Optimizing Convolutional Neural Network Hyperparameters by Enhanced Swarm Intelligence Metaheuristics

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

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.)

check for updates

Abstract: Computer vision is one of the most frontier technologies in computer science. It is used to build artificial systems to extract valuable information from images and has a broad range of applications in various areas such as agriculture, business, and healthcare. Convolutional neural networks represent the key algorithms in computer vision, and in recent years, they have attained notable advances in many real-world problems. The accuracy of the network for a particular task profoundly relies on the hyperparameters' configuration. Obtaining the right set of hyperparameters is a time-consuming process and requires expertise. To approach this concern, we propose an automatic method for hyperparameters' optimization and structure design by implementing enhanced metaheuristic algorithms. The aim of this paper is twofold. First, we propose enhanced versions of the tree growth and firefly algorithms that improve the original implementations. Second, we adopt the proposed enhanced algorithms for hyperparameters' optimization. First, the modified metaheuristics are evaluated on standard unconstrained benchmark functions and compared to the original algorithms. Afterward, the improved algorithms are employed for the network design. The experiments are carried out on the famous image classification benchmark dataset, the MNIST dataset, and comparative analysis with other outstanding approaches that were tested on the same problem is conducted. The experimental results show that both proposed improved methods establish higher performance than the other existing techniques in terms of classification accuracy and the use of computational resources.

Keywords: convolutional neural network; swarm intelligence; optimization; tree growth algorithm, firefly algorithm, enhanced metaheuristics

1. Introduction

Convolutional neural networks (ConvNets or CNN) are biologically-inspired structures and represent a special type of neural network. CNN is most commonly used to process 2D signals for digital image processing, but it can also be used to process 1D signals. ConvNets are very successful in different visual tasks, such as classifying images, detecting objects in pictures [1], segmenting images [2], generating image descriptions [3], recognizing faces [4], pose estimation [5], and others. LeCun et al. proposed the first CNN in 1998 [6]. It was successfully applied to digit recognition. AlexNet [7] was the first architecture that achieved significant results, popularized CNN, and brought a revolution in computer vision. AlexNet has a very similar architecture to LeNet, but it is much deeper. Some of the other well-known architectures are VGG [8], and more modern networks are GoogleNet [9], ResNet [10], DenseNet [11], and SENet [12].

The fundamental building blocks of CNNs are the convolution layer, the pooling layer, and the fully-connected layer (FC layer). By stacking these layers properly in a deep network, it can successfully solve different visual tasks. The classic CNN takes an input image, then it performs the convolution operation by sliding filters through the input image and applies the activation function, which results in feature maps, and after that, the pooling layer is utilized to reduce the dimensionality. This process is repeated several times, and in the end, one or more fully-connected layers exist in the architecture. The last FC layer represents the classifier. More modern architectures employ other building blocks and different operations, such as the residual block [10], the inception module [9], and the squeeze-and-excitation block [12].

The performance of any CNN highly depends on the parameters such as the number of convolutional layers and filters, along with the filter size, batch size, etc. These parameters are known as the CNN hyperparameters [13]. The values of the hyperparameters determine the CNN's success rate for solving a particular problem, and a CNN architecture that uniformly generates satisfying results for all problem instances does not exist. Determining the right values of hyperparameters (hyperparameters' optimization) for a particular problem represents NP hard tasks and one of the greatest challenges from the CNN domain [14]. The CNN architecture design can be perceived as the model selection problem in machine learning [15].

According to the literature review, most researchers focused on the CNN architecture improvements concerning classification accuracy, and only some of them addressed the problem of the hyperparameters' optimization [15–20].

The general hyperparameters in CNN that could be optimized are the following: the number of convolutional layers, the number of filters, the size of the filters, the stride, which is the number of steps that the filter takes when sliding through the input image, the type of padding, which can be either the same convolution or a valid convolution, the dropout probability, the batch size, and the number of training epochs. The choice of activation function, the regularization technique, significantly contributes to the network's performance. The most commonly used activation functions are sigmoid, hyperbolic tangent (TanH), and rectified linear unit (ReLU).

The most popular hyperparameter tuning techniques are grid search and random search [21]. Grid search is a systematic process for hyperparameter optimization. It is a trial and error method, where every combination of hyperparameters is specified and evaluated on the model. This technique requires expertise from the domain. The grid search method can be effective when the number of hyperparameters is small. However, as the number of hyperparameters increases, the utilization of computational resources grows exponentially. Random search is more effective than a grid search by allowing researchers to identify good hyperparameter values more accurately by sampling randomly from the search space. One of the main shortcomings of this technique is that it does not take into account previously produced results. In other words, it does not learn from previous tests, and repeated search with the same hyperparameters is possible.

The two above-mentioned techniques (grid search and random search) for hyperparameters' tuning require much time and knowledge from the domain. Addressing this issue, we propose an automatic method for hyperparameters' optimization and structure design by employing enhanced swarm intelligence algorithms. Opposite of random search, the swarm intelligence algorithm learns in each iteration and in each next iteration moves toward better hyperparameter values.

Since the CNN hyperparameters' optimization belongs to the group of NP hard problems, satisfying results could be obtained by employing metaheuristic-based approaches.

1.1. Swarm intelligence metaheuristic and literature review

Metaheuristic algorithms are stochastic approximate algorithms that do not assure generating the best solution, but they will result in a solution near the global optimum. These algorithms have two main phases, specifically diversification (also called exploration) and intensification (also called exploitation). Diversification refers to the global exploration of the search space. The diversification

phase is followed by the process of intensification. By conducting exploitation, the algorithm searches around the current best solutions in the population. Due to the stochastic nature of the metaheuristics optimization process, one of the most challenging tasks is to find the proper balance between diversification and intensification.

Metaheuristic algorithms are generally divided into two groups: evolutionary algorithms and swarm intelligence algorithms. Evolutionary algorithms (EA) are motivated by biological evolution, and they are further split into three sub-classes: evolutionary strategies [22], evolutionary programming (EP) [23], and genetic algorithms (GA) [24]. Swarm intelligence algorithms are population-based algorithms, which mimic the collective intelligence behavior of animal herds, a flock of birds, ant colonies, bacterial growth, and similar decentralized, self-organized systems in nature. The collective intelligence emerges from the cooperation between individuals locally and with their environment. Metaheuristic algorithm hybridization is achieved by the combination of the advantages of two or more metaheuristic algorithms or by incorporating additional operations in the existing metaheuristics, which yields a faster convergence rate, better accuracy, and more adequate balance between the exploration and exploitation process.

Some of the notable swarm intelligence algorithms are particle swarm optimization (PSO) [25], artificial bee colony (ABC) [26–28], ant colony optimization (ACO) [29], bacterial foraging optimization (BFO) [30], the firefly algorithm (FA) [31], the whale optimization algorithm (WOA) [32,33], the bat algorithm (BA) [34], cuckoo search (CS) [35,36], elephant herding optimization (EHO) [37], Monarch butterfly optimization (MBO) [38], and the tree growth algorithm (TGA) [39]. There are many successful applications of swarm intelligence algorithms on different real-life problems that were performed by many authors such as [40–69].

1.2. Research objectives and paper's structure

The network's performance highly depends on the hyperparameters' configuration, and the most critical issue is that there is no general method for determining the optimal set of CNN's hyperparameters and for designing the networks' structure. Moreover, another challenge is that there is no general architecture that can be applied for every problem ; each particular problem instance requires an individually designed CNN architecture. Considering that, the technique for finding the right set of hyperparameters and defining the CNNs architecture is commonly known as "guesstimating" (estimating by using guess).

The process of guessing the right CNN architecture (value of hyperparameters) is both computational resource intensive and time consuming [13] and requires a trial and error approach. For this reason, an approach that is able to choose the right CNN architecture automatically for a particular problem should be devised. Based on the facts that the swarm intelligence algorithms proved to be robust and efficient methods in solving NP hard tasks like CNN hyperparameter optimization and that those methods have already successfully been applied to this problem, the basic motivation behind the research is to develop improved swarm intelligence approaches that will generate CNNs architectures that are able to outperform other architectures in terms of classification accuracy.

According to the statement above, the research question behind the presented research can be formulated as follows: Is it possible to further enhance the classification accuracy of generated CNN architectures by using swarm intelligence algorithms?

The primary objective, as well as the scope of the research that is presented in this paper are to develop automatically a method by using swarm intelligence algorithms for designing CNN architectures that are able to obtain greater performance than existing methods from the literature. The secondary objective is to improve existing versions of two swarm intelligence algorithms (the tree growth algorithm and the firefly algorithm) by enhancing the exploitation and/or exploration processes of the original approaches.

When a new method is devised, it is good practice to test it on a standard set of benchmark functions. Only in this way, the real performance metrics of efficiency and robustness of a particular

method can be established. For this reason, we first tested the devised enhanced algorithms on a wider set of standard unconstrained benchmarks, and afterwards, we applied them to a practical NP hard CNN hyperparameter optimization challenge for the image classification task.

In the practical simulations, the hyperparameters that were taken into account during the optimization process were the number of convolution layers along with the number of kernels per layer and the kernel size and number of fully-connected (dense) layers along with the size of each fully-connected layer. We did not include the type of pooling layer, activation function, and regularization technique in the automatic search because of the limitations of available computing power.

To establish the performance of the proposed methods, we utilized simulation as the research methodology in a standard environment and with a standardized dataset.

The experiments were conducted on the MNIST benchmark dataset [70], which was comprised of handwritten digits from zero to nine. Each image was in grayscale, and it consisted of 28×28 pixels. The dataset consisted of 70,000 labeled samples. In the research, 50,000 samples were used for training, 10,000 samples for validation, and 10,000 samples for testing purposes. The grayscale images that had pixel values between zero and 255 were normalized in the range from zero to one.

The evaluation study results were compared with similar techniques that utilized metaheuristic algorithms and automatically designed the CNN architectures and were tested under the same experimental conditions [17]. We also performed comparative analysis with swarm approaches, tested under different conditions, that focused on dropout probability estimation [18] in order to further validate our approaches. Furthermore, the results were also compared to state-of-the-art architectures that were manually designed. In order to present unbiased performance comparison, we utilized the initial parameter settings in a framework similar to [17].

The scientific implications of the presented research are twofold: first, we managed to devise a framework that performed classification with greater accuracy than other state-of-the-art methods for this particular dataset, and secondly, we were able to improve basic versions of the tree growth and firefly algorithms.

We note that we have been researching this domain for some time and that the research presented in this paper is the extension of our previous research [71,72].

The remainder of this paper is as follows: Section 2 gives a concise explanation of the problem statement. Section 3 provides a description of the original swarm algorithms and the proposed enhanced versions. In Section 4, we present the details of the improved swarm implementations for the CNNs' design and the parameter setup. In Section 5, we show the results of the proposed method and the performance comparison study with other state-of-the-art algorithms, while Section 6 concludes the paper.

2. Problem formulation and related literature review

In this section, we first give a literature review of the metaheuristics applications for CNN optimization. Afterwards, we show a mathematical formulation of the model that was optimized in the experimental section by swarm intelligence algorithms.

2.1. Metaheuristics applications for CNN optimization

Based on the literature survey, it can be seen that both types of metaheuristics, evolutionary algorithms and swarm intelligence, have many applications and implementations in the domain of CNNs and computer vision. These methods were validated against various benchmark and practical datasets. For example, in [73], an approach that combined GA and CNNs was proposed for non-invasive classification of glioma using magnetic resonance imaging (MRI). The structure of the deep network was evolved by using GA, and in this way, a typical selection of neural network for a particular problem based on trial and error was avoided. The proposed method was validated against two case studies. In the first case study, 90.9% classification accuracy for three glioma grades

was obtained. In the second study, glioma, meningioma, and pituitary tumor types were classified with 94.2% accuracy. A framework for automating the CNN design by using EAs implemented in the Python programming environment was shown in [74]. The framework utilized two EAs: GA and grammatical evolution. It was proven that the proposed framework managed to obtain better results than many state-of-the-art CNN architectures.

Another interesting method was presented in [20], where the authors proposed the gradient-priority PSO (GPSO) with gradient penalties for selecting the CNN structure. The proposed method was tested by using three types of emotion states for each subject and simultaneously collecting EEG signals corresponding to each emotion category. Experimental results proved that the GPSO managed to obtain competitive classification performance over the dataset. An orthogonal learning particle swarm optimization (OLPSO) algorithm that optimized hyperparameters' values for VGG16 and VGG19 CNNs that have been developed for the task of plant disease diagnosis by classifying the leaf images as healthy and unhealthy was presented in [14]. By performing practical simulations, the authors proved that their approach managed to obtain higher performance than other methods that were tested for the same datasets. Another state-of-the-art method was presented in [75]. The developed method fell into the group of hybrid stochastic training algorithms. The authors employed the relatively novel grasshopper optimization algorithm (GOA) for multilayer perceptrons (MLPs) neural networks (GOAMLP). The GOAMLP model was applied to five practical datasets: Parkinson's, diabetes, breast cancer, coronary heart disease, and orthopedic patients.

In [13], a novel variant of the PSO algorithm, named cPSO-CNN, was proposed for determining the CNN architecture. The cPSO-CNN method enhanced the canonical PSO's exploration ability and redefined PSO's scalar acceleration coefficients as vectors to better adapt to the variant ranges of CNN hyperparameters. It was proven that the cPSO-CNN established competitive performance in terms of accuracy and overall computation costs when compared to seven other state-of-the-art methods. One more PSO-based approach used for defining CNN architecture that is worth mentioning was presented in [19]. By using their PSO adaptation, the authors managed to obtain better parameter settings than those found by the AlexNet configuration for five different image datasets.

2.2. Mathematical model

In this part of this section, we establish the context of this research. This research aims to develop an efficient method for automatically optimizing the hyperparameters and designing a CNN structure for a specific image classification task.

More details regarding the CNNs layers, evolution, and functional principles can be retrieved from [20,76].

The performance of the CNN architecture highly depends on the hyperparameters' values. Different real-world visual tasks require different network structures; thus, each network has to be configured independently, and the appropriate set of hyperparameters varies from task to task. Finding the right set of hyperparameters for a given problem is a very time-consuming process, and it requires knowledge from the domains of deep learning and CNNs. To address this issue, we propose an enhanced swarm intelligent-based method to search automatically for the right set of hyperparameters to build a structure that will produce high classification accuracy and that can be used by users who do not have prior knowledge from the area and can apply it to a specific problem of interest.

Since the optimization of all CNN hyperparameters would be a very time and computational resource consuming process, in the proposed research, we optimized the following CNN hyperparameters: the number of convolution layers, the number of filters, the filter size, the number of FC layers, and the hidden units in the FC layer.

In this work, the optimal CNN architecture was encoded as a set of hyperparameters, which is defined as:

$$h = \{h_{conv}, h_{fc}\} \tag{1}$$

where h_{conv} denotes the set of convolutional layer parameters and h_{fc} is the set of FC layer parameters.

The set of convolutional layer parameters is defined as:

$$h_{conv} = \{C_0, ..., C_{n-1}\} \tag{2}$$

where the number of convolutional layers is denoted by n.

Each convolution layer in the ith layer contains a two-tuple:

$$C_i = (k_{count}, k_{size}) \tag{3}$$

where the pairs k_{count} and k_{size} are the number of filters and the size of the filters of the ith layer, respectively.

The set of the fully-connected layers is formulated as:

$$h_{fc} = \{s_0, s_{m-1}\} \tag{4}$$

where s represents the hidden unit number in the FC layer of the ith layer and m is the number of FC layers.

By $\mathcal{H}$, we denote the possible set of CNN architecture configurations. The aim is to find a configuration $h \in \mathcal{H}$ that minimizes the error rate of a given task. Thus, in our research, we took the classification error rate (*err*) as an objective function.

This problem belongs to the class of NP hard challenges, and therefore, exact (deterministic) optimization algorithms are not applicable. However, swarm intelligence algorithms showed very promising results in solving NP hard optimization problems, and that was the basic reason why we implemented and adapted swarm intelligence algorithms for CNN hyperparameters' optimization.

3. Proposed swarm intelligence approaches

In this section, we first describe the original tree growth algorithm [39] followed by our proposed enhanced version. Afterwards, the original procedure of the second metaheuristic algorithm, the firefly algorithm [31], is described along with our proposed hybrid firefly algorithm.

3.1. Original tree growth algorithm overview

The tree growth algorithm (TGA) is a novel nature-inspired metaheuristic optimization algorithm, proposed by Cheraghalipour et al. in 2018. The algorithm was tested on 30 standard benchmark functions, three engineering benchmarks, and two industrial engineering optimization problems; the obtained results showed the robust performance of the algorithm.

TGA was inspired by the competition of trees in a forest. Trees are competing for absorbing sunlight and food. Two main phases of the algorithm are exploration and exploitation, respectively. In the exploration phase, the trees move towards the sunlight, which allows exploring new places. In the exploitation phase, the trees are already satisfied with the light; hence, they move toward better nutrients in the roots, analogous to moving toward the local or global optimum. The population of trees is divided into four groups in the forest:

1. Group of the best trees;
2. Group of trees that are competing for light;
3. Remove or replace group;
4. Reproduction group.

The first group of trees is satisfied with the light source, and they compete for a food source. In the second group, trees move between the two closest best solutions to compete over light. In the third group, the worst trees are removed and replaced by new trees. In the last group, new plants are created by the best trees.

In the first step, the algorithm randomly generates the initial population of trees (solutions) within the lower and upper bounds; and the fitness value is evaluated for each solution. The initial population is generated according to the following formula:

$$x_{i,j} = min_j + rand \cdot (max_j - min_j),$$ (5)

where $x_{i,j}$ represents the jth parameter of the ith solution in the population, *rand* is a random number drawn from a uniform distribution, and min_j is the lower bound and max_j the upper bound of the jth parameter.

In the next step, the population is sorted according to the value of the fitness, and the current best solution at the jth iteration is found. The global best solution is denoted by T_{GB}^j.

The best solutions are assigned to the first group (N_1), and the solutions form this population conduct the local search by using the following formula:

$$T_i^{j+1} = \frac{T_i^j}{\theta} + r \times T_i^j$$ (6)

where T_i^{j+1} is the new ith solution, T_i^j represents the ith solution in the jth iteration, the reduction rate of power is denoted by θ, and r is a random number between zero and one.

The greedy selection is applied to the newly generated solutions; if the new solution has a better fitness value than the current solution, the new solution replaces the current one; otherwise, we keep the current solution for the next generation.

The second-best solutions are assigned to N_2 subpopulation. Each solution from the N_2 group should be moved between the two closest solutions (from the first and second subpopulation) under different α angles. The distance between the solutions is calculated by the Euclidean distance as follows:

$$d_i = \left(\sum_{i=1}^{N_1+N_2} (T_{N_2}^j - T_i^j)^2 \right)^{\frac{1}{2}}$$ (7)

$$d_i = \begin{cases} d_i & \text{if } T_{N_2}^j \neq T_i^j \\ \infty & \text{if } T_{N_2}^j = T_i^j \end{cases}$$ (8)

where d_i represents the distance of the ith solution, the current tree is denoted by $T_{N_2}^j$, and the ith solution in the population is denoted by T_i^j.

After calculating the distance, the current solution moves toward the two nearest solutions with the minimal distance. The linear combination of the two closest solutions is calculated as follows:

$$y = \lambda x_1 + (1 - \lambda)x_2$$ (9)

where x_1 and x_2 are the first and second nearest solutions and λ is the control parameter, the values of which are between zero and one.

To move the solution between the nearest solutions under different α angles, the following formula is used:

$$T_{N_2}^{j+1} = T_{N_2}^j + \alpha_i y$$ (10)

where the current solution is denoted by $T_{N_2}^j$; the angle of the ith solution is denoted by α_i, and its value is between zero and one; y is the linear combination of the two adjacent solutions.

The third subpopulation N_3 contains the worst solutions from the population. These solutions are replaced by newly generated random solutions. N_3 is calculated as follows:

$$N_3 = N - N_1 - N_2 \tag{11}$$

where the population size is denoted by N; N_1 and N_2 are the first and second subpopulations, respectively.

In the next step, the new population (N) is created by summing the first three groups (N_1, N_2, and N_3).

$$N = N_1 + N_2 + N_3 \tag{12}$$

The final group N_4 consists of randomly generated new solutions. Each solution in N_4 is modified by using the mask operator with respect to the best solution from the first group (N_1) and the modified solutions merged with the population. The new population is sorted according to the fitness value, and the best N solutions are selected for the next iteration. The process is repeated until the termination criteria are met. In the final step, the best optimal solution is selected.

3.2. Enhanced tree growth algorithm

In every metaheuristics algorithm, it is essential to maintain the trade-off between the exploration and exploitation processes. The identified deficiencies of the original TGA metaheuristics refer to the insufficient exploitation power and to the inappropriate balance between exploitation and exploration. More about TGA's deficiencies can be found in [77,78].

In this work, we propose a modified TGA algorithm that enhances the exploitation capability of the original TGA with an additional operation in the algorithm's procedure. The proposed novel enhanced algorithm is named exploitation enhanced TGA (EE-TGA). The modification speeds up the convergence rate of the algorithm, and at the same time, the search process reaches the global optima faster.

In the best solution group (N_1), the first 40% of solutions in the subpopulation are scattered around the current best solution by using a normal distribution. The solution's position is updated according to the following formula:

$$T_i^{j+1} = T_{GB}^j + N(T_{GB}^j, \sigma_t^2) \tag{13}$$

where T_i^{j+1} is the new solution; T_{GB}^j denotes the current best solution; $N(\cdot)$ is the normal distribution, where the mean is the current best solution and the variance at iteration t is σ^2. The variance determines the step size, and it is decreasing over the course of the iteration.

The distribution around the current best solution is calculated as follows:

$$N(T_i^{j+1}; T_{GB}^j, \sigma_t^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(T_i^j - T_{GB}^j\right)^2 \Big/ 2\sigma^2} \tag{14}$$

Due to the implementation of the normal distribution, the new solutions are closer to the best solution, which yields increasing the possibility to find the global optimum faster.

As mentioned above, the step size (σ) is decreasing as the algorithm approaches the maximum iteration and gets closer to zero, $\sigma_t \to 0$. The step size updates at every iteration t according to the following equation:

$$\sigma_{t+1} = \sigma_0 - \sigma_0 t / t_{max} \tag{15}$$

where t is the current iteration, σ_0 indicates the initial step size, and t_{max} is the maximum iteration.

The pseudocode of the proposed EE-TGA optimization algorithm is shown in Algorithm 1.

Algorithm 1 Pseudocode of the exploitation enhanced TGA (EE-TGA).

Initialization:
Randomly generate the initial population of N trees x_i, $i = 1, 2, \ldots, n$
Define the initial step size σ
Set the number of maximum iterations (t_{max})
Set the current iteration (t) to 1
Calculate the fitness of each solution
Sort the population according to the fitness value
while t_{max} **do**

    for $i = 1$ to $ceil(N_1 * 0.4)$ **do**

        Perform local search by using Equation (13)
        If the new solution has a better fitness value than the current solution, replace the current

        solution with the new one
        Update the step size by utilizing Equation (15)
    end for
    for $i = ceil(N_1 * 0.4)$ to N_1 **do**

        Perform local search by using Equation (6)
        If the new solution has a better fitness value than the current solution, replace the current

        solution with the new one
    end for
    for $i = N_1$ to N_2 **do**

        Calculate the distance between the solutions by using Equation (7) and Equation (8)
        Move solutions between the two nearest solutions by using Equation (9) and Equation (10)
        If the new solution has a better fitness value than the current solution, replace the current

        solution with the new one
    end for
    Remove N_3 worst solutions, and replace with new random solutions
    Create new population N by using Equation (12)
    Generate N_4 subpopulation and apply the mask operator
    Evaluate the fitness of each solution
    Sort the population according to the fitness value
    Select N best solutions for the next iteration
end while
Return the best solution

In the presented pseudocode, t_{max} represents the maximum number of iterations in one algorithm's run.

3.3. Original firefly algorithm overview

The firefly algorithm (FA) was proposed by Xin-She Yang in 2008. In this algorithm, the fireflies represent the solutions. The fireflies are attracted by other fireflies based on their brightness level, and each solution moves toward brighter solutions. The algorithm procedure follows three simple rules:

1. Each firefly is unisex and can be attracted by any other firefly;
2. The brightness determines the fireflies' attractiveness; as the distance become shorter, the attractiveness of fireflies increases, and they become brighter;
3. The fireflies' brightness determines the fitness function.

The solutions are generated randomly within the lower and upper bounds:

$$x_{i,j} = lb_j + rand \cdot (ub_j - lb_j) \tag{16}$$

where lb_j and ub_j are the lower and upper bound, respectively. The solution is denoted by $x_{i,j}$; and *rand* is a random number between zero and one.

How close two solutions are to each other is calculated by the Euclidean distance, as follows:

$$r_{i,j} = ||x_i - x_j|| = \sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})} \tag{17}$$

where $r_{i,j}$ indicates the distance between the *i*th and *j*th solutions and *d* denotes the dimension of the problem.

The light intensity is calculated according to the following formula:

$$I(r) = \frac{I_0}{1 + \gamma r^2} \tag{18}$$

where the source's light intensity is denoted by I_0, the light intensity is denoted by $I(r)$, where *r* indicates the distance, and γ represents the light absorption coefficient.

The attractiveness level between two solution is evaluated by the following formula:

$$\beta(r) = \frac{\beta_0}{1 + \gamma r^2} \tag{19}$$

where β indicates the attractiveness and the attractiveness at distance zero is denoted by β_0.

The solutions reach a new location in the search space according to the movement formula, which is defined as follows:

$$x_i^{t+1} = x_i^t + \beta_0 r^{-\gamma r_{i,j}^2}(x_j - x_i) + \alpha(rand - 0.5) \tag{20}$$

where x_i^{t+1} denotes the new position of the solution; x_i^t denotes the current position, which moves toward the brighter solution x_j; α is a control parameter, and its value is drawn from the Gaussian distribution; and *rand* represents a random number between zero and one.

3.4. Enhanced firefly algorithm

In the original FA, the exploitation and exploration balance mostly depends on the value of parameter α, which is in most studies dynamically adjusted during one run of the algorithm according to the following equation [49,79–81]:

$$\alpha(t) = (1 - (1 - ((10^{-4}/9)^{1/t_{max}}))) * \alpha(t-1), \tag{21}$$

where t_{max} denotes the maximum number of iterations in one run. It means that at the beginning of a run, the value of α is greater (higher exploration power), and as the search progresses during iterations, the value of α is decreased (and thus, the exploration power decreases in favor of exploitation).

Despite the fact that the original FA exhibits relatively stable and efficient exploitation, according to the results of the performed simulations on standard unconstrained benchmarks, we concluded that the exploitation process of the original FA could be further enhanced, as well as the intensification-diversification trade-off by incorporating an additional procedure in the original FA algorithm.

Moreover, the original FA approach in some runs shows the behavior of premature convergence because the whole population converges towards the current best solutions. For this reason, we enhanced the exploration of the original FA by incorporating a procedure that introduced randomness in the population.

By incorporating two procedures: one that enhances the exploitation and one that improves exploration, we created the exploitation and exploration enhanced FA (E^3-FA).

The first procedure, the one that enhances the exploitation power of the original FA, introduces a new randomization parameter in the algorithm p, which determines whether the position of a solution is being updated by Equation (20) or will be distributed around the current best solution by utilizing the normal distribution:

$$x_i^{t+1} = x_{*,j}^t + N(x_{*,j}^t, \sigma_t^2) \tag{22}$$

where x_i^{t+1} is the new solution; $x_{*,j}^t$ denotes the current best solution; $N(\cdot)$ is the normal distribution, where the mean is the current best solution and the variance at iteration t is σ^2. The variance determines the step size, and it is decreasing over the course of iterations.

The distribution around the current best solution is calculated as follows:

$$N(x_i^t; x_{*,j}^t, \sigma_t^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(x_i^t - x_{*,j}^t\right)^2 \big/ 2\sigma^2} \tag{23}$$

By incorporating Equation (23), the new solutions are closer to the best solution, which yields increasing the possibility to find the global optimum faster.

As mentioned above, the step size (σ) is decreasing as the algorithm approaches the maximum iteration and gets closer to zero, $\sigma_t \to 0$. The step size updates at every iteration t according to the following equation:

$$\sigma_{t+1} = \sigma_0 - \sigma_0 t / t_{max} \tag{24}$$

where t is the current iteration, σ_0 indicates the initial step size, and t_{max} is the maximum iteration.

The second procedure (one that enhances exploration) operates in a way that at the end of each iteration, the solutions are sorted according to their fitness value, and the last 20% of worst solutions are replaced by a new random solution utilizing the initialization formula (Equation (16)). In this way, if an algorithm converges to the wrong part of the search space, new, random solutions are generated, and the trap of being stuck in suboptimal regions is avoided.

The pseudocode of the proposed E^3-FA is given in Algorithm 2.

Algorithm 2 Pseudocode of the exploitation and exploration enhanced FA (E^3-FA).

Objective function $f(x)$
Randomly generate the initial population of N fireflies $x_i, i = 1, 2, \ldots, n$
Calculate the light intensity by $I_i = f(x_i)$
Set light absorption coefficient γ
Set the iteration counter t to zero, and define the number of maximum iterations t_{max}
Initialize the step size σ at $t = 0$
while $t < t_{max}$ **do**

  for $i = 1$ to N **do**

    for $j = 1$ to i **do**

      if $I_j > I_i$ **then**

        if $p > 0.5$ **then**

          Move solution x_i toward x_j by using Equation (20)
        else

          Update the position according to Equation (22)
          Update the step size by Equation (24)
        end if
        Evaluate new solution, and update light intensity
      end if
    end for
  end for
  Sort all fireflies according to the fitness value, and save the current best solution
  for $ceil(N * 0.8)$ to N **do**

    Generate new random solutions by Equation (16)
  end for
end while
Return the best solution

In the presented pseudocode, t_{max} represents the maximum number of iterations in one algorithm's run.

4. CNN optimization simulation environment, the algorithms' adaptations, and parameter setup

This section represents the first empirical section of this paper, and it is devoted to the simulation environment setup for CNN architecture simulations. In the first part of this section, we show the EE-TGA and E^3-FA algorithms' adaptations and the solution encoding scheme for CNN hyperparameter optimization. The second part of this section gives detailed information regarding the environment, as well as the algorithms' control parameter values that were utilized in practical CNN experiments.

The same experimental conditions and setup were used in our previous research [71,72], as well as in [17].

4.1. Solutions' encoding and algorithms' adaptations

As was already mentioned in Section 1, CNN hyperparameters that were subject to optimization in the proposed model included the number of convolution layers along with the number of kernels per layer and the kernel size and number of fully-connected (dense) layers along with the size of each fully-connected layer. We did not include the type of pooling layer, activation function, regularization

technique, and other CNN hyperparameters in the automatic search because of the limitations of available computing power.

A detailed description of the utilized mathematical model is given in Section 2.

The possible hyperparameter values that were subject to the optimization in this study were defined within the lower and upper bounds (these bounds are given in Subsection 4.2), with the aim to exclude infeasible solutions of CNN architectures and reduce the utilization of computer resources.

The population at iteration t consisted of the set of N CNN configuration solutions, and the set is defined as follows:

$$H^t = \{h_0, h_1, \ldots, h_N\} \tag{25}$$

where each configuration h consists of h_{conv} and h_{fc}.

According to the mathematical formulation of the utilized model, given in Section 2, each solution in the population (network configuration h) consisted of variable length parameters. Since we did not include continuous parameters like the learning rate, all parameters took only discrete values.

The number of solution parameters was different for each solution in the population and depended on the number of convolutional layers (where each convolutional layer was further determined by two parameters: the number of kernels and the kernel size) and the number of FC network layers. For example, if a generated solution (network configuration) consisted of three convolutional layers and four FC layers, then such a solution would be encoding with 10 discrete parameters ($3 \cdot 2 + 4$).

A visual representation of the solutions' encoding scheme is given in Figure 1.
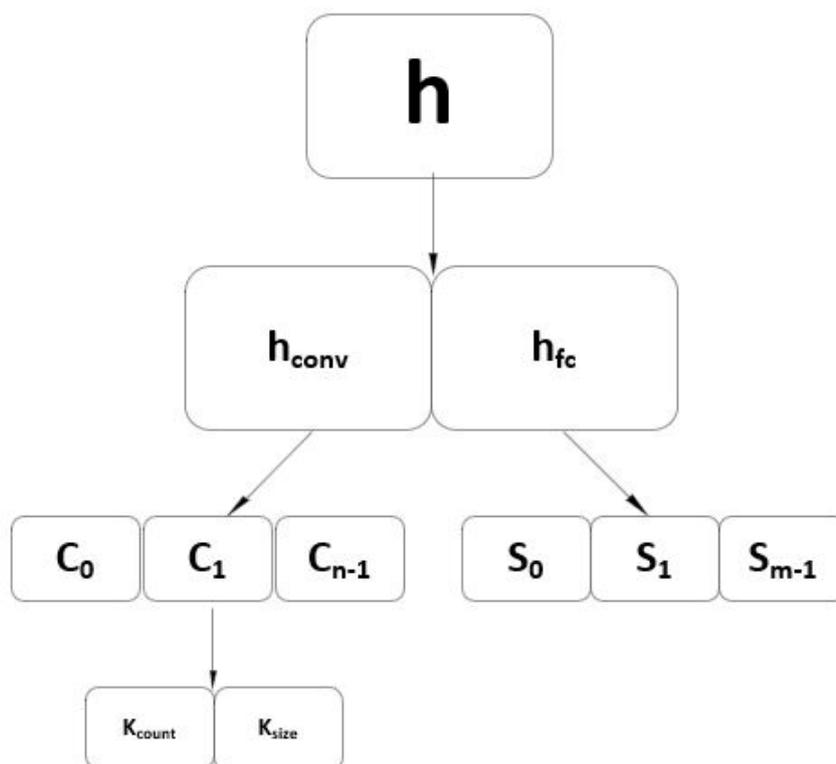
Figure 1. Solutions' encoding scheme.

To decrease the number of possible network configurations and to reduce the search domain, the convolutional layers were sorted in descending order, first concerning the filter size (k_{size}) and then based on the number of filters (k_{count}). The value of s, which denotes the FC layer size, was utilized for sorting dense layers.

In the beginning, the initial population of solutions (convolutional neural network structures) was randomly generated within the defined lower and upper bounds by using the following equation:

$$h_{i,j} = min_j + rand \cdot (max_j - min_j) \tag{26}$$

where $h_{i,j}$ represents the jth parameter of the ith solution in the population, $rand$ is a random number drawn from a uniform distribution, and min_j is the lower bound and max_j the upper bound of the jth parameter. Due to the relatively large range of parameter values, we did not adapt the proposed approaches for discrete optimization, and we just rounded the results to the nearest integer.

After generating the initial population, we evaluated the fitness value of each solution, and the current best solution in the population was determined. The fitness function that was utilized was simply inversely proportional to the classification error rate (*err*) of the respective CNN architecture *h*. For calculating the fitness of solution *i*, the following expression is used:

$$F(h_i) = \frac{1}{1 + |err_i|} \tag{27}$$

where the fitness function of CNN structure h_i is denoted by $F(h_i)$ and err_i represents the objective function of the concerned structure.

The fitness calculation process utilized much computational resource because each fitness evaluation required a CNN to be developed, trained, and validated against the dataset. For this reason, within the evaluation procedure of both devised approaches, EE-TGA and E^3-FA, we used classes from the deep learning programming library Deeplearning4j in order to generate and validate a network of a certain structure.

The high-level procedure of the CNN framework for both proposed algorithms is given in Algorithm 3.

Algorithm 3 High-level procedure of the CNN design framework.

Initialization
while $t < t_{max}$ **do**

   for $i = 1$ to N **do**

      Perform *evaluation* procedure
   end for
   Sort all solutions according to fitness
   Determine the current best solution
   Perform exploitation phase
   Perform exploration phase
end while
Return the best solution

In the presented pseudocode, t_{max} represents the maximum number of iterations in one algorithm's run.

The CNN optimization frameworks based on the EE-TGA and E^3FA are named EE-TGA-CNN and E^3FA-CNN, respectively.

4.2. Parameter setup

In the following few paragraphs, we briefly describe the EE-TGA and E^3-FA control parameter values that were utilized in the simulation with CNN hyperparameter optimization. The same environment was utilized in [71,72].

In EE-TGA, the population size N was set to eight CNN structures, and subpopulations N_1 and N_2 were set to three. The algorithm was iterated in 20 iterations (t_{max} = 20). In each iteration, two

worst solutions were removed from the population, and instead of them, two new random solutions were generated $(N3 = N - (N1 + N2))$. In the fourth group (N_4), in each iteration, one new solution was generated and modified with the max operator. The values of the control parameters λ and θ were set to 0.5 and 0.2, respectively. The initial step size σ_0 was set to three.

In the second enhanced metaheuristic, E^3-FA, the initial population size, maximum iteration, and the initial step size had the same values as in EE-TGA. The α randomization parameter was set to 0.5, the light absorption coefficient γ to 1.0, and the attractiveness at a distance of zero to 0.2. The metaheuristic control parameter setup of both enhanced algorithms is summarized in Table 1. The control parameters for the original TGA and FA were the same as for the enhanced versions, except the step size, which was not used in the basic TGA and FA implementations.

Table 1. Metaheuristic control parameters. EE-TGA, exploitation enhanced tree growth algorithm; E^3-FA, exploitation and exploration enhanced firefly algorithm.

Algorithm	Parameter	Value(s)
EE-TGA	Population size (N)	8
	Sub-population size (N_1)	3
	Sub-population size (N_2)	3
	Number of discarded solutions (N_3)	2
	Number of novel solutions (N_4)	1
	Reproduction rate (θ)	0.2
	Control parameter (λ)	0.5
	Maximum iteration number (t_{max})	20
	Initial step size (σ_0)	3
E^3-FA	Population size (N)	8
	Initial value for randomization parameter (α)	0.5
	Light absorption coefficient (γ)	1.0
	Attractiveness at $r = 0$ (β_0)	0.2
	Maximum iteration number (t_{max})	20
	Initial step size (σ_0)	3

The search space of the number of convolutional layers was set within the range between zero and six; the lower and upper bounds of the number of fully-connected layers were set within the interval $[1, 4]$; and the lower and upper boundaries of the filter size were adjusted between one and eight, respectively. The number of filters was set between one and 128, and the hidden unit number in the FC layer was bounded between values of 16 and 2048.

We restricted the possible set of CNNs hyperparameters' values to reduce the search space and to exclude the most infeasible CNN architectures. Lower and upper bounds for each hyperparameter were determined empirically and in this way, the proposed framework was much more efficient.

Moreover, we wanted to start the search process by generating less complex and, thus, less expensive CNN configurations. For this reason, in the initialization phase, we set an even narrower range of possible values for the number of convolutional layers (h_{conv}) and the number of FC layers (h_{fc}). Please refer to Table 2.

Table 2. CNN hyperparameter configuration.

Parameter	Value(s)
Optimizer	Adam
Learning rate (α)	0.0001
Activation Function	ReLU
Pooling layer (max pooling)	2×2
Batch size	54
Epoch number	1
Padding	valid
Stride	1
Loss function	MSE
Number of conv layers (h_{conv})	[0,6]
Initial conv layers (h_{conv}) at $t = 0$	[1,2]
Number of FC-layer (h_{fc})	[1,4]
Initial FC-layers (h_{fc}) at $t = 0$	[1,2]
Kernel size (k_{size})	[1,8]
Number of kernels per conv layer (k_{count})	[1,128]
FC-layer size (s)	[16,2048]

With the objectives of reducing the amount of parameters (weights), the computational burden, and to control overfitting, for each convolutional layer, we used a pooling (downsampling) layer of a fixed size of 2×2 along with the max pooling option. Further, we utilized the ReLU (rectified linear unit) activation function and employed the Adam optimizer with a fixed learning rate of $\theta = 0.0001$ and the mean squared error (MSE) loss function. We used valid convolution and did not apply padding on the input volume. None of the regularization techniques were employed.

To make a better and unbiased comparative analysis, the initial CNN hyperparameter configuration was similar to the configuration in [17]. The network was trained in one epoch with a batch size of 54. The hyperparameters that were not subject to the optimization were set to fixed values, and they are presented in Table 2.

The TGA, FA, EE-TGA, and E^3-FA metaheuristics were developed in Java SE 10 (18.3). The framework was developed and implemented by using deep learning programming library Deeplearning4j (https://deeplearning4j.org/), which was written for Java and Java Virtual Machine (JVM). To improve the execution speed of the performed study, due to the large utilization of computer resources, we adapted the framework for execution on a graphical processing unit (GPU) using NVIDIA's [TM]CUDA parallel computing platform and programming model. All experiments were performed on the computer platform with six NVIDIA GTX 1080 GPUs, each having 2560 [TM]CUDA cores.

5. Experimental results and discussion

The experimental section is divided into two parts. In the first part, we show the simulation results of our proposed metaheuristics for unconstrained benchmark functions along with comparative analysis with the basic versions, the control parameters' setup, and the mathematical formulations of the test functions.

When a modified method is created, its effectiveness should first be validated against a wider set of benchmarks, and the performance should be compared with the original method. This is the only way to establish a real improvement of the enhanced method over the original one.

In the second part, we describe the benchmark dataset that was used for the CNN hyperparameters' optimization, and we present the obtained experimental results along with a comparative analysis with other, similar methods, the results of which are retrieved from the modern computer science literature.

5.1. Performance evaluation of EE-TGA and E^3-FA on standard unconstrained benchmarks

For the sake of easier following, this subsection is also divided into two parts. The first part presents the enhanced tree growth algorithm (EE-TGA) performance evaluation, and the second part shows the evaluation of the enhanced firefly algorithm (E^3-FA) for standard unconstrained benchmarks along with respective comparative analysis with original versions.

5.1.1. Performance evaluation of EE-TGA

For the purpose of the EE-TGA evaluation, we tested the algorithm on ten standard benchmark functions. The details of the functions and their characteristics are described in Table 3 and Table 4, respectively.

Table 3. Benchmark function details (EE-TGA simulations).

ID	Function Name	Function Definition
F1	Ackley Function	$f(x) = -20e^{(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})} - -e^{(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))} + 20 + e^{(1)}$
F2	Becker–Lago Function	$f(x) = (|x_1| - 5)^2 + (|x_2| - 5)^2$
F3	Branin Function	$f(x) = \left(x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$
F4	Dekker–Aarts Function	$f(X) = 10^5 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4$
F5	Rastrigin Function	$f(x) = \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i) + 10]$
F6	Cosine Mixture Function	$f(x) = 0.1\sum_{i=1}^{n}\cos(5\pi x_i) - \sum_{i=1}^{n}x_i^2$
F7	Gulf Research Problem	$f(X) = \sum_{j=1}^{m}\left(e^{-\frac{\left|y_j - x_2\right|^{x_3}}{x_1}} - t_j\right), \quad m = 99$ where $t_j = \frac{j}{100}, y_j = 25 + \sqrt{\left[-50\log(t_j)\right]^2}$
F8	Modified Rosenbrock Function	$f(X) = 100\left(x_2 - x_1^2\right)^2 + \left[6.4(x_2 - 0.5)^2 - x_1 - 0.6\right]^2$
F9	Rosenbrock Function	$f(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
F10	Schwefel 2.26 Function	$f(x) = \sum_{i=1}^{n}[-x_i \sin(\sqrt{|x_i|})]$

The parameters of the EE-TGA were set up according to the best results of the parameter tuning in the paper [39] to make an unbiased comparison. The maximum iteration was set to 250 for the problems with a large dimension. In contrast, for the problems with a small dimension, the maximum iteration was set to 50, and we obtained the test results from 30 independent trials of the algorithms. The population size N was set to 100, and the subpopulations N_1, N_2, and N_4 are set to 20, 20, and 30, respectively. The control parameter λ was set to 0.1, and θ had a value of 0.8. Specific control parameters of the EE-TGA were set as shown in Table 1.

The comparison between the original TGA and the EE-TGA is given in Table 5, where the results of the TGA algorithm were taken from the original TGA paper [39]. We took the best, mean, and standard deviation as performance metrics. In the presented table, the bold values indicated better results. We note that we have also performed tests with the basic TGA and that we got the same results as in [39].

Table 4. Benchmark function characteristics (EE-TGA simulations).

ID	Modality	Dimension	Input Domain	Global Minimum
F1	Multimodal	30	$[-30,30]$	$f(x^*) = 0$ at $x^* = (0, ..., 0)$
F2	Multimodal	2	$[-30,30]$	$f(x^*) = 0$ at $x^* = \pm 5$
F3	Multimodal	2	$-5 \leq x_1 \leq 10$	$f(x^*) = 0.39788735772973816$ at
			$0 \leq x_2 \leq 15$	$x_i^* \approx (-\pi, 12.275), (\pi, 2.275), (9.42478, 2.475)$
F4	Multimodal	2	$[-20,20]$	$f(x^*) = -24777$ at $x_i^* = (0, \pm 15)$
F5	Multimodal	10	$[-5.12,5.12]$	$f(x^*) = 0$ at $x^* = (0, ..., 0)$
F6	Unimodal	2,4	$[-1,1]$	$f(x^*) = n/10$ at $x^* = n$
F7	Unimodal	3	$[0,100]$	$f(x^*) = 0$ at $x^* = (50, 25, 1.5)$
F8	Unimodal	2	$[-5,5]$	$f(x^*) = 0$ at
				$x^* = \{(0.341307503353524, 0.116490811845416), (1,1)\}$
F9	Unimodal	10	$[-30,30]$	$f(x^*) = 0$ at $x^* = (1, ..., 1)$
F10	Unimodal	10	$[-500,500]$	$f(x^*) = -418.9829 \times n$ at $x^* = (420.9687)$

Table 5. Comparative analysis of the tree growth algorithm (TGA) and the enhanced tree growth algorithm (EE-TGA).

ID	$f(x^*)$	TGA			EE-TGA		
		Best	*Mean*	*StdDev*	*Best*	*Mean*	*StdDev*
F1	0	0.00	0.00	0.00	0.00	0.00	0.00
F2	0	1.07×10^{-8}	3.70×10^{-7}	5.84×10^{-7}	**0.00**	**3.04×10^{-9}**	**1.47×10^{-9}**
F3	0.39789	0.39789	0.39789	0.00	0.39789	0.39789	0.00
F4	-24777	-24775.72	-24775.66	0.0944586327	**-24777**	**-24776.97**	**0.0139247**
F5	0	0.00	0.00	0.00	0.00	0.00	0.00
F6	0.2	0.2	0.2	0.00	0.2	0.2	0.00
F7	0	1.52×10^{-5}	7.05×10^{-2}	2.60×10^{-1}	**5.36×10^{-6}**	**1.50×10^{-4}**	**5.72×10^{-2}**
F8	0	0.00	2.23×10^{-3}	3.58×10^{-3}	**0.00**	**0.00**	**0.00**
F9	0	0.56	0.8231	0.3419	**0.00**	**0.2159**	**0.1510**
F10	-418.9829	-418.9702	-418.9292	0.23796	**-418.9829**	**-418.9829**	**0.00**

From the results presented in Table 5, it can be clearly seen that EE-TGA obtained better solution quality in most tests by improving the exploitation of the original TGA.

Since the most significant performance difference between EE-TGA and the original TGA could be noticed in the *F*9 benchmark, in Figure 2, we show the convergence speed graph for this problem of the best generated solutions. In the case of this test, the mean values generated by the EE-TGA were much better than the value obtained in the best run by the original TGA.
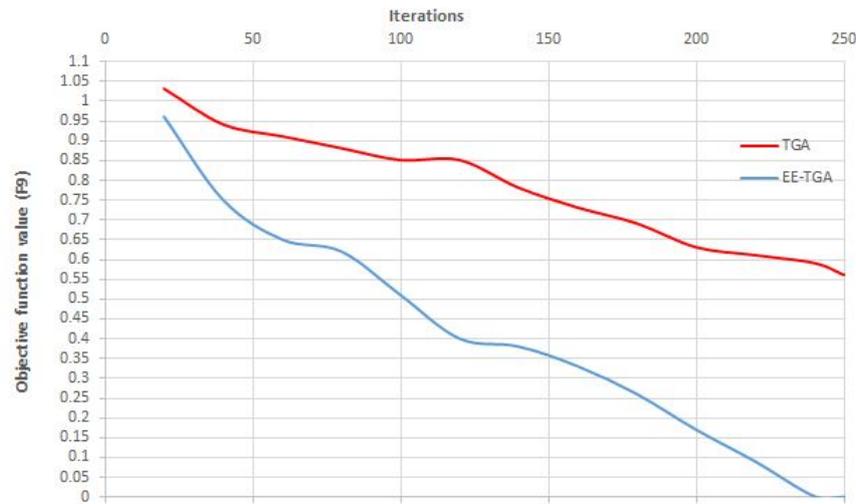
Figure 2. EE-TGA vs. TGA convergence speed graph for the F9 benchmark.

5.1.2. Performance evaluation of E^3-FA

For the purpose of the E^3-FA evaluation, we tested the algorithm on eight standard benchmark functions. The details of the functions and their characteristics are described in Table 6 and Table 7, respectively.

Table 6. Benchmark function details (E^3-FA simulations).

ID	Function Name	Function Definition
F1	Michalewicz Function	$f(x) = -\sum_{j=1}^{n} \sin\left(x_j\right) \left[\sin\left(\dfrac{jx_j^2}{\pi}\right)\right]^{2m}, \quad m = 10$
F2	Rosenbrock Function	$f(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
F3	De Jong (Sphere) Function	$f(x) = \sum_{i=1}^{n} x_i^2$
F4	Ackley Function	$f(x) = -20e^{(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2})} - -e^{(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))} + 20 + e^{(1)}$
F5	Rastrigin Function	$f(x) = \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i) + 10]$
F6	Easom Function	$f(x) = -\cos(x_1)\cos(x_2) e^{\left[-(x_1-\pi)^2 - (x_2-\pi)^2\right]}$
F7	Griewank Function	$f(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$
F8	Shubert Function	$f(x) = f(x_1, ..., x_n) = \prod_{i=1}^{n}\left(\sum_{j=1}^{5} cos((j+1)x_i + j)\right)$

Table 7. Benchmark function details (E^3-FA simulations).

ID	Modality	Dimension	Input domain	Global minimum
F1	Multimodal	2	$[0, \pi]$	$f(x^*) = -0.8013$ at $x^* = (2.20, 1.57)$
F2	Unimodal	16	$[-30, 30]$	$f(x^*) = 0$ at $x^* = (1, ..., 1)$
F3	Unimodal	256	$[-100, 100]$	$f(x^*) = 0$ at $x^* = (0, ..., 0)$
F4	Multimodal	128	$[-30, 30]$	$f(x^*) = 0$ at $x^* = (0, ..., 0)$
F5	Multimodal	2	$[-5.12, 5.12]$	$f(x^*) = 0$ at $x^* = (0, ..., 0)$
F6	Unimodal	2	$-10 \leq x_i \leq 10, i = 1, 2$	$f(x^*) = -1$ at $x^* = \pi$
F7	Multimodal	2	$[-600, 600]$	$f(x^*) = 0$ at $x^* = (0, ..., 0)$
F8	Multimodal	2	$[-10, 10]$	$f(x^*) = -186.7309$

The parameter configuration of E^3-FA was conducted based on the suggested control parameter values in [31]. The simulation of E^3-FA was conducted in 100 independent runs, and the maximum number of iterations was set to 100. The population size was set to 40, α to 0.2, the light absorption coefficient (γ) to 1, and $\beta_0 = 1$. The E^3-FA specific parameters were set as shown in Table 1.

Sample plots of the benchmark functions are depicted in Figure 3.

(a) Sphere

(b) Rosenbrock

(c) Rastrigin

(d) Schwefel

(e) Griewank

(f) Ackley
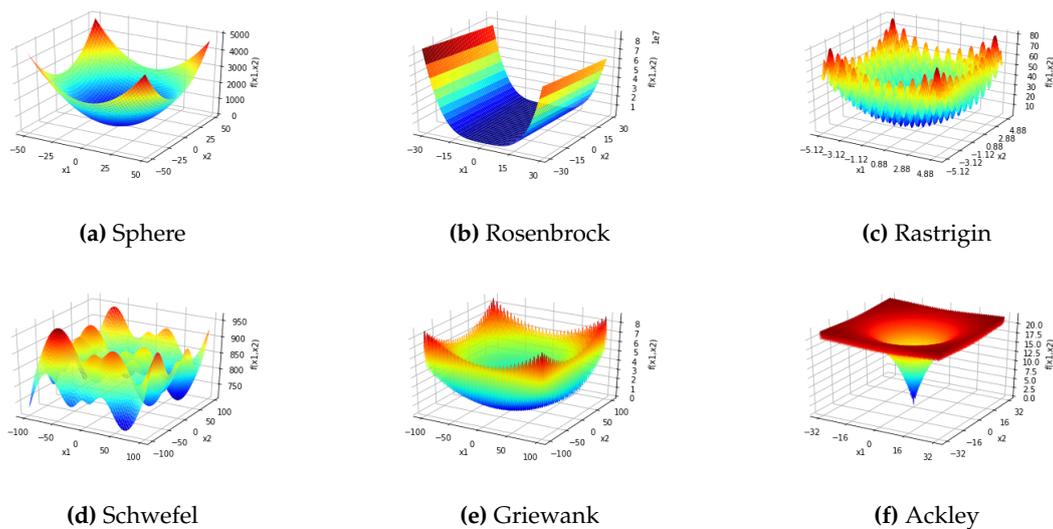
Figure 3. Benchmark function plots.

The comparison of the FA and E^3-FA is presented in Table 8, where the results of FA were taken from [82]. However, after conducting the simulations of the benchmark functions with FA, we got the same results as in [82]. As in the experiment with EE-TGA, we took the best, mean, and standard deviation as the performance metrics. The bold values in Table 8 refer to the best values.

Table 8. Comparative analysis of the firefly algorithm (FA) and the enhanced firefly algorithm (E^3-FA).

ID	$f(x^*)$	FA			E^3-FA		
		Best	*Mean*	*StdDev*	*Best*	*Mean*	*StdDev*
F1	-1.8013	-1.8013	-1.8013	0.00	-1.8013	-1.8013	0.00
F2	0	0.00	0.00	0.00	0.00	0.00	0.00
F3	0	1.24×10^{-4}	1.32×10^{-4}	3.87×10^{-5}	$\mathbf{1.52 \times 10^{-5}}$	$\mathbf{5.61 \times 10^{-5}}$	$\mathbf{2.49 \times 10^{-6}}$
F4	0	3.23×10^{-5}	5.14×10^{-5}	1.12×10^{-5}	$\mathbf{1.97 \times 10^{-5}}$	$\mathbf{3.48 \times 10^{-5}}$	$\mathbf{1.07 \times 10^{-6}}$
F5	0	4.48×10^{-11}	0.895	1.28	$\mathbf{0.00}$	$\mathbf{1.82 \times 10^{-3}}$	$\mathbf{5.25 \times 10^{-2}}$
F6	-1	-1	-3.33×10^{-1}	0.50	-1	$\mathbf{-1}$	$\mathbf{0.00}$
F7	0	1.71×10^{-8}	5.93×10^{-3}	4.23×10^{-3}	$\mathbf{0.00}$	$\mathbf{5.98 \times 10^{-6}}$	$\mathbf{1.62 \times 10^{-4}}$
F8	-186.7309	-186.7309	-186.7309	0.00	-186.7309	-186.7309	0.00

Similar to the case of EE-TGA simulations, based on the results presented in Table 8, the E^3-FA approach managed to establish better exploitation and exploration than the original FA and, in most benchmark functions, obtained better results.

The comparative analysis of convergence speed between the FA and E^3FA for *F5* benchmark is given in Figure 4. From the presented figure, it can be seen that E^3FA managed to obtain the optimum solution somewhere between the 80th and 85th iteration.
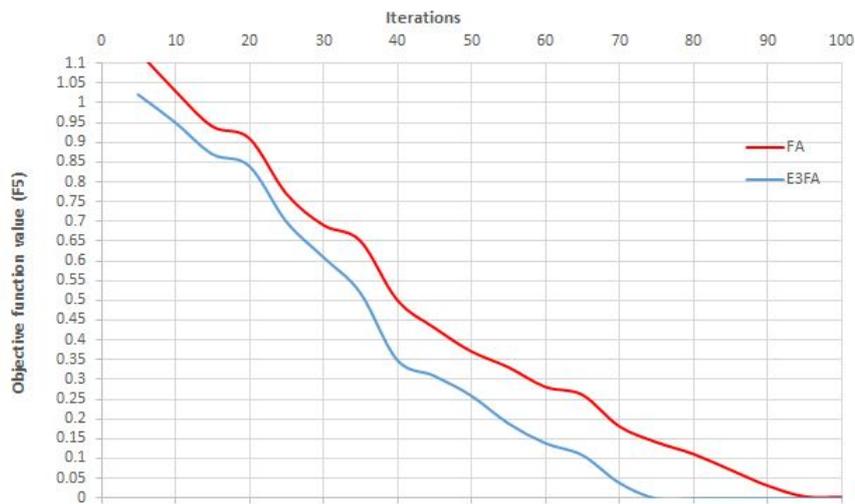
Figure 4. E^3FA vs. FA convergence speed graph for the F5 benchmark.

5.1.3. EE-TGA and E^3FA benchmark simulations' overall conclusion

Based on the experimental output of the unconstrained functions' simulations, it could be concluded that in the cases of both enhanced implementations, the algorithms outperformed the original metaheuristics. In [31,39], TGA and FA were compared to other metaheuristic algorithms, and comparing those algorithms to our proposed approaches EE-TGA and E^3-FA, it could be concluded that our algorithms outperformed most of the metaheuristics used for comparison purposes in [31,39].

In the case of EE-TGA, the global optimum was reached at seven functions. On the other hand, the TGA algorithm reached the global optimum only at five functions out of ten. The second enhanced algorithm, E^3-FA, reached the global optimum on four unconstrained functions, while the original FA managed to do the same for only three functions. Hence, EE-TGA had a 63.6% success rate at reaching the global optimum versus 45.5% fir TGA. E^3-FA reached the global optimum in 50% and the original FA in 38% test cases.

5.2. Convolutional neural network hyperparameter optimization experiments

In the second part of the experimental section, first we describe the benchmark dataset used for the purpose of CNN hyperparameter optimization and architecture design, then the simulation results are presented along with the comparative analysis with similar methods.

5.2.1. Benchmark dataset

The TGA, FA, and the proposed EE-TGA and E^3-FA methods were tested and validated on the MNIST benchmark dataset [70]. The MNIST database is a large database of handwritten digits from zero to nine; each image is in grayscale; and it consists of 28×28 pixels. The dataset consists of 70,000 labeled samples; in the experiment, 50,000 examples were used for training, 10,000 examples for validation, and 10,000 examples for testing purposes. The grayscale images that had pixel values between zero and 255 were normalized in the range from zero to one. Samples of the digit images are shown in Figure 5.
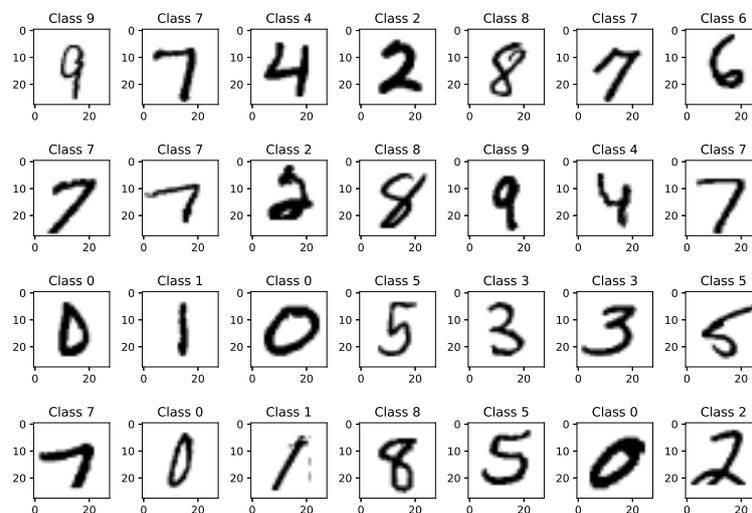
Figure 5. Samples of the MNIST dataset

5.2.2. Convolutional neural network hyperparameter optimization results and analysis

After testing the algorithms on the unconstrained benchmark functions, all four metaheuristics were applied to the CNN hyperparameter optimization problem. The performance of the method could not be assessed by the result of a single test, due to the random character of EE-TGA and E^3-FA. More than one run with independent random population initialization should be made in order to estimate the performance of the introduced approaches. Accordingly, in this study, the results were collected in 10 independent experiments, where for each run, different pseudo-random seeds were employed. Within the each run, the best CNN structure was generated in 20 iterations. The best reported result was taken as the network configuration (solution) that obtained the lowest classification test error from all 10 runs.

The algorithms' control parameters, as well as the CNN hyperparameter range of values that were taken in the simulation are presented in detail in Subsection 4.2.

As already stated, in all conducted experiments, the objective function that was subject to the process of optimization was the classification test error rate.

The results of both proposed metaheuristics (EE-TGA and E^3FA) were compared to the peer competitors that were tested under the same simulation conditions [17], the original implementations of the TGA [71] and FA [72] metaheuristics and with the same state-of-the-art network configurations that were manually designed.

In [17], the authors proposed a CNN design framework based on the evolutionary algorithm: evolutionary-based hyper-parameter optimization for independent CNNs (IEA-CNN). Furthermore, the authors developed an extension of their framework towards a joint optimization of a committee of CNNs with the goal of leveraging specialization and cooperation between individual networks (joint optimization for a committee of multiple CNNs (CEA-CNN)). A committee represents a set of K trained CNNs where the classification is performed by merging the individual CNN scores. In the proposed approach [17], the average of the decision values over K CNNs was employed.

In all experiments, the committee size was set to 34 ($K = 34$). Moreover, the authors conducted simulations with CEA-CNN by varying penalization k and found that the best CNN configurations were generated when k was set to two. The results with k set to 1, 2, and 3 were presented in [17].

According to the results presented in [17], CEA-CNN produced larger filter sizes than both our proposed approaches, which extended from 3×3 - 4×4 to 8×8 -7×7. On the other hand, some of the best generated solutions by EE-TGA-CNN and E^3FA consisted of three convolutional layers and one fully-connected layer with the filter sizes 2×2, 3×3, and 4×4, while the number of filters per convolutional layer ranged from 42 to 128, and the number of hidden units in the fully-connected layers were between 50 and 128. The network structures were significantly less complex than the architectures in [17].

We performed careful analysis of the empirical results generated from the conducted tests and concluded that the kernel size of the convolutional layers had a very significant influence on the network performance compared to the number of filters (kernels). The details of the five best performance network architectures (solutions) from the total of 80 generated solutions (eight solutions in the population · 10 runs) are summarized in Table 9.

Table 9. Ranges of the parameters for the best network configurations (solutions).

Parameter	Best 5 solutions
Kernel 1	2-4
Output 1	43-110
Kernel 2	3-4
Output 2	96-128
Kernel 3	2
Output 3	100-120
FC Layer 1	50-128

Some of generated network architectures by TGA-EE-CNN and E^3FA-CNN are given in Figure 6.

Since the study presented in this paper was an extension of our previous study where we tested the basic TGA [71] and FA [72] metaheuristics for the same problem and under the same experimental conditions, we also included in the comparative analysis these two methods.

Within the comparative analysis, we also encompassed manually designed state-of-the-art networks, namely LeNet-5 [6], deeply supervised net [83], shallow CNN [84], recurrent CNN [85], and gated pooling CNN [86].
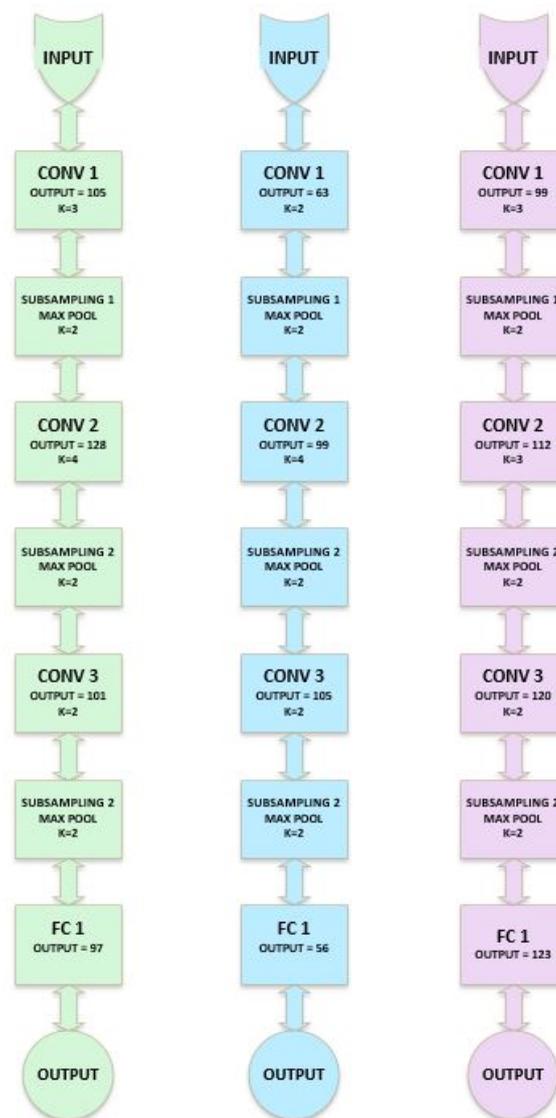
Figure 6. Some of the generated network architectures by TGA-EE-CNN and E^3FA-CNN.

The EE-TGA-CNN approach resulted in the best accuracy and lowest test error rate, that is 0.19%; E^3-FA-CNN resulted in the classification error rate of 0.21%; TGA-CNN 0.23% [71]; and the fourth approach, FA-CNN, was 0.23% [72]. IEA-CNN [17] obtained the best classification error rate of 0.34%, while CEA-CNN [17] generated the best architecture with a classification error rate of 0.24% with penalization k set to two. With k set to one and three, CEA-CNN managed to establish classification error rates of 0.26% and 0.28%, respectively [17].

On the other side, some manually designed CNN configurations established a much higher classification error rate. The highest error rate of 0.95% was produced by LeNet-5 [6], while the deeply supervised net [83] and shallow CNN [84] obtained error rates of 0.39% and 0.37%, respectively. The recurrent CNN, which managed to establish an error rate of 0.31%, and the gated pooling CNN with an error rate of 0.34% generated better results than the IEA-CNN approach.

In comparison to the peer competitors and other architectures that were manually designed, besides the high classification accuracy, the proposed enhanced metaheuristic techniques (TGA-CNN and E^3FA-CNN) had fewer parameters and were computationally less expensive.

The results of the proposed methods with other comparable techniques that utilized metaheuristics algorithms and manually designed state-of-the-art architectures are presented in

Table 10. We note once again that the best network configuration (solution) from all 10 independent runs is reported.

Table 10. Comparative analysis with other metaheuristics. IEA-CNN, evolutionary-based hyper-parameter optimization for independent CNNs ; CEA-CNN, joint optimization for a committee of multiple CNNs.

Method	Test Error Rate (%)
LeNet-5 [6]	0.95
Deeply Supervised Net [83]	0.39
Shallow CNN [84]	0.37
Recurrent CNN [85]	0.31
Gated Pooling CNN [86]	0.29
IEA-CNN [17]	0.34
CEA-CNN, k=1 [17]	0.26
CEA-CNN, k=2 [17]	0.24
CEA-CNN, k=3 [17]	0.28
FA-CNN [72]	0.23
TGA-CNN [71]	0.23
E^3-FA-CNN	0.21
EE-TGA-CNN	**0.19**

A visual representation of the comparative analysis is depicted in Figure 7. For the sake of better visibility, CEA-CNN results with *k* set to one and three are excluded from the graph.
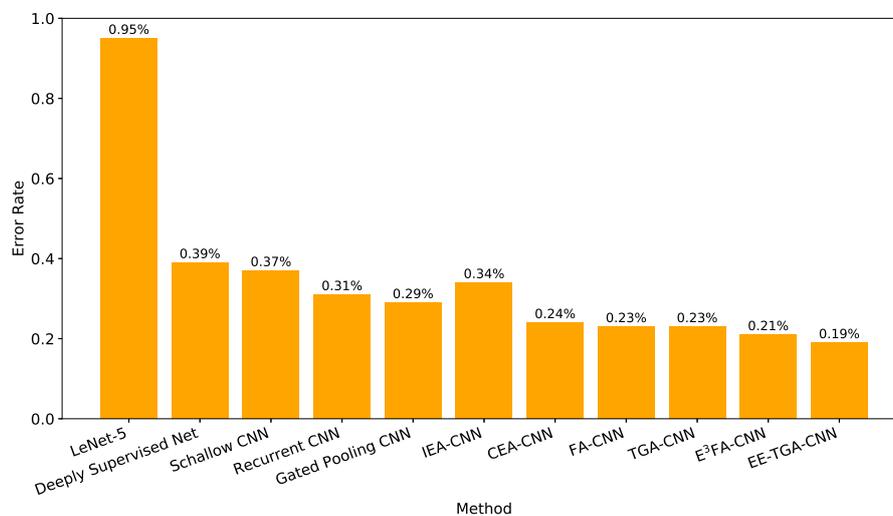
Figure 7. Visual representation of the error rates by different methods.

Since we implemented two methods in our current study and two in previous studies [71,72] and with the goal of establishing the visual representation of the quality of the proposed approaches, the convergence speed graph of the accuracy for TGA-CNN, FA-CNN, EE-TGA-CNN, and E^3-FA-CNN is given in Figure 8. The results presented in the figure were taken from one of the best runs.
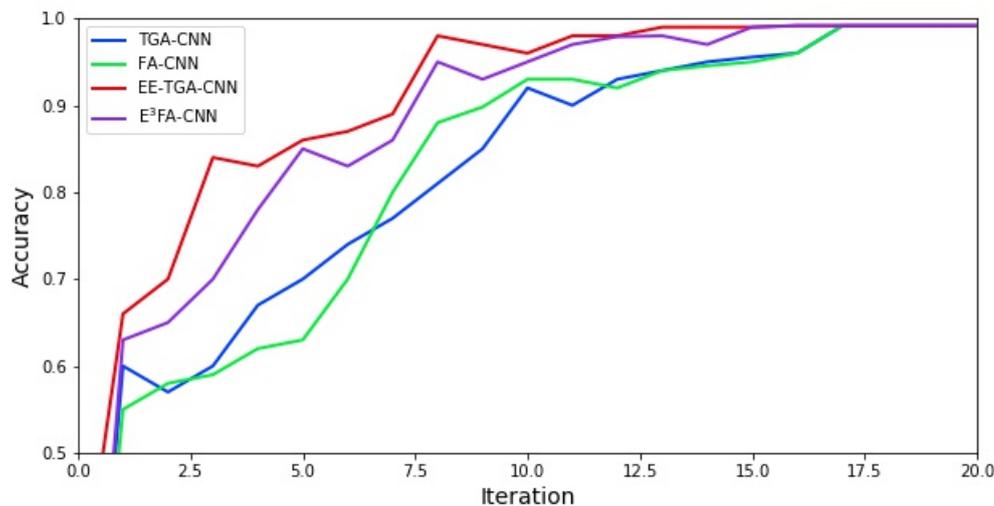
Figure 8. Convergence graph of the accuracy.

From the presented figure, it was clear that EE-TGA-CNN and E^3-FA-CNN established better convergence than the original TGA and FA algorithms. Moreover, the enhanced metaheuristics managed to converge to the best solutions after only 15 iterations.

In order to further access the solution quality of the proposed approaches, we also compared the EE-TGA-CNN and $E^3FA-CNN$ methods with another swarm intelligence-based CNN optimizer presented in [18]. However, the methods presented in [18] were tested under different experimental conditions, but for the same MNIST dataset. For this reason, comparative analysis with these approaches was not entirely realistic.

In [18], as one of the main drawbacks of CNN models, the over-fitting problem was addressed. In order to overcome this issue, the authors proposed four swarm intelligence methods (FA, BA, CS, and PSO) for proper selection of the regularization parameter dropout and reported the average accuracy over 20 independent runs.

Our first approach, EE-TGA-CNN, established an average accuracy over 10 independent runs of 99.18%, while E^3FA-CNN for the same metric obtained a result of 99.16%. At the other hand, the basic FA and TGA metaheuristics that we implemented obtained an average accuracy of 99.13% and 99.14%, respectively. We note that the results reported for the FA in [18] were different from our paper since different experimental conditions were used.

The results of the comparative analysis between EE-TGA-CNN and E^3FA-CNN and the approaches presented [18] are summarized in Table 11, while the visual representation is given in Figure 9.

Table 11. Comparative analysis with other swarm algorithms. CS, cuckoo search.

Method	Accuracy (%)
PSO [18]	99.17
FA [18]	99.16
CS [18]	99.14
BA [18]	99.13
FA-CNN [72]	99.13
TGA-CNN [71]	99.14
EE-TGA-CNN	**99.18**
E^3FA-CNN	99.16
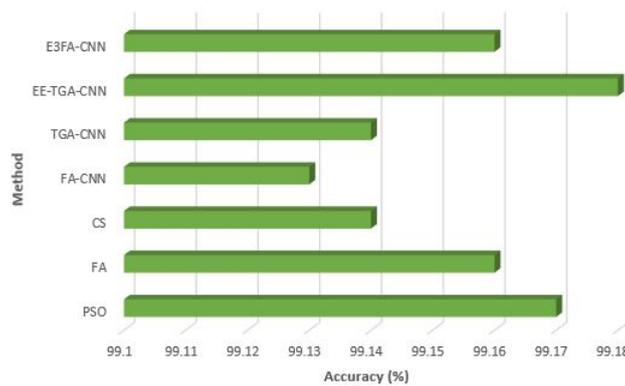
Figure 9. Comparative analysis with other swarm intelligence approaches for the accuracy.

Although it was not an entirely realistic comparative analysis, we could conclude that the average accuracy of EE-TGA-CNN was the greatest among all compared approaches.

Finally, in the experimental section of this paper, we wanted to give some information regarding the computational burden (costs) of the proposed approaches. The computational burden can be calculated in many ways, for example by determining the time and space complexity, computation time, number of fitness function evaluations, etc. The time complexity uses the number of elementary operations executed by the method, while the space complexity takes into account the memory required by an algorithm to run as a function of the input size. Computational time is a simple measurement in time units taken by the algorithm to perform, and the drawback of this approach is that it highly depends on the overall speed and performance of the computer platform.

However, in the case of metaheuristics, it is more suitable to use the number of fitness function evaluations, which was especially emphasized in the simulations that were presented in this paper. Every time the quality of the solution from the population is estimated (fitness calculation), the evaluation of a particular CNN is performed, which is a very time and resource consuming process. For this reason, inspired by [18], for calculating the computation burden of the proposed approaches, we used the metric of the number of evaluation function (fitness function) calls.

In both proposed approaches (EE-TGA-CNN and E^3FA-CNN), as well as in the TGA-CNN and FA-CNN methods, the total number of function evaluations depended on the population size N and the maximum number of iterations (t_{max}) in one run. In the initialization phase, N fitness function evaluations were performed, and then, in each iteration, N more evaluations were executed (that is a total in one run $N \cdot t_{max}$, one for each solution from the population), and finally, at the end of a run, one final evaluation was executed. For calculating the computational burden, we used the following equation, as in [18]:

$$burden = N + (N \cdot t_{max}) + 1 \tag{28}$$

Since in our simulations, we used values $N = 8$ and $t_{max} = 20$ in each run of both proposed methods, as well as in TGA-CNN and FA-CNN, 169 fitness function calls were used ($8 + 8 \cdot 20 + 1$). However, according to the simulation results, we concluded that EE-TGA-CNN and E^3FA-CNN managed to establish the best CNN accuracy after only 15 iterations (please refer to Figure 8); in this case, the computational burden was 129 evaluation function calls ($8 + 8 \cdot 15 + 1$).

The methods proposed in this paper had a higher computational burden than the state-of-the-art algorithms presented in [18] and significantly lower computational costs than the evolutionary approaches shown in [17].

As shown above, the proposed metaheuristics were tested only for an MNIST dataset. Moreover, the simulations with different control parameter adjustments could also be performed to further establish the validity and efficiency of the proposed approaches and potentially to obtain better results. This represents one of the most significant threats and limitations of the proposed research. Since

the simulations took a long time to execute, we could not encompass more datasets in one set of the simulations, and we plan do this in some of our future research works in this domain.

6. Conclusions

In this paper, we proposed two enhanced versions of TGA and FA swarm intelligence metaheuristics for optimizing the CNN's hyperparameters. The first proposed approach (EE-TGA) enhanced the exploitation ability of the original TGA, while the second proposed metaheuristics (E^3FA) improved both the exploration and exploitation of the basic FA.

Based on our previous experience with the TGA method, we came to the conclusion that the exploitation ability, as well as the intensification-diversification balance of this algorithm could be improved in order to establish better solution quality and convergence. We addressed this issue by scattering some of the newly generated solutions in each iteration around the current best solution by using a normal distribution. In this way, the convergence speed and exploitation-exploration balance of the original TGA version were improved.

Similarly, founded on our previous experience with the FA metaheuristic, despite the fact that the original FA exhibited relatively stable and efficient exploitation, according to the results of the performed simulations on the standard unconstrained benchmarks, we concluded that the exploitation process of the original FA could be further enhanced and also that the exploitation-exploration balance could be better adjusted. Moreover, the original FA approach in some runs showed behavior of premature convergence because the whole population converged towards the current best solutions. By incorporating two procedures: one that enhanced the exploitation and one that improved exploration, into the original, we addressed both issues.

When a modified/improved/hybridized method is created, its effectiveness should first be validated against a wider set of benchmarks, and the performance should be compared with the original method. This is the only way to establish a real improvement of the enhanced method over the original one. For that reason, we firstly performed simulations of the proposed EE-TGA and E^3FA metaheuristics on standard unconstrained benchmarks and compared the generated solutions' quality and convergence with the original algorithms. In most cases, the methods proposed in this paper managed to obtain better results than the original implementations.

After conducting tests on standard benchmarks, we adapted and tested the proposed EE-TGA and E^3FA CNNs' hyperparameter optimization. The EE-TGA-CNN and E^3FA-CNN frameworks aimed to optimize the following CNN hyperparameters: the number of convolutional layers, the number of filters, filter size, the number of FC layers, and the the number of hidden units in the FC layers. The proposed frameworks searched for the right set of hyperparameters and automatically designed the CNN architecture for a given task. We did not include the type of pooling layer, activation function, regularization technique, and other CNN hyperparameters in the automatic search because of the limitations of available computing power.

The proposed metaheuristics were applied to the well-known MNIST benchmark dataset, and the obtained results were compared to other similar methods that were conducted under the same and similar conditions, as well as to state-of-the-art architectures. The results of both proposed frameworks (EE-TGA-CNN and E^3FA-CNN) were compared to the peer evolutionary competitors that were tested under the same simulation conditions [17], the original implementations of the TGA [71] and FA [72] metaheuristics and with same of the state-of-the-art network configurations that were manually designed. Finally, we included one more comparative analysis with state-of-the-art swarm intelligence approaches [18], which were tested under slightly different experimental conditions.

The results proved that the proposed method was robust, achieved high classification accuracy, was competitive in terms of the employed computational resources, efficient in real-world classification tasks, and did not require the user to have any prior expertise in the domain.

The EE-TGA and E^3FA metaheuristics were developed in Java SE 10 (18.3). The framework was developed and implemented by using the deep learning programming library Deeplearning4j

(https://deeplearning4j.org/), which was written for Java and Java Virtual Machine (JVM). To improve the execution speed of the performed study, due to the large utilization of computer resources, we adapted the framework for the execution on the graphical processing unit (GPU) using NVIDIA's ^{TM}CUDA parallel computing platform and programming model. All experiments were performed on the computer platform with six NVIDIA GTX 1080 GPUs, each having 2560 ^{TM}CUDA cores.

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

Based on the details of the presented research, the scientific contribution was twofold: we first managed to setup an automatic framework for CNNs' architecture generation that outperformed other well-known approaches and that could be used in practical challenges; secondly, we managed to improve the original FA and TGA implementations.

In our future research from this domain, first we want to test and improve other swarm algorithms and to adapt them for solving the CNN hyperparameter optimization problem. Furthermore, we plan to include other CNN hyperparameters in the optimization problem, such as the dropout regularization technique, learning rate, etc. Moreover, we plan to utilize other datasets as well to further establish and to try to improve the classification accuracy and to minimize the classification error. Some of the datasets that we plan to include in our future research are the following: the USPS, CIFAR-10, Semeion Handwritten Digits, and SEEDdatasets.

References

1. Ren, S.; He, K.; Girshick, R.B.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Available online: Advancesinneuralinformationprocessingsystems (accessed on 10 February 2020).
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] [PubMed]
3. Karpathy, A.; Fei-Fei, L. Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2015; pp. 3128–3137.
4. 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.
5. 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; pp. 1653–1660.
6. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
7. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. Available online: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (accessed on 10 February 2020).
8. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2004**, arXiv:1409.1556. Available online: https://arxiv.org/pdf/1409.1556.pdf (accessed on 10 February 2020).

9. 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.

10. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016.

11. Huang, G.; Liu, Z.; v. d. 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.

12. Hu, J.; Shen, L.; Sun, G. Squeeze-and-Excitation Networks. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018.

13. 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]

14. 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]

15. Suganuma, M.; Shirakawa, S.; Nagao, T. A Genetic Programming Approach to Designing Convolutional Neural Network Architectures. In Proceedings of the Genetic and Evolutionary Computation Conference, New York, NY, USA, 15–19 July 2017; pp. 497–504.

16. 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 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 26–30 June 2017; pp. 1285–1290.

17. 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.

18. 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]

19. 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.

20. 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]

21. Bergstra, J.; Bengio, Y. Random Search for Hyper-parameter Optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.

22. Beyer, H.G.; Schwefel, H.P. Evolution strategies—A comprehensive introduction. *Nat. Comput.* **2002**, *1*, 3–52. [CrossRef]

23. Fogel, D.; Society, I.C.I. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 3rd ed.; Wiley: Hoboken, NJ, USA, 2006.

24. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed.; Addison-Wesley Longman Publishing Company: Boston, MA, USA, 1989.

25. 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.

26. Karaboga, D.; Basturk, B. On the performance of artificial bee colony (ABC) algorithm. *Appl. Soft Comput.* **2008**, *8*, 687–697. [CrossRef]

27. Tuba, M.; Bacanin, N.; Stanarevic, N. Adjusted artificial bee colony (ABC) algorithm for engineering problems. *WSEAS Trans. Syst.* **2012**, *11*, 111–120.

28. Bacanin, N.; Tuba, M. Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Improved with Genetic Operators. *Stud. Informat. Contr.* **2012**, *21*, 137–146. [CrossRef]

29. Dorigo, M.; Birattari, M. *Ant Colony Optimization*; Springer: Berlin, Germany, 2010.

30. Passino, K.M. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Syst. Mag.* **2002**, *22*, 52–67.

31. Yang, X.S. Firefly Algorithms for Multimodal Optimization. In *Stochastic Algorithms: Foundations and Applications*; Watanabe, O., Zeugmann, T., Eds.; Springer: Berlin, Germany, 2009; Volume 5792.

32. Mirjalili, S.; Lewis, A. The whale optimization algorithm. *Adv. Eng. Software.* **2016**, *95*, 51–67. [CrossRef]

33. 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]

34. Yang, X.S.; Hossein Gandomi, A. Bat algorithm: A novel approach for global engineering optimization. *Eng. Comput.* **2012**, *29*, 464–483. [CrossRef]

35. Yang, X.; Deb, s. Cuckoo Search via Lévy flights. In Proceedings of the 2009 World Congress on Nature Biologically Inspired Computing (NaBIC), Coimbatore, India, 9–11 December 2009; pp. 210–214.

36. Bacanin, N. Implementation and performance of an object-oriented software system for cuckoo search algorithm. *Int. J. Math. Comput. Simul.* **2010**, *6*, 185–193.

37. 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.

38. Wang, G.G.; Deb, S.; Cui, Z. Monarch Butterfly Optimization. *Neural. Comput. Appl.* **2015**, 1–20. [CrossRef]

39. 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]

40. 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.

41. Bacanin, N.; Tuba, M. Fireworks Algorithm Applied to Constrained Portfolio Optimization Problem. In Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan, 25–28 May 2015.

42. Strumberger, I.; Bacanin, N.; Beko, M.; Tomic, S.; Tuba, M. Static Drone Placement by Elephant Herding Optimization Algorithm. In Proceedings of the 25th Telecommunications Forum (TELFOR), Belgrade, Serbia, 21–22 November 2017.

43. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Modified Monarch Butterfly Optimization Algorithm for RFID Network Planning. In Proceedings of the 6th International Conference on Multimedia Computing and Systems (ICMCS), Rabat, Morocco, 10–12 May 2018; pp. 1–6.

44. Strumberger, I.; Beko, M.; Tuba, M.; Minovic, M.; Bacanin, N. Elephant Herding Optimization Algorithm for Wireless Sensor Network Localization Problem. In *Technological Innovation for Resilient Systems*; Camarinha-Matos, L.M., Adu-Kankam, K.O., Julashokri, M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 175–184.

45. 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.

46. 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*; Yang, X.S., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 139–162.

47. Tuba, M.; Bacanin, N.; Beko, M. Fireworks algorithm for RFID network planning problem. In Proceedings of the 25th International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, Czech Republic, 21–22 April 2015; pp. 440–444.

48. 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.

49. Strumberger, I.; Bacanin, N.; Tuba, M. Enhanced Firefly Algorithm for Constrained Numerical Optimization. In Proceedings of the IEEE International Congress on Evolutionary Computation (CEC 2017), San Sebastian, Spain, 5–8 June 2017; pp. 2120–2127.

50. Strumberger, I.; Bacanin, N.; Tuba, M. Constrained Portfolio Optimization by Hybridized Bat Algorithm. In Proceedings of the 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS), Bangkok, Thailand, 25–27 January 2016; pp. 83–88.

51. Tuba, M.; Bacanin, N.; Pelevic, B. Artificial Bee Colony Algorithm for Portfolio Optimization Problems. *Int. J. Math. Model. Meth. Appl. Sci.* **2013**, *7*, 888–896.

52. Strumberger, I.; Sarac, M.; Markovic, D.; Bacanin, N. Moth Search Algorithm for Drone Placement Problem. *Int. J. Comput.* **2018**, *3*, 75–80.

53. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Wireless Sensor Network Localization Problem by Hybridized Moth Search Algorithm. In Proceedings of the 14th International Wireless Communications Mobile Computing Conference (IWCMC), Limassol, Cyprus, 25–29 June 2018; pp. 316–321.

54. 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.

55. Tuba, M.; Bacanin, N.; Alihodzic, A. Multilevel image thresholding by fireworks algorithm. In Proceedings of the 25th International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, Czech Republic, 21–22 April 2015; pp. 326–330.

56. Tuba, E.; Strumberger, I.; Bezdan, T.; Bacanin, N.; Tuba, M. Classification and Feature Selection Method for Medical Datasets by Brain Storm Optimization Algorithm and Support Vector Machine. *Procedia Comput. Sci.* **2019**, *162*, 307–315. [CrossRef]

57. Bacanin, N.; Tuba, E.; Bezdan, T.; Strumberger, I.; Tuba, M. Artificial Flora Optimization Algorithm for Task Scheduling in Cloud Computing Environment. In *International Conference on Intelligent Data Engineering and Automated Learning*; Springer: Cham, Switzerland, 2019; pp. 437–445.

58. Bacanin, N.; Bezdan, T.; Tuba, E.; Strumberger, I.; Tuba, M.; Zivkovic, M. Task Scheduling in Cloud Computing Environment by Grey Wolf Optimizer. In Proceedings of the 27th Telecommunications Forum (TELFOR), Belgrade, Serbia, 26–27 November 2019; pp. 1–4.

59. 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.

60. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Modified and Hybridized Monarch Butterfly Algorithms for Multi-Objective Optimization. In *Hybrid Intelligent Systems*; Madureira, A.M., Abraham, A., Gandhi, N., Varela, M.L., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 449–458.

61. Tuba, M.; Bacanin, N.; Beko, M. Multiobjective RFID Network Planning by Artificial Bee Colony Algorithm with Genetic Operators. In *Advances in Swarm and Computational Intelligence*; Tan, Y., Shi, Y., Buarque, F., Gelbukh, A., Das, S., Engelbrecht, A., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 247–254.

62. Tuba, M.; Bacanin, N.; Pelevic, B. Framework for constrained portfolio selection by the firefly algorithm. *Int. J. Math. Model. Meth. Appl. Sci.* **2014**, *7*, 1888–896.

63. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Monarch butterfly optimization algorithm for localization in wireless sensor networks. In Proceedings of the 28th International Conference Radioelektronika (RADIOELEKTRONIKA), Prague, Czech Republic, 19–20 April 2018; pp. 1–6.

64. Strumberger, I.; Sarac, M.; Markovic, D.; Bacanin, N. Hybridized Monarch Butterfly Algorithm for Global Optimization Problems. *Int. J. Comput.* **2018**, *3*, 63–68.

65. Strumberger, I.; Tuba, M.; Bacanin, N.; Tuba, E. Cloudlet Scheduling by Hybridized Monarch Butterfly Optimization Algorithm. *J. Sens. Actuator Netw.* **2019**, *8*, 44. [CrossRef]

66. Tuba, M.; Tuba, E. Generative Adversarial Optimization (GOA) for Acute Lymphocytic Leukemia Detection. *Stud. Informat. Contr.* **2019**, *28*, 245–254. [CrossRef]

67. 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.

68. 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.

69. Tuba, E.; Strumberger, I.; Bacanin, N.; Jovanovic, R.; Tuba, M. Bare Bones Fireworks Algorithm for Feature Selection and SVM Optimization. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 2207–2214.

70. LeCun, Y.; Cortes, C. MNIST handwritten digit database. 2010. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 10 February 2020).

71. 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.

72. 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), Costa da Caparica, Portugal, 10 May 2019; pp. 59–65.

73. 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]

74. Baldominos, A.; Saez, Y.; Isasi, P. Evolutionary convolutional neural networks: An application to handwriting recognition. *Neurocomputing* **2018**, *283*, 38–52. [CrossRef]

75. Heidari, A.A.; Faris, H.; Aljarah, I.; Mirjalili, S. An efficient hybrid multilayer perceptron neural network with grasshopper optimization. *Soft Comput.* **2017**, *23*, 7941–7958. [CrossRef]

76. 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]

77. 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]

78. 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.

79. Tuba, M.; Bacanin, N. Improved seeker optimization algorithm hybridized with firefly algorithm for constrained optimization problems. *Neurocomputing* **2014**, *143*, 197–207. [CrossRef]

80. 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]

81. 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] [PubMed]

82. Gupta, M.; Gupta, D. A New Modified Firefly Algorithm. *Int. J. Recent Contrib. Eng. Sci. IT* **2016**, *4*, 18–23. [CrossRef]

83. Lee, C.Y.; Xie, S.; Gallagher, P.W.; Zhang, Z.; Tu, Z. Deeply-Supervised Nets. *arXiv* **2014**. arXiv:1409.5185. Available online: https://arxiv.org/pdf/1409.5185.pdf (accessed on 10 February 2020).

84. McDonnell, D.M.; Vladusich, T. Enhanced Image Classification With a Fast-Learning Shallow Convolutional Neural Network. In Proceeding of the International Joint Conference on Neural Networks (IJCNN), Killarney, Ireland, 12–17 July 2015; Volume 7, pp. 1–7.

85. Liang, M.; Hu, X. Recurrent convolutional neural network for object recognition. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 3367–3375.

86. Lee, C.Y.; Gallagher, P.W.; Tu, Z. Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree. 2016. Available online: http://proceedings.mlr.press/v51/lee16a.pdf (accessed on 10 February 2020).