Intelligent Systems with Applications 26 (2025) 200532

Contents lists available at ScienceDirect

Intelligent Systems with Applications

journal homepage: www.journals.elsevier.com/intelligent-systems-with-applications

Metaheuristics in automated machine learning: Strategies for optimization

Francesco Zito [a,*], El-Ghazali Talbi [b], Claudia Cavallaro [a], Vincenzo Cutello [a], Mario Pavone [a,*]

[a] Department of Mathematics and Computer Science, University of Catania, viale Andrea Doria 6, Catania, 95125, Italy
[b] University of Lille, CNRS, INRIA, Polytech'Lille, Cité Scientifique, Villeneuve d'Ascq, Lille, 59655, France

ARTICLE INFO

Keywords:
Metaheuristics
Hyperparameter optimization
Automated machine learning
Deep learning
Artificial neural network

ABSTRACT

The present work explores the application of Automated Machine Learning techniques, particularly on the optimization of Artificial Neural Networks through hyperparameter tuning. Artificial Neural Networks are widely used across various fields, however building and optimizing them presents significant challenges. By employing an effective hyperparameter tuning, shallow neural networks might become competitive with their deeper counterparts, which in turn makes them more suitable for low-power consumption applications. In our work, we highlight the importance of Hyperparameter Optimization in enhancing neural network performance. We examine various metaheuristic algorithms employed and, in particular, their effectiveness in improving model performance across diverse applications. Despite significant advancements in this area, a comprehensive comparison of these algorithms across different deep learning architectures remains lacking. This work aims to fill this gap by systematically evaluating the performance of metaheuristic algorithms in optimizing hyperparameters and discussing advanced techniques such as parallel computing to adapt metaheuristic algorithms for use in hyperparameter optimization with high-dimensional hyperparameter search space.

1. Introduction

Artificial neural networks (ANN) are widely used across various fields including biology, finance, statistics, industry, operational research, and computer vision (Abiodun et al., 2018; Cutello et al., 2024b). Their success lies on their capability to extract features from raw data and process them individually (Zhong et al., 2016). However, building an artificial neural network is a very challenging task (Dargan et al., 2020). First, the network architecture must be tailored to the specific problem at hand. Secondly, preprocessing of the training dataset is essential to eliminate noise and handle missing values. Thirdly, training an artificial neural network can be time-consuming and demands a substantial amount of data. Lastly, evaluation of the neural network should be conducted on a partition of the dataset that has not been previously used. If the network does not meet expectations, adjustments to the architecture should be made, followed by iterative repeating of the process (Sarker, 2021).

In the rapidly evolving landscape of machine learning, one of the most significant challenges is selecting the most effective model for a given task. The vast number of machine learning models available can be overwhelming, often leading to suboptimal choices that hinder performance and efficiency. To simplify this process, researchers are focused on developing techniques in *Automated Machine Learning* (AutoML) (Hutter et al., 2019). These innovative approaches primarily concentrate on two key areas: Hyperparameter Optimization and Neural Architecture Search. *Hyperparameter Optimization* (HPO) involves adjusting the settings that control how a model learns, helping it to perform better. An example of this method is the use of Bayesian Optimization to search for the optimal hyperparameters of a neural network (Wu et al., 2019; Zito et al., 2023a). On the other hand, *Neural Architecture Search* (NAS) automates the process of designing the structure of neural networks, allowing for the identification of the best setup for different types of data and tasks (Zito et al., 2024). Additionally, NAS can be viewed as a multi-objective optimization problem (Lu et al., 2022). Automated Machine Learning techniques aim to simplify the process of building and deploying machine learning models by automatically selecting the best algorithm and hyperparameters based on the data. These techniques have gained popularity due to their ability to eliminate the need for human intervention (Salehin et al., 2024).

Over the past decade, numerous studies have investigated various metaheuristic algorithms as effective techniques for optimizing hyperparameters in machine learning models across diverse fields. These algorithms are essential for improving the performance of machine learning models and, in turn, positively impacting a wide range of application domains. Despite significant advancements in this area, to

* Corresponding authors.
E-mail addresses: francesco.zito@phd.unict.it (F. Zito), mpavone@dmi.unict.it (M. Pavone).

https://doi.org/10.1016/j.iswa.2025.200532
Received 19 September 2024; Received in revised form 17 April 2025; Accepted 5 May 2025
Available online 26 May 2025

our knowledge, there is no comprehensive study which systematically compares different metaheuristic algorithms for hyperparameter optimization in deep learning models. Thus, there is a considerable gap in literature concerning the direct comparisons of these algorithms across various deep learning architectures.

For this reason, in the present work we will explore in more details the concept of Automated Machine Learning. Different optimization algorithms, specifically metaheuristic algorithms, will be compared to evaluate their effectiveness in optimizing the hyperparameters of common neural network architectures. The aim of this work is to show how AutoML techniques can simplify the model selection process and help data scientists extract valuable insights from their data more efficiently. Advanced techniques, such as parallel computing, will be discussed to adapt metaheuristic algorithms for optimizing neural network architectures with large hyperparameter search spaces. By leveraging parallel processing, these algorithms can explore a wider range of hyperparameter combinations simultaneously, leading to faster convergence and more optimal solutions. It is important to note that while machine learning refers to a wide range of areas, this article will focus specifically on the field of Deep Learning (DL). However, many of the concepts and techniques discussed here can also be generalized and applied to other areas of machine learning, such as supervised learning, unsupervised learning, and reinforcement learning.

The present work is structured as follows. Section 2 introduces artificial neural networks and their various applications. Section 3 explores the recent trends and applications of automated machine learning. In Section 4, we analyze how metaheuristic algorithms (single solution and population-based) can be used to optimize the hyperparameters of common neural network architectures. Section 5 showcases techniques to adapt metaheuristics in real-world application domains, analyzing neural networks with higher complexity and a larger number of hyperparameters. Section 6 discusses the results obtained from our experiments. Finally, Sections 7 and 8 provide some final comments and highlight possible future research developments.

2. Artificial neural networks

An artificial neural network is a machine learning model that draws inspiration from the architecture of the human brain. It is composed of interconnected nodes, known as artificial neurons, which process information by responding to external inputs. ANNs are usually organized in layers, each with distinct properties (Wilamowski & Irwin, 2018). A typical neural network consists of three main types of layers: an input layer, one or more hidden layers, and an output layer (Fig. 1). The specific types of layers included in an ANN are determined primarily by its intended application. Each layer has unique properties and can be defined by parameters that rule its behavior. These parameters, referred to as hyperparameters, are separate from the network's internal parameters (Wang, 2003). More specifically, hyperparameters are variables that establish the structure and characteristics of a neural network and are defined before the training phase (Akay et al., 2022). These may include the output size of a fully connected layer, the dropout rate in a dropout layer, the choice of activation functions, and more. Additionally, parameters related to training, such as the number of epochs, the optimizer, and the initial learning rate, can also be categorized as hyperparameters. On the other hand, parameters determine the network's behavior and are determined during training, where optimal weights are chosen to minimize the loss function (Aljarah et al., 2018). Unlike hyperparameters, these parameters rely on the input data and are exclusively determined by the training process. Overall, the performance of a neural network is influenced by both its hyperparameters and parameters. The selection of suitable hyperparameters can significantly impact the efficiency of the training process and the subsequent performance of the network.

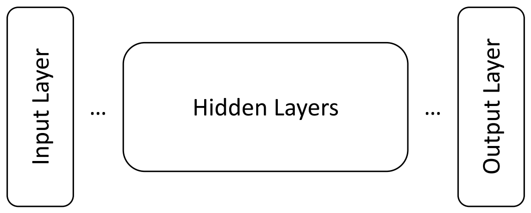


Fig. 1. Artificial neural network.

Deep Neural Networks (DNNs) extend the concept of traditional neural networks by incorporating multiple hidden layers. This increased depth allows DNNs to learn more intricate patterns and representations of data (Islam et al., 2023). One of the most significant advantages of deep neural networks is their ability to process information directly without the use of a strong preprocessing of data, making them particularly well-suited for computer vision tasks (Chai et al., 2021). For example, in tasks such as image recognition, each layer can extract progressively more abstract features, starting from basic edges and shapes in the initial layers to complex objects in the deeper layers. Additionally, DNNs are highly effective at extracting relationships between variables that are unseen by humans, making them well-suited for tasks such as pattern recognition and time-series forecasting.

3. Recent trends and applications

Automated Machine Learning has gained significant attention in recent years due to the increasing complexity of machine learning models. The primary goal of AutoML is to automate the process of building and optimizing machine learning models, which reduces the need for manual intervention and accelerates the model development process (Hutter et al., 2019). This section will explore recent trends and applications of AutoML, highlighting its potential to revolutionize various industries and fields (Salehin et al., 2024).

3.1. Interdisciplinary impact of automated machine learning

In the last ten years, the number of publications in the field of automated machine learning has shown a remarkable growth, as can be observed in Fig. 2, reflecting the growing interest and advancements in automated machine learning technologies and methodologies. To provide further insight, Fig. 3 depicts the distribution of research publications across various subject areas related to automated machine learning from 2014 to 2024. Each segment of the pie chart represents a different field, highlighting the interdisciplinary nature of automated machine learning research, its impact across diverse scientific domains, and the interest in investing financially in such a research field.

More in details, AutoML has found numerous applications in the healthcare industry, particularly in areas such as medical image analysis and disease prediction (Waring et al., 2020). While much of the existing AutoML literature has focused on supervised learning, recent work emphasizes the need for advancements in unsupervised learning, particularly in anomaly detection. This highlights the importance of developing automated methods for identifying anomalies, which are critical across various domains (Bahri et al., 2022). In the field of cybersecurity, AutoML has shown great promises. As cyber threats become increasingly sophisticated, traditional rule-based approaches are less effective (Alrowais et al., 2023). AutoML can enhance threat detection and response by automating the analysis of vast amounts of data. The financial sector has also benefited from advancements in AutoML. Machine learning models developed using AutoML techniques facilitate rapid decision-making and optimization of trading strategies, as discussed in Larsen and Becker (2021). Another significant application of

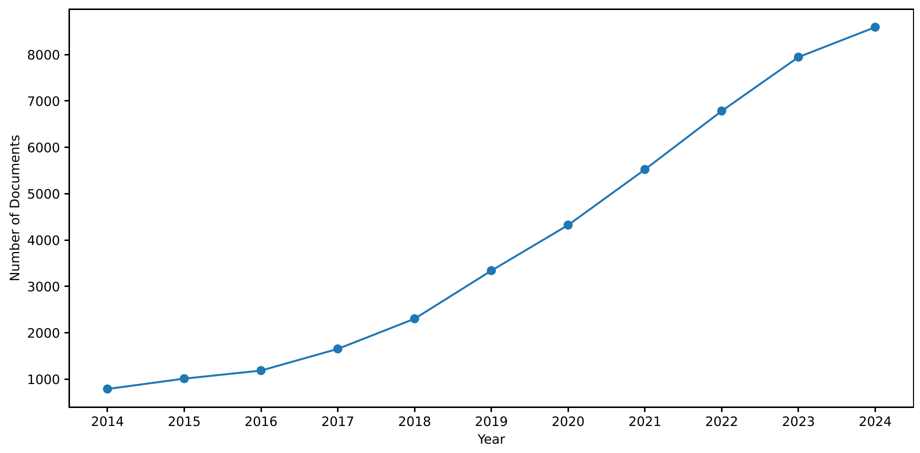


Fig. 2. Publication growth in the field of Automated Machine Learning (2014–2024). Data obtained from the Scopus Database.

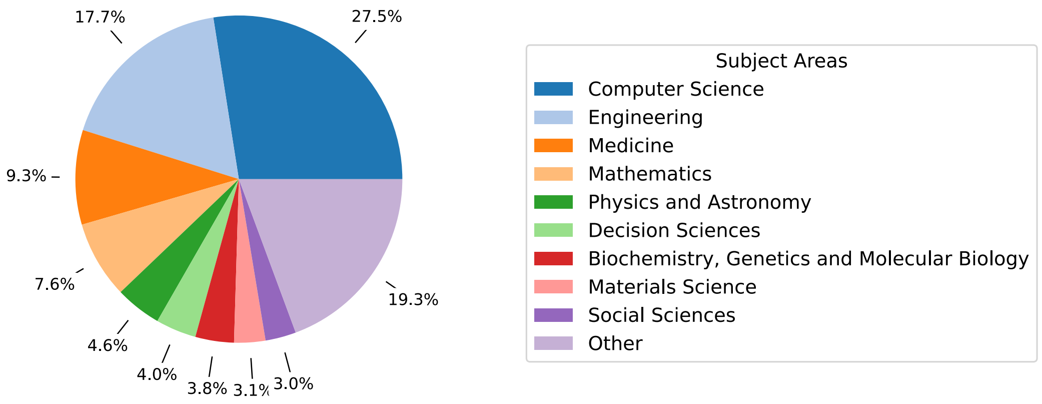


Fig. 3. Distribution of research areas for Automated Machine Learning (2014–2024). Data obtained from the Scopus Database.

AutoML lies in forecasting. AutoML can develop neural network models that effectively capture relationships between variables so to generate accurate predictions. This capability is particularly valuable in decision-making processes and gene regulation, where a good understanding of the intricate interactions among genes and their regulatory elements is essential (Zito et al., 2023b).

3.2. Importance of hyperparameter optimization

An interesting aspect to consider is the number of techniques proposed for AutoML that have been published between 2014 and 2024. Fig. 4 shows the number of research results that addressed two areas of AutoML: hyperparameter optimization (in blue) and neural architecture search (in orange). The figure highlights a significant increase in publications related to hyperparameter optimization, particularly in 2024, while publications on neural architecture search show a more gradual rise, peaking in 2023. This comparison underscores the growing interest and research activity in hyperparameter optimization in recent years. Hyperparameter optimization is indeed a crucial step in developing machine learning models, as it involves fine-tuning the parameters that govern the learning process (Goswami et al., 2023). AutoML has made significant advancements in this area, employing techniques such as Bayesian optimization and evolutionary algorithms to enable efficient and effective hyperparameter tuning (Zhang et al., 2020). Given the vast search space for optimization algorithms, various solutions have been proposed in the literature (Yang & Shami, 2020). An example is found in Morteza et al. (2023), where a hyperparameter optimization algorithm is used to optimize the demand predictions for a building. In addition, to simplify the complexity of building machine learning models, in Li et al. (2022), a framework for AutoML is proposed, aimed at building blocks to create machine learning models. Another notable trend in AutoML is Neural Architecture Search techniques. In recent years, the number of papers published in this area has declined compared to hyperparameter optimization; this is because NAS are resource-intensive algorithms due to their automatic design of neural network architectures tailored to specific tasks (Talbi, 2021a). A significant challenge in NAS is evaluating candidate neural network architectures. This evaluation can be resource-intensive with respect to computational time and hardware resources. Therefore, it is necessary to adapt strategies to speed up this evaluation process (Prechelt, 1998; Zito et al., 2024). Some effective strategies include early stopping, model pruning, transfer learning, and parallel training.

As shown in Fig. 4, several optimization algorithms, in particular metaheuristic algorithms, are used for hyperparameter optimization in various application areas (Bibaeva, 2018). We reported some recent ones in Table 1 along with the year of publication, the specific machine learning model used, and their strengths and weaknesses. While the literature on metaheuristics for hyperparameter optimization is extensive, this table emphasizes research that has made a significant impact in the field. This is clearly shown in Fig. 5, where it is depicted the trend of applications of metaheuristic algorithms for hyperparameter optimization, which were published from 2014 to 2024, as obtained from the Scopus Database. It is well-known that metaheuristic algorithms are commonly used when traditional optimization methods falter, particularly in high-dimensional spaces where the search for optimal hyperparameters becomes computationally very expensive. As demonstrated in recent studies (Saber & Salem, 2023; Salem, 2023), metaheuristics are well-suited for tackling complex optimization problems, including photovoltaic cell optimization and the 0–1 knapsack

Table 1
Recent metaheuristic algorithms for hyperparameter optimization.

Reference	Algorithm	Application domain	ML model	Strengths	Weaknesses
Bouktif et al. (2018)	Genetic Algorithm	Forecasting	LSTM	Exploration of search space prevents overfitting; robustness	Parameter sensitivity
Nakisa et al. (2018)	Differential Evolution	Classification	LSTM	High diversity in finding optimal solutions	Premature convergence; computationally expensive
Elmasry et al. (2020)	Particle Swarm Optimization	Classification	DNN, LSTM	Fast convergence	Lack of diversity; parameter sensitivity
Tsai et al. (2020)	Simulated Annealing	Forecasting	DNN	Robustness in large search spaces	Low convergence rate
Turkoglu and Kaya (2020)	Artificial Algae Algorithm	Classification	MLP	Capability in searching solutions without reaching local minimum	Lack of exploration in high-dimensional datasets
Ma et al. (2023)	Iterated Local Search	Prediction	BiGRU	Fast convergence	Parameter sensitivity; effectiveness depends on local search quality
Chen (2024)	Tabu Search	Regression	MLP	Efficiency for high-dimensional datasets; prevents cycling back to previously visited solutions	May require many iterations for high-quality solution
Sarwar et al. (2024)	Ant Colony Optimization	Classification	ResUNet	Adapts to dynamic problems; handles large-scale problems	High parameter sensitivity; high computational cost

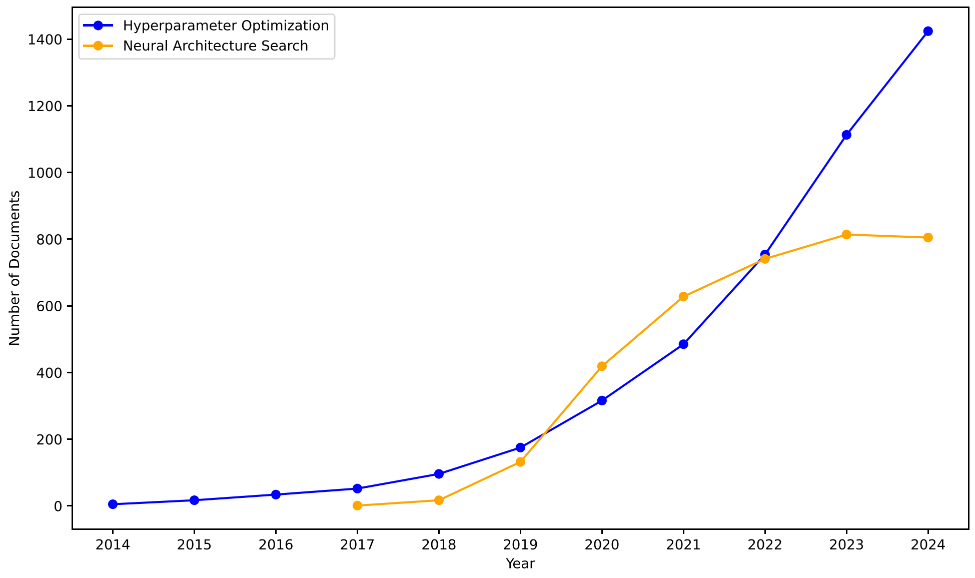


Fig. 4. Comparison of research fields over last decade (2014–2024). Data obtained from the Scopus Database. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

problem, which is known to be an optimization problem with a high-dimensional search space. In addition, they find several applications in real-world scenarios to estimate model parameters and to explore unknown environments, as demonstrated in Cavallaro, Crespi et al. (2024) and Crespi et al. (2024).

However, despite the growing interest in using metaheuristic algorithms for hyperparameter optimization, there is a notable lack of studies that compare the most commonly used metaheuristic algorithms across various neural network architectures of differing complexities and tasks. This gap in the literature highlights the need for comprehensive analyses that evaluate how different metaheuristics perform in optimizing hyperparameters for diverse neural network models. In the next section, we will explore this topic in greater detail so to provide the reader with a comprehensive overview of which metaheuristic achieves a better performance while balancing the performance and the time required to optimize a neural network architecture.

4. Hyperparameter optimization based on metaheuristics

The application of metaheuristics has evolved significantly in recent years, with numerous studies in literature demonstrating their effectiveness in enhancing machine learning performance (Talbi, 2021b). Metaheuristic algorithms are designed to be versatile, allowing them to be applied to a wide range of optimization problems. Solutions are generated iteratively, with candidate solutions being continuously refined to achieve optimal or near optimal outcomes (Bianchi et al., 2009). One of the advantages of metaheuristics is their simplicity, which makes them suitable for addressing several types of problems (Stork et al., 2022). Before analyzing the performance of metaheuristic algorithms for optimizing neural networks, it is essential to define three key components necessary for employing an optimization algorithm. First, defining the search space involves specifying the range of possible solutions that the algorithm can explore. Next, the objective function

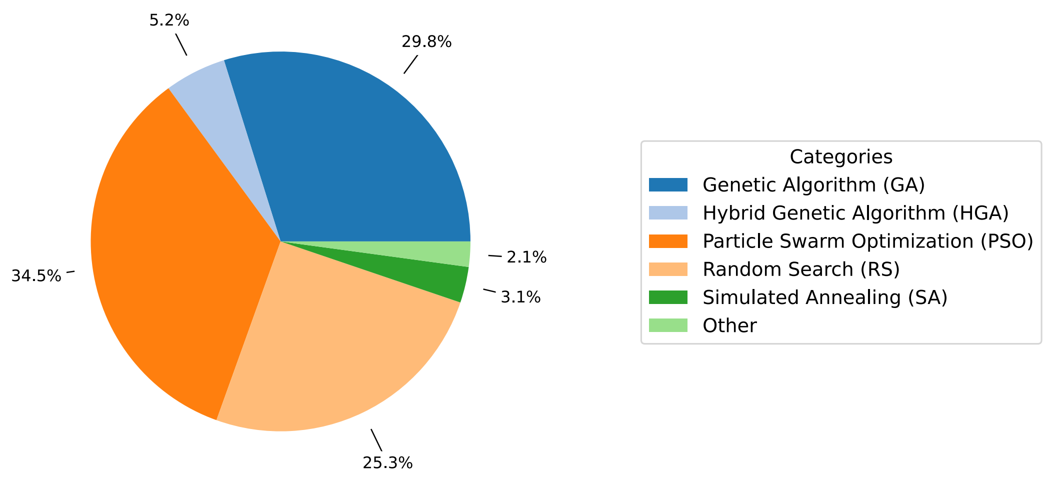


Fig. 5. Distribution of popular metaheuristics in Automated Machine Learning (2014–2024). Data obtained from the Scopus Database.

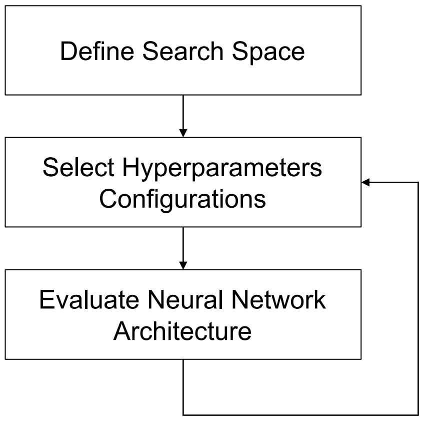

Define Search Space
Select Hyperparameters Configurations
Evaluate Neural Network Architecture

Fig. 6. Hyperparameter optimization schema.

evaluates the quality of the solutions within the search space, guiding the optimization process. Finally, defining search strategies determines how the algorithm explores the search space to find the optimal solution. All of these components will be described in detail below. Fig. 6 depicts how any hyperparameter optimization algorithm works. Basically, we have an iterative algorithm where at each iteration a hyperparameter configuration is found based on the search strategy and, subsequently, it is evaluated and, possibly, updated.

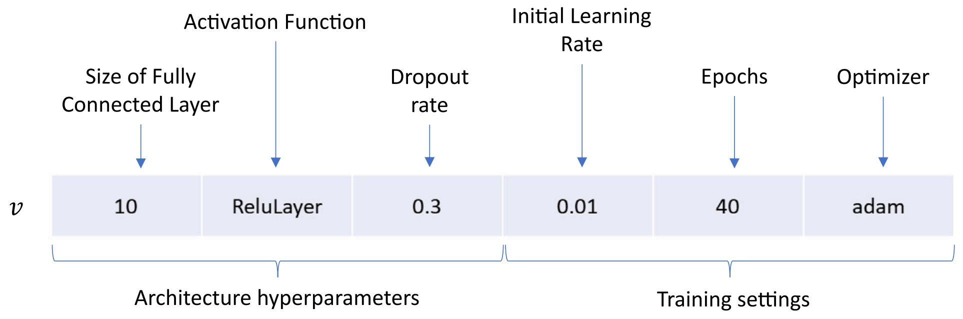
4.1. Hyperparameter search space

Before delving into the details of the various application domains and the neural network architectures considered in this study, it is essential to clarify that each neural network employed in a particular application domain has a unique set of hyperparameters. These hyperparameters include both the architectural hyperparameters and the training settings. In general, these neural networks can be classified as parametric artificial neural networks, where the characteristics of each layer and the training settings are treated as variables within the model. Consequently, we can define a *hyperparameter vector*, essentially a collection of values representing these variables (including neural layer hyperparameters and training settings), that needs to be optimized through various algorithms. The primary objective of a HPO algorithm is to find a hyperparameter vector that enables a neural network to achieve optimal performance.

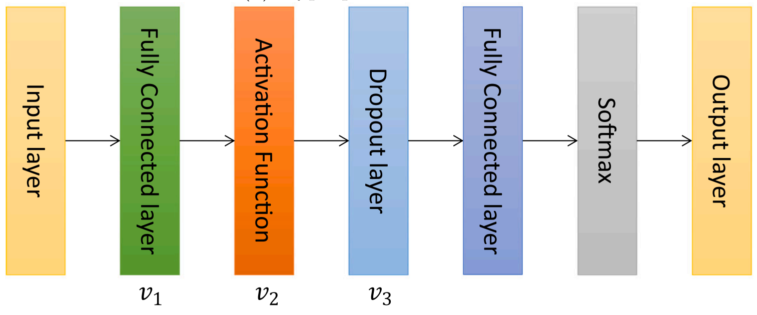
Formally, a hyperparameter vector is a vector v in which each element refers to a single hyperparameter of an artificial neural network. A hyperparameter vector contains two information groups. The first group contains architecture hyperparameters, whereas the second group contains training setting. Fig. 7 shows an example of the representation of a hyperparameter vector and the relative parametric artificial neural network. In this study, we focus on the case of discrete optimization. Given that each element of a hyperparameter vector v can take on a finite set of values, we denote the set of all possible values that the ith hyperparameter can assume as U_i. Thus, a hyperparameter vector v is defined as $v \in U$, where $U = U_1 \times U_2 \times \cdots \times U_k$, and k represents the number of hyperparameters in an artificial neural network. In this notation, U represents the *hyperparameter search space*, which is used by the optimization algorithm to explore various configurations of the hyperparameters for optimization. Due to the vast size of the hyperparameter space, Wistuba et al. (2015) has adopted strategies to effectively navigate and optimize within it.

4.2. Objective function

A hyperparameter optimization algorithm searches a hyperparameter vector for the neural architecture under consideration by employing a strategy that depends on the type of algorithm itself. To assess the quality of the hyperparameter vector and determine whether the corresponding neural network performs well, an evaluation process must be conducted. However, during the assignment of hyperparameters to a parametric neural network, it is possible to encounter an invalid architecture. In this case, the cost of the hyperparameter vector is set to be an infinite cost so to avoid it in future selections. Conversely, if the hyperparameter vector is valid, the evaluation process yields a real value representing the cost of the solution. The cost is a metric used to quantify the performance of a neural network and should be as small as possible. Typically, this value can be equal to the loss value computed by the loss function. In hyperparameter optimization, algorithms can utilize multiple objective functions that take various conditions into account when evaluating a candidate neural network. These conditions may include loss values, model complexity, the number of hidden layers, and more (Morales-Hernández et al., 2022; Smithson et al., 2016). However, in this study, we focus solely on a single-objective optimization approach, where the objective function represents the cost associated with the loss incurred during the evaluation of a neural network. Algorithm 1 illustrates this process. A hyperparameter vector is used to configure each layer of the ANN and set the training parameters. The resulting ANN is then trained on a training dataset, and its performance is evaluated using a test set. The cost returned in line 5 is a metric used to quantify the performance of a neural network and should be as small as possible. Typically, this value can be equal to the loss value computed by the loss function.



(a) Hyperparameter vector



(b) Parametric Artificial Neural Network

Fig. 7. Representation of the hyperparameter vector v (Fig. 7(a)) and the corresponding parametric artificial neural network (Fig. 7(b)).

Algorithm 1: Hyperparameter Vector Evaluation Function

Inputs :

- A_k: a parametric artificial neural network
- v: a hyperparameter vector that is divided into two components: v_{net} containing the architecture hyperparameter and v_{train} containing the training settings
- D: a dataset divided into training set D_{train} and test set D_{test}

Output: *cost*: represent the error obtain during the phase of evaluation by using the test set

```
Function HPEvaluation($A_k$,$v$,$D$):
    $net \leftarrow$ setNetHyperparameters($A_k$,$v_{net}$) ;
    $D_{train}, D_{test} \leftarrow$ splitDataset($D$);
    train($net$,$D_{train}$,$v_{train}$) ;
    $cost \leftarrow$ eval($net$,$D_{test}$) ;
    return $cost$ ;
```

4.3. Search strategies

Each optimization algorithm employs its own search strategies, which are influenced by the intrinsic characteristics of the algorithm itself. However, as mentioned in the Introduction Section and according to the reasons explained in Section 3, we focus here exclusively on metaheuristic algorithms. They are visually categorized in Fig. 8, where they are divided into two distinct groups: single-solution and population-based metaheuristics. Additionally, we briefly summarized them in Table 2 along with some key corresponding bibliographical references.

The selected algorithms represent some of the most established and widely recognized approaches in the literature. Given the continuous growth in the number of metaheuristic algorithms published each year, it is impractical to include all of them in this study. Recent reviews, such as Rajwar et al. (2023) and Li et al. (2024), provide a comprehensive overview of metaheuristic algorithms, offering a novel taxonomy and critical analysis of their proliferation. Additionally, recent innovations in metaheuristic approaches for NP-hard combinatorial optimization tasks have been addressed in Abdipoor et al. (2023). In this study, we have instead chosen to focus on algorithms that leverage fundamental search strategies common to many others, as discussed in Section 3. This approach allows us to emphasize foundational algorithms without delving into every existing variation that employs similar mechanisms. Furthermore, in the context of hyperparameter optimization, the primary objective is to identify the best hyperparameter configuration within a predefined amount of time. Unlike other optimization problems where rapid convergence is critical, the emphasis here is on achieving the highest-quality solution within the available time.

When employing metaheuristics for hyperparameter optimization, the initial step involves representing the hyperparameters as candidate solutions managed by the metaheuristic algorithm. Each hyperparameter is associated with discrete values, as discussed in the definition of the Hyperparameter Space in Section 4.1. These discrete values can be effectively encoded as integers. Therefore, a candidate solution, which is a hyperparameter vector v, is encoded as an integer vector. If $U_i = \{u_{i1}, u_{i2}, \dots, u_{n_i}\}$ represents a finite set of possible values that the ith hyperparameter can assume, the ith element of the vector is assigned a value $u_{ij} \in U_i$, with $1 \le j \le n_i$.

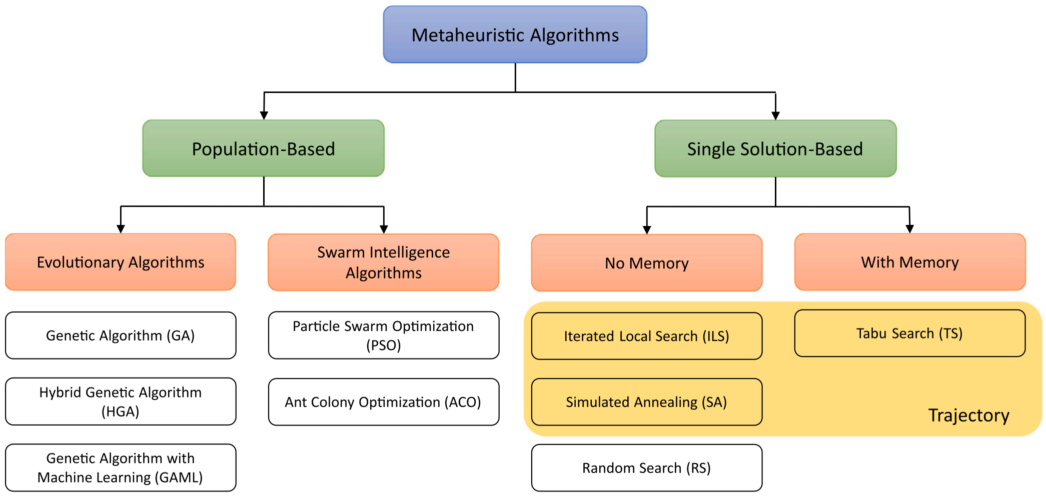

Metaheuristic Algorithms
Population-Based
Single Solution-Based
Evolutionary Algorithms
Swarm Intelligence Algorithms
No Memory
With Memory
Genetic Algorithm (GA)
Hybrid Genetic Algorithm (HGA)
Genetic Algorithm with Machine Learning (GAML)
Particle Swarm Optimization (PSO)
Ant Colony Optimization (ACO)
Iterated Local Search (ILS)
Simulated Annealing (SA)
Random Search (RS)
Tabu Search (TS)
Trajectory

Fig. 8. Visual representation of metaheuristics algorithms, categorized into two main groups: single-solution and population-based metaheuristics.

Table 2
Classification of metaheuristic algorithms into two main categories: single-solution (S1) and population-based (PB).

Algorithm	Acronym	Category	Reference
Iterated Local Search	ILS	S1	Rego and Glover (2007) and Talbi (2009)
Tabu Search	TS	S1	Lai et al. (2018) and Wu et al. (2020)
Simulated Annealing	SA	S1	Yabi et al. (2024)
Random Search	RS	S1	Bergstra and Bengio (2012)
Genetic Algorithm	GA	PB	Talbi (2009)
Hybrid Genetic Algorithm	HGA	PB	Gharsalli and Guérin (2019) and Talbi (2009)
Particle Swarm Optimization	PSO	PB	Tani and Veelken (2024)
Genetic Algorithm with Machine Learning	GAML	PB	Cavallaro, Cutello et al. (2024)
Ant Colony Optimization	ACO	PB	Dorigo and Socha (2018)

4.4. Single solution search

The basic idea of a metaheuristic algorithm based on a single solution is to modify and improve a single candidate solution at each iteration. The search process consists of jumping from one feasible solution to another in the search space. The strategy used to discover a new solution depends on the chosen type of algorithm.

Iterated local search. An Iterated Local Search (ILS) algorithm is employed to find a solution to a problem by starting at an initial state and iteratively improving it until a desired goal state is reached. In the context of hyperparameter optimization, a state represents a potential combination of hyperparameters. Consequently, the algorithm begins with a random hyperparameter combination and, at each iteration, it transitions between states based on a defined rule that considers a cost function. For example, hill-climbing algorithms choose a neighbor that has the lowest cost among all neighbors (Jacobson & Yücesan, 2004). A local search algorithm operates by focusing on the current state and its neighboring solutions, rather than the entire search space. A neighbor is a potential solution that differs minimally from the current solution (Rego & Glover, 2007). For a given hyperparameter combination (candidate solution) v, the neighbors consist of all hyperparameter combinations that are similar to v, differing by only a few values. Algorithm 2 shows the pseudocode of a simple version of the local search algorithm. It starts by generating an initial solution s and then, iteratively, improves it by randomly generating a neighborhood of the solution, using a given distance radius value ρ, and picking the best solution in the neighborhood. If it is better than the current solution s according to some objective function f the current solution is updated to the new solution. The algorithm runs for a given maximum time, T_{max}.

Tabu Search. Tabu Search (TS) algorithm employs a guided local search procedure to avoid or escape from local optima. It allows moves that may temporarily worsen the objective function value and rejects moves to points already visited in the search space using a tabu list (Wu et al., 2020) of a given constant size L. The algorithm iterates from one potential solution to an improved solution in the neighborhood of the current solution, generated using the given distance radius ρ, until a stopping criterion, such as an overall limit T_{max} is met (Cutello et al., 2025; Glover, 1986). Unlike other single-solution algorithms, as highlighted in Fig. 8, TS employs a memory structure, referred to as the Tabu List, to store previously explored solutions with a size equal to L. The algorithm runs for a given maximum time, T_{max}. The pseudocode of Tabu Search algorithm is reported in Algorithm 3.

Simulated Annealing. Simulated Annealing (SA) is a probabilistic optimization algorithm inspired by the annealing process in metallurgy. It is used to find the global optimum of a function by iteratively exploring the solution space. The algorithm starts with an initial solution and progressively modifies it according to a specified distance radius ρ, which defines the proximity between two solutions. In each iteration, the algorithm accepts new solutions that improve the objective function value. Additionally, it may accept worse solutions with a certain probability, which decreases over time. This acceptance probability is determined by the current temperature τ. Such a strategy allows the algorithm to escape local optima and explore a larger portion of the solution space (Aarts et al., 2006). The pseudocode of the algorithm is reported in Algorithm 4.

Random Search. Random Search (RS) algorithm explores the solution space of a problem randomly (Zabinsky, 2011). The Random Search

Algorithm 2: Iterated Local Search Algorithm

```
Function IteratedLocalSearch($T_{max} = 18000, \rho = 2$):
    $s \leftarrow$ generate_initial_solution();
    while $time() \leq T_{max}$ do
        $S \leftarrow$ generate_neighbor($s$,$\rho$)
        Select the best solution $\hat{s} \in S$;
        if $f(\hat{s}) < f(s)$ then
            $s \leftarrow \hat{s}$;
        end
    end
return $s$;
```

Algorithm 3: Tabu Search Algorithm

```
Function TabuSearch($T_{max} = 18000, \rho = 2, L = 10$):
    $x^* \leftarrow$ generate_initial_solution();
    $tabu\_list \leftarrow \emptyset$;
    while $time() \leq T_{max}$ do
        $best\_candidate \leftarrow None$;
        $best\_candidate\_f = \inf$;
        for $y$ in generate_neighbor($x$,$\rho$) do
            if $y$ not in $tabu\_list$ and $f(y) < best\_candidate\_f$ then
                $best\_candidate = y$;
                $best\_candidate\_f = f(y)$;
            end
        end
        if $best\_candidate = None$ then
            break;
        end
        $x = best\_candidate$;
        if $f(x) < f(x^*)$ then
            $x^* = x$;
        end
        Add $(x, best\_candidate)$ to tabu list;
        if tabu list size exceeds $L$ then
            Remove the oldest solutions from tabu list;
        end
    end
```

Algorithm 4: Simulated Annealing Algorithm

```
Function SimulatedAnnealing($T_{max} = 18000, \rho = 1, \tau = 10$):
    current_solution $\leftarrow$ generate_initial_solution();
    best_solution $\leftarrow$ current_solution;
    $i \leftarrow 0$;
    while $time() \leq T_{max}$ do
        Generate a new solution $s_{new}$ near current_solution increase all element of $\rho$ steps ;
        if $f(current\_solution) < f(best\_solution)$ then
            best_solution $\leftarrow$ current_solution;
        end
        $\Delta \leftarrow f(s_{new}) - f(\text{current\_solution})$ ;
        $t \leftarrow \tau/(i+1)$
        if $\Delta < 0$ or $random(0,1) < e^{-\Delta/t}$ then
            current_solution $\leftarrow s_{new}$;
        end
        $i \leftarrow i + 1$;
    end
    return best_solution;
```

algorithm generates and evaluates random inputs to the objective function until a satisfactory solution is found, or a stopping criterion is met. This algorithm follows a similar concept to traditional local search, but with a key difference: at each iteration, a random solution is selected from the search space without any specific strategy. If this random solution outperforms the current best solution, it is then considered the new best solution. The algorithm runs for a given maximum time, T_{max}. The pseudocode is reported in Algorithm 5.

Algorithm 5: Random Search Algorithm

```
Function RandomSearch($T_{max}$ = 18000):
    while time() $\leq T_{max}$ do
        $s \leftarrow$ generate_random_solution();
        if $f(s) < f(s_{best})$ then
            $s_{best} \leftarrow s$;
        end
    end
    return $s_{best}$;
```

Algorithm 6: Genetic Algorithm

```
Function GeneticAlgorithm($T_{max} = 18000$, $\rho_m = 0.4$, $\rho_c = 0.9$, $n = 10$, $tournament\_size = 3$):
    $t \leftarrow 0$;
    Let $P^{(t)} = (I_1, I_2, \dots, I_n)$ be an initial population of $n$ individuals;
    FitnessOperator($P^{(t)}$) ;
    while time() $\leq T_{max}$ do
        Select a subset of individuals $P^\star$ such that $P^\star \subseteq P^{(t)}$ ;
        CrossoverOperator($P^\star$, $p_c$);
        MutationOperator($P^\star$, $p_m$);
        FitnessOperator($P^\star$) ;
        $P^{(t+1)} \leftarrow P^\star$;
        $t \leftarrow t + 1$;
    end
    return the best individual in $P^{(t)}$;
```

4.5. Population based search

Population-based algorithms evolve a set of candidate solutions, referred to as a population, to solve optimization problems. Each individual in the population represents a possible combination of hyperparameters, and through various strategies, it either undergoes some changes, called mutations, or it is combined with other individuals to generate new individuals. Thus, the population evolves at each iteration based on a predefined strategy specific to the chosen algorithm. To avoid the population becoming excessively large, a filtering strategy is implemented to select candidate solutions.

Genetic Algorithm. Genetic Algorithm (GA) operates on the principles of Darwinian natural selection, simulating the process of natural evolution to find solutions to problems (Plebe & Pavone, 2018). The algorithm starts with an initial population of n candidate solutions, which are typically represented as strings of binary digits. The fitness of each solution, which is a measure of how well the solution solves the problem at hand, is then evaluated. The algorithm then proceeds to create a new generation of solutions by selecting the fittest individuals from the current population and applying genetic operators such as crossover and mutation to create new offspring. The new generation is then evaluated, and the process is repeated until a satisfactory solution is found or a maximum number of generations is reached (Talbi, 2009). In our implementation of a genetic algorithm, we used the following operators: Tournament Selection as the selection operator, which randomly selects a subset of individuals from the population and chooses the best one for the next step; Single-Point Cross-Over where the selected parent individuals are cut at a randomly selected location, and the genetic information to the left (or right) of the point is swapped between the two parent individuals to produce two new offspring. The mutation operator selects a gene of an individual randomly, representing a hyperparameter, and changes this value by randomly selecting another valid value for the specific hyperparameter. The mutation and crossover operators are applied based on probabilities denoted as ρ_m and ρ_c, respectively. The pseudocode of the genetic algorithm is reported in Algorithm 6.

Hybrid Genetic Algorithm. Hybrid Genetic Algorithm (HGA), also known as a memetic algorithm (El-mihoub et al., 2006), is a type of population-based method that combines a genetic algorithm with other techniques to improve the quality of solutions. Our version of the hybrid genetic algorithm uses local search to try to further improve the solutions obtained by the crossover and mutation operators (Talbi, 2009). In other words, a local search algorithm, as described in Algorithm 2, is applied immediately after the mutation operators in Algorithm 6.

Genetic Algorithm with Machine Learning. Genetic Algorithm with Machine Learning (GAML) is an improvement of GA, proposed in Cavallaro, Cutello et al. (2024), that use integration of a machine learning unit can guide the algorithm towards an acceptable solution. The targeted search of GAML efficiently directs computational resources towards a neighborhood where an optimal solution is likely to be found. Indeed, the integration between machine learning and metaheuristics seems to be a promising approach. These algorithms can automatically adjust their parameters, allowing for control over the exploration and exploitation of the search algorithm. As a result, the algorithm can automatically change the search direction and focus its computational resources to seek solutions in other areas of the search space where it is likely to find better solutions (Cutello et al., 2024a). In our experiments, we used the same parameters and configurations as reported in Cavallaro, Cutello et al. (2024).

Particle Swarm Optimization. Particle Swarm Optimization (PSO) is inspired by the social behavior of bird flocking or fish schooling (Wang et al., 2017). In PSO, a group of candidate solutions, known as particles, traverse the search space to discover the optimal solution. Each particle adjusts its position based on its individual experience and that of its neighbors. The number of particles in the swarm is represented by $n\_particles$. The cognitive parameter, denoted as $cognitive$, influences the degree to which a particle is attracted to its own best-known position. The social parameter, represented by $social$, determines how much a particle is influenced by the best-known position of its neighbors. Finally, $inertia$ regulates the balance between exploration and exploitation of the particles within the swarm. The algorithm aims to converge towards the global optimum by iteratively updating the particles' positions (Shi & Eberhart, 1999). The pseudocode of the algorithm is presented in Algorithm 7.

Algorithm 7: Particle Swarm Optimization

```
Function ParticleSwarmOptimization($T_{max} = 18000$, $n\_particles = 10$, $cognitive = 0.5$, $social = 0.3$, $inertia = 0.9$):
    while $time() \leq T_{max}$ do
        for each particle do
            Update particle's velocity;
            Update particle's position;
            Evaluate particle's fitness;
            if particle's position is better than personal best then
                Update personal best;
            end
            if particle's position is better than global best then
                Update global best;
            end
        end
    end
```

Ant Colony Optimization. Ant Colony Optimization (ACO) is a part of the general field of Swarm Intelligence, presented in Colorni et al. (1992) and Dorigo and Socha (2018). It is an algorithm inspired by the behavior of real ants, which are known for their ability to find the shortest path between their nest and food resources. In ACO, artificial ants are used to construct solutions to a given problem. They move through the problem space, constructing solutions by making probabilistic decisions based on pheromone trails. These pheromone trails are updated based on the quality of the solutions found, with better solutions resulting in higher pheromone levels. This process is repeated until a termination condition is met, such as a maximum number of iterations or a satisfactory solution being found. For our case study, we build a graph where each vertex represents a possible combination of hyperparameter values, that is a hyperparameter vector. Two vertices are adjacent if their Hamming distance is 1, i.e. they differ by only one hyperparameter value. Ants move from one vertex to another based on the following rule:

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{h\in Adj(i)} \tau_{ih}^{\alpha}\eta_{ih}^{\beta}} \tag{1}$$

Here, τ_{ij} represents the pheromone level on the edge (u_i, u_j), and η_{ij} is the desirability defined as the inverse of the distance, $\eta_{ij} = 1/d_{ij}$, where the distance d_{ij} is calculated as:

$$d_{ij} = \left|cost(u_i) - cost(u_j)\right| \tag{2}$$

and $cost(u_i)$ returns the cost of the hyperparameter vector u_i, and it is computed according to the Algorithm 1. The pheromone guides the search direction to find a path with the lowest cost. The cost of a path π is determined as:

$$c_{\pi} = \sum_{(u_i,v_j)\in\pi} d_{ij} \tag{3}$$

4.6. Neural networks models and data

In literature, different types of neural network architectures have been developed for application across different fields. However, in this study, we selected five representative neural network architectures that are commonly used in real-world applications such as Forecasting, Anomaly Detection, and Image Classification. The focus of this study is not on the methodologies for constructing a neural network or selecting its layers, but rather on strategies for determining optimal hyperparameters to improve the neural network's performance. In specific domains such as the Internet of Things, home automation, and automotive industries, where simplicity is crucial to minimize power consumption and energy demands (Profentzas et al., 2021), optimizing the neural network models through careful adjustment of their hyperparameters can serve as a valid alternative to improve the performance without increasing the complexity of such models.

In the preliminary analysis of how metaheuristic algorithms can enhance the performance of artificial neural networks through hyperparameter optimization, we selected three standard and widely-used neural network architectures: Multi-Layer Perceptron, Convolutional Neural Network, and Recurrent Neural Network. As illustrated in Fig. 9, each artificial neural network is represented as a sequence of layers, with each layer possessing its own unique properties and hyperparameters. Table 3 provides details on the hyperparameters considered for each type of layer, along with the parameters used during the training of the neural networks. This information defines the Hyperparameter Space within which the metaheuristic algorithms operate. Below, we will provide some details regarding the considered neural network architectures, as well as the datasets and application domains in which they have been employed.

4.6.1. Multi-Layer Perceptron (MLP)

A MLP is a type of artificial neural network that comprises multiple layers of perceptrons connected by weighted edges. A perceptron receives a vector x with m features as input and it produces a single output value y based on the activation function $f(\cdot)$, which considers the linear combination of the features x and the weights ω (Goodfellow et al., 2016). The neural network architecture used in our experiment is shown in Fig. 9(a) and is applied to a regression task to forecast Credit Default Prediction, a critical issue for financial institutions that has been extensively studied in the literature (Yeh, 2016).

4.6.2. Convolutional Neural Network (CNN)

The CNNs are a class of specialized artificial neural network primarily used for image processing. CNNs can learn hierarchical representations of images by stacking multiple convolutional layers. The lower layers capture basic features such as edges and corners, while the higher layers capture more complex features such as shapes and objects. By exploiting the spatial structure and locality of images, CNNs can achieve high accuracy and efficiency in image classification (Indolia et al., 2018). In our experiments, we use the architecture illustrated in Fig. 9(b) and the MNIST dataset, which contains handwritten digits from 0 to 9 and is commonly used for optical character recognition (OCR) tasks (Deng, 2012).

4.6.3. Recurrent Neural Network (RNN)

A RNN is characterized by feedback connections between its units, enabling it to process sequential data like time-series data (Hewamalage et al., 2021). RNNs can learn from both current input and previous output or hidden state. However, RNNs have some limitations such as the vanishing gradient problem, which hinders their ability to learn long-term dependencies in the data. To address this issue, a specialized type of RNN known as Long Short-Term Memory (LSTM) was developed. LSTM incorporates memory cells that can retain and

Algorithm 8: Ant Colony Optimization

```
1 Function AntColonyOptimization($T_{max}$ = 18000, $n\_ants$ = 10, $Q$ = 10, $\alpha$ = 1, $\beta$ = 1, $\rho$ = 0.01):
2     initialization;
3     while time() ≤ $T_{max}$ do
4         construct solutions;
5         update pheromones;
6         daemon actions;
7     end
```

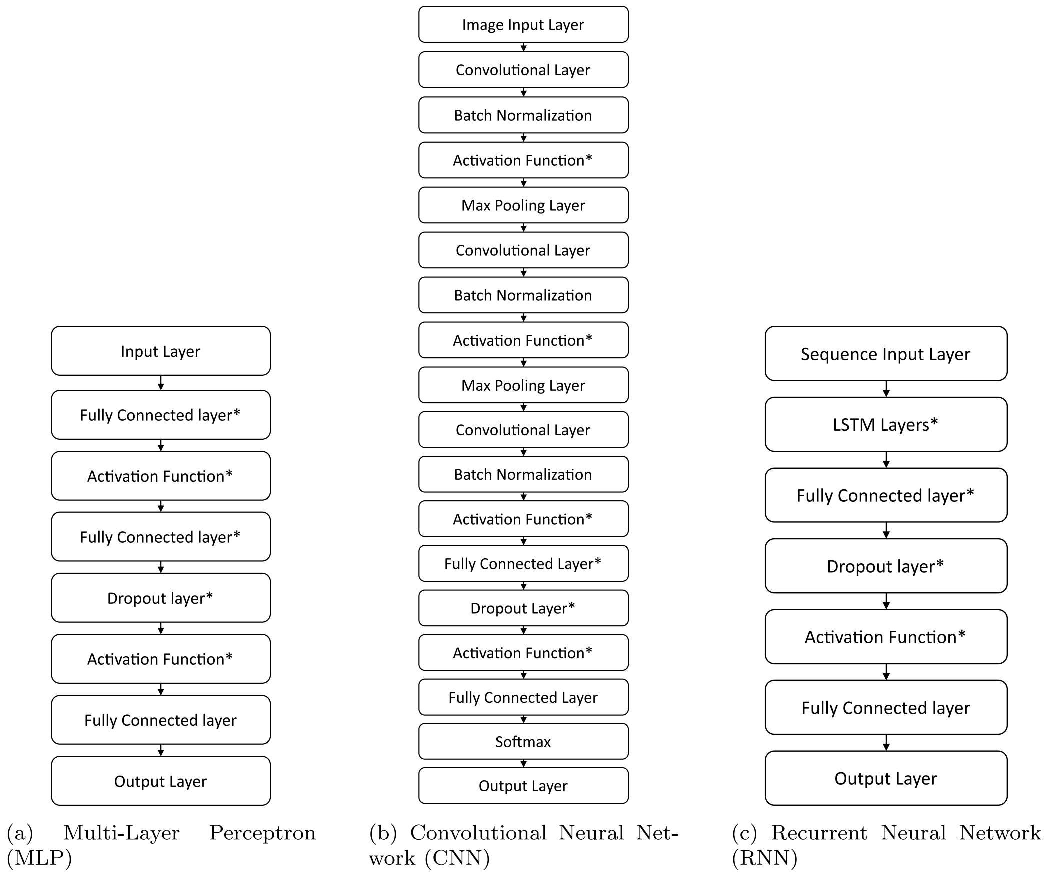


Fig. 9. Architectures of artificial neural networks. Layers marked with * indicate those whose hyperparameters are optimized using metaheuristic algorithms.

Table 3
Neural network hyperparameters.

Layer name	Hyperparameter	Possible values
Fully connected	Output size	From 10 to 200
Dropout layer	Dropout rate	From 0 to 1
Activation function	Nonlinear function name	ReLU, LReLU, ELU, Tanh
LSTM layer	Number of units	From 50 to 400
Training setting	**Description**	**Possible values**
Initial learn rate	Initial learning rate used for training	From 0.001 to 0.1
Max epochs	Maximum number of epochs to use for training	From 2 to 100
Shuffle	Criteria used for data shuffling	Once, never, every-epoch
Optimizer	Solver for training network	sgdm, adam

recall information over extended periods by using gates to regulate information flow (DiPietro & Hager, 2020). It leverages its internal state to retain pertinent information from past inputs and outputs, which is then used to generate the output sequence or classify the input sequence. The dataset used for this purpose is sourced from Saxena and Goebel (2008) and pertains to Turbofan Engine Degradation Simulation while the neural network architecture is depicted in Fig. 9(c).

4.7. Methodology

To effectively compare various metaheuristics for optimizing the hyperparameters of neural network architectures, it is essential to establish criteria that consider two key aspects: the quality of the solutions, evaluated through performance metrics, and the time required to achieve these performance metrics. The metaheuristic algorithms are compared by considering the best hyperparameter combination

Algorithm 9: Method for comparing the results of different metaheuristic algorithms for the same ANN

```
Data:
    - $n$: number of metaheuristic algorithms under consideration
    - $\Delta t$: time interval used to check the best solution. It corresponds to the average duration of the evaluation process (Algorithm 1)
    - $T_{max}$: all metaheuristics stop within this time
    - $X$: a set of size $(1 + \lfloor T_{max}/\Delta t \rfloor)$ containing the score vector of all metaheuristics that change over time
    - $T$: a time vector of size $(1 + \lfloor T_{max}/\Delta t \rfloor)$ containing all-time points from 0 to $T_{max}$ and $\Delta t$ as sample time

$X \leftarrow \emptyset$ ;
for $t \leftarrow 0; t \le T_{max}; t \leftarrow t + \Delta t$ do
    $x \leftarrow$getCurrentScores($t$)
    $X \leftarrow X \cup \{x\}$ ;                    /* add $x$ to set $X$ */
end
$f_{min} \leftarrow$ the lowest score obtained taking into account all metaheuristics and runs performed ;
$f_{max} \leftarrow$ the highest score obtained taking into account all metaheuristics and runs performed ;
Transform each element of $X$ using Eq. (5) or (4) according to the performance metric used for this ANN ;
return $(T, X)$;
```

found within a fixed amount of time. Additionally, a hyperparameter optimization algorithm should aim to find an optimal solution by minimizing the number of evaluations of a hyperparameter vector as defined in Algorithm 1. The time taken for this evaluation can vary depending on the complexity of the neural network architecture and the size of the training set. If a metaheuristic algorithm converges faster than the others, it can save time and provide a neural network for immediate use.

Since metaheuristic algorithms may not reach the same solution by the end of their search process, we examine how the score obtained by the Algorithm 1 of the best found solution changes over time. For the metaheuristics based on a single solution, we record the score of the current solution only if it is better than the previous one. For population-based metaheuristics, we instead record the score of the best solution in the current population. Thus, at each time point, we are able to know the score of the best solution for every metaheuristic algorithm. In our experiments, the score represents a Root Mean Squared Error (RMSE) for the model MLP and RNN, and classification accuracy for the model CNN. Considering that the score value can range within a small interval for a neural network (A_k), we decide to use a percentage value of it instead. Eqs. (4) and (5) transform Classification Accuracy and RMSE (denoted by f), respectively, into percentage values.

$$f'_{\%} = \frac{f - f_{min}}{f_{max} - f_{min}} \cdot 100 \tag{4}$$

$$f'_{\%} = \left(1 - \frac{f - f_{min}}{f_{max} - f_{min}}\right) \cdot 100 \tag{5}$$

where f_{min} and f_{max} are respectively the lowest and highest score reached during the search process for A_k, considering all metaheuristic algorithms. Algorithm 9 illustrates in details the procedure for comparing all metaheuristics for a given ANN. The function getCurrentScores(t) returns an n-tuple where each element is associated with a metaheuristic and represents the score of the best solution obtained up to t. If multiple runs are performed for a metaheuristic algorithm, this score is the average of all runs. If the metaheuristic does not yet have a candidate solution because it is too early, this value is set to $\emptyset$.

4.8. Results

This section presents a primary comparison between the previously mentioned metaheuristic algorithms and three different types of artificial neural networks (i.e., MLP, CNN, and RNN). In these experiments, the metaheuristics were implemented using MATLAB, and all experiments were conducted on an Intel Xeon 2.40 GHz processor with 16 GB of RAM. The results reported in this section were obtained using the average of the results obtained with 10 runs. Consequently, the total number of experiments per run is given by:

$$E = M \cdot D$$

where E represents experiments, M is the number of metaheuristic algorithms, and D denotes the number of application domains. Each experiment corresponds to a unique combination of one metaheuristic algorithm and one application domain. For each experiment, two types of information are collected: the best solution obtained at the end of the algorithm, and the computation time required to obtain the best solution.

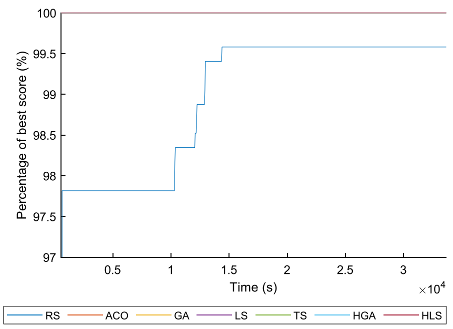
Table 4
Mean of the best score in each experiment after 10 runs.

	MLP (↓)	CNN (↑)	RNN (↓)
ACO	0.0999886	0.9850800	16.9978153
GA	0.0999848	**0.9992000**	16.9473436
HGA	**0.0999846**	0.9976400	**16.6612061**
LS	0.0999863	0.9942800	17.8173269
RS	0.0999877	0.9907200	17.8919289
TS	0.0999874	0.9681600	17.7157497

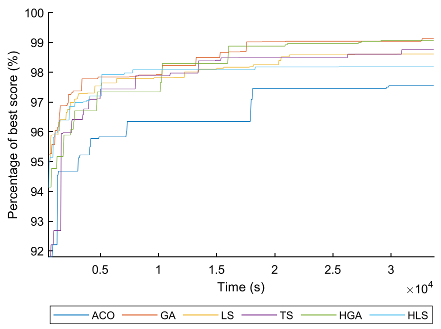
Table 4 shows the mean of the best score obtained in each experiment after 10 runs were performed. Note that the value given for MLP and RNN represents the RMSE, therefore it must be minimized, while for CNN it denotes the classification accuracy that we need to maximize.

As it can be seen by inspecting Table 4, the population-based metaheuristics seem to have better performances than the ones based on a single-solution. As a matter of fact, GA and HGA find a solution of higher quality than the other methods. In addition, HGA is an improvement of GA, which integrates a local search into the latter. Therefore, it benefits from both the advantages of a population-based algorithm and those of a single solution-based algorithm. Considering that a population-based metaheuristic analyzes many possible hyperparameter vectors in the same iteration, a higher number of possible neural network are tested than in a single-solution metaheuristic. As a result, this may have had an impact on the quality of the final results of such an algorithm.

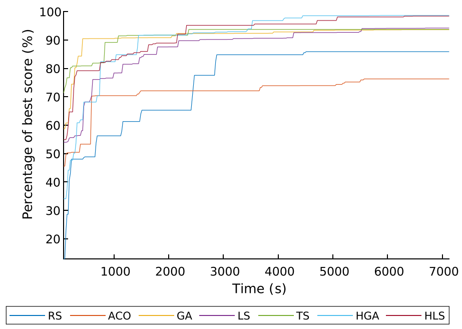
Another aspect to consider is the speed at which an algorithm achieves the best solution. Indeed, the quality of the solution alone is not sufficient to evaluate a metaheuristic algorithm. Usually, metaheuristic algorithms are used because of their ability to find an acceptable solution in less time than deterministic algorithms. In many optimization problems where they are used, computing the fitness of a feasible solution takes a small or even an insignificant amount of



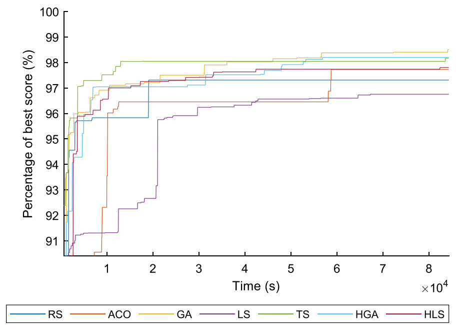
(a) Regression Problem (MLP)



(b) Regression Problem (MLP)



(c) Image Classification (CNN)



(d) Time-Series Forecasting (RNN)

Fig. 10. Performance comparison between the metaheuristic algorithms using the three artificial neural networks. The plots were generated using Algorithm 9. Unlike Figs. 10(c) and 10(d), Fig. 10(b) does not include the results of the *Random Search* because it returned values over a wide range, making the plot less interpretable. This is evident in Fig. 10(a).

time. In this problem, however, the evaluation of a hyperparameter vector requires a large amount of time. Therefore, an ideal metaheuristic algorithm should be able to find an acceptable solution with a few evaluations of the objective function, that is, evaluate a small number of hyperparameter vectors, since one single evaluation is quite time-consuming.

The computation time of all metaheuristic algorithms was analyzed using Algorithm 9 and the results are shown in Fig. 10.

In the case of the *Regression Problem*, we report the results of the experiments in Fig. 10(a) considering all algorithms. However, we underly the fact that all algorithms, except RS, achieve scores close to 100%. In contrast, RS has a much lower score, making it the least effective optimization algorithm. This difference occurs because the “Percentage of the Best Score” is computed using Eq. (5). As a result, the large gap between the maximum and minimum scores affects the percentage values for all algorithms. In general, Random Search can produce highly variable results due obviously to its inherent randomness. In some cases, it may perform well, but in others, especially with limited iterations, it may yield poor results due to the stochastic nature of these methods. In addition, RS may not effectively navigate the hyperparameter space to find configurations that minimize these error metrics, particularly when the relationship between hyperparameters and the resulting predictions is complex and non-linear, as can be the case in regression problems with MLP. To make our comparison clearer, we decided to exclude RS from this analysis. Fig. 10(b) does not include the results from Random Search because its scores vary widely, which would make the plot harder to understand. In this updated figure, we see that the HGA not only achieves the best solution, as shown in Table 4, but also finds better solutions faster than the other algorithms. Its speed and effectiveness make HGA an excellent choice when used with an MLP for regression problems.

On the other hand, if we consider *Image Classification* (Fig. 10(c)), we can see that TS initially finds an acceptable solution faster than the others, even if such a solution is further improved by HGA and HLS over time. This shows that although TS does not achieve a better solution than population-based metaheuristics, it can still be a good alternative because it balances the quality of the solution with the time required to achieve it. This is a good aspect since in the application of artificial neural networks, time is a key factor. In addition, we note that a new version of local search (HLS), presented in Zito et al. (2023c, 2023d), performs better than traditional local search.

With regard to *Time-Series Forecasting* (Fig. 10(d)), TS is confirmed to be a valid alternative to population-based metaheuristics such as GA and HGA, as indicated for image classification. Moreover, in this application domain, it is very close to the solutions of GA and HGA, which instead obtain the best solution in each domain.

5. Adapting metaheuristics for large hyperparameter space

In the previous section, we focused our analysis on how metaheuristics can assist in optimizing hyperparameters for three different types of common neural networks. In this section, we delve deeper into evaluating the performance of metaheuristic algorithms as hyperparameter optimization tools. We compare their effectiveness using more complex neural networks that are commonly found in real-world applications and have a higher number of hyperparameters. We explore five application domains, listed in Table 5, where neural networks are used to provide valuable insights when selecting the most suitable hyperparameter optimization algorithm based on the application domain and complexity of the architectures. Nine metaheuristic algorithms, categorized as population-based and single solution, were used to optimize hyperparameters for five distinct types of neural networks with varying levels of complexity.

Table 5
Application domains.

Application domain	Neural network model	Dataset	Ref.
Regression	Simple-FCNN	House price	Anna Montoya (2016)
Binary classification	Fully Connected Neural Network (FCNN)	Credit default prediction	Yeh (2016)
Image classification	AlexNet (CNN)	MNIST	Deng (2012)
Forecasting	Recurrent Neural Network (RNN)	Turbofan engine degradation simulation data set	Saxena and Goebel (2008)
Anomaly detection	AutoEncoder (AE)	NLD-KDD	Tavallaee et al. (2009)

Table 6
Hyperparameters of hidden layers in Simple-FCNN.

No.	Hidden layer	Hyperparameters	Possible values
1	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh
2	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh
3	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh
4	Dropout	Dropout rate	[0,1]

5.1. Models and data

We now discuss the architecture of the neural networks used for our analyses that find applications in real-world problems. For each neural network, we provide a table containing the neural layers along with their hyperparameters.

The first application domain we analyzed is in the field of Regression. One example of an application of such neural networks is in the field of finance, such as forecasting house prices over time in the market (Anna Montoya, 2016). In this context, the challenge is to forecast the house price taking into account several factors such as the building class, type of road access, lot size in square feet, and other 76 features. The neural network architecture denoted as *Simple-FCNN* used for this task is shown in Table 6.

In the context of *Binary Classification*, a fully connected neural network, denoted with *FCNN*, can serve as a classifier to discern the underlying relationship between input features and the binary output classes. The neural network architecture used for this particular application domain is outlined in Table 7 which shows the hidden layers along with their corresponding hyperparameters. The dataset employed for this study is The Miscellaneous Default of Credit Card Clients Dataset, also known as the PHM08 Challenge Dataset, which focuses on the default payment behavior of credit card clients in Taiwan. This dataset was sourced from credit card customers in Taiwan and offers valuable insights into the factors that influence credit card defaults (Yeh, 2016).

For what concerns *Image Classification*, the neural network architecture employed is *AlexNet*, a deep CNN architecture with eight layers, including five convolutional and three fully connected layers. It uses ReLU activation, dropout regularization, and data augmentation to enhance performance and prevent overfitting (Krizhevsky et al., 2012). Table 8 outlines all the neural layers with their respective hyperparameters and their possible values.

For *Forecasting*, we employed a *Recurrent Neural Network* in the analysis conducted in Section 4. The choice of recurrent units used in the RNN is itself a hyperparameter of the neural network and can include Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Fully-connected RNN (FCRNN) (ArunKumar et al., 2022). The dataset used in this study is sourced from Saxena and Goebel (2008) and focuses on Turbofan Engine Degradation Simulation. The neural network architecture employed for this purpose is outlined in Table 9.

Table 7
Hyperparameters of hidden layers in FCNN.

No.	Hidden layer	Hyperparameters	Possible values
1	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh, Linear
2	Batch normalization		
3	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh, Linear
4	Batch normalization		
5	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh, Linear
6	Dropout	Dropout rate	[0,1]

Table 8
Hyperparameters of hidden layers in AlexNet.

No.	Hidden layer	Hyperparameters	Possible values
1	Convolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU
2	Max pooling		
3	Convolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU
4	Max pooling		
5	Convolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU
6	Convolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU
7	Convolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU
8	Max pooling		
9	Flatten		
10	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh
11	Dropout	Dropout rate	[0,1]
12	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh
13	Dropout	Dropout rate	[0,1]
14	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	Linear

Finally, the last application domain is in the field of *Anomaly Detection*. *AutoEncoder* (AE) is a type of artificial neural network that is commonly used for anomaly detection due to its ability for dimensional

Table 9
Hyperparameters of hidden layers in RNN.

No.	Hidden layer	Hyperparameters	Possible values
1	Recurrent	Number of units	10, 50, 100, 200, 250, 500
		Activation	ReLU, ELU, Tanh
		Type	LSTM, GRU, FRNN
2	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh
3	Dropout	Dropout rate	[0,1]
4	Fully connected	Output size	25, 50, 100, 150, 200, 250, 500, 800
		Activation	ReLU, ELU, Tanh
5	Dropout	Dropout rate	[0,1]

Table 10
Hyperparameters of hidden layers in AE.

No.	Hidden layer	Hyperparameters	Possible values
1	Convolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU
2	Dropout	Dropout rate	[0,1]
3	Convolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU
4	Deconvolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU
5	Dropout	Dropout rate	[0,1]
6	Deconvolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU
7	Deconvolutional	Filters	8, 16, 32, 64, 128, 256
		Kernel size	2, 3, 5, 7, 11
		Activation	ReLU

reduction and data compression. It consists of an encoder network that compresses the input data into a lower-dimensional representation, and a decoder network that reconstructs the original input data from the compressed representation (Yang et al., 2022). The AutoEncoder learns to accurately reconstruct the input data, capturing the underlying patterns and features of the normal data. AutoEncoders are effective for anomaly detection because they can capture complex patterns and relationships in the data, making them sensitive to deviations from normal behavior (Chow et al., 2020). To optimize the hyperparameters of an AutoEncoder whose architecture is described in Table 10, we used the NSL-KDD dataset (Tavallaee et al., 2009), which is an improved version of the KDD CUP 99 dataset for intrusion detection. The dataset comprises various types of attacks that can compromise the security of a computer network by unauthorized individuals. It consists of 41 features, 125,973 records in the training set, and 22,544 records in the test set. The attack classes are divided into four different categories: U2R (User to Root), R2L (Remote to Local), Probe (surveillance), and DoS (Denial of Service) (Cavallaro et al., 2023).

In addition to the hyperparameters of the neural network architecture, we include other hyperparameters used for setting the training process and which are reported in Table 11a. By considering that each application domain has different metrics to evaluate the resultant neural network as reported in Table 11b, we regard a performance metric as a value that allows us to determine the goodness of a solution, that is, the hyperparameter vector.

5.2. Complexity analysis

When attempting to apply the metaheuristics described in Section 4, we initially observed poor performance. This was attributed to the lower convergence rates of the optimization algorithms and the high computational costs associated with evaluating candidate solutions. The dimensions of the search space for these algorithms, as presented in Table 12, indicate the scope within which they search for optimal hyperparameter combinations to enhance performance. The size of the search space is directly influenced by the number of hyperparameters considered in a neural network. The selected neural networks, known for their complexity and practical applications, present significant challenges in this context. Evaluating combinations of hyperparameters can be time-consuming, particularly given the complexity of the neural networks and the expansive nature of the search space. The dimensionality of the hyperparameter search space significantly affects the convergence of a metaheuristic algorithm. The complexity of metaheuristic algorithms can be analyzed by examining both population-based and single-solution approaches.

In population-based metaheuristic algorithms, such as Genetic Algorithm and Particle Swarm Optimization, a group of candidate solutions (the population) evolves over iterations. The time complexity of these algorithms is influenced by several factors. First, the population size plays a crucial role; a larger population can improve exploration of the search space but increases computational costs. The time complexity is clearly $\Omega(n \cdot g)$, where n is the population size and g is the number of generations (Omidvar et al., 2022). Since each candidate solution must be evaluated, we need to consider the computational cost of the objective function and the number of hyperparameters involved. If evaluating one single solution takes $T(m)$ time, where m is the size of each individual in the population, the overall evaluation cost becomes $\mathcal{O}(n \cdot g \cdot T(m))$.

Conversely, single-solution metaheuristics like Simulated Annealing or Tabu Search focus on iteratively improving a single candidate solution. It can be noted that the time complexity is $\mathcal{O}(g \cdot T(m))$, where g is the number of iterations, m the size of a solution and $T(m)$ is the time taken for each iteration's evaluation. Single-solution methods often evaluate a limited number of neighboring solutions at each iteration, which can reduce computational costs compared to population-based methods. However, this approach may limit their ability to escape local optima.

5.3. Methodology

To address the challenges posed by the extensive search space and high computational demands, we implemented several techniques aimed at accelerating the search process. A key strategy involved adopting a parallel evaluation approach, whereby multiple candidate solutions are evaluated simultaneously whenever feasible. This method leverages each core of the CPU to process evaluations of hyperparameter combinations, thereby reducing the overall time required for algorithm evaluation. This parallel evaluation strategy was employed across all previously described metaheuristic algorithms, albeit with variations in implementation. In population-based metaheuristic algorithms, individuals within the population whose fitness values had not yet been determined were evaluated concurrently at each iteration. Conversely, in single-solution metaheuristic algorithms, the approach varied based on the specific strategies employed. For example, in iterated local search or tabu search, which uses neighborhood structures, all candidate solutions were evaluated simultaneously. The effectiveness of this parallel evaluation approach is contingent upon available hardware resources: more CPU cores facilitate a greater number of simultaneous evaluations. All algorithms were implemented in Python, and experiments were conducted on an Intel Xeon 2.40 GHz processor with 8 cores. The machine learning framework used for this analysis is Keras, a widely recognized tool in the field.

5.4. Results

The first challenge is to determine which algorithm can find the optimal hyperparameter combination for a neural network in the shortest possible time. All algorithms were tested by using the same maximum

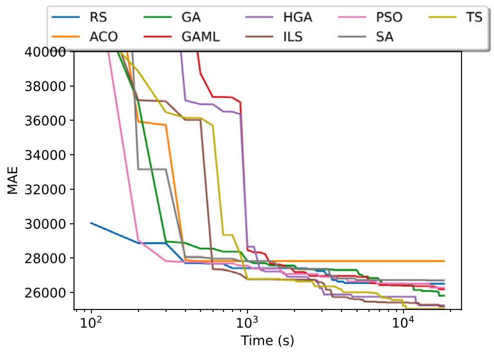
Table 11
Training details.

(a) Training setting.

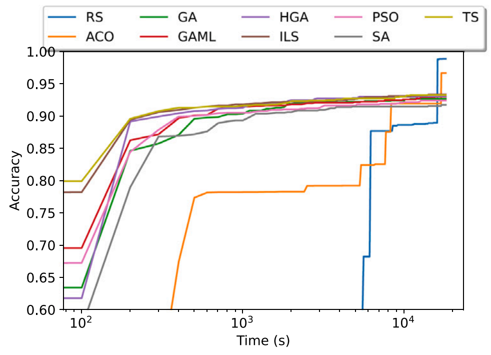
Training setting	Possible values
Batch size	16, 32, 64, 128
Shuffle	Yes, No
Optimizer	SGDM, ADAM, RMSprop
Learning rate	0.001, 0.01, 0.1
Max epochs	5, 10, 20, 40, 50, 100

(b) Loss functions for application domains.

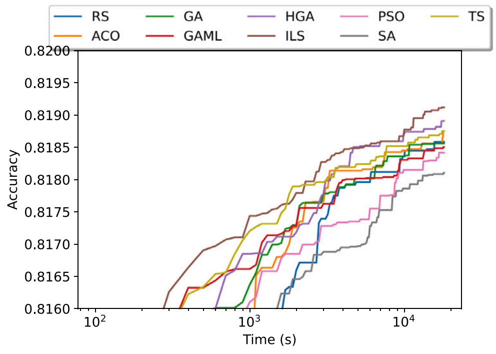
Application domain	Loss function
Regression	Mean absolute error
Binary classification	Binary cross-entropy
Image classification	Categorical cross-entropy
Forecasting	Mean absolute error
Anomaly detection	Categorical cross-entropy



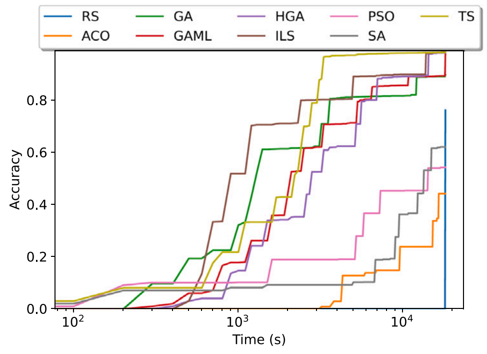
(a) Regression (Simple-FCNN)



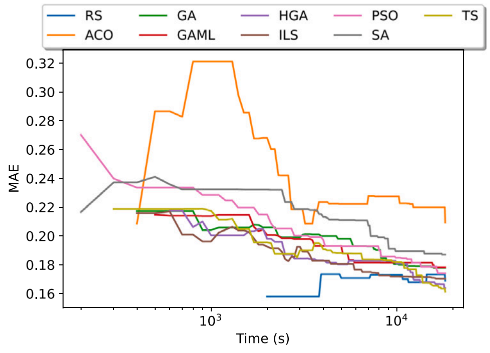
(b) Anomaly Detection (AutoEncoder)



(c) Binary Classification (FCNN)



(d) Image Classification (AlexNet)



(e) Forecasting (RNN)

Fig. 11. Comparison of performance metrics over time for different metaheuristic algorithms across various application domains. Each subfigure highlights the evolution of fitness values for a specific neural network model.

time, which served as the stopping criterion for the search process. We used a time limit equal to five hours. The reason why the algorithms were let run for such a maximum time is that in real applications we need an optimized neural network within a certain amount of time. Obviously, a longer time allocation typically results in a more optimized neural network.

Table 13 displays the average performance metric across the runs for each neural network model considered. Note that the performance metrics given for the models RNN and MLP, that are associated with the application domains Forecasting and Regression respectively, represents the Mean Squared Error (MSE), instead for the other models they denote the accuracy. As it is possible to see, in those cases

Table 12
Search space size.

Neural network model	Hyperparameters	Search space size
Simple-FCNN	12	5.9720E+07
FCNN	12	1.4156E+08
AlexNet	22	4.8373E+15
RNN	14	1.3437E+09
Autoencoder	17	1.0498E+12

where the search space is larger, such as with AutoEncoder models, RS appears to be the preferred method for selecting hyperparameters for the neural network. Random Search explores the search space randomly, increasing the likelihood of discovering an acceptable solution. However, this approach may not always be the optimal choice due to its purely random approach. On the other hand, for other applications, a single solution approach performs better compared to population-based algorithms such as genetic algorithms and their variants, ACO and PSO. Specifically, Iterated Local Search employs efficient search strategies that enable the algorithm to obtain a hyperparameter configuration with the highest accuracy in complex neural network as AlexNet, which has a vast search space and high complexity compared to other neural networks discussed here.

5.5. Statistical analysis

In analogy to Section 4, the computation time of all metaheuristic algorithms was analyzed using Algorithm 9, and the evolution of the performance metric over time is illustrated in Fig. 11. In addition to the conclusions drawn in the previous section, it is evident that all algorithms tend to converge towards a good solution, although some algorithms are more efficient than others. For instance, ILS along with TS are particularly effective in identifying the optimal hyperparameter configuration that maximizes the performance of the optimized neural network. One of the challenges in hyperparameter optimization is that different hyperparameter configurations for a given neural network can yield similar fitness values, which may mislead the optimization algorithm. This issue is well-known in the field of hyperparameter optimization, and it can be observed in the curves presented in those figures. Clearly, this is an ongoing challenge that deserves further investigation in the future (Bischl et al., 2023). In this regard, this study could serve as a foundational reference for a more detailed analysis of this problem in the future.

Another important aspect to consider when comparing algorithms, in addition to the goodness of the solution, is their reliability. Since they involve random components that influence the search process, it is impossible to control the outcomes beforehand. The randomness in the search strategy employed by metaheuristic algorithms leads to variations in results across multiple runs of the same algorithm. To address such a variability, each algorithm is run 10 times to assess its reliability compared to others. A statistical analysis is then carried out in order to demonstrate which algorithm is more reliable than others. We stress the fact that by reliability of an algorithm we mean its ability to obtain the same results, that is the hyperparameters combination for a neural network, over independent runs, where each run is initialized with a different random seed. In other words, reliability corresponds to the precision of an algorithm, and it is inversely proportional to the degree of dispersion of an algorithm running several times. In statistics, the degree of dispersion is measured by the InterQuartile Range (IQR), which is, specifically, the difference between the third quartile (Q3) and the first quartile (Q1) in a set of values that represent the best fitness obtained by the algorithm in each run. Lower IQR values indicate less variability in the quality of the solution produced by each run and therefore more homogeneity of the solutions. Higher IQR values, instead, suggest greater variability and, as a consequence, a lower reliability of the algorithm (Zambrano-Vega et al., 2017). Indeed, the fact that such an algorithm can reach a solution better than the solutions obtained by other algorithms with a low IQR, it might simply be due to randomness, and it is, clearly, a probabilistically unlikely event. By calculating the average IQR value for each algorithm across all models, we can create a ranking of the algorithms based on these criteria and match this ranking with that obtained in the previous paragraph.

Thus, let us now analyze the dispersion of the algorithms so to measure their reliability. To conduct this analysis, we used box plots that display the performance metrics achieved by each algorithm in a specific application domain. Fig. 12 illustrates the box plots for various application domains. Each plot visually represents the distribution using key values such as the minimum, first quartile (Q1), median, third quartile (Q3), and maximum. It is evident that algorithms such as RS and ACO exhibit a high degree of dispersion compared to others. The dispersion levels of the other algorithms, instead, vary depending on the application domain in which they are applied. For example, SA and PSO show high dispersion in image classification due to the numerous hyperparameters of models such as AlexNet, making them less suitable for large search spaces. On the other hand, SA and PSO demonstrate the lowest dispersion in anomaly detection, although they still fall short of the performance achieved by RS and ACO in terms of performance metrics. To determine which algorithm performs the best across all models in the provided dataset, we can analyze the interquartile range values for each of them and compare their overall performance. In Fig. 13, we can observe that population-based algorithms generally exhibit a wider range of performance, as indicated by their larger interquartile ranges compared to single solution-based algorithms. For example, algorithms like ACO, GA, and PSO show relatively higher IQR values across different models, suggesting greater variability in their performance. On the other hand, single solution-based algorithms like ILS, RS, and TS tend to have smaller IQR values, denoting a more consistent performance within the dataset. These algorithms may converge more quickly to a solution but might struggle in exploring a diverse solution space when compared to population-based algorithms.

6. Discussion of results

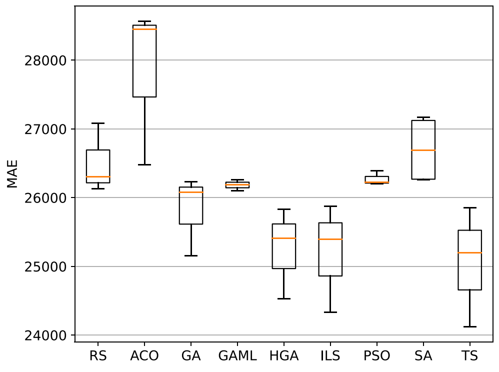
Our study delved into the application of metaheuristic algorithms to the optimization of the hyperparameters of artificial neural networks. Through a series of experiments across various application domains, the behavior and reliability of different metaheuristic algorithms were explored. Given the vast number of potential hyperparameter combinations for parametric artificial neural networks, deterministic methods cannot efficiently identify the optimal combination within a reasonable amount of time. We compared two classes of metaheuristics population-based and single solution-based for hyperparameter optimization of artificial neural networks. Each algorithm was run for the same amount of time equal to five hours. The challenge of our experiments was to get the best configuration of hyperparameters within this amount of time.

The obtained results show that population-based metaheuristics, particularly GA and its hybrid version HGA, excel in optimizing neural network models such as Multi-Layer Perceptron, Convolutional Neural Network, and Recurrent Neural Network with a few numbers of hyperparameters. This is because population-based algorithms operate on multiple individuals simultaneously, ensuring a diverse set of solutions within a population and enabling exploration of different regions of the search space. As future work, to improve the efficiency and reliability of population-based algorithms, following the work in Cutello et al. (2010a), we can make use of the concept of entropy to study their effectiveness and infer specific convergence properties.

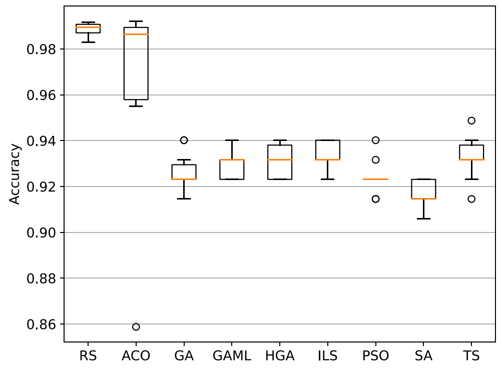
Conversely, when dealing with neural network models with an extensive hyperparameter space (see Section 5), single solution algorithms demonstrate superior performance in optimizing neural networks and algorithm reliability. Examples are Tabu Search, Iterated Local Search, and Random Search. Among these, random search emerges

Table 13
Average performance metrics of metaheuristic algorithms across different neural network models.

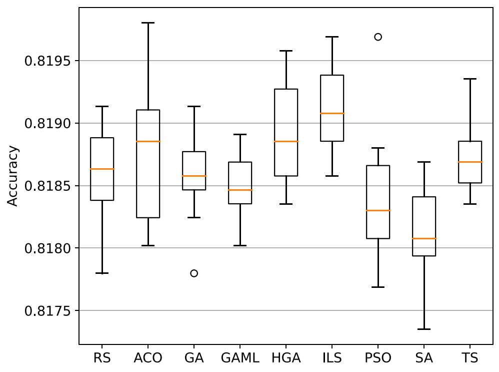
Model	RS	ACO	GA	GAML	HGA	ILS	PSO	SA	TS
AE (↑)	**0.9884**	0.9665	0.9265	0.9299	0.9308	0.9333	0.9239	0.9171	0.9325
AlexNet (↑)	0.7619	0.4417	0.8973	0.9793	0.9820	**0.9843**	0.5435	0.6212	0.9837
RNN (↓)	0.1692	0.2096	0.1782	0.1786	0.1641	0.1707	0.1745	0.1875	**0.1616**
FCNN (↓)	0.8186	0.8188	0.8186	0.8185	0.8189	0.8191	0.8184	**0.8181**	0.8188
Simple-FCNN (↓)	26 505	27 832	25 820	26 184	25 255	25 199	26 273	26 702	**25 057**



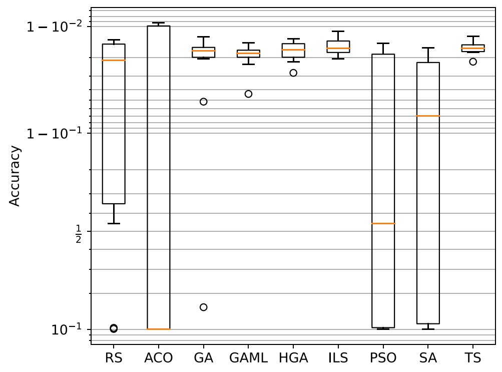
(a) Regression (Simple-FCNN)



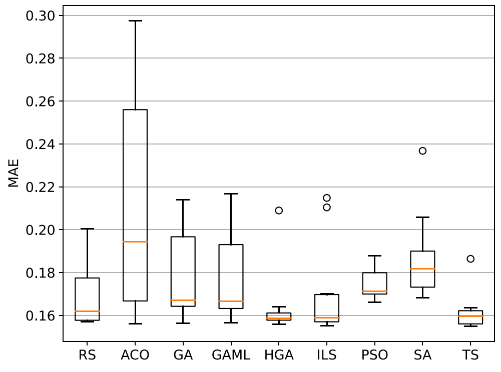
(b) Anomaly Detection (AutoEncoder)



(c) Binary Classification (FCNN)



(d) Image Classification (AlexNet)



(e) Forecasting (RNN)

Fig. 12. Box plots illustrating the reliability and variability of metaheuristic algorithms across different application domains.

as the preferred optimization method for neural network architecture optimization, especially in scenarios with large search space sizes. Random search facilitates solution diversification and exploration of high-dimensional search spaces. However, it is essential to note that while random search is effective in many cases, it may not always be the optimal choice. Indeed, from the statistical analysis, we concluded that random search could not be reliable due to its inherent random nature. For example, in image classification applications utilizing AlexNet on the MNIST dataset, along with ACO, PSO, and SA showed a low reliability, likely due to the high number of hyperparameters in the neural network.

7. Future work

As a future line of research, we propose to analyze another well-known metaheuristic based on population, namely Immune Algorithms.

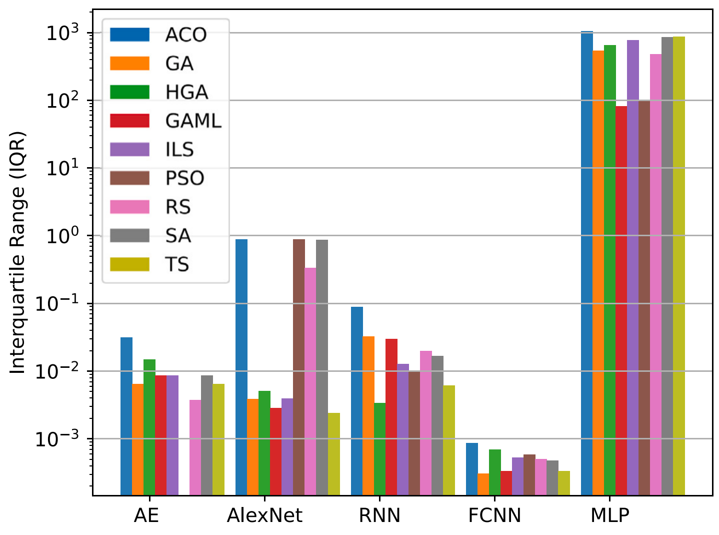


Fig. 13. Interquartile range comparison of algorithms across neural network architectures.

These algorithms represent a well-established population-based metaheuristic (de Castro & Timmis, 2002), which takes inspiration from observing the dynamics of the immune system through which it efficiently carries out detection, recognition, and learning tasks. A special class of immune algorithms are those based on the clonal selection principle (Cutello et al., 2010b; de Castro & Von Zuben, 2002; Timmis et al., 2008), which have been shown to be reliable and efficient for purely mathematical or applied optimization problems (Stracquadanio et al., 2015) and for which many interesting problems, in the initial configuration, arise such as the setting of the age assignment at birth for each individual, see Vitale et al. (2019) for instance.

In addition, we aim to analyze the significance of hyperparameters in neural networks. Hyperparameters play a crucial role in determining the performance and stability of these models. By understanding how different hyperparameter settings affect outcomes, we can gain valuable insights that enhance the effectiveness of algorithms across various applications. Moreover, a thorough analysis of hyperparameters can help reduce the search space during optimization. By identifying which hyperparameters are most influential, we can focus our search efforts on the most significant configurations, thereby improving efficiency and reducing computational costs.

It is essential to recognize that HPO primarily focuses on fine-tuning the hyperparameters of a model without altering its underlying architecture, which remains fixed. In contrast, NAS algorithms are designed for scenarios where the architecture of the neural network needs to be automatically generated based on the data. While both approaches aim to improve model performance, HPO is concerned with optimizing a predefined neural network architecture, whereas NAS generates an entirely new architecture from scratch. Additionally, NAS can also optimize hyperparameters, as demonstrated in Zito et al. (2024). As a direction for future research, we plan to investigate several NAS algorithms and compare them with metaheuristic-based hyperparameter optimization techniques.

8. Conclusions

In the present work, we investigated the application of Automated Machine Learning techniques for the selection of appropriate machine learning models tailored to specific tasks. Our findings highlight the critical role of hyperparameter optimization in enhancing the performance of neural networks, enabling these models to fully leverage their capabilities. This optimization is especially advantageous in scenarios with hardware resource constraints, where the performance and the efficiency of the model are very important.

We demonstrated the effectiveness of metaheuristic algorithms in hyperparameter optimization, providing insights into key implementation details such as solution representation, evaluation metrics, and strategies for accelerating algorithm convergence. Given the extensive search space of hyperparameters in neural network architectures, often reaching up to 10^{15} in real-world applications, our study underscores the necessity for robust implementation mechanisms including the parallelization methods discussed in this article.

Overall, our work contributes to the growing body of knowledge on AutoML and metaheuristics, offering practical approaches that can significantly improve model selection and performance in diverse application domains.

CRediT authorship contribution statement

Francesco Zito: Conceptualization, Methodology, Investigation, Formal analysis, validation, Writing manuscript. **El-Ghazali Talbi:** Conceptualization, Methodology, Investigation, Formal analysis, validation, Writing manuscript. **Claudia Cavallaro:** Conceptualization, Methodology, Investigation, Formal analysis, validation, Writing manuscript. **Vincenzo Cutello:** Conceptualization, Methodology, Investigation, Formal analysis, validation, Writing manuscript. **Mario Pavone:** Conceptualization, Methodology, Investigation, Formal analysis, validation, Writing manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors wish to thank to the anonymous reviewers for their important and valuable comments.

This research is supported by the project Future Artificial Intelligence Research (FAIR), Italy – PNRR MUR Cod. PE0000013 - CUP: E63C22001940006.

Data availability

No data was used for the research described in the article.

References

Aarts, E., Korst, J., & Michiels, W. (2006). *Simulated annealing* (pp. 187–210). Springer US, http://dx.doi.org/10.1007/0-387-28356-0_7.

Abdipoor, S., Yaakob, R., Goh, S. L., & Abdullah, S. (2023). Meta-heuristic approaches for the university course timetabling problem. *Intelligent Systems with Applications*, *19*, Article 200253. http://dx.doi.org/10.1016/j.iswa.2023.200253.

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, *4*(11), Article e00938. http://dx.doi.org/10.1016/j.heliyon.2018.e00938.

Akay, B., Karaboga, D., & Akay, R. (2022). A comprehensive survey on optimizing deep learning models by metaheuristics. *Artificial Intelligence Review*, *55*(2), 829–894. http://dx.doi.org/10.1007/s10462-021-09992-0.

Aljarah, I., Faris, H., & Mirjalili, S. (2018). Optimizing connection weights in neural networks using the whale optimization algorithm. *Soft Computing*, *22*(1), 1–15. http://dx.doi.org/10.1007/s00500-016-2442-1.

Alrowais, F., Althahabi, S., S. Alotaibi, S., Mohamed, A., Ahmed Hamza, M., & Marzouk, R. (2023). Automated machine learning enabled cybersecurity threat detection in internet of things environment. *Computer Systems Science and Engineering*, *45*(1), 687–700. http://dx.doi.org/10.32604/csse.2023.030188.

Anna Montoya, D. (2016). House prices - advanced regression techniques. (Accessed 17 January 2024).

ArunKumar, K., Kalaga, D. V., Mohan Sai Kumar, C., Kawaji, M., & Brenza, T. M. (2022). Comparative analysis of gated recurrent units (GRU), long short-term memory (LSTM) cells, autoregressive integrated moving average (ARIMA), seasonal autoregressive integrated moving average (SARIMA) for forecasting COVID-19 trends. *Alexandria Engineering Journal*, *61*(10), 7585–7603. http://dx.doi.org/10.1016/j.aej.2022.01.011.

Bahri, M., Salutari, F., Putina, A., & Sozio, M. (2022). AutoML: state of the art with a focus on anomaly detection, challenges, and research directions. *International Journal of Data Science and Analytics*, *14*(2), 113–126. http://dx.doi.org/10.1007/s41060-022-00309-0.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(2), 281–305.

Bianchi, L., Dorigo, M., Gambardella, L. M., & Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, *8*(2), 239–287. http://dx.doi.org/10.1007/s11047-008-9098-4.

Bibaeva, V. (2018). Using metaheuristics for hyper-parameter optimization of convolutional neural networks. In *2018 IEEE 28th international workshop on machine learning for signal processing* (pp. 1–6). http://dx.doi.org/10.1109/MLSP.2018.8516989.

Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., Deng, D., & Lindauer, M. (2023). Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data Mining and Knowledge Discovery*, *13*(2), http://dx.doi.org/10.1002/widm.1484.

Bouktif, S., Fiaz, A., Ouni, A., & Serhani, M. (2018). Optimal deep learning LSTM model for electric load forecasting using feature selection and genetic algorithm: Comparison with machine learning approaches †. *Energies*, *11*(7), 1636. http://dx.doi.org/10.3390/en11071636.

Cavallaro, C., Crespi, C., Cutello, V., Pavone, M., & Zito, F. (2024). Group dynamics in memory-enhanced ant colonies: The influence of colony division on a maze navigation problem. *Algorithms*, *17*(2), 63. http://dx.doi.org/10.3390/a17020063.

Cavallaro, C., Cutello, V., Pavone, M., & Zito, F. (2023). Discovering anomalies in big data: a review focused on the application of metaheuristics and machine learning techniques. *Frontiers in Big Data*, *6*, http://dx.doi.org/10.3389/fdata.2023.1179625.

Cavallaro, C., Cutello, V., Pavone, M., & Zito, F. (2024). Machine learning and genetic algorithms: A case study on image reconstruction. *Knowledge-Based Systems*, *284*, Article 111194. http://dx.doi.org/10.1016/j.knosys.2023.111194.

Chai, J., Zeng, H., Li, A., & Ngai, E. W. (2021). Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, *6*, Article 100134. http://dx.doi.org/10.1016/j.mlwa.2021.100134.

Chen, C. (2024). Artificial intelligence aided pharmaceutical engineering: Development of hybrid machine learning models for prediction of nanomedicine solubility in supercritical solvent. *Journal of Molecular Liquids*, *397*, Article 124127. http://dx.doi.org/10.1016/j.molliq.2024.124127.

Chow, J., Su, Z., Wu, J., Tan, P., Mao, X., & Wang, Y. (2020). Anomaly detection of defects on concrete structures with the convolutional autoencoder. *Advanced Engineering Informatics*, *45*, Article 101105. http://dx.doi.org/10.1016/j.aei.2020.101105.

Colorni, A., Dorigo, M., & Maniezzo, V. (1992). *A genetic algorithm to solve the timetable problem* (Ph.D. thesis), Politecnico di Milano, Milan, Italy.

Crespi, C., Cutello, V., Pavone, M., & Zito, F. (2024). An agent framework to explore pathfinding strategies in maze navigation problem. *Le Matematiche*, *79*(2), 555–583. http://dx.doi.org/10.4418/2024.79.2.17.

Cutello, V., Mezzina, A., Pavone, M., & Zito, F. (2025). A real-time adaptive tabu search for handling zoom In/Out in map labeling problem. In P. Festa, D. Ferone, T. Pastore, & O. Pisacane (Eds.), *Learning and intelligent optimization* (pp. 108–122). Cham: Springer Nature Switzerland.

Cutello, V., Nicosia, G., Pavone, M., & Stracquadanio, G. (2010a). Entropic divergence for population based optimization algorithms. In *IEEE congress on evolutionary computation* (pp. 1–8). IEEE, http://dx.doi.org/10.1109/CEC.2010.5586044.

Cutello, V., Nicosia, G., Pavone, M., & Stracquadanio, G. (2010b). An information-theoretic approach for clonal selection algorithms. In E. Hart, C. McEwan, J. Timmis, & A. Hone (Eds.), *Artificial immune systems* (pp. 144–157). Berlin, Heidelberg: Springer Berlin Heidelberg.

Cutello, V., Pavone, M., & Zito, F. (2024a). Improving an immune-inspired algorithm by linear regression: A case study on network reliability. *Knowledge-Based Systems*, *299*, Article 112034. http://dx.doi.org/10.1016/j.knosys.2024.112034.

Cutello, V., Pavone, M., & Zito, F. (2024b). In D. Cantone, & A. Pulvirenti (Eds.), *Vol. 14070*, *Inferring a gene regulatory network from gene expression data. An overview of best methods and a reverse engineering approach* (pp. 172–185). Cham: Springer Nature Switzerland, http://dx.doi.org/10.1007/978-3-031-55248-9_9.

Dargan, S., Kumar, M., Ayyagari, M. R., & Kumar, G. (2020). A survey of deep learning and its applications: A new paradigm to machine learning. *Archives of Computational Methods in Engineering*, *27*(4), 1071–1092. http://dx.doi.org/10.1007/s11831-019-09344-w.

de Castro, L., & Timmis, J. (2002). *Artificial immune systems: A new computational intelligence approach*. London: Springer Verlag, URL: https://ci.nii.ac.jp/ncid/BA61106585.

de Castro, L., & Von Zuben, F. (2002). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, *6*(3), 239–251. http://dx.doi.org/10.1109/TEVC.2002.1011539.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

DiPietro, R., & Hager, G. D. (2020). Chapter 21 - deep learning: RNNs and LSTM. In S. K. Zhou, D. Rueckert, & G. Fichtinger (Eds.), *The Elsevier and MICCAI society book series*, *Handbook of medical image computing and computer assisted intervention* (pp. 503–519). Amsterdam, The Netherlands: Academic Press, http://dx.doi.org/10.1016/B978-0-12-816176-0.00026-0.

Dorigo, M., & Socha, K. (2018). An introduction to ant colony optimization. In T. F. Gonzalez (Ed.), *Handbook of approximation algorithms and metaheuristics, second edition* (pp. 395–408). Boca Raton: Chapman and Hall/CRC, http://dx.doi.org/10.1201/9781351236423-23.

El-mihoub, T., Hopgood, A., Nolle, L., & Alan, B. (2006). Hybrid genetic algorithms: A review. *Engineering Letters*, *3*(2), 124–137.

Elmasry, W., Akbulut, A., & Zaim, A. H. (2020). Evolving deep learning architectures for network intrusion detection using a double PSO metaheuristic. *Computer Networks*, *168*, Article 107042. http://dx.doi.org/10.1016/j.comnet.2019.107042.

Gharsalli, L., & Guérin, Y. (2019). A hybrid genetic algorithm with local search approach for composite structures optimization. In *8th European conference for aeronautics and space sciences* (p. 9). Madrid: AIP Publisher.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533–549. http://dx.doi.org/10.1016/0305-0548(86)90048-1, Applications of Integer Programming.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge, MA: The MIT Press, URL: https://dl.acm.org/citation.cfm?id=3086952.

Goswami, C., Gupta, M., Sharma, S., Ravindar, B., Al-Hilali, A. A., & Bader Alazzam, M. (2023). A theoretical analysis of recent development in statistical kind of machine learning. In *2023 3rd international conference on advance computing and innovative technologies in engineering*. IEEE, http://dx.doi.org/10.1109/icacite57410.2023.10182672.

Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, *37*(1), 388–427. http://dx.doi.org/10.1016/j.ijforecast.2020.06.008, URL: https://arxiv.org/pdf/1909.00590.

Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). *The springer series on challenges in machine learning*, *Automated machine learning: methods, systems, challenges*. Springer International Publishing, http://dx.doi.org/10.1007/978-3-030-05318-5.

Indolia, S., Goswami, A. K., Mishra, S., & Asopa, P. (2018). Conceptual understanding of convolutional neural network- a deep learning approach. *Procedia Computer Science*, *132*, 679–688. http://dx.doi.org/10.1016/j.procs.2018.05.069, International Conference on Computational Intelligence and Data Science.

Islam, M. T., Zhou, Z., Ren, H., Khuzani, M. B., Kapp, D., Zou, J., Tian, L., Liao, J. C., & Xing, L. (2023). Revealing hidden patterns in deep neural network feature space continuum via manifold learning. *Nature Communications*, *14*(1), http://dx.doi.org/10.1038/s41467-023-43958-w.

Jacobson, S. H., & Yücesan, E. (2004). Analyzing the performance of generalized hill climbing algorithms. *Journal of Heuristics*, *10*(4), 387–405. http://dx.doi.org/10.1023/b:heur.0000034712.48917.a9.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Vol. 25*, *Advances in neural information processing systems* (pp. 1097–1105). Curran Associates, Inc..

Lai, X., Yue, D., Hao, J.-K., & Glover, F. (2018). Solution-based tabu search for the maximum min-sum dispersion problem. *Information Sciences*, *441*, 79–94. http://dx.doi.org/10.1016/j.ins.2018.02.006.

Larsen, K., & Becker, D. (2021). *Automated machine learning for business*. Oxford University Press, URL: https://books.google.it/books?id=d-krEAAAQBAJ.

Li, Y., Shen, Y., Zhang, W., Zhang, C., & Cui, B. (2022). VolcanoML: speeding up end-to-end AutoML via scalable search space decomposition. *The VLDB Journal*, *32*(2), 389–413. http://dx.doi.org/10.1007/s00778-022-00752-2.

Li, G., Zhang, T., Tsai, C.-Y., Yao, L., Lu, Y., & Tang, J. (2024). Review of the metaheuristic algorithms in applications: Visual analysis based on bibliometrics. *Expert Systems with Applications*, *255*, Article 124857. http://dx.doi.org/10.1016/j.eswa.2024.124857.

Lu, Z., Cheng, R., Jin, Y., Tan, K. C., & Deb, K. (2022). Neural architecture search as multiobjective optimization benchmarks: Problem formulation and performance assessment. *IEEE Transactions on Evolutionary Computation*, *28*, http://dx.doi.org/10.1109/TEVC.2022.3233364, 1–1. URL: https://arxiv.org/abs/2208.04321.

Ma, H., Peng, T., Zhang, C., Ji, C., Li, Y., & Nazir, M. S. (2023). Developing an evolutionary deep learning framework with random forest feature selection and improved flow direction algorithm for NOx concentration prediction. *Engineering Applications of Artificial Intelligence*, *123*, Article 106367. http://dx.doi.org/10.1016/j.engappai.2023.106367.

Morales-Hernández, A., Van Nieuwenhuyse, I., & Rojas Gonzalez, S. (2022). A survey on multi-objective hyperparameter optimization algorithms for machine learning. *Artificial Intelligence Review*, *56*(8), 8043–8093. http://dx.doi.org/10.1007/s10462-022-10359-2.

Morteza, A., Yahyaeian, A. A., Mirzaeibonehkhater, M., Sadeghi, S., Mohaimeni, A., & Taheri, S. (2023). Deep learning hyperparameter optimization: Application to electricity and heat demand prediction for buildings. *Energy and Buildings*, *289*, Article 113036. http://dx.doi.org/10.1016/j.enbuild.2023.113036.

Nakisa, B., Rastgoo, M. N., Rakotonirainy, A., Maire, F., & Chandran, V. (2018). Long short term memory hyperparameter optimization for a neural network based emotion recognition framework. *IEEE Access*, *6*, 49325–49338. http://dx.doi.org/10.1109/ACCESS.2018.2868361.

Omidvar, M. N., Li, X., & Yao, X. (2022). A review of population-based metaheuristics for large-scale black-box global optimization—Part I. *IEEE Transactions on Evolutionary Computation*, *26*(5), 802–822. http://dx.doi.org/10.1109/TEVC.2021.3130838.

Plebe, A., & Pavone, M. (2018). Multi-objective genetic algorithm for interior lighting design. In G. Nicosia, P. Pardalos, G. Giuffrida, & R. Umeton (Eds.), *Vol. 10710*, *Machine learning, optimization, and big data* (pp. 222–233). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-72926-8_19.

Prechelt, L. (1998). Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, *11*(4), 761–767. http://dx.doi.org/10.1016/s0893-60809800010-0.

Profentzas, C., Almgren, M., & Landsiedel, O. (2021). Performance of deep neural networks on low-power IoT devices. In *CPS-IoT week '21*, *Proceedings of the workshop on benchmarking cyber-physical systems and internet of things*. ACM, http://dx.doi.org/10.1145/3458473.3458823.

Rajwar, K., Deep, K., & Das, S. (2023). An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. *Artificial Intelligence Review*, *56*(11), 13187–13257. http://dx.doi.org/10.1007/s10462-023-10470-y.

Rego, C., & Glover, F. (2007). *Local search and metaheuristics* (pp. 309–368). Boston, MA: Springer US, http://dx.doi.org/10.1007/0-306-48213-4_8, chapter 8.

Saber, S., & Salem, S. (2023). High-performance technique for estimating the unknown parameters of photovoltaic cells and modules based on improved spider wasp optimizer. *Sustainable Machine Intelligence Journal*, *5*, http://dx.doi.org/10.61185/smij.2023.55102.

Salehin, I., Islam, M. S., Saha, P., Noman, S., Tuni, A., Hasan, M. M., & Baten, M. A. (2024). AutoML: A systematic review on automated machine learning with neural architecture search. *Journal of Information and Intelligence*, *2*(1), 52–81. http://dx.doi.org/10.1016/j.jiixd.2023.10.002.

Salem, S. (2023). An improved binary quadratic interpolation optimization for 0-1 knapsack problems. *Sustainable Machine Intelligence Journal*, *4*, http://dx.doi.org/10.61185/smij.2023.44101.

Sarker, I. H. (2021). Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, *2*(6), 420. http://dx.doi.org/10.1007/s42979-021-00815-1.

Sarwar, N., Irshad, A., Naith, Q. H., D.Alsufiani, K., & Almalki, F. A. (2024). Skin lesion segmentation using deep learning algorithm with ant colony optimization. *BMC Medical Informatics and Decision Making*, *24*(1), http://dx.doi.org/10.1186/s12911-024-02686-x.

Saxena, A., & Goebel, K. (2008). *Turbofan engine degradation simulation data set*. NASA Ames Prognostics Data Repository.

Shi, Y., & Eberhart, R. (1999). Empirical study of particle swarm optimization. *Vol. 3*, In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)* (pp. 1945–1950). IEEE, http://dx.doi.org/10.1109/CEC.1999.785511, Vol. 3.

Smithson, S. C., Yang, G., Gross, W. J., & Meyer, B. H. (2016). Neural networks designing neural networks: Multi-objective hyper-parameter optimization. In *2016 IEEE/ACM international conference on computer-aided design* (pp. 1–8). http://dx.doi.org/10.1145/2966986.2967058.

Stork, J., Eiben, A. E., & Bartz-Beielstein, T. (2022). A new taxonomy of global optimization algorithms. *Natural Computing*, *21*(2), 219–242. http://dx.doi.org/10.1007/s11047-020-09820-4.

Stracquadanio, G., Greco, O., Conca, P., Cutello, V., Pavone, M., & Nicosia, G. (2015). Packing equal disks in a unit square: an immunological optimization approach. In *2015 international workshop on artificial immune systems* (pp. 1–5). http://dx.doi.org/10.1109/AISW.2015.7469239.

Talbi, E.-G. (2009). *Metaheuristics: From design to implementation*. Hoboken: Wiley Publishing.

Talbi, E.-G. (2021a). Automated design of deep neural networks: A survey and unified taxonomy. *ACM Computing Surveys*, *54*(2), 1–37. http://dx.doi.org/10.1145/3439730.

Talbi, E.-G. (2021b). Machine learning into metaheuristics: A survey and taxonomy. *ACM Computing Surveys*, *54*(6), 1–32. http://dx.doi.org/10.1145/3459664.

Tani, L., & Veelken, C. (2024). Comparison of Bayesian and particle swarm algorithms for hyperparameter optimisation in machine learning applications in high energy physics. *Computer Physics Communications*, *294*, Article 108955. http://dx.doi.org/10.1016/j.cpc.2023.108955, URL: https://arxiv.org/pdf/2201.06809.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications* (pp. 1–6). IEEE, http://dx.doi.org/10.1109/CISDA.2009.5356528.

Timmis, J., Hone, A., Stibor, T., & Clark, E. (2008). Theoretical advances in artificial immune systems. *Theoretical Computer Science*, *403*(1), 11–32. http://dx.doi.org/10.1016/j.tcs.2008.02.011, URL: https://www.sciencedirect.com/science/article/pii/S0304397508001059.

Tsai, C.-W., Hsia, C.-H., Yang, S.-J., Liu, S.-J., & Fang, Z.-Y. (2020). Optimizing hyperparameters of deep learning in predicting bus passengers based on simulated annealing. *Applied Soft Computing*, *88*, Article 106068. http://dx.doi.org/10.1016/j.asoc.2020.106068.

Turkoglu, B., & Kaya, E. (2020). Training multi-layer perceptron with artificial algae algorithm. *Engineering Science and Technology, an International Journal*, *23*(6), 1342–1350. http://dx.doi.org/10.1016/j.jestch.2020.07.001.

Vitale, A., Di Stefano, A., Cutello, V., & Pavone, M. (2019). The influence of age assignments on the performance of immune algorithms. In A. Lotfi, H. Bouchachia, A. Gegov, C. Langensiepen, & M. McGinnity (Eds.), *Advances in computational intelligence systems* (pp. 16–28). Cham: Springer International Publishing.

Wang, S.-C. (2003). *Artificial neural network* (pp. 81–100). Boston, MA: Springer US, http://dx.doi.org/10.1007/978-1-4615-0377-4_5, chapter 5.

Wang, D., Tan, D., & Liu, L. (2017). Particle swarm optimization algorithm: an overview. *Soft Computing*, *22*(2), 387–408. http://dx.doi.org/10.1007/s00500-016-2474-6.

Waring, J., Lindvall, C., & Umeton, R. (2020). Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine*, *104*, Article 101822. http://dx.doi.org/10.1016/j.artmed.2020.101822.

Wilamowski, B. M., & Irwin, J. D. (2018). *Intelligent systems*. Boca Raton: CRC Press, http://dx.doi.org/10.1201/9781315218427.

Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015). Hyperparameter search space pruning - a new component for sequential model-based hyperparameter optimization. In *Machine learning and knowledge discovery in databases* (pp. 104–119). Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-23525-7_7.

Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., & Deng, S.-H. (2019). Hyperparameter optimization for machine learning models based on Bayesian optimizationb. *Journal of Electronic Science and Technology*, *17*(1), 26–40. http://dx.doi.org/10.11989/JEST.1674-862X.80904120.

Wu, Q., Wang, Y., & Glover, F. (2020). Advanced tabu search algorithms for bipartite Boolean quadratic programs guided by strategic oscillation and path relinking. *INFORMS Journal on Computing*, *32*(1), 74–89.

Yabi, C. P., Agongbe, S. W., Koto Tamou, B. C., Noroozinejad Farsangi, E., Alamou, E., & Gibigaye, M. (2024). Prediction of CBR by deep artificial neural networks with hyperparameter optimization by simulated annealing. *Indian Geotechnical Journal*, http://dx.doi.org/10.1007/s40098-024-00870-4.

Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, *415*, 295–316. http://dx.doi.org/10.1016/j.neucom.2020.07.061, URL: https://arxiv.org/pdf/2007.15745.

Yang, Z., Xu, B., Luo, W., & Chen, F. (2022). Autoencoder-based representation learning and its application in intelligent fault diagnosis: A review. *Measurement*, *189*, Article 110460. http://dx.doi.org/10.1016/j.measurement.2021.110460.

Yeh, I.-C. (2016). *Default of credit card clients*. UCI Machine Learning Repository, http://dx.doi.org/10.24432/C55S3H.

Zabinsky, Z. B. (2011). *Random search algorithms*. Hoboken: John Wiley & Sons, Ltd, http://dx.doi.org/10.1002/9780470400531.eorms0704.

Zambrano-Vega, C., Nebro, A. J., García-Nieto, J., & Aldana-Montes, J. F. (2017). Comparing multi-objective metaheuristics for solving a three-objective formulation of multiple sequence alignment. *Progress in Artificial Intelligence*, *6*(3), 195–210. http://dx.doi.org/10.1007/s13748-017-0116-6.

Zhang, Q., Hu, W., Liu, Z., & Tan, J. (2020). TBM performance prediction with Bayesian optimization and automated machine learning. *Tunnelling and Underground Space Technology*, *103*, Article 103493. http://dx.doi.org/10.1016/j.tust.2020.103493.

Zhong, G., Wang, L.-N., Ling, X., & Dong, J. (2016). An overview on data representation learning: From traditional feature learning to recent deep learning. *The Journal of Finance and Data Science*, *2*(4), 265–278. http://dx.doi.org/10.1016/j.jfds.2017.05.001.

Zito, F., Cutello, V., & Pavone, M. (2023a). Deep learning and metaheuristic for multivariate time-series forecasting. In P. García Bringas, H. Pérez García, F. J. Martínez de Pisón, F. Martínez Álvarez, A. Troncoso Lora, A. Herrero, J. L. Calvo Rolle, H. Quintián, & E. Corchado (Eds.), *Lecture notes in computer science*: *vol. 13838*, *18th international conference on soft computing models in industrial and environmental applications* (pp. 249–258). Cham: Springer Nature Switzerland, http://dx.doi.org/10.1007/978-3-031-42529-5_24.

Zito, F., Cutello, V., & Pavone, M. (2023b). A machine learning approach to simulate gene expression and infer gene regulatory networks. *Entropy*, *25*(8), http://dx.doi.org/10.3390/e25081214, URL: https://www.mdpi.com/1099-4300/25/8/1214.

Zito, F., Cutello, V., & Pavone, M. (2023c). A novel reverse engineering approach for gene regulatory networks. In H. Cherifi, R. N. Mantegna, L. M. Rocha, C. Cherifi, & S. Miccichè (Eds.), *Vol. 1077*, *Complex networks and their applications XI* (pp. 310–321). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-031-21127-0_26.

Zito, F., Cutello, V., & Pavone, M. (2023d). Optimizing multi-variable time series forecasting using metaheuristics. In L. Di Gaspero, P. Festa, A. Nakib, & M. Pavone (Eds.), *Metaheuristics* (pp. 103–117). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-031-26504-4_8.

Zito, F., Cutello, V., & Pavone, M. (2024). A general-purpose neural architecture search algorithm for building deep neural networks. In M. Sevaux, A.-L. Olteanu, E. G. Pardo, A. Sifaleras, & S. Makboul (Eds.), *Vol. 14754*, *Metaheuristics* (pp. 126–141). Cham: Springer Nature Switzerland, http://dx.doi.org/10.1007/978-3-031-62922-8_9.