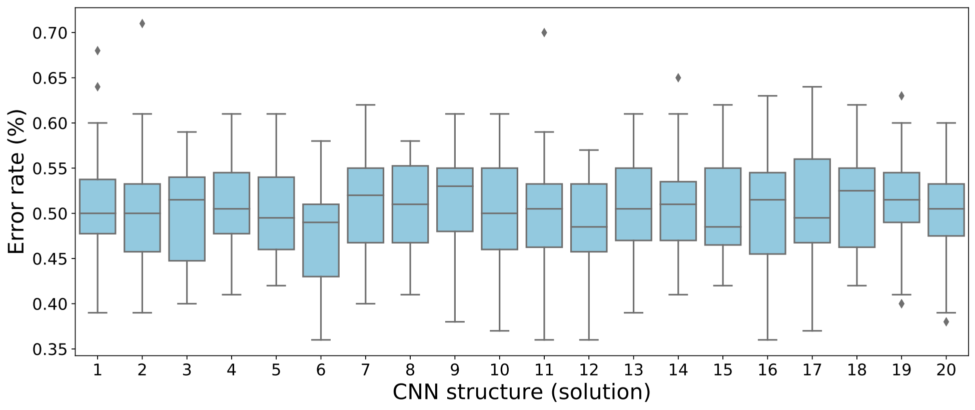


Figure 5. Boxplot of the error rate distribution of the best 20 solutions generated by the MBO algorithm.

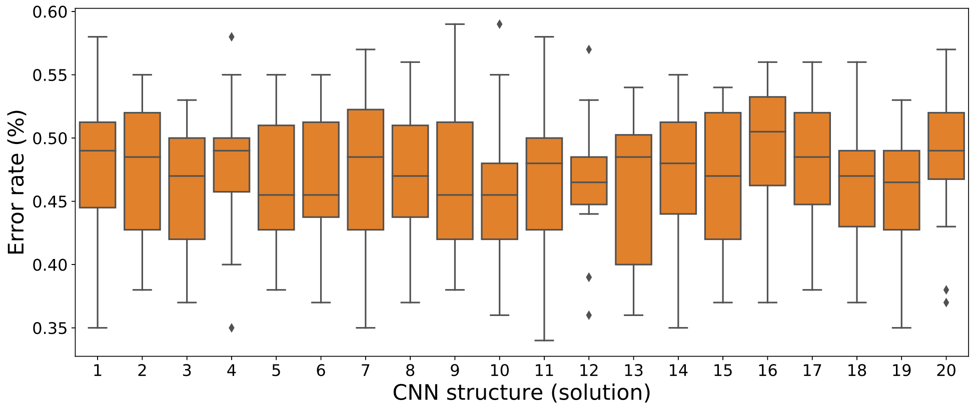


Figure 6. Boxplot of the error rate distribution of the best 20 solutions generated by the MBO-ABCFE algorithm.

The minimum classification error rate of the 20 best CNN structure generated by the MBO algorithm ranged between 0.36 % and 0.5%, with a median value of 0.44%; on the other hand, the maximum classification error rate ranged between 0.45% and 0.71%, with a median value of 0.575%. In the case of the proposed MBO-ABCFE algorithm, the minimum classification error rate of the 20 best CNN structures ranged between 0.34% and 0.47%, with a median value of 0.415%, and the maximum classification error rate ranged between 0.39% and 0.59%, with a median value of 0.545%.

The best architecture resulted in a 0.36% error rate on the test set. The optimized architecture consisted of two convolutional layers, the filter size in the first layer being 64 and in the second layer 128. The filter size was 5×5 in the first layer and in the second 3×3. The pooling layers' size was 2×2 after each convolution. In the convolution layers, as well as the FC layers, the ReLU activation function was used. The architecture had only one full-connected layer before the classification layer, and it had 1024 hidden units. As regularization, only a dropout with probability 0.5 of keeping was selected. The batch size was 100, and the structure was trained by the adamax optimizer with a learning rate $\alpha = 10^{-3}$. The best resulted architecture is depicted in Figure 7.

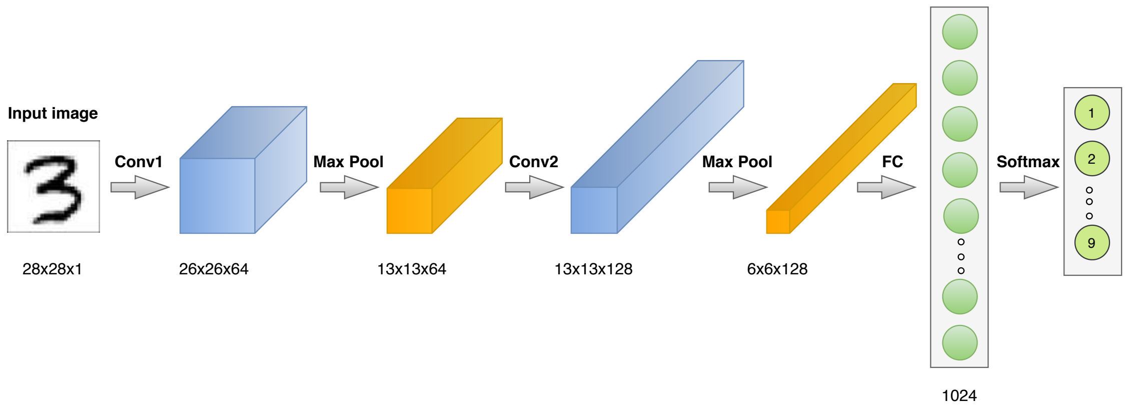


Figure 7. CNN architecture.

The resulting structures, which were generated automatically by metaheuristics algorithms, may have the same or similar structure and design as the traditional CNN structures. The main difference was that when using evolutionary or some other metaheuristics, hyperparameters optimization was performed automatically by the algorithm, instead of manually by performing "trial and error", as is the case in traditional approaches. In both cases (traditional and metaheuristics), the main layers stacked on top of one another and formed the full structure; also, in both cases, additional building blocks could be incorporated. The utilization of a metaheuristic approach allowed the evolution of a better accuracy rate without manual modification of hyperparameter values, which is the case in the traditional approach.

As already noted a few times, the state-of-the-art method that was presented and tested under the same experimental conditions [31] was utilized as a direct comparison with our proposed MBO-ABCFE. In [31], the neuroevolution framework by using GA and grammatical evolution was proposed. The proposed framework managed to obtain the lowest classification error rate on the MNIST dataset of 0.37%. Compared to this framework, the original MBO established slightly better performance, resulting in a 0.36% error rate, while the proposed MBO-ABCFE managed to perform classification of the MNIST dataset with only a 0.34% error rate.

A comparative analysis between the proposed MBO-ABCFE metaheuristics, MBO, and the neuroevolution framework by using GA and grammatical evolution is given in Table 9. With the goals of establishing more objective validation of the proposed MBO-ABCFE-CNN framework and to provide a more informative and extensive review to the evolutionary computation and deep learning communities, in the presented comparative analysis, we also included the results of some traditional CNNs tested on the same dataset. All results were reported and retrieved from the relevant literature sources.

Table 9. Comparative analysis of the classification error rate on the MNIST dataset.

Method	Error Rate (%)
CNN LeNet -5 [1]	0.95
CNN (2 conv, 1 dense, ReLU) with DropConnect [22]	0.57
CNN (2 conv, 1 dense, ReLU) with dropout [22]	0.52
CNN (3 conv maxout, 1 dense) with dropout [101]	0.45
CNN with multi-loss regularization [102]	0.42
Verbancsics et al. [103]	7.9
EXACT [104]	1.68
DEvol [105]	0.60
Baldominos et al. [31]	0.37
MBO-CNN	0.36
MBO-ABCFE-CNN	0.34

The visual representation of the comparative analysis between the proposed MBO-ABCFE-CNN framework and some methods, the results of which are presented in Table 9, is given in Figure 8.

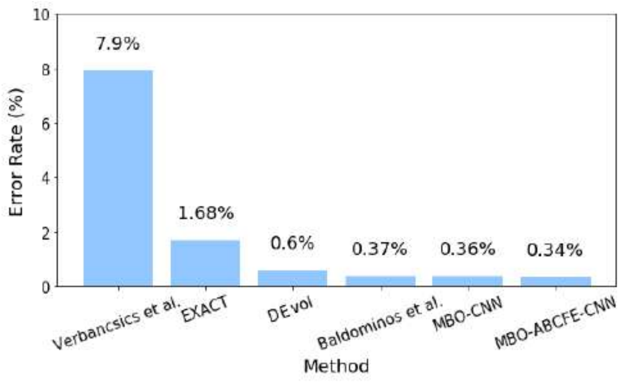


Figure 8. Visual representation of the comparative analysis.

It can be concluded that the proposed MBO-ABCFE approach is very promising in the application of CNN design. The metaheuristic approach led to very good results, and it did not require expertise in the domain of the convolutional neural network for fine-tuning the hyperparameters. Since the design was done automatically, it would save the time and effort of the researchers who are conducting the experiments.

6. Conclusions

Convolutional neural networks, as a fast growing field, have a wide range of applications in different areas and represent important machine learning methods. One of their major limitations is that a structure for a given problem requires the fine-tuning of the hyperparameter values in order to achieve better accuracy. This process is very time consuming and requires a lot of effort and researchers expertise in this domain.

In the literature survey, research that addressed this problem could be found. The general idea was to develop an automatic framework for generating CNN structures (design) that would perform specific classification tasks with high accuracy. Recently, the term “neuroevolution” for generating such networks was proposed. Since the CNN hyperparameters’ optimization belongs to the group of NP-hard challenges, all research has proposed heuristic and/or metaheuristics methods for its solving.

The research proposed in this paper is an extension of our previously conducted simulations and experiments from this domain. However, in the research that was presented in this paper, with the objective to generate network structures that would perform a given classification task with higher accuracy than other networks, we included more CNN hyperparameters for the optimization process than previously presented works from this domain: the number of convolutional layers along with the number of kernels, the kernel size and activation function of each convolutional layer, the pooling size, the number of dense (fully-connected) layers with the number of neurons, the connectivity pattern, the activation function, weight regularization, the dropout for each dense layer, as well as the batch size, learning rate, and rule as general CNN hyperparameters.

We tried to generate state-of-the-art CNN structures by developing an automatic framework using hybridized MBO swarm intelligence metaheuristics, which was also proposed in this paper. By conducting practical simulations with the original MBO, we noticed some deficiencies that were particularly emphasized in its exploration process and in the established balance between intensification and diversification. Moreover, we concluded that the basic MBO’s exploitation process could be further improved. To address these issues, we hybridized the original MBO with ABC and FA swarm algorithms. We first adopted the exploration mechanism and one parameter that adjusted intensification from the ABC metaheuristics. Second, we incorporated a very efficient FA search equation into our approach.

In compliance with the established practice in the scientific literature, the proposed hybridized MBO-ABCFE was firstly tested on a standard group of unconstrained benchmarks, and later, it was applied to the practical CNNs' neuroevolution problem. For the tests on unconstrained function instances, we performed comparative analysis with the original MBO, as well as with one other improved state-of-the-art MBO algorithm. Our proposed MBO-ABCFE proved to be a significantly better approach in terms of convergence (mean values) and also in the results' quality (best individuals).

In the second group of practical simulations, by using the proposed MBO-ABCFE, we developed the framework for CNN hyperparameters' optimization and performed tests on the standard MNIST dataset. Since the implementation of the original MBO for this challenge was not found in the literature survey, to establish a more precise comparative analysis, we also implemented original MBO for this problem. The proposed hybrid metaheuristics was compared with several other state-of-the-art methods that were tested on the same dataset and under the same experimental conditions, and we obtained better classification accuracy. Moreover, the original MBO managed to establish high classification accuracy.

The scientific contributions of the presented research can be summarized as follows:

- An automated framework for "neuroevolution" based on the hybridized MBO-ABCFE algorithm, which managed to design and generate CNN architectures with high performance (accuracy) for image classification tasks, was developed;
- In the CNN hyperparameters' optimization problem, we included more CNN hyperparameters than most previous works from this domain;
- We managed to enhance the original MBO approach significantly by performing hybridization with other state-of-the-art swarm algorithms; and
- The original MBO was implemented for the first time for tackling CNNs' optimization challenge.

The detailed description of the experimental conditions along with the control parameters' values were presented in this paper.

Since the domain of the proposed research represents a very promising area, in our future work, we plan to continue research, experiments, and simulations with CNNs' design challenge. We will also try to improve other swarm intelligence algorithms and adjust them to tackle this problem. Moreover, we plan to invest significant effort into developing frameworks that will include even more CNN hyperparameters in the optimization process and perform tests on other datasets as well, like CIFAR-10, SEED, and Semeion Handwritten Digits.

Author Contributions: N.B. and T.B. proposed the idea. N.B., T.B., I.S., and E.T. implemented, adapted, and adjusted the algorithms and simulation environment. The entire research project was conceived of and supervised by M.T. The original draft was written by I.S., N.B., and E.T. Review and editing was performed by M.T. All authors participated in the conducted experiments and in the discussion of the experimental results. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
2. Farabet, C.; Couprie, C.; Najman, L.; LeCun, Y. Learning Hierarchical Features for Scene Labeling. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1915–1929. [CrossRef]
3. Stoean, C.; Stoean, R.; Becerra-García, R.A.; García-Bermúdez, R.; Atencia, M.; García-Lagos, F.; Velázquez-Pérez, L.; Joya, G. Unsupervised Learning as a Complement to Convolutional Neural Network Classification in the Analysis of Saccadic Eye Movement in Spino-Cerebellar Ataxia Type 2. In *Advances in Computational Intelligence*; Rojas, I., Joya, G., Catala, A., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 26–37.

4. Karpathy, A.; Li, F.-F. Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3128–3137.
5. Taigman, Y.; Yang, M.; Ranzato, M.; Wolf, L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 1701–1708. [CrossRef]
6. Samide, A.; Stoean, C.; Stoean, R. Surface study of inhibitor films formed by polyvinyl alcohol and silver nanoparticles on stainless steel in hydrochloric acid solution using convolutional neural networks. *Appl. Surf. Sci.* **2019**, *475*, 1–5. [CrossRef]
7. Toshev, A.; Szegedy, C. DeepPose: Human Pose Estimation via Deep Neural Networks. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; IEEE Computer Society: Washington, DC, USA, 2014; pp. 1653–1660. [CrossRef]
8. Stoean, R.; Stoean, C.; Samide, A.; Joya, G. Convolutional Neural Network Learning Versus Traditional Segmentation for the Approximation of the Degree of Defective Surface in Titanium for Implantable Medical Devices. In *Advances in Computational Intelligence*; Rojas, I., Joya, G., Catala, A., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 871–882.
9. Hubel, D.H.; Wiesel, T.N. Receptive fields of single neurones in the cat's striate cortex. *J. Physiol.* **1959**, *148*, 574–591. [CrossRef] [PubMed]
10. Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybern.* **1980**, *36*, 193–202. [CrossRef] [PubMed]
11. LeCun, Y.; Boser, B.E.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.E.; Jackel, L.D. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1990; pp. 396–404.
12. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*; Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2012; pp. 1097–1105.
13. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 1–9. [CrossRef]
14. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015.
15. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely Connected Convolutional Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 2261–2269. [CrossRef]
16. Hu, J.; Shen, L.; Sun, G. Squeeze-and-Excitation Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017.
17. Duchi, J.C.; Hazan, E.; Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* **2011**, *12*, 2121–2159.
18. Zeiler, M.D. ADADELTA: An Adaptive Learning Rate Method. *arXiv* **2012**, arXiv:cs.LG/1212.5701.
19. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:cs.LG/1412.6980.
20. Ng, A.Y. Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*; ACM: New York, NY, USA, 2004; p. 788. [CrossRef]
21. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
22. Wan, L.; Zeiler, M.; Zhang, S.; Le Cun, Y.; Fergus, R. Regularization of neural networks using dropconnect. In Proceedings of the International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 1058–1066.
23. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*; Bach, F., Blei, D., Eds.; PMLR: Lille, France, 2015; Volume 37, pp. 448–456.
24. Nair, V.; Hinton, G.E. Rectified Linear Units Improve Restricted Boltzmann Machines. In Proceedings of the 27th International Conference on International Conference on Machine Learning, Scotland, UK, 26 June–1 July 2010; pp. 807–814.

25. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2017; p. 800.
26. Wang, Y.; Zhang, H.; Zhang, G. cPSO-CNN: An efficient PSO-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. *Swarm Evol. Comput.* **2019**, *49*, 114–123. [CrossRef]
27. Darwish, A.; Ezzat, D.; Hassanien, A.E. An optimized model based on convolutional neural networks and orthogonal learning particle swarm optimization algorithm for plant diseases diagnosis. *Swarm Evol. Comput.* **2020**, *52*, 100616. [CrossRef]
28. Yamasaki, T.; Honma, T.; Aizawa, K. Efficient Optimization of Convolutional Neural Networks Using Particle Swarm Optimization. In Proceedings of the 2017 IEEE Third International Conference on Multimedia Big Data (BigMM), Laguna Hills, CA, USA, 19–21 April 2017; pp. 70–73. [CrossRef]
29. Qolomany, B.; Maabreh, M.; Al-Fuqaha, A.; Gupta, A.; Benhaddou, D. Parameters optimization of deep learning models using Particle swarm optimization. In Proceedings of the 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 26–30 June 2017; pp. 1285–1290. [CrossRef]
30. Bochinski, E.; Senst, T.; Sikora, T. Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP), Beijing, China, 17–20 September 2017; pp. 3924–3928. [CrossRef]
31. Baldominos, A.; Saez, Y.; Isasi, P. Evolutionary convolutional neural networks: An application to handwriting recognition. *Neurocomputing* **2018**, *283*, 38–52. [CrossRef]
32. Strumberger, I.; Tuba, E.; Bacanin, N.; Jovanovic, R.; Tuba, M. Convolutional Neural Network Architecture Design by the Tree Growth Algorithm Framework. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–8. [CrossRef]
33. Strumberger, I.; Tuba, E.; Bacanin, N.; Zivkovic, M.; Beko, M.; Tuba, M. Designing Convolutional Neural Network Architecture by the Firefly Algorithm. In Proceedings of the 2019 International Young Engineers Forum (YEF-ECE), Caparica, Portugal, 10 May 2019; pp. 59–65. [CrossRef]
34. Bacanin, N.; Bezdan, T.; Tuba, E.; Strumberger, I.; Tuba, M. Optimizing Convolutional Neural Network Hyperparameters by Enhanced Swarm Intelligence Metaheuristics. *Algorithms* **2020**, *13*, 67. [CrossRef]
35. Wang, G.G.; Deb, S.; Cui, Z. Monarch Butterfly Optimization. *Neural Comput. Appl.* **2015**, 1–20. [CrossRef]
36. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Monarch butterfly optimization algorithm for localization in wireless sensor networks. In Proceedings of the 2018 28th International Conference Radioelektronika (RADIOELEKTRONIKA), Prague, Czech Republic, 19–20 April 2018; pp. 1–6. [CrossRef]
37. Wang, G.G.; Hao, G.S.; Cheng, S.; Qin, Q. A Discrete Monarch Butterfly Optimization for Chinese TSP Problem. In *Advances in Swarm Intelligence*; Tan, Y., Shi, Y., Niu, B., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 165–173.
38. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Modified and Hybridized Monarch Butterfly Algorithms for Multi-Objective Optimization. In *International Conference on Hybrid Intelligent Systems*; Springer: Berlin, Germany, 2018; pp. 449–458.
39. Strumberger, I.; Tuba, M.; Bacanin, N.; Tuba, E. Cloudlet Scheduling by Hybridized Monarch Butterfly Optimization Algorithm. *J. Sensor Actuator Networks* **2019**, *8*, 44. [CrossRef]
40. Strumberger, I.; Sarac, M.; Markovic, D.; Bacanin, N. Hybridized Monarch Butterfly Algorithm for Global Optimization Problems. *Int. J. Comput.* **2018**, *3*, 63–68.
41. Wang, G.G.; Deb, S.; Zhao, X.; Cui, Z. A new monarch butterfly optimization with an improved crossover operator. *Oper. Res.* **2018**, *18*, 731–755. [CrossRef]
42. Suganuma, M.; Shirakawa, S.; Nagao, T. A Genetic Programming Approach to Designing Convolutional Neural Network Architectures. In *GECCO '17, Proceedings of the Genetic and Evolutionary Computation Conference, Berlin, Germany, 15–19 July 2017*; ACM: New York, NY, USA, 2017; pp. 497–504. [CrossRef]
43. De Rosa, G.H.; Papa, J.P.; Yang, X.S. Handling dropout probability estimation in convolution neural networks using meta-heuristics. *Soft Comput.* **2018**, *22*, 6147–6156. [CrossRef]
44. Ting, T.O.; Yang, X.S.; Cheng, S.; Huang, K. Hybrid Metaheuristic Algorithms: Past, Present, and Future. *Recent Adv. Swarm Intell. Evol. Comput. Stud. Comput. Intell.* **2015**, *585*, 71–83.
45. Bacanin, N.; Tuba, M. Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Improved with Genetic Operators. *Stud. Inform. Control* **2012**, *21*, 137–146. [CrossRef]
46. Dorigo, M.; Birattari, M. *Ant Colony Optimization*; Springer: Berlin/Heidelberg, Germany, 2010.

47. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948. [CrossRef]
48. Karaboga, D.; Basturk, B. On the performance of artificial bee colony (ABC) algorithm. *Appl. Soft Comput.* **2008**, *8*, 687–697. [CrossRef]
49. Yang, X.S. Firefly Algorithms for Multimodal Optimization. In *Stochastic Algorithms: Foundations and Applications*; Watanabe, O., Zeugmann, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 169–178.
50. Strumberger, I.; Bacanin, N.; Tuba, M. Enhanced Firefly Algorithm for Constrained Numerical Optimization, IEEE Congress on Evolutionary Computation. In Proceedings of the IEEE International Congress on Evolutionary Computation (CEC 2017), San Sebastián, Spain, 5–8 June 2017; pp. 2120–2127.
51. Gandomi, A.H.; Yang, X.S.; Alavi, A.H. Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems. *Eng. Comput.* **2013**, *29*, 17–35. [CrossRef]
52. Bacanin, N. Implementation and performance of an object-oriented software system for cuckoo search algorithm. *Int. J. Math. Comput. Simul.* **2010**, *6*, 185–193.
53. Yang, X.S.; Hossein Gandomi, A. Bat algorithm: A novel approach for global engineering optimization. *Eng. Comput.* **2012**, *29*, 464–483. [CrossRef]
54. Mirjalili, S.; Lewis, A. The whale optimization algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [CrossRef]
55. Wang, G.G.; Deb, S.; dos S. Coelho, L. Elephant Herding Optimization. In Proceedings of the 2015 3rd International Symposium on Computational and Business Intelligence (ISCBI), Bali, Indonesia, 7–9 December 2015; pp. 1–5.
56. Strumberger, I.; Bacanin, N.; Tuba, M. Hybridized Elephant Herding Optimization Algorithm for Constrained Optimization. In *Hybrid Intelligent Systems*; Abraham, A., Muhuri, P.K., Muda, A.K., Gandhi, N., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 158–166.
57. Strumberger, I.; Tuba, E.; Zivkovic, M.; Bacanin, N.; Beko, M.; Tuba, M. Dynamic Search Tree Growth Algorithm for Global Optimization. In *Technological Innovation for Industry and Service Systems*; Camarinha-Matos, L.M.; Almeida, R.; Oliveira, J., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 143–153.
58. Strumberger, I.; Bacanin, N.; Tomic, S.; Beko, M.; Tuba, M. Static drone placement by elephant herding optimization algorithm. In Proceedings of the 2017 25th Telecommunication Forum (TELFOR), Belgrade, Serbia, 21–22 November 2017; pp. 1–4.
59. Cheraghalipour, A.; Hajiaghaei-Keshteli, M.; Paydar, M.M. Tree Growth Algorithm (TGA): A novel approach for solving optimization problems. *Eng. Appl. Artif. Intell.* **2018**, *72*, 393–414. [CrossRef]
60. Mucherino, A.; Seref, O. Monkey search: A novel metaheuristic search for global optimization. In *Data Mining, Systems Analysis and Optimization in Biomedicine*; Seref, O., Kundakcioglu, E., Pardalos, P., Eds.; American Institute of Physics Conference Series; American Institute of Physics: Melville, NY, USA, 2007; Volume 953, pp. 162–173. [CrossRef]
61. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Hybridized moth search algorithm for constrained optimization problems. In Proceedings of the 2018 International Young Engineers Forum (YEF-ECE), Costa da Caparica, Portugal, 4 May 2018; pp. 1–5. [CrossRef]
62. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
63. Yang, X.S. Flower pollination algorithm for global optimization. In Proceedings of the International Conference on Unconventional Computing and Natural Computation, Orléans, France, 3–7 September 2012; pp. 240–249.
64. Strumberger, I.; Bacanin, N.; Tuba, M. Hybridized elephant herding optimization algorithm for constrained optimization. In Proceedings of the International Conference on Health Information Science, Moscow, Russia, 7–9 October 2017; pp. 158–166.
65. Strumberger, I.; Sarac, M.; Markovic, D.; Bacanin, N. Moth Search Algorithm for Drone Placement Problem. *Int. J. Comput.* **2018**, *3*, 75–80.
66. Strumberger, I.; Tuba, E.; Bacanin, N.; Tuba, M. Modified Moth Search Algorithm for Portfolio Optimization. In *Smart Trends in Computing and Communications*; Zhang, Y.D., Mandal, J.K., So-In, C., Thakur, N.V., Eds.; Springer: Singapore, 2020; pp. 445–453.
67. Tuba, E.; Strumberger, I.; Bacanin, N.; Zivkovic, D.; Tuba, M. Brain Storm Optimization Algorithm for Thermal Image Fusion using DCT Coefficients. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 234–241.

68. Tuba, E.; Strumberger, I.; Zivkovic, D.; Bacanin, N.; Tuba, M. Mobile Robot Path Planning by Improved Brain Storm Optimization Algorithm. In Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8. [CrossRef]
69. Tuba, E.; Strumberger, I.; Bacanin, N.; Tuba, M. Optimal Path Planning in Environments with Static Obstacles by Harmony Search Algorithm. In *Advances in Harmony Search, Soft Computing and Applications*; Kim, J.H., Geem, Z.W., Jung, D., Yoo, D.G., Yadav, A., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 186–193.
70. Bacanin, N.; Tuba, M. Firefly Algorithm for Cardinality Constrained Mean-Variance Portfolio Optimization Problem with Entropy Diversity Constraint. *Sci. World J.* **2014**, *2014*, 16. [CrossRef]
71. Tuba, M.; Bacanin, N. Artificial bee colony algorithm hybridized with firefly metaheuristic for cardinality constrained mean-variance portfolio problem. *Appl. Math. Inf. Sci.* **2014**, *8*, 2831–2844. [CrossRef]
72. Strumberger, I.; Minovic, M.; Tuba, M.; Bacanin, N. Performance of Elephant Herding Optimization and Tree Growth Algorithm Adapted for Node Localization in Wireless Sensor Networks. *Sensors* **2019**, *19*, 2515. [CrossRef] [PubMed]
73. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Wireless Sensor Network Localization Problem by Hybridized Moth Search Algorithm. In Proceedings of the 2018 14th International Wireless Communications Mobile Computing Conference (IWCMC), Limassol, Cyprus, 25–29 June 2018; pp. 316–321. [CrossRef]
74. Tuba, M.; Bacanin, N. Hybridized bat algorithm for multi-objective radio frequency identification (RFID) network planning. In Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan, 25-28 May 2015; pp. 499–506. [CrossRef]
75. Bacanin, N.; Tuba, M.; Strumberger, I. RFID network planning by ABC algorithm hybridized with heuristic for initial number and locations of readers. In Proceedings of the 2015 17th UKSim-AMSS International Conference on Modelling and Simulation (UKSim), Cambridge, UK, 25–27 March 2015; pp. 39–44.
76. Bacanin, N.; Tuba, M.; Jovanovic, R. Hierarchical multiobjective RFID network planning using firefly algorithm. In Proceedings of the 2015 International Conference on Information and Communication Technology Research (ICTRC), Abu Dhabi, UAE, 17–19 May 2015; pp. 282–285. [CrossRef]
77. Strumberger, I.; Bacanin, N.; Tuba, M.; Tuba, E. Resource Scheduling in Cloud Computing Based on a Hybridized Whale Optimization Algorithm. *Appl. Sci.* **2019**, *9*, 4893. [CrossRef]
78. Strumberger, I.; Tuba, E.; Bacanin, N.; Tuba, M. Hybrid Elephant Herding Optimization Approach for Cloud Computing Load Scheduling. In *Swarm, Evolutionary, and Memetic Computing and Fuzzy and Neural Computing*; Zamuda, A., Das, S., Suganthan, P.N., Panigrahi, B.K., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 201–212.
79. Strumberger, I.; Tuba, E.; Bacanin, N.; Tuba, M. Dynamic Tree Growth Algorithm for Load Scheduling in Cloud Environments. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 65–72. [CrossRef]
80. Magud, O.; Tuba, E.; Bacanin, N. Medical ultrasound image speckle noise reduction by adaptive median filter. *Wseas Trans. Biol. Biomed.* **2017**, *14*, 38–46.
81. Hrosik, R.C.; Tuba, E.; Dolicanin, E.; Jovanovic, R.; Tuba, M. Brain Image Segmentation Based on Firefly Algorithm Combined with K-means Clustering. *Stud. Inform. Control* **2019**, *28*, 167–176. [CrossRef]
82. Tuba, M.; Bacanin, N.; Alihodzic, A. Multilevel image thresholding by fireworks algorithm. In Proceedings of the 2015 25th International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, Czech Republic, 21–22 April 2015; pp. 326–330. [CrossRef]
83. Tuba, M.; Alihodzic, A.; Bacanin, N. Cuckoo Search and Bat Algorithm Applied to Training Feed-Forward Neural Networks. In *Recent Advances in Swarm Intelligence and Evolutionary Computation*; Springer International Publishing: Cham, Switzerland, 2015; pp. 139–162. [CrossRef]
84. Tuba, E.; Strumberger, I.; Bacanin, N.; Tuba, M. Bare Bones Fireworks Algorithm for Capacitated p-Median Problem. In *Advances in Swarm Intelligence*; Tan, Y., Shi, Y., Tang, Q., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 283–291.
85. Sulaiman, N.; Mohamad-Saleh, J.; Abro, A.G. A hybrid algorithm of ABC variant and enhanced EGS local search technique for enhanced optimization performance. *Eng. Appl. Artif. Intell.* **2018**, *74*, 10–22. [CrossRef]
86. Ghosh, S.; Kaur, M.; Bhullar, S.; Karar, V. Hybrid ABC-BAT for Solving Short-Term Hydrothermal Scheduling Problems. *Energies* **2019**, *12*, 551. [CrossRef]

87. Bacanin, N.; Tuba, E.; Bezdan, T.; Strumberger, I.; Tuba, M. Artificial Flora Optimization Algorithm for Task Scheduling in Cloud Computing Environment. In Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning, Manchester, UK, 14–16 November 2019; pp. 437–445.
88. Tuba, E.; Strumberger, I.; Bacanin, N.; Zivkovic, D.; Tuba, M. Acute Lymphoblastic Leukemia Cell Detection in Microscopic Digital Images Based on Shape and Texture Features. In *Advances in Swarm Intelligence*; Tan, Y., Shi, Y., Niu, B., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 142–151.
89. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed.; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1989.
90. Fogel, D.; Society, I.C.I. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*; IEEE Series on Computational Intelligence; Wiley: Hoboken, NJ, USA, 2006.
91. Beyer, H.G.; Schwefel, H.P. Evolution strategies—A comprehensive introduction. *Nat. Comput.* **2002**, *1*, 3–52. [CrossRef]
92. Gao, Z.; Li, Y.; Yang, Y.; Wang, X.; Dong, N.; Chiang, H.D. A GPSO-optimized convolutional neural networks for EEG-based emotion recognition. *Neurocomputing* **2020**, *380*, 225–235. [CrossRef]
93. Martín, A.; Vargas, V.M.; Gutiérrez, P.A.; Camacho, D.; Hervás-Martínez, C. Optimising Convolutional Neural Networks using a Hybrid Statistically-driven Coral Reef Optimisation algorithm. *Appl. Soft Comput.* **2020**, *90*, 106144. [CrossRef]
94. Anaraki, A.K.; Ayati, M.; Kazemi, F. Magnetic resonance imaging-based brain tumor grades classification and grading via convolutional neural networks and genetic algorithms. *Biocybern. Biomed. Eng.* **2019**, *39*, 63–74. [CrossRef]
95. Fernando, C.; Banarse, D.; Reynolds, M.; Besse, F.; Pfau, D.; Jaderberg, M.; Lanctot, M.; Wierstra, D. Convolution by Evolution: Differentiable Pattern Producing Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*; Association for Computing Machinery: New York, NY, USA, 2016; pp. 109–116. [CrossRef]
96. Davison, J. DEvol: Automated Deep Neural Network Design via Genetic Programming. Available online: https://github.com/joeddav/devol (accessed on 1 March 2020).
97. Karaboga, D.; Akay, B. A modified Artificial Bee Colony (ABC) Algorithm for constrained optimization problems. *Appl. Soft Comput.* **2011**, *11*, 3021–3031. [CrossRef]
98. Ghanem, W.A.; Jantan, A. Hybridizing artificial bee colony with monarch butterfly optimization for numerical optimization problems. *Neural Comput. Appl.* **2018**, *30*, 163–181. [CrossRef]
99. Tuba, M.; Nebojsa. Improved seeker optimization algorithm hybridized with firefly algorithm for constrained optimization problems. *Neurocomputing* **2014**, *143*, 197–207. [CrossRef]
100. LeCun, Y.; Cortes, C. MNIST Handwritten Digit Database. 2010. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 1 March 2020).
101. Jarrett, K.; Kavukcuoglu, K.; Ranzato, M.; LeCun, Y. What is the best multi-stage architecture for object recognition? In Proceedings of the 2009 IEEE 12th International Conference on Computer Vision, Kyoto, Japan, 29 September–2 October 2009; pp. 2146–2153.
102. Xu, Y.F.; Lu, W.; Rabinowitz, J.D. Avoiding Misannotation of In-Source Fragmentation Products as Cellular Metabolites in Liquid Chromatography–Mass Spectrometry-Based Metabolomics. *Anal. Chem.* **2015**, *87*, 2273–2281. [CrossRef]
103. Verbancsics, P.; Harguess, J. Generative NeuroEvolution for Deep Learning. *arXiv* **2013**, arXiv:cs.NE/1312.5355.
104. Desell, T. Large Scale Evolution of Convolutional Neural Networks Using Volunteer Computing. *arXiv* **2017**, arXiv:cs.NE/1703.05422.
105. Baldominos, A.; Saez, Y.; Isasi, P. Model selection in committees of evolved convolutional neural networks using genetic algorithms. In Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning, Madrid, Spain, 21–23 November 2018; pp. 364–373.