Archives of Computational Methods in Engineering
https://doi.org/10.1007/s11831-025-10292-x

REVIEW

Optimizing Convolutional Neural Networks: A Comprehensive Review of Hyperparameter Tuning Through Metaheuristic Algorithms

Mohammed Q. Ibrahim[1] · Nazar K. Hussein[1] · David Guinovart[2] · Mohammed Qaraad[2]

Received: 7 March 2025 / Accepted: 2 May 2025

Abstract
Convolutional neural networks have become essential in computer vision, especially for image classification. They depend heavily on hyperparameters, and there is no practical way to manually tune these numerous settings through trial and error. This made it necessary for automated methods, especially those that come with metaheuristic algorithms, to optimize the hyperparameters and build good network architectures. Metaheuristic algorithms provide an easy way of determining the best hyperparameters by generating and testing various combinations using intuitive strategies and principles of solution-finding. This review provides a comprehensive discussion of convolutional neural networks, such as their layers, architectural designs, types, and ways of improvement, with a focus on optimization using metaheuristic algorithms. It highlights prominent algorithms and recent studies aimed at improving hyperparameter selection. By combining results of current and future research, this review should be a helpful resource for researchers, serving as the basis for further research and innovation in automated hyperparameter optimization using metaheuristic approaches, contributing significantly to further development in this field. The study concludes that metaheuristic algorithms significantly enhance the performance of convolutional neural networks with a simple yet effective replacement for manual tuning and high future prospects for automated optimization breakthroughs.

1 Introduction

Artificial intelligence (AI) has evolved over decades into a broad discipline that aims to develop systems with the capacity to perform tasks involving human-like cognitive abilities, such as learning, decision-making, and problem-solving [1]. Under AI, deep learning (DL), a machine learning (ML) variant, has gained popularity due to its ability to handle complex information through multi-layered neural networks, with convolutional neural networks (CNNs) serving as a cornerstone for computer vision tasks like image classification. This is achieved via a hierarchical framework that executes convolution operations and extracts features at various levels [2, 3]. The term "Convolution" was first used in 1989 by LeCun et al. [4] to describe a CNN, they were created with the intention of recognizing hand-written zip codes; that network became the basis of LeNet. A variety of shallow neural networks were proposed in the 1990s, such as generic regression [5] and Chaotic [6]. A number of key features are shared between ANNs and CNNs, such as the use of layers and weighted and biased electric neurons. In contrast to ANNs, CNNs process pictures and operate on the assumption that the input may be either a two-dimensional or three-dimensional picture. Convolutions, not inner products, are applied by CNNs' initial layers, thus the word "convolutional" to describe them. This is the opposite of ANNs [7–10].

However, the behavior of CNNs heavily depends on their hyperparameters—such as learning rate, network depth, and filter sizes—that control their architecture and training process, but are hard to adjust manually due to their huge combinatorial space [11, 12]. Recent research on hyperparameter optimization relies mostly on suboptimal manual

✉ Mohammed Qaraad
qaraa001@umn.edu

Mohammed Q. Ibrahim
mohammed.q.ibrahim35509@st.tu.edu.iq

Nazar K. Hussein
nazar.dikhil@tu.edu.iq

David Guinovart
guino001@umn.edu

1 Department of Mathematics, College of Computer Sciences and Mathematics, Tikrit University, Tikrit 34001, Iraq

2 The Hormel Institute, University of Minnesota, 801 16th Ave NE, Austin, MN 55912, USA

trial-and-error or computationally expensive methods like grid search, which do not scale with the complexity of modern CNNs, while automated techniques like Bayesian optimization and random search lack the flexibility needed for diverse problem domains [13]. In addition, gradient-based optimization fails with non-differentiable hyperparameter spaces, and existing studies have not thoroughly investigated metaheuristic algorithms, which provide transparent, population-based methods for global optimization [14, 15]. These shortfalls call for a structured, efficient solution for CNN hyperparameter tuning that can balance computational cost against performance benefit [16, 17]. This review addresses these gaps through three key contributions.

- First, it provides a detailed synthesis of CNN fundamentals, including their architecture, types, and improvement techniques, offering a foundational resource for understanding their optimization needs.
- Second, it comprehensively evaluates the application of metaheuristic algorithms—such as Particle Swarm Optimization, Genetic Algorithms, and Sine Cosine Algorithm—for tuning CNN hyperparameters, addressing the inefficiency of traditional methods by highlighting adaptable, automated solutions.
- Third, it bridges historical and contemporary research by analyzing recent studies, providing a consolidated reference that overcomes the fragmented exploration of metaheuristic approaches in prior work and sets the stage for future innovations in CNN optimization.

The paper is organized as follows: Sect. 2 illustrates the related work. Section 3 provides the background that includes an overview of artificial intelligence (AI) and related subfields, including ML, DN, and COV, as well as their interconnections. Section 4 explains convolutional neural networks (CNNs), their origin, structure, variations, platforms that facilitate their construction, and the methodologies used to enhance their performance. Section 5 defines CNN hyperparameters and optimization problems. Section 6 explains metaheuristic algorithms and how they have been applied to CNN hyperparameter optimization through global research examples. Section 7 provides Convolutional Neural Network Architectures Optimized via Metaheuristic Algorithms. Finally, Sect 8 offers conclusions and recommendations for future research.

2 Related Work

It is important to optimize convolutional neural networks (CNNs) to enhance their performance, stability, and efficiency. Hyperparameter optimization (HPO) is an important part of this process, as it determines the optimal parameter settings for a CNN to provide high accuracy in classification problems [18, 19]. This process of selection is time-consuming and complicated as it entails the tuning of a large number of parameters that consume considerable computational resources. Historically, hyperparameters are chosen by heuristic approaches and manually tuned with a single configuration, potentially running anywhere from several hours to even days.

To address this issue, a number of various HPO algorithms have been developed for parameter selection automation [20–23]. By employing some minor adjustments to hyperparameter levels, these methods can enhance model performance and significantly reduce training time. The optimization process is typically iterative, with continuous updates to parameter values until the optimal parameters are reached. Although HPO is important, the majority of survey studies [24–28] lack a comprehensive review of the different HPO algorithms, thus the necessity for detailed discussions about their variants. Figure 1 displays the core operational flow of HPO algorithms.

Hyperparameter optimization in CNNs has been explored in recent studies across many different applications, including healthcare [29], education [30], business [31], stock prediction [32], and agriculture [33], among others [34, 35].

In these works, hyperparameter tuning has led to drastic performance gains in model performance. In addition, HPO algorithms have been employed to optimize architectures for prediction tasks, hyperparameter tuning of models such as long short-term memory (LSTM), CNNs, and multilayer perceptrons (MLPs) [36]. A number of studies [37, 38] also highlight the importance of hyperparameter tuning in the improvement of the performance of deep learning models. Many HPO techniques have been used in different applications. Gradient-based optimization has been used for COVID-19 and colon cancer diagnosis [39], and bio-inspired algorithms were used for breast cancer detection [40]. The fuzzy tree model was employed for image classification, while evolutionary computation methodologies [41] were employed for network intrusion detection. All of these emphasize how the optimization of the hyperparameters is crucial for improving the accuracy and performance of CNN-based applications. Heuristic, meta-heuristic, and statistical-based HPO strategies for CNNs are all examined in this paper.

Mehrdad Kaveh and Mohammad Saadi Mesgari [42] cover the use of meta-heuristic (MH) algorithms for optimizing artificial neural networks (ANNs) and deep learning (DL) models to avoid local minima and computational expenses of gradient-based approaches. They highlight the ability of MH algorithms for hyperparameter adjustment and improving training in all applications with the advent of hybrid MH solutions. The study classifies newer MH methods, evaluates their performance, and recognizes their sparse

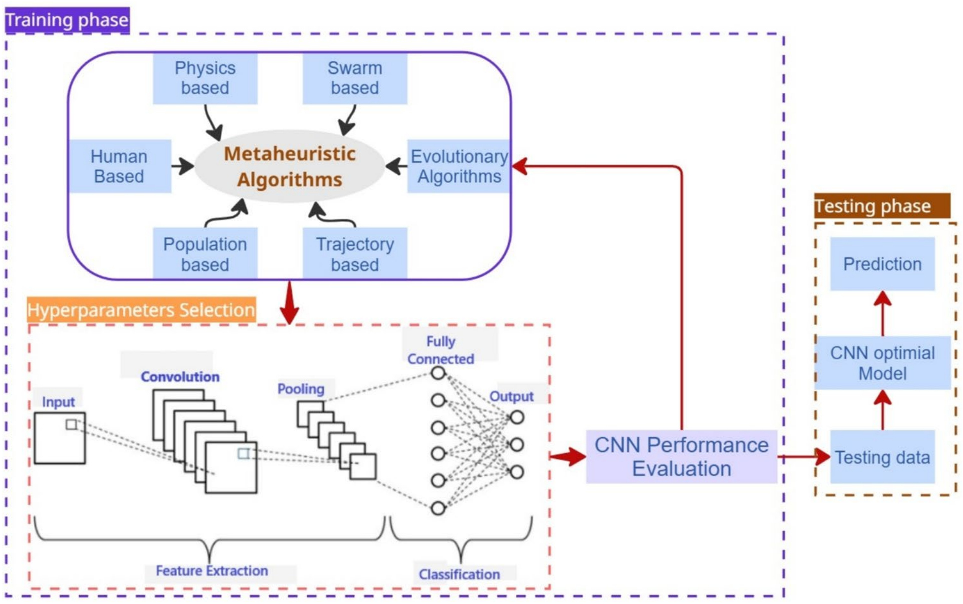


Fig. 1 Workflow of HPO algorithms applied to a CNN-based system

exploration of evolutionary hybrid structures. They suggest MH-DL integration can enhance training, although there is limited research in this area.

Mohaimenul et al. [43] carry out a systematic review of HPO in CNNs with emphasis on the time-consuming process of manual tuning. They categorize the HPO algorithms as metaheuristic, statistical, sequential, and numerical types and evaluate their performance during tuning CNN hyperparameters. Through widespread literature and comparison, they compare these methods based on strategy, error analysis, and accuracy for various datasets. The paper highlights ongoing problems, unresolved issues, and research directions for the future, aimed at assisting researchers in selecting the right HPO approaches for specific problems. Veeranjaneyulu et al. [44] explains how swarm intelligence-based metaheuristic optimization algorithms are used to optimize artificial neural network (ANN) models for the diagnosis of breast cancer, a leading cause of death in women globally. The article compares various ANN-based approaches, explaining how the optimization of selected features and hyperparameter tuning improves classification accuracy while reducing computational time. It goes on to talk about wrapper and filter methods to distinguish cancer cells from normal cells, highlighting the advantages and disadvantages of swarm intelligence-optimized ANN models. Directions of future research are also recommended, and the importance of evaluation measures in ascertaining the effectiveness of such diagnostic methodologies is emphasized.

Benaissa et al. [45] discuss the adaptability of metaheuristic optimization algorithms to the extent that their ability to address challenging optimization problems without gradient information or specific problem structures is of interest. They balance exploration and exploitation, enable global search and multi-objective optimisation, but lack guarantees of finding the global optimum and varying convergence rates. They are contrasted with metaphor-based algorithms in the paper, with the latter characterised as intuitively appealing but a troublesome simplification. Brahim's article explores the core concepts of such algorithms, addressing search mechanisms and providing graphical intuition into their operation, emphasizing their continued relevance despite limitations.

Taji et al. [46] explain the use of technology in agriculture to improve yield and quality through early plant leaf disease diagnosis, which is a vital aspect of affecting the crop output as well as farmers' wellbeing. Machine learning, and indeed deep learning, is investigated to automatically identify and classify apple and maize plant diseases and recommend an ensemble deep learning approach. This hybrid technique combines pre-trained convolutional neural network (CNN) features with local binary pattern (LBP) features, which are metaheuristic algorithm-optimized (Binary Dragonfly (BDA), Ant Colony Optimization, and Moth Flame Optimization (MFO)) features. Sophisticated machine learning methods are used to classify the optimized features, closing the research gaps in plant disease analysis for fruits and vegetables. Latif et al. [47] evaluate the enhanced use of artificial intelligence (AI) and machine learning (ML) in medical diagnosis, noting their ability to transform medical imaging diagnosis (MID) through auto tools that reduce costs, human errors, and increase accessibility. The article provides a comprehensive review of deep convolutional neural network (CNN) optimization techniques employed in MID, focusing on improving image classification accuracy. Using the COVID-19 radiological images example, Latif addresses the impact of the datasets and AI/ML approaches, as well as common challenges and barriers. Based on the review, it holds that AI/ML will remain important in MID, and further efforts in classification accuracy are to be expected.

Together, these works validate HPO's essential contribution to the development of CNN applications, but there are still gaps in investigating hybrid algorithms, standardizing testing, and scaling solutions across domains. This work draws on these foundations, seeking to overcome such challenges and contribute to the developing body of CNN optimization.

3 Background

Artificial neural networks and deep learning are the foundational pillars of modern machine learning, driving advancements in applications such as computer vision and image classification. In what follows, this explanation of the interdependent relationship between them offers a framework for building a platform on which convolutional neural networks can be optimized, at the heart of this debate. Beginning with the fundamental workings of artificial neural networks, their architecture and mode of learning, we proceed to deep learning as a higher-level form, dealing with the architecture and capability of convolutional neural networks. All these topics are combined to provide the building blocks from which one might tackle the problems and techniques of hyperparameter optimization, particularly through metaheuristic approaches.

3.1 Deep Learning

Artificial Intelligence (AI) is the broad collection of techniques for getting machines to do things related to cognition. One of the most popular subfields of AI is machine learning (ML), in which algorithms are trained on large data to learn patterns and make predictions. One particularly powerful version of ML is deep learning (DL), in which multi-layer neural networks are employed to generate complex representations of data. DL has been demonstrated with unprecedented success in a wide range of applications, from computer vision, speech recognition, natural language processing, to autonomous systems [1]. Unlike traditional ML models, deep learning models—more precisely deep neural networks (DNNs)—employ hierarchical feature extraction through multiple layers, which enables them to discover intricate patterns from raw data. However, these models come with challenges, e.g., higher computational demands, the risk of overfitting due to their huge parameter space, and the need for specialized hardware (e.g., GPUs or TPUs) for their rapid training [48]. Some of the most well-known DL architectures are convolutional neural networks (CNNs), which excel at processing grid-like data, e.g., images. CNNs employ convolutional layers to automatically discover spatial hierarchies of features—from edges and textures in early layers to more intricate objects in deeper layers—making them extremely successful at tasks like image classification, object detection, and medical image analysis [49]. They are successful at visual representation learning by maintaining spatial relationships and increasingly abstracting features through pooling and nonlinear activation functions [50]. Besides CNNs, other DL models such as RNNs and transformers have revolutionized sequential data processing (e.g., language modeling and time-series forecasting) by their capacity to model long-range dependencies and contextual information [51]. The rapid development of DL has been fueled by the existence of large-scale datasets (e.g., ImageNet, COCO), optimized libraries (e.g., TensorFlow, PyTorch), and improved training techniques (e.g., batch normalization, dropout) [52]. (Fig. 2 illustrates the hierarchical relationship between CNNs, deep learning, machine learning, and artificial intelligence.).

3.2 Artificial Neural Network (ANN)

Artificial Neural Networks (ANNs) are computational models of the biological neural networks within the human brain, coded to mimic intelligent decision-making. The human brain consists of approximately 86 billion neurons interlinked through synapses, with massively parallel processing of information using immense efficiency, beyond standard computers at operations like pattern recognition and adaptive learning [53]. ANNs replicate this structure with artificial neurons (or "nodes") that share information through weighted connections, which are the foundation for modern-day deep learning architectures. A biological neuron (Fig. 3) contains three main components:

– Dendrites—Receive signals from other neurons.

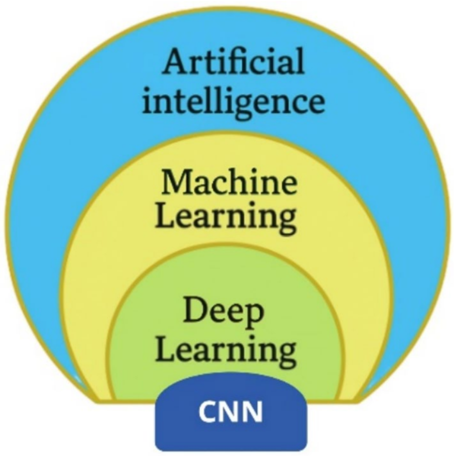


Fig. 2 Connections among (CNNs), (ML), (DL), and (AI)

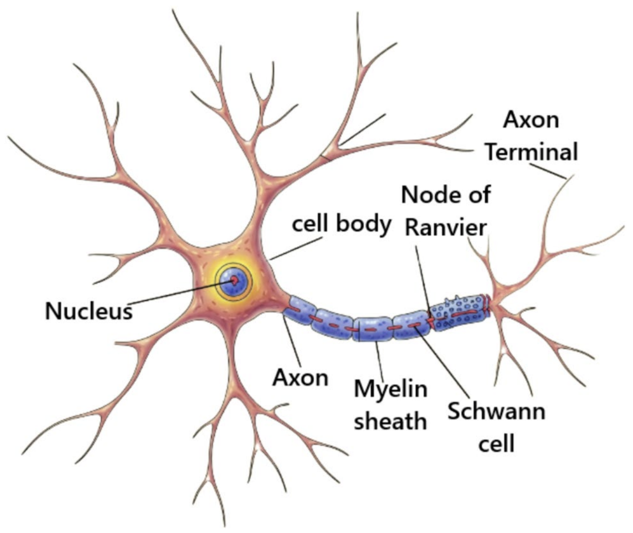


Fig. 3 The neural network's biological design

- Soma (Cell Body)—Processes inputs and generates an output signal.
- Axon—Transmits the output signal to downstream neurons.

An artificial neuron (Fig. 4) replicates this analogy:

- Inputs ($\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$)—Signals or features from previous neurons.
- Weights ($\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n$)—Scaling values that multiply the impact of each input, the network's "memory."
- Activation Function (φ)—Introduces nonlinearity, determining whether the neuron "fires" (e.g., ReLU, Sigmoid).
- Output (y)—Forward to subsequent layers or produces a final prediction.

The interconnect strengths (weights) are adapted iteratively during training by backpropagation, with error optimized using techniques such as gradient descent [54]. Adaptation makes ANNs competent in learning complex mappings of data, i.e., from image classification or forecasting time-series trends.

As shown in Fig. 4, ($\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$) represent input signals. ($\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n$) are the associated signal weights, and b stands for the bias, which may be seen as a numerical value indicating the threshold at which the artificial node or neuron generates a meaningful output [11–17, 55]. Before the input signal reaches the node, it is multiplied by the weight. The weighted total, which is the result of adding all of the weighted signals at a given node, is then fed through the nonlinear activation function together with the score and bias. Equations may therefore be used to represent it.

$$k = \left(x_1 \times w_1\right) + \cdots + \left(x_1 \times w_1\right) + b \tag{1}$$

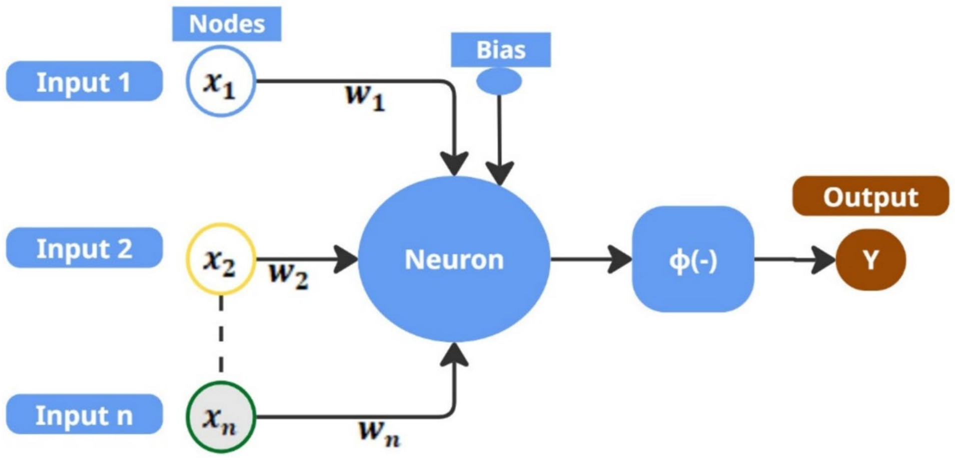
$$k = \sum_{i=1}^{n} \left(x_i \times w_i\right) + b \tag{2}$$

$$y = \varphi(k) \tag{3}$$

where x_1 is the input signal, w_1 represents weight corresponding to the ith input, b is the bias term, φ denotes the activation function (e.g., ReLU, sigmoid), and y represents the neuron's output.

Although the basic structure of an ANN is a single-layer perceptron, more complex architectures have been designed so that these models can resolve different computational issues. FNNs are the most basic structure in which information only goes in one direction from input to output layers, and so are good for regression and classification tasks. Recurrent neural networks (RNNs) incorporate cyclic connections for sequential input, such as speech and

Fig. 4 The diagram of a standard artificial neuron



text, enabling them to learn temporal correlations. This is crucial for tasks such as time-series prediction and language modeling [56]. Alternatively, by maintaining spatial connections and increasingly extracting higher-level features, Convolutional Neural Networks (CNNs) utilize hierarchical convolutional layers to process grid-structured input, such as images, effectively [57]. These design breakthroughs highlight the versatility of ANNs to different data modalities, with each version being tuned to specific problem domains. Table 1 compares the structural components of biological and artificial neurons, while Fig. 4 illustrates a standard artificial neuron's design.

3.3 Activation Functions in Neural Networks

Because they introduce non-linearity and allow the model to learn complicated patterns in data, activation functions are an elementary building block in artificial neural networks (ANN). Without activation functions, a network would be a mere linear regression network incapable of complicated tasks like language processing or face recognition. How well the network performs, is stable during training, and is able to converge depends heavily upon the activation function chosen.

The most popular activation function is the Rectified Linear Unit (ReLU) [58]. ReLU is used very extensively in hidden layers since it is computationally inexpensive and does not suffer from the vanishing gradient problem in deep networks. It suffers from the "dying ReLU" problem, though, where it occurs when always negative inputs are provided, and neurons are killed. On the other hand, learnable small slopes for the negative inputs are incorporated into other variants, including Leaky ReLU [59] and Parametric ReLU (PReLU) [60] that promote gradient flow and model stability. For binary classification tasks, the Sigmoid function [61] is generally applied in the output layer since it squashes values to 0 and 1, representing probabilities. However, its saturation regions lead to vanishing gradients in backpropagation. Even Hyperbolic Tangent (Tanh) [62] scales inputs to the range [− 1,1], which are zero-centered and may be employed to accelerate convergence in certain architectures.

All the activation functions have their own set of strengths and weaknesses, summarized in Table 2, listing their mathematical representation, output values, and actual-world trade-offs. The selection of an appropriate activation function depends on the problem, architecture, and efficiency.

Table 1 Neural artificial networks and networks of biological neurons are equivalent

Network of biological neurons	Neural artificial
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

4 CNNs (Convolutional Neural Networks)

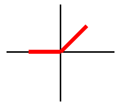
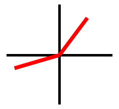
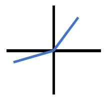
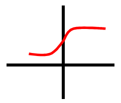
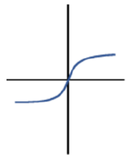
Over the recent decades, CNNs have become a cornerstone of deep learning, breaking through domains previously deemed intractable. Applications include intelligent medical diagnosis, autonomous vehicles, face recognition, and cashierless store checking systems, revolutionizing sectors by capitalizing on high-level computer vision capabilities [4, 63]. CNNs are a particular type of artificial neural network (ANN), whose beginnings trace back to the McCulloch-Pitts (MP) neuron model of 1943 [64]. Later milestones are Rosenblatt's perceptron during the 1950s–60s [54] and backpropagation-based multi-layer network by Hinton et al. in 1986 [65], shattering single-layer limitations. Revolutionary CNN variants started during the 1980s–90s, such as Waibel's Time-Delay Neural Network (TDNN) for speech recognition [66] and LeCun's LeNet-5 for handwritten digit classification [4, 49], introducing convolutional operations as a paradigm shift.

While promising, initial deep networks were marred by problems like vanishing gradients and overfitting. A turning point arrived in 2006 with Hinton's layer-wise pre-training technique [67], re-sparking enthusiasm for deep learning. The field picked up speed after AlexNet's 2012 ImageNet victory [68], establishing CNNs as the default for visual tasks. Follow-up work used CNNs to detect objects, segment images, and more, even if a lot of the experimentation ignored more general theoretical advances made possible by CNN research.

CNNs and ANNs have several fundamental similarities, including the use of layers and artificial neurons with weights and biases. The primary difference between CNNs and ANNs is that CNNs function with images, implicitly assuming the image appears in either two or three dimensions as the input. Because of this, the term "convolutional" refers to the fact that the initial layers of CNN use convolutions rather than inner products, as ANNs do. Figure 5 depicts the construction of a CNN, which shares a similar set of layers with ANN, except for the addition of convolutional, pooling, and fully connected layers. The CNN framework comprises three core layers:

a. **Convolutional Layer:** The fundamental component of CNN is the convolutional layer. It uses a collection of learnable filters to apply a convolutional operation to the input image. Sliding a small array over the input picture, each step calculates the dot product between

Table 2 A brief overview of the many activation functions

Activation function	Characteristics	The formula	Range	Power	Weakness	Shape
The Rectified Linear Unit (ReLU)	Provides a linear response to positive inputs and zero to negative ones	$= \begin{cases} x, ifx \geq 0 \\ 0, ifx < 0 \end{cases}$	$[0, +\infty)$	efficient computing, deep learning, hidden layers, gradient avoidance	Unbounded from above, asymmetrical gradients, Dying ReLU for negative inputs	
Parametric ReLU (P ReLU)	Negative inputs always produce zero output, P. ReLU teaches the network the negative slope. Non-linear: Learning the parameter α increases model flexibility	$= \begin{cases} x, ifx > 0 \\ 0, ifx \leq 0 \end{cases}$	$(-\infty, +\infty)$	Better performance, slope-adaptable	Increased complexity and potential overfitting can lead to increased training time	
Leaky ReLU	Non-zero Negative Input Output: This is in contrast to the normal ReLU, which outputs zero for negative inputs	$= \begin{cases} x_i, ifx_i \geq 0 \\ \frac{x_i}{a_i}, x_i < 0 \end{cases}$	$(-\infty, +\infty)$	A powerful alternative to ReLU improves learning and balances negativity and positivity	Over-adaptation, costly computation	
Sigmoid	The output ranges from 0 to 1, which helps with binary classification difficulties. The functional curve is S-shaped	$= \frac{1}{1+e^{-z}}$	(0,1)	Smooth Curve, Easy Derivative	Vanishing Gradients Problem, Non-Zero-Centered, Limited Output Range	
Hyperbolic Tangent (Tanh)	outputs that span from -1 to 1, rendering it zero-centered	$= \frac{e^x - e^{-x}}{e^x + e^{-x}}$	(-1,1)	Zero-centered output with a smooth curve the output represents both positive and negative values	Vanishing gradients problem, higher computational cost	

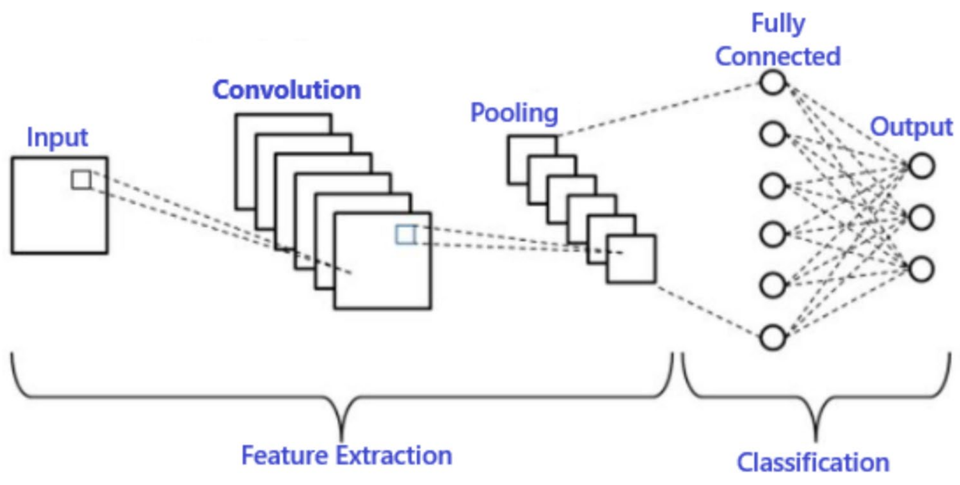


Fig. 5 CNN's fundamental architecture

the filter and a little piece of the input. A convolutional layer produces a set of feature maps, where each map represents a different feature in the input picture [7]. The convolutional layer produces a set of feature maps by applying a convolutional operation and a set of learnable filters to the input picture.

b. **Pooling layer:** The convolutional layer's feature maps' spatial scale is reduced by the pooling layer [8, 9]. By averaging or maximizing the values of the non-overlapping areas, each feature map is down-sampled independently. Without compromising data quality, pooling helps reduce the network's computational complexity. Therefore, the abundance of parameters in the network will amplify its sensitivity to even small changes in the input images. Activation. The pooling layer generates feature maps that the convolutional layer uses to shrink the feature set by averaging or maximizing the values in the non-overlapping areas. A full comparison of different pooling layer topologies in CNNs is listed in Table 3, together with information on their unique features, usage in the real world, and performance trade-offs.

c. **Fully Connected Layer:** Usually, the final layer in a CNN's architecture is a classifier that determines the input image's class using characteristics that the CNN has found and retrieved from the pooling and convolutional layers. In a fully connected layer, the number of neurons is directly proportional to the number of classes or categories. A single one-dimensional vector is used to flatten feature maps before they are loaded into fully linked layers [10].

4.1 Traditional CNN Models

Over the years, researchers have invented and developed many diverse, deeper, wider, and lighter CNN models and architectures, each with its own unique features and characteristics. The following is a rundown of a few of the designs that this part intends to present:

a. **LeNet-5:** A robust convolutional neural network trained using the backpropagation approach, LeNet-5, was created with the aim of recognizing handwritten characters

Table 3 Comparative analysis of pooling layer architectures in CNNs

Method	Operational Principle	Mathematical Formulation	Optimal Applications	Performance Limitations
Max Pooling [69]	Extracts dominant features through regional maxima selection	$= down\left(\max\left(y_{i,j}\right)\right), i,j \in p$	- Position-independent feature detection [70, 71] - Edge and texture preservation in visual data	- Degrades precise spatial relationships [72] - May discard relevant sub-maximal features
Mean Pooling [73]	Computes the arithmetic mean across receptive fields	$= \tanh(\beta \sum_{N\times M} M_i^{n\times n} + b)$	- Noise reduction in medical imaging [74, 75] - Regularization for overfit-prone models	- Feature dilution in high-variance regions [76] - Blurs sharp discriminative features
Rank-Weighted Pooling (RWP) [77]	Performs order-statistic weighted averaging	$= \frac{1}{t} \sum_{i\epsilon R_j, r_i<t} a_i$	- Outlier-resistant processing in sensor data [78] - Robust feature extraction from noisy inputs	- $O(n\log n)$ sorting overhead [79] - Memory-intensive for high-resolution inputs

in 1998 by LeCun et al. [5]. With three fully connected layers, two pooling layers, and two convolutional layers, LeNet-5 is a seven-layer trainable network. By combining shared weights with adjacent receptive zones and using spatial or temporal subsampling, LeNet-5—one of the first CNNs—could provide a level of invariance against scale, shift, and distortion. Modern CNN may be able to trace its origins there. Despite LeNet-5's use for document reading and handwriting recognition, traditional techniques of boosting and support vector machines (SVM) still surpass it. That is why LeNet-5 was not given enough credit when it was first released. Using small filters, the Conv_Lays and Pool-Lays obtain features from the supplied image. A probability distribution for all potential classes is generated by the completely linked layers using a softmax activation from [80, 81].

b. **Alex Net:** The ImageNet 2012 title was won by AlexNet, which was proposed by Alex et al. [82]in 2012. AlexNet has eight layers total, three of which are fully connected and five of which are convolutional. Extending LeNet's work by applying CNN's essential notions, AlexNet uses a deep and wide network architecture with bigger filters than LeNet. Thanks to its innovative use of the ReLU activation function in conjunction with dropout and LRN, it revolutionizes CNN. To speed up calculations, AlexNet takes advantage of GPUs all at once.

c. **VGG:** In 2014, Karen Simonyan and Andrew Zisserman created the VGG CNN architecture, which on the ImageNet dataset produced recent results [80, 83]. In the VGG architecture, the max Pooling layer comes after many layers of tiny 3×3 convolutional filters. ReLU activation and dropout regularization are used by the fully linked layers to avoid overfitting. [81] The creators of VGGNets demonstrate how adding depth to neural networks might somewhat enhance the network's overall performance. The following are VGGNet's advantages over AlexNet:

   (1) The creator of VGGNets eliminated the LRN layer after discovering that the impact of LRN in DCNNs was not readily apparent.
   (2) VGGNets employ three-by-three convolution kernels instead of five-by-five or five-by-five ones since several tiny kernels have more non-linear variations and the same receptive field as bigger ones. For example, one five-by-five kernel and three-by-three kernels have the same receptive field. However, there exists a 45% reduction in the number of parameters, and there are three nonlinear variations for each of the three kernels.

d. **ResNet:** As DNNs are capable of extracting more intricate and adequate characteristics from pictures, they should theoretically perform better than shallow ones. Yet, DNNs have a tendency to make gradients vanish as the number of layers rises. Issues blowing out, etc. The winning method for the ILSVRC 2015 image classification and object recognition challenge was a 34-layer residual network, as suggested by Kaiming He et al. [84]. When compared to Google LeNet Inception v3, ResNet performs better. Among ResNet's one of the most important innovations is the 50-layer shortcut connection-based 2-layer residual block, one hundred and one layers, and one hundred fifty-two-layer ResNet, which use 3-layer residual blocks rather than two-layer ones. Owing to its two smaller extremities than its center, a three-layer residual block is sometimes referred to as the bottleneck module. Not only may the network's nonlinearity be significantly increased by using a 1×1 convolution kernel, but it can also lower its parameter count. ResNet may ameliorate the gradient vanishing issue in deep neural networks without degenerating, as shown by several experiments in [38], because gradients may ideally flow directly via shortcut links. The performance of the original ResNet has been enhanced by several research studies, including broad ResNet [85], Stochastic Depth ResNets (SDR) [86], pre-activation ResNet [87], and ResNet in ResNet (RiR) [88].

e. **InceptionNet (GoogLeNet):** Christian Szegedy et al. (2014) created InceptionNet, or GoogLeNet, a CNN architecture that makes optimal use of computing resources by introducing the idea of inception modules [45]. Here are four variants of the Inception network: (Inception v1 [89]), (Inception v2 [90]), (Inception v3 [91]), and (Inception v4 [92]). The output of an Inception module is formed by concatenating many convolutional filters of varying sizes plus pooling layers. Complementary classifiers aid in training and network regularization as part of the InceptionNet design.

A full comparison of Classical CNNs Architectures is listed in Table 4, together with information on their unique features, usage in the real world, and performance trade-offs.

4.2 Resources for Constructing Convolutional Neural Networks (CNNs)

The construction of CNNs and other deep learning (DL) architectures is facilitated by a wide array of platforms and libraries. These resources support model development, training, evaluation, and deployment [93, 94]. Below is a concise overview of several widely adopted platforms in the machine learning community:

Table 4 Comparative analysis of classical CNN architectures

Model	Key innovation	Architecture	Strengths	Limitations	Impact & legacy
LeNet-5	Pioneered the convolutional-pooling alternation; Introduced weight sharing	7 layers (2 conv, 2 pool, 3 FC); Small 5×5 filters	Translation invariance; Efficient for digit recognition	Not as scalable as modern deep learning models; Limited depth	Foundation for modern CNNs; inspired spatial feature extraction
AlexNet	ReLU activation; GPU parallelization; Dropout & LRN	8 layers (5 conv, 3 FC); 11×11 and 5×5 filters	Won ImageNet 2012; Scalable to complex tasks	High parameter count; Computationally intensive	Popularized deep CNNs; established ReLU as standard
VGG	Depth emphasis; Stacked 3×3 convs (vs. large filters)	16–19 layers; Uniform 3×3 convs + max-pooling	45% fewer params than AlexNet; Better feature hierarchy	Memory-heavy due to FC layers	Demonstrated depth's importance; widely used for transfer learning
ResNet	Residual blocks; Identity shortcuts	34–152 layers; Bottleneck modules (1×1 convs)	Solves vanishing gradients; Won ILSVRC 2015	Complex design for very deep variants	Enabled 100+layer networks; inspired ResNet variants (e.g., WideResNet, SDR)
InceptionNet	Inception modules (multi-scale filters); Auxiliary classifiers	v1–v4 variants; Parallel conv/pool branches	Computational efficiency; High accuracy with fewer params	Design complexity; Training instability in early versions	Introduced multi-scale processing; influenced efficient architecture design

a. **Tensor-Flow:** Developed by Google, TensorFlow is a widely used open-source deep learning library that supports the construction and deployment of CNNs through an extensive ecosystem of tools and APIs [95]:

- TensorFlow Core: The foundational module that enables efficient definition and execution of computational graphs, with a comprehensive library of operations for building CNNs and other neural networks. It supports full control over model architecture, layers, and training processes.

  – The TensorFlow Hub is A repository of reusable machine learning modules, including pre-trained CNN models such as Inception, ResNet, and MobileNet. These modules can be integrated easily into custom architectures.
  – TensorFlow datasets: A collection of preprocessed datasets frequently used in computer vision, such as MNIST, CIFAR-10, and ImageNet. It simplifies the process of importing and preprocessing data for training and evaluation.
  – TensorFlow Estimators: A high-level API designed to streamline the development, training, and deployment of machine learning models, including CNNs. It abstracts much of the low-level implementation to facilitate rapid experimentation.
  – TensorFlow Model Optimization Toolkit: A suite of tools for model compression and optimization through pruning, quantization, and weight sharing—crucial for deploying CNNs on resource-constrained devices.
  – TensorFlow Lite: A lightweight version of TensorFlow tailored for mobile and embedded devices, enabling efficient inference of CNN models in environments with limited computational capacity.

b. **Keras:** Keras is a high-level API that runs on top of TensorFlow and offers a user-friendly interface for constructing CNNs and other neural networks. It is ideal for beginners and for rapid prototyping. Keras includes support for model evaluation, visualization, and transfer learning using pre-trained models such as VGG16, Inception, and ResNet [89–94, 96, 97].

c. **Py-Torch:** Developed by Facebook, PyTorch is an open-source machine learning framework known for its dynamic computation graph, which allows real-time modifications to the network architecture. It supports both high-level (e.g., TorchVision) and low-level APIs, enabling flexibility for CNN construction and training. PyTorch includes tools for visualization, distributed training, and model deployment [98, 99].

d. **MXNet:** An open-source DL framework maintained by Apache, MXNet supports both imperative and symbolic programming via its Gluon API. It provides multi-language support (e.g., Python, Scala, R), tools for distributed training, and access to pre-trained models such as ResNet, SqueezeNet, and SSD. Its scalability makes it suitable for training on large datasets [100, 101].

e. **Caffe:** Created by the Berkeley AI Research (BAIR) group, Caffe is a deep learning framework optimized for speed and modularity. It includes a suite of pre-trained CNN models, including AlexNet, GoogLeNet, and YOLO, and offers tools for visualization and model deployment. Its efficiency is particularly beneficial for large-scale vision tasks [102–104].

f. Theano: Although no longer under active development, Theano was foundational in the evolution of deep learning frameworks and played a key role in CNN research. It enabled symbolic expression and efficient evaluation of mathematical computations, laying the groundwork for libraries like Keras [105, 106].

A full comparison of popular deep learning frameworks for CNN construction is listed in Table 5, highlighting core features such as developer, programming support, computation graph type, API level, key strengths, and pre-trained model support.

4.3 Optimization Approaches for CNNs

Optimizing Convolutional Neural Networks (CNNs) involves a multitude of strategies designed to improve model generalization, convergence, and computational efficiency. These methods are problem-specific and resource-sensitive; thus, determining the optimal combination often requires iterative tuning and experimentation. Key categories of optimization include architectural design, regularization, data augmentation, optimization algorithms, and hyperparameter tuning.

a. **Designing network architecture:** CNN performance is highly dependent on its architectural design. Over the years, researchers have developed various architectures, including LeNet-5 [83], Alex Net [82], VGGNet [80, 83], and GoogleNET (Inception) [89–92]. These networks incorporate innovations such as deeper layers, skip connections, and specialized blocks (e.g., normalization, dropout, and pooling), allowing improved feature extraction and hierarchical learning [7–10].

b. **Regularization methods:** Regularization is essential to prevent overfitting, especially when models contain significantly more parameters than training examples. Widely used techniques include L1 (Lasso) and L2 (Ridge or Tikhonov) regularization. L1 encourages sparsity by penalizing the absolute magnitude of weights,

Table 5 Comparative analysis of popular deep learning frameworks for CNN construction

Framework & developer	Language support	Computation graph	API Level	Key strengths	Pre-trained model support
TensorFlow Google	Python, C++, Java, Swift	Static (Eager optional)	High & Low	Scalable and production-ready; strong deployment tools (TF Lite, TF Serving); large ecosystem	Yes (TF Hub)
Keras Google (built on TensorFlow)	Python	TensorFlow backend	High	Simple API; fast prototyping; excellent for newcomers and education	Yes (Keras Applications)
PyTorch Facebook	Python, C++	Dynamic	High & Low	Intuitive interface; real-time debugging; popular in research and experimentation	Yes (TorchVision)
MXNet Apache Foundation	Python, Scala, R, Julia	Hybrid (Imperative + Symbolic)	High & Low	Scalable; multi-language support; efficient on large-scale systems	Yes
Caffe Berkeley AI Research	C++, Python	Static	Low	High-speed model execution; optimized for vision tasks	Yes (Model Zoo)
Theano MILA (University of Montreal)	Python	Static	Low	Efficient symbolic computation; pioneered GPU acceleration for DL	Limited (legacy models)

often aiding in feature selection. L2 regularization, also known as weight decay, discourages large weights by penalizing their squared magnitude [107, 108]. Regularization modifies the loss function as:

$$\theta = \operatorname{arg} min \frac{1}{N} \sum_{i=1}^{N} (L(\hat{y}_i, y) + \lambda R(w)) \tag{4}$$

where:

$$R_{L2}(w) = \|w\|_2^2 \tag{5}$$

$$R_{L1}(w) = \|w\|_1 \tag{6}$$

To address non-differentiability in L1, smoothed approximations such as:

$$|w|_1 \approx \sum_{k=1}^{D} \sqrt{w_k^2 + \in} \tag{7}$$

are used in practice, especially in first-order optimization routines.

c. **Augmentation of data:** Data augmentation helps improve model generalization by synthetically increasing the dataset size using transformations such as rotation, cropping, flipping, and color jittering. Techniques like Principal Component Analysis (PCA)-based color augmentation, as introduced by Krizhevsky et al. [49], have shown success in ImageNet classification. Empirical evidence by Bengio et al. [109] further supports its effectiveness in deep architectures. Zhang et al. [84] combined explicit regularizes with augmentation to enhance generalization. Domain-specific studies (e.g., leaf classification) have also demonstrated its benefit [110].

d. **Optimization algorithms:** Efficient optimization algorithms are vital for training CNNs. Adaptive optimizers like Adam [111], RMSprop [112], and traditional methods like Stochastic Gradient Descent [113] are commonly used. These algorithms vary in convergence rate, sensitivity to hyperparameters, and ability to escape local minima. For highly complex, non-convex loss landscapes, metaheuristic approaches—including Particle Swarm Optimization (PSO), Grey Wolf Optimizer (GWO), and Biogeography-Based Optimization (BBO) variants—have superior capabilities to escape local optima through biologically inspired search processes, as demonstrated in recent medical informatics and general classification problems [Li et al. [114]; Yang [115]]. The emerging trend of hybrid optimization (e.g., the integration of Adam's adaptive property with GWO's exploratory capability) is particularly promising for addressing CNN-specific problems like vanishing gradients without compromising computational efficiency.

e. **Hyperparameter Tuning:** Hyperparameters such as learning rate, batch size, dropout rate, and filter size significantly influence model performance. Tuning strategies include grid search, random search, and automated methods like Bayesian optimization [116, 117]. Optimal tuning enhances both convergence speed and final model accuracy.

These are just a few of the methods that have been used to enhance CNNs; which ones are chosen is relevant to the issue, the resources at hand, and any constraints that may be imposed. Finding the optimal combination of methods to achieve optimal network performance often requires iterative optimization and experimentation. In this paper, we will concentrate on optimizing the parameters of a CNN using meta-heuristic algorithms. A comparative analysis of optimization approaches for CNNs is listed in Table 6, highlighting description and key strengths.

4.4 Applications of Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have revolutionized artificial intelligence and data processing, achieving remarkable success across a variety of domains. Their superior performance has led to widespread adoption in sectors such as healthcare, finance, security, and education. To enhance CNNs' effectiveness, many of these applications incorporate hyperparameter optimization (HPO) methods. Below, we present a concise overview of major research areas and practical applications of CNNs enhanced through HPO methodologies.

a. **Computer Vision and Related Tasks:** Computer Vision (CV) is focused on developing models that extract meaningful insights from visual data such as images and videos. Common applications include face recognition, pose estimation, and action detection. Face recognition, for example, must contend with variations in lighting, facial expressions, and occlusions. To address these issues, Farfade (2015) introduced a deep CNN capable of detecting faces at multiple angles and occlusion levels [118]. Later, Zhang (2016) developed a cascaded CNN for face recognition that outperformed contemporary methods [119].

Pose estimation remains one of the most complex tasks in CV due to the diversity in human body configurations. Bulat and Tzimiropoulos (2016) proposed a cascaded CNN approach that initially identifies heat maps and then performs regression analysis for pose detection [120]. Similarly,

Table 6 Comparative analysis of optimization approaches for CNNs

Technique	Description	Key strengths	References
Architecture design	Structural configuration of CNN layers	Enhances feature extraction and hierarchical learning	[7–10, 80, 82, 83, 89–92]
L1 regularization	Penalizes absolute weights	Promotes sparsity; useful for feature selection	[107, 108]
L2 regularization	Penalizes squared weights (weight decay)	Prevents large weights; improves generalization	[107, 108]
Data augmentation	Introduces data variability via transformations	Reduces overfitting; improves generalization	[49, 84, 109, 110]
Adam optimizer	Adaptive learning rates per parameter	Fast convergence; efficient for sparse gradients	[111]
RMSprop	Uses moving average of squared gradients	Effective for non-stationary problems; popular in RNNs	[112]
SGD	Basic gradient descent with random batch updates	Strong generalization; foundation of most optimization schemes	[113]
Bayesian optimization	Probabilistic model-based hyperparameter tuning	Efficient tuning with fewer trials; handles expensive evaluations	[116, 117]
Grid/random search	Systematic or stochastic search of hyperparameter space	Easy to implement; baseline for hyperparameter optimization	[114]

for action detection, Wang (2017) presented a 3D CNN that excels at recognizing temporal patterns in video frames [121], surpassing previous models based on Hidden Markov Models (HMMs) and motion history photography [122]. Ji (2010) also proposed a 3D CNN architecture capable of extracting spatial–temporal features from sequential frames [123].

b. **Processing of Natural Language:**

- **Statistical Modeling of Language:** Statistical modeling in NLP often involves predicting missing information within sequences. Kim et al. [124]. combined character-level CNNs with LSTM for improved sequence modeling. Kalchbrenner et al. [123] introduced ByteNet, a CNN architecture using dilated convolutions to capture long-range dependencies efficiently. Gu et al. [125] proposed a hierarchical CNN that integrates with recurrent networks, outperforming traditional LSTM models. Gated CNNs [126] further improved sequence modeling by mimicking LSTM's gating mechanism, though limitations remain in capturing hierarchical structures and long-term dependencies.
- **Text Classification:** CNNs have shown exceptional performance in text classification by capturing local dependencies in temporal and hierarchical structures. Approaches vary in architecture depth: some employ a single convolutional layer [127, 128], while others use multiple layers for richer feature extraction [129, 130]. Yin et al. [131] extended these architectures with hierarchical convolutional designs. Pooling methods such as k-max pooling [130] retain key features while preserving sequence order. Conneau et al. [132] explored deep CNNs (up to 49 layers) and found that shortcut connections significantly improve performance in deeper networks.

c. **Speech Recognition:** While CNNs are traditionally favored for image-related tasks, their application in speech recognition is gaining traction. Hamid (2012) demonstrated a CNN model that reduced error rates by 10% compared to previous approaches [81]. CNNs have also proven effective in phoneme recognition and emotion detection in speech. Huang et al. [133] combined CNN and LSTM architectures to capture emotional cues in both verbal and non-verbal speech content.

d. **Classification of Images:** CNNs for a long time have been widely employed in the area of picture classification [9, 89, 134, 135]. CNNs outperform other methods in terms of their capacity to attain higher accuracy in classifying huge datasets [136, 137]. This is because CNNs are capable of simultaneously learning both features and classifiers. The advent of large-scale picture categorization occurred in 2012. Krizhevsky et al. [136] created the AlexNet and achieved the highest level of efficiency in the ILSVRC 2012 competition. Following the achievement of AlexNet, Several studies have shown that by decreasing the total number of filters, classification accuracy may be significantly improved [9]or increasing the network's depth [136, 138, 139]. CNN relies heavily on medical imaging, in particular for the evaluation of histological pictures and the objective of detecting cancer. In their study, reference [140] used CNN to analyze breast cancer images and conducted a comparison with a pre-trained network that applied manually crafted descriptors to a dataset. Data augmentation is used in the second step to address the issue of class skewness. Multiple widely used pre-trained networks for image categorization are already available. picture categorization may be simplified if a dataset with clear labels is available for the desired picture.

e. **Recognition and Detection of Text:**

   - **Text Detection:** A seminal study that used CNN with respect to the purpose of scene text identification is referenced as [141]. The CNN model used by [141] trains on clipped the scene patches to include both text and non-text elements in order to differentiate between the two. Using the input's multiscale image pyramid, the CNN filters generate response maps that include text identification. Xu and others have reduced the scope of text recognition research. [142] suggest using Maximally Stable Extremal Regions (MSER) to generate a collection of potential characters, which are then filtered using CNN categorization. Another study that integrates MSER and CNN for text detection is referenced as [143]. In [143], researchers use a CNN to differentiate among MSER components that resemble text and those that do not. Additionally, crowded parts of text are separated by using CNN in a movable window fashion, followed by Non-Maximal Suppression NMS. In addition to text localization, there is a fascinating study [144] that utilizes convolutional neural networks (CNN) to ascertain if an input picture includes text without providing specific information about the actual location of the text. In [144] text candidates are acquired by MSER and then fed inputted in a CNN to provide visual characteristics. Finally, the global features of the pictures are formed via combining the CNN features inside a Bag-of-Words (BoW) framework.
   - **Text Recognition:** Goodfellow et al. propose a convolutional neural network that, in the final layers, has several softmax classifiers [145]. Text prediction at each successive location in the multi-digit input image is assigned to each classifier in the model. To tackle the issue with text identification without a vocabulary or lexicon, Jaderberg and others [146] present another CNN model that is similar to Conditional Random Fields (CRF) and learns to generate bigrams and forecast character sequences for scene text recognition all at once. Newer text recognition techniques include variations of (Recurrent Neural Networks RNN) to enhance the modeling of Sequential dependencies in text, in addition to traditional convolutional neural network (CNN) models. While LSTM handles the sequence labeling in [147], CNN Image patches generated by sliding windows are used to detect certain features of visual information [148]. Compared to [148], the approach given in [149] is almost identical; however, the latter allows for the consideration of lexicon, intending to enhance text recognition efficiency.

f. **End-to-End Text Spotting:** Wang et al. use a CNN that has been developed for character classification to achieve text detection from beginning to end [150]. Consistent with previous work, the convolutional neural network CNN suggested in [151] This allows for the sharing of features across the four separate tasks that make up an end-to-end text spotting system: text detection, character classification (both sensitive to case and case-insensitive), bigram classification, and text matching. Jaderberg et al. [152] do comprehensive beginning-to-end text detection using CNNs. Different CNN models handle the three main subtasks of the system suggested in Ref. [152]. Classification of text inside bounding boxes, regression of text within bounding boxes, and text recognition.

g. **Segmentation and Object Detection:** A necessary and long-standing issue in computer vision is object detection [133, 153, 154]. Finding the precise and efficient location of objects in still photos or video frames is often the biggest challenge. CNNs usage for detection and localization purposes since the 1990s [155]. However, prior to 2012, CNN-based object identification made little headway owing to insufficient training data and processing resources. Success with convolutional neural networks in the ImageNet competition in 2012 has reignited interest in CNN-based object identification in recent years [156]. Prior research [155–157] suggests extensively testing the CNN classifier on windows collected at different locations and scales using sliding window-based methods. Methods like Pascal VOC [158], ImageNet, and MSCOCO [153] are computationally expensive and hence unfit for use on datasets of great magnitude because of the vast candidate count windows present in images. There has been a lot of research on object-proposal-based algorithms recently [133, 159–162].

In order to identify whether a window sample contains possible objects, these algorithms often use quick and general measures. If they do, they then transmit the output object proposals to more advanced detectors, which then decide if the objects are background or belong to a certain class. The (Region-based Convolutional Neural Network R-CNN) is one of dependent best-known CNN detectors that uses object proposals [163]. Approximately two thousand bottom-up area suggestions that are likely to include items are extracted by R-CNN using selective search (SS) [160]. Next, a pre-trained CNN is used to derive characteristics from these distorted area suggestions, which are then scaled to a fixed size of 227 × 229. Detection is then carried out using a binary SVM classifier. Finding good training sets is another critical challenge in object detection, due to the fact that the quantity and quality of positive and negative samples determine efficiency. Recently, there has been a growing interest in online bootstrapping (or hard negative mining) for training CNNs because of its effectiveness in enabling intelligent cognitive systems to interact with dynamic contexts [164].

Online hard example mining (OHEM), a new bootstrapping method for CNN-based detection model training, is suggested in [165]. Because it chooses the challenging instances automatically, training is made easier. Meanwhile, it only needs to calculate an image's feature maps once before passing them on to each and every region of interest (RoI) in the picture. For this reason, it may locate the challenging instances at a little more computational expense. Newer methods like YOLO [166] and SSD [167] make it possible to discover classes with a single pipeline. The reason for this is because it uses related class probabilities and separated bounding boxes. YOLO [166] approaches object recognition as a regression issue. It is possible to improve the whole detection pipeline based on detection performance alone, because during a single evaluation, a single network is able to forecast both the bounding boxes and the class probabilities from the whole image. With SSD [167] for each location in the feature map, the output space of bounding boxes is discretized into a set of default boxes that cover a range of aspect ratios and sizes. Compared to YOLO, SSD's accuracy is much higher with this many scale configurations and their matching approach. An additional network is educated to take into consideration such division selections in a top-down search method proposed by Lu et al. [122] to recursively split a window into sub-windows using the advantages of super-resolution.

h. **Low-resolution images:** Resolution improvements in images have been achieved by many ML researchers using CNN-based image enhancement methods [168–170]. A DCNN-based approach it was utilized by Peng and others to detect objects in low-resolution images [169]. Chevalier et al. developed LR-CNN for the categorization of images with poor resolution [168]. Yet another technique based on DL for action detection from low-resolution thermal images is described by Kawashima et al. [170]. This method makes use of convolutional layers in conjunction with an LSTM one.
i. **Video Sequence Action Recognition:** In contrast to the two-dimensional pure spatial signals that conventional CNNs are trained to represent, videos have a third axis—time—that is fundamentally distinct from the spatial fluctuations seen in still pictures, making CNN application to videos a difficult task [171, 172]. Another challenge with using convolutional networks on video data is that their sizes are higher-order compared to picture signals. Ji et al. suggest a network of three-dimensional convolutional layers for use with video inputs [12], to treat the time axis similarly to other spatial axes. In recent research, Tran et al. [173] highlight the advantages of this strategy above others by analyzing its performance, efficiency, and efficacy.

Alternatively, as suggested by [14] one may utilize 2D convolutions and combine the feature maps of successive frames to apply CNNs to movies. They compare the CNN efficiency on individual frames to various fusion strategies, including slow, early, and late fusion. If we follow Simonyan and Zisserman's [174]suggestion and train distinct CNNs for spatial and temporal changes, we may further improve action recognition using CNNs that account. In the initial influx of this architecture, a standard CNN is utilized to each frame. In the second influx, the dense optical flow of the input movies is used to train a CNN that is structurally and dimensionally similar to the spatial stream. At the stage of fusing class scores, the two streams' outputs are blended. The authors, Ch'eron et al. [175] demonstrate how to increase action recognition performance by aggregating part-based local CNN descriptors using two-stream CNN on localized human body parts. One alternative to spatial fluctuations for representing video dynamics is to input the CNN-based properties of individual frames into a sequence learning module, such a recurrent neural network. The authors, Donahue et al. [176] investigate several implementations of this framework's sequence learner using LSTM units.

5 Convolutional Neural Network Hyperparameters

Convolutional Neural Network hyperparameters play a crucial role in determining model performance independently of learned weights and biases. These fixed values, including architectural aspects like layer depth and kernel size, along with optimization parameters like learning rate, must be appropriately selected before training because they cannot be inferred from the data [11, 12]. Their significance lies in the way they essentially define the structure of the network as well as its learning process, and hence hyperparameter tuning is an imperative part of CNN development. While parameters are optimized via backpropagation, hyperparameters need various optimization methods—a challenge that we shall look into through reviewing typical CNN hyperparameters and a variety of hyperparameter optimization methods later in this section.

5.1 General Hyperparameters

To increase the performance of CNN architectures, you may modify hyperparameters such as batch size, learning rate, and epochs, in addition to the hyperparameters of convolutional and fully connected layers, as illustrated below [177, 178].

– **Learning rate:** The learning rate determines the frequency of parameter updates for the best outcomes. Incorrect learning rates may affect model performance. Georgeakopoulos et al. [179] suggest that increasing LR increases convergence when two subsequent steps have the same gradient vector direction. Cyclical learning rates balance exploration and exploitation but require parameter fine-tuning for CNN training. Fixed learning rates are simple but lack adaptability. Each learning rate performs better for various situations, emphasizing the necessity of selecting the right method depending on the neural network being trained. Various learning rate strategies have been developed for CNNs, each with unique advantages and limitations (see Table 7 for comparison).
– **Batch size:** The batch size refers to the quantity of data handled concurrently during gradient estimation. A large batch size impedes network convergence. Limited batch sizes may destabilize networks and produce suboptimal results. Consequently, a model must determine the appropriate batch size. Kandel et al. [180] assert that smaller batch sizes, namely between 2 and 32, provide superior outcomes compared to larger quantities. People often use batch sizes of 16, 32, and 64 with various hyperparameters.
– **Epochs:** The quantity of epochs [181] specifies the total number of complete iterations (forward and backward) for a neural network during the training process. Merely processing all training data once is insufficient to ascertain the characteristics of a neural network. The dataset may need additional inputs to enhance generalization, perhaps several times. Overfitting is a significant challenge in a network, necessitating an optimal number of epochs to mitigate this problem. The optimal epoch size for all datasets remains uncertain. Sinha et al. [182] advised using a self-organized map (SOM) approach to determine the ideal number of epochs. SOM assists in selecting data for network training and testing to prevent overfitting and determine the ideal epoch size.

Table 7 Comparative analysis of learning rate strategies in CNNs

Learning rate strategy	Description	Advantages	Disadvantages
Fixed learning rate [57]	A constant learning rate throughout training	Simple to implement; works well in some scenarios	May not adapt well to training progress; it can lead to slow convergence or divergence
Step decay [49]	Reduces the learning rate by a factor every few epochs	Helps model converge more finely in later stages	Requires careful tuning of steps and decay factor
Exponential decay [183]	Reduces learning rate exponentially over time	Smooth transition; widely used in practice	May decay too quickly or too slowly if hyperparameters are not well-tuned
1/t decay (inverse time decay) [184]	Learning rate decays inversely with epoch number	Simple and stable; avoids too rapid decay	It can be too conservative in early epochs
Adaptive learning rates (Adagrad) [185]	Adapts the learning rate for each parameter based on past gradients	Good for sparse data; no need for manual decay scheduling	Learning rate can become too small over time
RMSprop [186]	Similar to Adagrad but uses a moving average of squared gradients	Prevents aggressive decay; better suited for non-stationary objectives	Requires tuning decay factor
Adam (adaptive moment estimation) [187]	Combines momentum and RMSprop for better convergence	Widely used; works well out of the box; adaptive	May generalize worse than SGD in some cases
Cyclical learning rates (CLR) [188]	Learning rate cyclically varies between bounds	Can escape saddle points; faster convergence	Requires tuning of cycle length and bounds
Cosine annealing [189]	Learning rate follows a cosine decay schedule	Smooth decay; helps in regularization	It can be sensitive to the total epoch count
Warm restarts (SGDR) [189]	Cosine annealing with periodic restarts	Helps escape local minima; improved generalization	Adds complexity in scheduling restarts
Learning rate finder [190]	A strategy to select the optimal initial LR by gradually increasing and plotting	Empirical and data-driven; useful in practical model training	Requires preliminary training run

5.2 Strategies for Hyper-Parameter Optimization (HPO)

5.2.1 Model-Free Algorithms

The ability to deal with complicated and unfamiliar situations is a strength of model-free algorithms over their model-dependent counterparts. On the other hand, they could have sample inefficiency and call for a lot of interactions with the environment to converge. Due to their reliance on trial-and-error learning, they may also struggle in environments that are difficult to explore. You may find model-free algorithms in several forms, such as:

(1) **Grid Search (GS):** When investigating possible hyperparameter configurations, grid search (GS) is a popular tool [191]. GS examines every possible combination of hyperparameters input into the grid of configurations, making it both an exhaustive search and a brute-force approach [192]. To function, GS takes a finite set of values that the user specifies and calculates their Cartesian product [175]. On its own, GS is unable to fully capitalize on the areas that are functioning successfully. In order to find the global optimum, it is necessary to manually execute the following technique [176]:

  (a) To begin, set the search space and step size to be quite large.
  (b) Reduce the search area and step size based on the results of successful hyper-parameter setups.
  (c) For optimal results, repeat step 2 as needed.

GS is readily implementable and can be parallelized. The fundamental problem with GS, however, is that it is inefficient for high-dimensional hyper-parameter configuration spaces; Having more hyperparameters causes the number of assessments to grow exponentially. The phenomenon of exponential increases in dimensions is sometimes referred to as the “curse of dimensionality” issue [191]. Under the assumption that there are (k) parameters With (n) a finite number of distinct values, the computational of GS expands with an exponential rate of O(n^k) [192]. For GS to work as an HPO method, the hyperparameter area of configuration has to be small.

(2) **Random Search (RS)**: In order to address some of GS’s shortcomings, random search (RS) was suggested in [15]. Parallel to GS, RS picks are established in an advanced quantity of samples in between the upper and lower boundaries at random to serve as potential values for hyperparameters. Subsequently, it proceeds to train these predetermined values until they are exhausted. The scientific foundation of RS states that, given a sufficiently vast configuration space, the global optimums, or possibly their approximations, can be found. Even though RS has a smaller budget than GS, it can cover more ground in its search [15]. Due to the independence of each assessment, RS is well-suited for parallelization and resource allocation. To increase system efficiency and decrease the likelihood of spending time on a tiny, poor-performing area, RS uses a predetermined distribution to select a specific number of possible parameter choices, unlike GS. There is a fixed value n for the total number of evaluations in RS before optimization begins; hence, the computational difficulty of RS becomes O(n) [193]. Furthermore, with sufficient funding, RS can identify the worldwide or near-global optimum [175]. While RS outperforms GS in vast search areas, it fails to capitalize on previously successful regions, leading to an excessive number of needless function evaluations [176]. Finally, the biggest problem with RS and GS is that they both spend a lot of time assessing underperforming parts of the search space because each assessment in their iterations is separate from the others. Bayesian optimization, which takes into account past assessment records to ascertain the upcoming evaluation, is one alternative optimization strategy that may address this problem [194].

(3) **Trial and Error:** One simple way to tune hyperparameters is by using gradient descent (GSD), which is also known as babysitting [195]. This approach is often used by academics and students and is executed via meticulous hand adjustment. Simply put, after creating an ML model, students try out an extensive array of hyperparameter values employing their knowledge, intuition, or analysis of past results. This cycle continues until either the student reaches a time limit for completion or is content with the outcomes. Hence, this strategy requires a significant amount of prior information and skills to determine the best hyper-parameter values within a constrained timeframe. In many cases, manual tuning is not feasible due to a large number of hyperparameters, complex models, long model evaluations, and non-linear interactions among hyperparameters [196]. Because of these considerations, there has been a surge in studies aimed at developing methods for the autonomous optimization of hyperparameters [197].

5.2.2 Bayesian Optimization (BO)

Most often, iterative algorithms like Bayesian optimization (BO) [13] are used to solve HPO issues. Bo uses the data from prior evaluations to decide the points for future evaluations, unlike GS and RS. Two essential parts of BO are the process of acquisition and the substitute model; these are used to determine the subsequent hyper-parameter setup [198]. All of the points found must be fitted into the goal function simultaneously for the surrogate model to be considered successful. Once the acquisition function has the distribution of probabilistic surrogate model predictions, it determines the usage of various points by balancing exploration and exploitation. While exploitation involves sampling in prospective places the most probable location for the optimal global value may be determined by analyzing the posterior distribution, exploration involves sampling in areas that have not been explored before. BO models find the sweet spot between exploration and exploitation to find the most optimum regions right now while avoiding losing out on superior configurations in uncharted places [197]. As a general rule, BO works like this [13]:

(a) Construct an objective function surrogate model using probability theory.
(b) Get the surrogate model's hyperparameters set to their optimum settings.
(c) To assess them, insert these hyperparameter values into the actual objective function.
(d) Refresh the surrogate model using the latest findings.
(e) Once you've reached the maximum quantity of iterations, stop completing processes 2–4.

Therefore, BO modifications the surrogate model after each objective function evaluation. With BO, you may discover the best hyper-parameter combinations by looking at the attempted values, which makes it more efficient than GS and RS, and running a surrogate model is usually significantly cheaper than running the whole objective function. Executed sequentially using previously-tested values, Bayesian optimization models are challenging to parallelize [199], despite the fact that they often discover near-optimal hyperparameter configurations within a few rounds. The (Tree Parzen Estimator TPE) [200], the random forest (RF) [201], and the Gaussian process (GP) [202] are common BO surrogate models. Based on their surrogate models, the three main types of Bayesian Optimization (BO) algorithms are BO-GP, BO-RF, and BO-TPE. SMAC, which stands for Sequential Model-Based Algorithm Configuration, is an alternative term for BO-RF [201].

5.2.3 Optimization with Gradients

Among the more conventional methods of optimization, gradient descent [16] uses the gradient of variables to find the best possible course of action. Following a data point selection at random, the method follows the direction with the highest gradient in order to find the next data point. Consequently, convergence allows for the attainment of a local optimum. A convex function's local optimum is its global optimum. For optimal k, the time complexity is O(n^k) using gradient-based methods [203]. It is feasible to tune the hyperparameters of certain ML algorithms by calculating their gradient and then using gradient descent. However, there are a number of drawbacks to using gradient-based algorithms, even if they converge to the local optimum more quickly than. They can only optimize continuous hyperparameters as other types of hyperparameters, including categorical hyperparameters, do not have gradient directions. Furthermore, they are ineffective when applied to non-convex functions, which might lead to a local optimum instead of a global one [176]. Consequently, gradient-based techniques are limited to situations where hyper-parameter gradients may be obtained, such as when improving the learning rate in neural networks [204]. Nonetheless, finding global optimums via gradient-based optimization approaches is no guarantee for ML systems.

5.2.4 Algorithms for Multifidelity Optimization

Long execution time is a big problem with HPO, and it becomes worse as the hyper-parameter configuration space and dataset sizes grow. The time required for execution could range from a few hours to days or more [205]. To surmount the constraints of restricted time and resources, multifidelity optimization methods are often used. Using just some of the features or the initial dataset could help someone save time [206]. A multifidelity assessment combines uses at low and high fidelity evaluations for workable solutions [207]. While low-fidelity evaluations are inexpensive, they only evaluate a small fraction and have poor generalizability. However, low-fidelity assessments are more cost-effective, and high-fidelity evaluations assess a larger group with more generalizability while incurring more expense. By the end of each round of evaluating hyperparameters on the created subsets, multi-fidelity optimization algorithms remove configurations that performed badly, allowing them to focus on training on the best configurations. Deep learning optimization issues have been successfully addressed by bandit-based algorithms classified as multifidelity optimization methods [208]. Successive Halving [209]and Hyperband [210]two commonly used bandit-based approaches.

(1) **Successive Halving:** In theory, by testing each possible combination of hyperparameters, exhaustive approaches may find the one that works best. Practical implementations, however, need to take a number of things taking factors like available time and materials into consideration. We refer to these considerations as budgets (B). Successive halving techniques were suggested in [209] to enhance efficiency and circumvent the restrictions of (GS) and (RS). Below is the primary

method for applying consecutive algorithms that divide the search space in half for Hyperparameter Optimization (HPO). Starting with the premise that there exist n possible combinations of hyperparameters and that each one is tested using a budget that is evenly distributed ($b = \frac{B}{n}$). Next, advanced the hyperparameter configurations that performed better to the subsequent iteration with double budgets ($b_{i+1} = 2 * b_i$), and rejected half of the configurations that performed badly based on the evaluation findings for each iteration. We keep iterating until we find the perfect mix of hyperparameters. Compared to (RS), consecutive halving is more efficient, although this is dependent on a tradeoff between the number of hyperparameter configurations and the budgets given to each configuration [175]. Budget management and the trade-off between testing more configurations with a smaller budget and fewer configurations with a bigger budget are, hence, the fundamental challenges of successive halves [176].

(2) **Hyperband:** To address the problem with consecutive halving methods, Hyperband [210] is suggested as a solution that can dynamically choose an appropriate number of variants. This method divides the entire budget (B) into n parts and assigns one piece to each hyper-parameter configuration ($b = \frac{B}{n}$), to achieve a compromise between the number of configurations (n) and their assigned spending plans. To enhance efficiency and remove badly performing hyperparameter combinations, a subroutine called successive halving is applied to each batch of random configurations [176]. Algorithm 1 demonstrates the primary procedures of hyperband algorithms.

Hyperband algorithm

```
Input : $b_{max}, b_{min}$
1 : $s_{max} = log(\frac{b_{max}}{b_{min}})$
2 : $fors \in \{b_{max}, b_{min} - 1, \dots, 0\}do$
3 : $n = Determine\ Budget(s)$
4 : $\eta = SampleConfigurations(n)$
5 : $SuccessiveHalving(\eta)$
6: $endfor$
7 : $returnthebestconfigurationsofar$
```

The budget restrictions b_{min} and b_{max} are determined by the number of data points, the least number of instances required to train an acceptable model, and the feasible budgets. Next, in steps 2–3 of Algorithm 1, we use b_{min} and b_{max} to determine the combined quantity of configurations denoted as 'n', plus the allocated financial resources, referred to as the budget. The amount that is assigned to each configuration. The configurations are randomly selected. according to n and b and thereafter sent to the consecutive (Halving model) shown in steps 4–5. While moving on to the following iteration, the consecutive halving method discards the configurations that were found to be underperforming. Iteratively pursuing the optimal possible hyperparameter configuration is the end goal. The computational complexity of the Hyperband is O($nlog(n)$) [210] due to its use of the successive halving searching strategy.

(3) **Bayesian Optimization Hyper Band (BOHB):** The combination of Bayesian optimization with hyperband results in a very advanced approach for hyperparameter optimization dubbed Bayesian Optimization Hyper Band (BOHB) [211]. Takes advantage of both approaches while avoiding their downsides. To scan the hyper-parameter configuration space, the original Hyperband used an inefficient random search. BOHB optimizes all kinds of hyperparameters to provide high performance with minimal execution time by efficiently employing parallel resources, replacing the RS technique. In BOHB, the usual stand-in for BO uses multidimensional kernel density estimators, which are a variation on the Tree-structured Parzen Estimator (TPE). Accordingly, BOHB also has a complexity of O($nlog(n)$) [211]. Among several optimization methods, BOHB has shown superior performance when fine-tuning DL and SVM models [211]. To avoid decreased convergence speed compared to typical BO models, BOHB only works if assessments on tiny budget subsets are indicative of evaluations on the whole training set.

6 Metaheuristic Algorithms

Metaheuristic algorithms [17] are effective methods to find near-optimal solutions to difficult optimization problems by efficacious exploration of search spaces, frequently inspired by physical phenomena, biological processes, or nature-inspired systems. Metaheuristics are superior to traditional optimization methods when solving non-smooth, discontinuous, and non-convex problems that are difficult for traditional methods. Every algorithm possesses distinct search mechanisms, explores versus exploits differently, and makes different assumptions [212, 213]. These methods have been applied to be particularly beneficial in numerous applications, including engineering design, scheduling, logistics, and machine learning problems. While all methods possess unique strengths, their performance is dependent on the correct matching of algorithmic strengths with problem requirements—a goal that naturally results in the adaptation of these methods to specific applications. This chapter briefly outlines ten popular metaheuristic algorithms organized into five taxonomic classes, whose classification hierarchy is presented in Fig. 6.

6.1 Particle Swarm Optimization (PSO)

Improve continuous nonlinear functions with the use of a particle swarm optimization algorithm. Kennedy et al. first proposed the method [214] with the aim of promoting social behavior. Another stochastic population evolutionary optimization method is particle swarm optimization, or PSO. Every particle in PSO stands for a different solution [215]. The process begins by setting the starting values for the primary parameters. Next, generate a completely random figure for the location. The algorithm terminates by producing the optimal outcome in accordance with the termination specification. If it doesn't, we update the particle velocities and locations, record the local and global best outcomes, and assess the particle fitness values. After each iteration of the PSO algorithm, the global search employs two equations to update the locations of each particle. Below [216], there are the equations:

$$V_i^{t+1} = w * V_i^t + c1r1\left(XBest_i^t - X_i^t\right) + c2r2(gBest_i^t - X_i^t) \tag{8}$$

$$X_{id}(t+1) = X_{id}(t) + V_{id}(t+1) \tag{9}$$

Equation (8) refers to the velocity of each piece as V_i^{t+1}. The initial inertia is denoted by w, the particle's previous velocity is denoted by V_i^t, the random numbers are $r1$ and $r2$, the corresponding accelerations are $c1$ and $c2$, the best fitness value of the local particle is $XBest_i^t$, and the best fitness value of the global particle is $gBest_i^t$. The prior locations of the swarm i at time t are represented by X_i^t. In order to determine the swarm's updated locations, we use Eq. (9).

In order to discover the CNN's architectures and hyperparameters all at once, Pratibha et al. [217] presented a hybrid multilevel PSO method. The experiments are carried out on two levels: first, the designs are optimized, and then the hyperparameters are optimized. To determine the best setup for the model, this technique employs numerous swarms. The experiment began by randomly assigning values to the swarm's convolutional, pooling, and fully connected layers. Every particle in the first swarm initiates a second swarm at level 2. Consequently, multiple swarms emerge at level 2, representing variables like padding, output neuron count, stride size, filter size, and number of filters. The operation repeats when it meets the halted condition. After careful calculation, the fitness function is marked as the optimal choice. The swarm at level 2 repeats the process for every swarm at level 1. As a result, the algorithms allow the CNN model to achieve its utmost accuracy.

Shaikh et al. [218] provide a fresh perspective on estimating transmission line characteristics, which are critical for reliable power distribution and excellent service. By integrating moth-flame optimization (MFO) and PSO techniques, it introduces a hybrid approach that fixes the problems of previous systems. The hybrid algorithm does better than traditional PSO and MFO methods in terms of convergence time and solution quality, as shown by multiple simulations and mathematical validations. This makes it an appealing choice for accurately estimating transmission line parameters.

6.2 Ant Colony Optimization (ACO)

One method for maximizing food collection efficiency is ant colony optimization, or ACO [219]. This method takes advantage of ant search patterns. A small path guides ants from their nest to their food source. In order to find the quickest path, they leave pheromones behind. Though they eventually fade and become less dense, ants are able to detect and decipher pheromones. The high-density pheromone aids in the discovery of food by other ant species by means of density. The density of pheromones increases as the number of ants depositing them on shorter paths increases. Conversely, the pheromone on the opposite route gradually diminishes or disappears. Updates to ant-m pheromone values are the topic of this approach. All of the m ants who have solved the problem may change the pheromone's value in each cycle in the following way [219]:

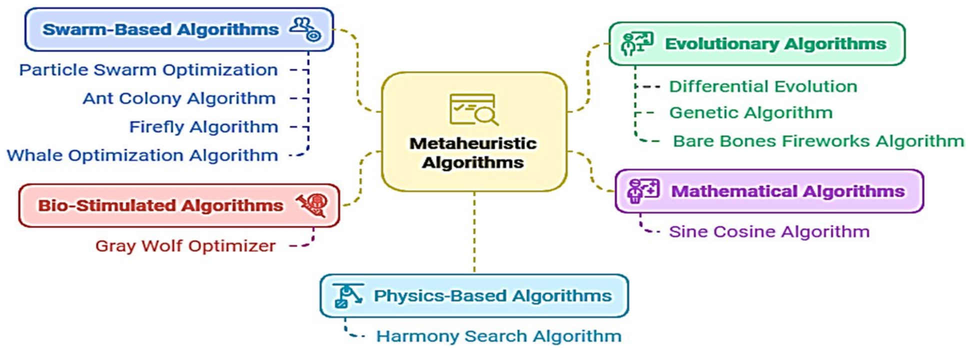


Fig. 6 Taxonomy of metaheuristic algorithms

$$\tau_{ij} \leftarrow (1-\rho).\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{10}$$

where i is the source,j is the destination, and τ_{ij} is the pheromone value. The evaporation rate is denoted by ρ, the number of ants is represented by m, and the amount of pheromone on routes i,j by the ant k is denoted by $\Delta\tau_{ij}^{k}$.

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & ifantkusededgei,j \\ 0 & otherwise \end{cases} \tag{11}$$

Q is a constant, and L_k is the touring route length. A stochastic method determines the ant's movement. If the ant is currently at position i, the probability of it moving to position j is as follows:

$$P_{ij}^{k} = \begin{cases} \frac{\tau_{ij}^{k}.X_{ij}^{\beta}}{\sum C_{il} \in N(s^p) \tau_{il}^{\alpha}.\eta_{il}^{\beta}} & ifC_{il} \in N(s^p) \\ 0 & otherwise \end{cases} \tag{12}$$

Here, $N(s^p)$ denotes the set of possible components, which is an edge (i,l), where ant k has not yet visited l. α and β deal with the significance of pheromones and η_{ij} heuristic information.

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{13}$$

In this case, d_{ij} represents the separation between i and j.

In order to discover the CNN's architectures and hyperparameters, Lankford et al. [220] employed ACO to choose the optimal neural network and hyperparameters in a separate investigation. Two setups with different ant numbers and epoch counts are generated by the procedure. The first configuration uses eight ants and thirty epochs, whereas the second uses sixteen ants and fifteen epochs. You can tweak the size of the kernel, rate of dropout, and number of layers (maximum and minimum) as additional hyperparameters. To start fine-tuning, they eliminated completely linked layers from the top of the model. They tested the model on two more datasets and found it effective.

6.3 Firefly Algorithm (FA)

Yang created the firefly algorithm (FA) in 2008 and has since used it to solve several optimization issues [221], including CNN HPO [222]. The algorithm's three-rule foundation was primarily inspired by natural phenomena like firefly flashing [221]:

– Fireflies are unisex; therefore, any other firefly may attract any other firefly.
– The intensity of the attraction is proportional to the fireflies' luminosity. A closer proximity of the firefly results in a stronger attraction, while a closer distance results in a weaker tone.
– How bright a firefly is a favorable indicator of its fitness level.

Initially, the process uses the following equation to create fireflies (solutions) at random, with the lower limit and upper bound being:

$$X_{i,j} = lb_j + rand(ub_j - lb_j) \tag{14}$$

In this context, $X_{i,j}$ denotes the ith solution and its corresponding jth element; the lower bound is represented by lb_j and the upper bound by ub_j. $rand$ is a stochastic number.

The firefly uses the following formula to update its location:

$$X_i^{t+1} = X_i^t + \beta_0 r^{-\gamma r_{i,j}^2}\left(X_j - X_i\right) + \alpha(rand - 0.5) \tag{15}$$

Here, the current position is represented by X_i^t, while the new location is represented X_i^{t+1}. At a distance of zero, β_0 indicates attraction. X_j is the brighter answer, and the existing solution switches to it. The parameter for randomization is alpha. The Euclidean distance between two fireflies, X_i and X_j, is as follows:

$$r_{i,j} = \|X_i - X_j\| = \sqrt{\sum_{k=1}^{d} (X_{i,k} - X_{j,k})^2} \tag{16}$$

where $r_{i,j}$ represents the distance between the solutions X_i and X_j. After calculating the distance, we must determine the new attractive value as follows:

$$\beta(r) = \frac{\beta_0}{1+\gamma r^2} \tag{17}$$

where the new attractiveness value is denoted by β(r). Here is the formula for determining the light intensity:

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

where γ stands for the light absorption coefficient, $I(r)$ represents the light intensity at a distance r.

By using an improved FA method, Nebojsa et al. [222] achieve HPO. This work optimizes the following CNN hyperparameters: convolutional layer count, kernel size, fully connected layer count, and number of hidden units in the fully connected layer. We maintain the randomization parameter and light absorption coefficient at 0.5 and 1.0, respectively. They leave all other parameters irrelevant to the optimization problem unchanged. For example, they employ ReLU as an activation function, use the Adam optimizer with a learning rate of 0.0001, and maintain a pooling layer size of 2×2. They train the model in one epoch with a batch size of 54. This ensures that they get optimal hyperparameters. Similarly, Aswanandini et al. [223] tune hyperparameters using FA to identify those that provide the best classification accuracy. To achieve this goal, they establish two heuristics: a low-level approach that uses firefly as a selection criterion and a high-level method that uses that set of solutions to determine which one is better. The size of the kernel, the number of convolutional layers, the number of pooling layers, and the number of fully connected layers with hidden units are the hyperparameters in question. They have maintained β at 0.02 and γ between 0.01 and 100 to ensure optimum exploitation. For this research, we have not optimized the learning rate, dropout rate, or a few other parameters with actual values. Finally, they test the model on the NSL-KDD dataset and select the configuration with the best classification error rate.

6.4 Whale Optimization Algorithm (WOA)

Whales are the largest sentient mammals on Earth. Whales include several species, including humpback, finback, and orca whales. This inquiry focuses on the hunting behaviors of humpback whales. Mirjalili et al. [224] created a novel swarm-based optimization approach called the Whale Optimization Algorithm (WOA). They designed it based on the hunting methods of humpback whales. This approach involves three operators who simulate the hunting behavior of humpback whales:

- Foraging for Prey: Whales seek locations abundant in food sources.
- Circling Prey: Once they locate sustenance, they encircle it to ensnare it.
- Employing Bubble Nets: They use bubble nets to efficiently capture fish.

Bubble net feeding refers to this specific foraging behavior. They mathematically model the optimization of spiral bubble net feeding, which includes prey encirclement, spiral bubble net feeding, and prey seeking. Humpback whales are required to identify the locations of their prey, discern them, and then encircle them. Since the exact position of the prey is not known, the WOA algorithm first selects the current solution as the most suitable option for the target prey. The optimal whale is recognized as having the most effective solution. According to the findings, the other agents/whales adjust their positions to approach the optimal whale. The equations below illustrate this behavior:

$$\vec{D} = \left| \vec{C} \cdot \vec{X}^{*}(t)\_\vec{X}(t) \right| \tag{19}$$

$$\vec{X}(t+1) = \vec{X}^{*}(t) - \vec{A} \cdot \vec{D} \tag{20}$$

where $\vec{X}$ is the position vector, $\vec{X}^{*}$ is the position vector of the best solution achieved so far, A and C are coefficient vectors, and t is the current iteration. X^{*} is updated after every iteration whenever a better solution is found. Below, we present the computation of A and C.

$$\vec{A} = 2\,\vec{a} \cdot \vec{r} - \vec{a} \tag{21}$$

$$\vec{C} = 2 \cdot \vec{r} \tag{22}$$

A random vector $\vec{r}$ is in the interval [0, 1], and for each iteration, $\vec{a}$ is linearly lowered from 2 to 0. To find food, whales use spiral updates and diminishing encircling techniques in their bubble-net behavior. Equation 3 represents the shrinking encircling approach. Here is a representation of the humpback whales' movement in spiral space:

$$\vec{X}(t+1) = \vec{\grave{D}} \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}^{*}(t) \tag{23}$$

here, $\vec{\grave{D}} = \vec{X}^{*}(t) - \vec{X}(t)$ represents the distance between the whale and its prey, b is used to characterize the logarithmic spiral form, and l signifies a random value inside the interval $[-1, 1]$. The whales navigate a diminishing circle and arc concurrently to find their meal. A 50% likelihood of transitioning between shrinking, encircling, and spiral forms characterizes this activity. The following equation modifies the whales' location:

$$\vec{X}(t+1) = \left\{ \begin{array}{ll} \vec{X}(t) - \vec{A} \cdot \vec{D} & if \quad p < 0.5 \\ \vec{\grave{D}} \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}^{*}(t) & if \quad p \geq 0.5 \end{array} \right\} \tag{24}$$

p is an arbitrary integer between 0 and 1. Additionally, whales may randomly seek prey using the following equations:

$$\vec{D} = \left| \vec{C} \cdot \vec{X}_{rand}(t)\_\vec{X}(t) \right| \tag{25}$$

$$\vec{X}(t+1) = \vec{X}_{rand}(t) - \vec{A} \cdot \vec{D} \tag{26}$$

In order to discover the CNN's architectures and hyperparameters, Sruthi et al. [225] used the whale optimization technique to enhance the efficacy of convolutional neural networks (CNN). This approach entails optimizing the hyperparameters of convolutional neural networks and picking relevant features from the input data, primarily for classifying lung nodules on CT scans. The Whale Optimization technique efficiently explores the solution space and enhances model correctness by repeatedly updating candidate solutions. The study's results show that the improved CNN model was better at telling the difference between nodules and other things than the previous improved CNN models. This illustrates the efficacy of the Whale Optimization method in improving the diagnostic proficiency of CNNs for lung cancer diagnoses. In conclusion, the research introduces an innovative method to enhance CNN's performance using WOA, resulting in favorable outcomes for classification accuracy and metrics.

6.5 Grey Wolf Optimizer (GWO)

A metaheuristic algorithm that mimics biological processes is known as the grey wolf optimization (GWO) algorithm. By studying the social dynamics, leadership structure, and hunting grounds of grey wolves, Mirjalili et al. [226] developed GWO in 2014. The name gives away the fact that grey wolves impacted the algorithm. The GWO mathematical model is shown by considering α wolf as the best choice, followed by β wolf, δ wolf, and the other possibilities by ω wolf. The behavior of gray wolves indicates that they initially encircle their victims. This encircling behavior is characterized by the mathematical formulas provided below:

$$\vec{D} = \left| \vec{C} \cdot \vec{X}_P(t)\_\vec{X}(t) \right| \tag{27}$$

$$\vec{X}(t+1) = \vec{X}_P(t) - \vec{A} \cdot \vec{D} \tag{28}$$

The position vector of the prey is represented by X_P, the position vector of the gray wolf is denoted by $\vec{X}$, t is the current iteration, and $\vec{A}$ and $\vec{C}$ are coefficient vectors.

$$\vec{A} = 2\vec{a}\,\vec{r}_1 - \vec{a} \tag{29}$$

$$\vec{C} = 2\vec{r}_2 \tag{30}$$

where $\vec{r}_1$ and $\vec{r}_2$ are arbitrary numbers between 0 and 1, and $\vec{a}$ is a diminishing parameter from 2 to 0. To replicate hunting, α, β, and δ are presumed to possess the most comprehensive understanding of the prey. The ensuing equations are used to revise the values of these three categories of wolves first, followed by the locations of the additional search agents (ω wolves).

$$\vec{D}_\alpha = \left|\vec{C}_1 \cdot \vec{X}_\alpha\_\vec{X}\right|, \vec{D}_\beta = \left|\vec{C}_2 \cdot \vec{X}_\beta\_\vec{X}\right|, \vec{D}_\delta = \left|\vec{C}_3 \cdot \vec{X}_\delta\_\vec{X}\right| \tag{31}$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_\alpha \cdot \left(\vec{D}_\alpha\right), \vec{X}_2 = \vec{X}_\beta - \vec{A}_\beta \cdot \left(\vec{D}_\beta\right), \vec{X}_3 = \vec{X}_\delta - \vec{A}_\delta \cdot \left(\vec{D}_\delta\right) \tag{32}$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{33}$$

Gray wolves finish their hunt by attacking their prey after it stops moving. Because $\vec{A}$ may take on values between $-a$ and a, the value of a decreases as iterations continue. So, the wolves are forced to attack the prey since $|A| < 1$. Another part of GWO that promotes exploration is $\vec{C}$, which helps to avoid local optimum and promote exploration. Aside from the initial and final random numbers between 0 and 2, it's completely bias-free, which is excellent for the exploration notion. The last step in the GWO method is to complete an end criterion.

In order to discover the CNN's architectures and hyperparameters, Shukla et al. [227] followed these procedures and used GWO to determine the optimal values for the CNN tuning parameters for transmission line protection. Using the power system model, they were able to gather current and voltage data at relaying buses in real-time for different types of faults. Subsequently, they established the maximum-minimum restrictions, iterations, and number of gray wolves for each CNN hyperparameter. They then used the search space to randomly create the beginning population for each design variable. Next, we trained CNN using the default hyperparameters for each individual population set. When the search is complete, the termination criteria halt the process, and the alpha wolf position shows the optimal CNN hyperparameter value for the current fault classification job.

In order to detect skin cancer, Mohakud et al. [228] tuned hyperparameters using GWO in four steps. Instructions for building the next-generation population include parameter encoding, population initialization, fitness assessment, and construction. The approach was used to optimize a variety of parameters, including the dropout rate, kernel size in the pooling layer, number of kernels associated with the convolution layer, and the size of the kernel in the convolution layer. The values of these parameters may be any integers within a certain range. They saved time during model training by utilizing a lightweight model with less predictability, and they minimized unnecessary repetitions by deciding whether the change in accuracy was minimal. By following this approach, they achieved the optimum outcome.

6.6 Harmony Search Algorithm (HAS)

Another metaheuristic optimization method is the harmony search (HS) algorithm. It draws inspiration from random occurrences in music, most notably jazz improvisation

[229]. In order to discover the optimal solution with a minimum number of iterations, the HS algorithm mimics the process by which a musician discovers a superior harmonic tone. A harmony search algorithm gets its value either from memory or randomly, much as a musician may play a note [230]. Value initialization, harmony memory (HM) initialization, new harmony construction, updating HM, and repetition operations are the five basic components of the HS algorithm [231].

- Value initialization: The HS method for HPO starts with initializing values. The harmony search method requires the configuration of the harmony memory size (HMS), harmony memory consideration ratio (HMCR), and pitch adjustment ratio (PAR).
- HM initialization: The HS method use HM to determine the ideal solution and refine the process by preserving superior harmonies and discarding worse ones.
- Formation of new harmony: This procedure eliminates the inferior harmonies and incorporates the superior ones. This procedure uses HMCR to determine whether to create a new harmonic or subtly modify the current harmonies.
- Update HM: We contrast the new harmonies of HMCR and PAR with HM's least favorable performance. New harmonies that surpass prior iterations are included into the HM, while the inferior ones are removed.
- Repetition operations: For the specified number of iterations, we generate new harmony and update HM steps, culminating in the HPO by selecting the optimal harmony from the final HM.

In order to find the optimal hyperparameter, Lee et al. [230] suggested adjusting the layer's feature map size during CNN feature extraction using the parameter-setting-free HS methodology. They begin their method by setting up the PAR, HMCR, and HMS. In the next step, they adjust the CNN's hyperparameters to achieve harmony. One way to construct the first HM is to determine the CNN loss for every harmonic. As many times as HMS is needed to construct an initial HM, this technique is done. To teach the CNN, they use a gradient descent method with a cross-entropy function serving as the loss function. For HMCR and PAR, they tweak the equations using the sigmoid function. For the convolutional layer, they dictate the kernel size and stride; for the pooling layer, they dictate the kernel size and stride; and for the overall number of channels, they serve as harmonies. Using the MNIST and the CIFAR-10 datasets, they assess how well their suggested technique performs.

Huang et al. conducted another study, proposing a self-adaptive HS method for CNN optimization [232]. They break down the system architecture into two parts in their technique. The first step is to find the optimal CNN layer length and then to produce CNN candidates using a self-adaptive harmony search (SAHS) method. The SAHS algorithm terminates when it reaches the maximum number of iterations. After completing all feasible iterations, the system advances to phase 2. In phase 2, they generate various hyperparameter values using the SAHS method. After running through all possible iterations, the algorithm produces an optimal CNN. Their suggested solution was tested on well-known datasets such as MNIST, CIFAR-10, and Caltech-101. HMCR, PAR, and *hms* are SAHS algorithm parameters that are fine-tuned for each of the three datasets in order to get the best possible results.

6.7 Sine Cosine Algorithm (SCA)

Mirjalli proposed an analytical model that makes use of the sine and cosine functions, and this led to the development of the sine cosine algorithm (SCA) approach [233]. This technique emphasizes the exploration and exploitation of the search space at different optimization phases by using several random and customized factors. Research demonstrates that the SCA can navigate through a search space, identify intriguing regions for investigation, steer clear of local optima, and ultimately reach the global optimum. During the exploration phase, an optimization algorithm finds potentially intriguing parts of the search space by suddenly blending the random solutions into a collection of solutions with a high degree of randomness. The random solutions do change somewhat during exploitation, but the random variations are far less than they were during exploration. SCA uses the following equations to update positions during the exploration and exploitation phase:

$$X_i^{t+1} = X_i^t + r_1.\sin(r_2).|r_3P_i^t - X_i^t| \tag{34}$$

$$X_i^{t+1} = X_i^t + r_1.\cos(r_2).|r_3P_i^t - X_i^t| \tag{35}$$

P_i^t denotes the position of the ith dimension, r_1, r_2, and r_3 range from 0 to 1, and X_i^t represents the current solution location in the *ith* dimension at the tth iteration. The absolute value (positive value) will only be used for the transformation $r_3P_i^t - X_i^t$. Here's how to combine the two equations:

$$X_i^{t+1} = \begin{cases} X_i^t + r_1.\sin(r_2).\left|r_3P_i^t - X_i^t\right|, r_4 \le 0.5 \\ X_i^t + r_1.\cos(r_2).\left|r_3P_i^t - X_i^t\right|, r_4 > 0.5 \end{cases} \tag{36}$$

where r_4 denotes a random variable within the interval [0, 1]. Equation (36) illustrates that the SCA employs four primary parameters, which govern either the position or the destination. Consequently, the search process achieves equilibrium between alternatives for effective coordination toward the optimal outcome. The periodic characteristics of sine and cosine functions guarantee the use of the region delineated between two observed solutions. The solutions have the ability to expand their search beyond the boundaries of their respective objectives, thereby exploring the entire search space. Modifications to the ranges of the sine and cosine functions achieve this. To ensure comprehensive exploration of the search region, the range adjustments of sine and cosine require solutions to revise their positions beyond the confines of the space between each solution. Randomness is achieved by selecting an integer randomly for r_2 in Eq. (36), which is between 0 and 2π. To achieve balance between the exploration phase and the exploitation phase, the SCA employs the following equation:

$$r_1 = a - t\frac{a}{T} \tag{37}$$

where T is the maximum number of rounds, t is the current round, and a is a constant integer. However, testing the original SCA using basic CEC unconstrained benchmarks can further explore the metaheuristics. Initially, the algorithm may lack sufficient exploration power, trapping it in less-than-ideal areas of the search space. The quality of the solutions will suffer as a consequence of this flaw. Basic SCA incorporates a basic exploration mechanism to tackle this problem. In each cycle, the expression replaces the worst solution from the population with an arbitrary individual within the search space's bounds.

$$X^j_{rnd} = L^j + \varnothing.(U^j - L^j) \tag{38}$$

where X^j_{rnd} is the jth component of the recently generated random solution, *phi* represents the value obtained from the uniform distribution, and U^j and L^j denotes the upper and lower bounds of the jth parameter, respectively. The adjustments enhance the computational complexity of the proposed algorithm relative to the basic version, yet the performance benefits of the suggested approach over the original SCA warrant this increase.

By leveraging the eSCA metaheuristics as an optimizer, Bacanin, Nebojsa, et al. [234] brought a new approach to CNN hyperparameter optimization. The paper focuses on finding the right architecture and hyperparameters because every problem requires training the convolutional neural network (CNN). The suggested eSCA-CNN method has shown encouraging outcomes in its attempt to achieve a more effective CNN structure. Using the CIFAR-10 dataset for tests proved that eSCA-CNN could be used to automatically choose the best CNN structures with no human oversight.

6.8 Differential Evolution (DE)

Storn and Prince presented differential evolution (DE) in 1997 [235], a meta-heuristic approach for population-based search. The DE algorithm is simple to use, adaptable, and multipurpose [236]. Similar to the genetic algorithm, the DE algorithm employs operators for initial population formation, mutation, crossover, and selection. The main distinction is that genetic algorithms rely on crossover and differential evolution for mutations [237]. The three primary procedures of the DE algorithm are outlined below [238]: Mutation, crossover, and selection operators, as well as Np real-coded vectors, make up the DE algorithm. The Np real-coded vectors here represent the possible solutions. The following equation depicts all possible solutions, where each element of the solution is in the interval $X^{(t)}_{i,1} \in [X^{(L)}_i, X^{(U)}_i]$, with $X^{(L)}_i$ representing the lower bound and $X^{(U)}_i$ the upper bound of the *ith* variable.

$$X^{(t)}_i = \left(X^{(t)}_{r1}, X^{(t)}_{r2}, \cdots, X^{(t)}_{r3}\right), fori = 1, \cdots, N_p \tag{39}$$

Creating a mutation vector for every target vector is the next stage in the mutation process. The following equation represents the mutation operation:

$$U^{(t)}_i = X^{(t)}_{r1} + F.\left(X^{(t)}_{r2} - X^{(t)}_{r3}\right), fori = 1, \cdots, N_p \tag{40}$$

The variables $r1$, $r2$, and $r3$ are chosen at random from the range $(1, \cdots, N_p)$, and F is the scaling factor that controls the pace of change.

$$X^{t+1}_i = \begin{cases} U^{(t)}_{i,j} rand_j(0,1) \le CR \vee j = j_{rand} \\ \quad X^{(t)}_{r,j} \quad otherwise \end{cases} \tag{41}$$

The crossover probability rate, denoted as $CR \in [0,1]$, controls the proportion of parameters transferred to the trial solution. The last step, selection, checks whether the generated vector meets the greedy requirement and continues with creation if so. One might formulate the choice as follows:

$$X^{t+1}_i = \begin{cases} W^{(t)}_i \quad if \quad f\left(W^{(t)}_i\right) \le f\left(X^{(t)}_i\right) \\ \quad X^{(t)}_i \quad otherwise \end{cases} \tag{42}$$

The fitness function is represented by $f\left(X^{(t)}_i\right)$. If you want the DE algorithm to work properly, you have to specify the parameters N_p, F, and CR appropriately. The population size is denoted by Np in this case. In equation [239], F is the differential weight that indicates the distance from point X_i that the offspring should be formed. According to [236], CR is the crossover probability rate that controls the number of expectation variables changed in a population member. Through the use of CNN transfer learning with HPO based on DE, Vili et al. [238] created a system for image-based sports category classification. They found the optimal outcome by comparing their suggested CNN model to traditional CNN models. For the model, the DE algorithm optimized a number of hyperparameters. To get the best outcome, plenty of parameters are tweaked, including batch size, epochs, issue dimension, population size, scaling factor, crossover rate, learning rate, optimizer, and dropout.

More recently, Ghasemi et al. [240] improved the traditional DE technique's exploration capabilities for optimization issues by developing a self-competitive DE method. The study reveals that DE algorithms work better than their older versions when they use a competitive control parameter and a new mutation mechanism. These changes make it easier to tune a PID controller in an AVR system and solve real-world optimization problems.

6.9 Genetic Algorithm (GA)

One computing model that mimics the genetic mechanics and natural selection underlying Darwin's theory of biological evolution is the genetic algorithm (GA) [241]. Every iteration maintains a set of potential solutions by mimicking the natural processes of mutation, crossover, and reproduction [242]. Regardless of the domain or nature of the challenge, genetic algorithms provide a generic framework for solving complex optimization problems in systems. The coding mechanism, fitness function, genetic operator (selection, crossover, and mutation), and control parameters are the major parts of a genetic algorithm. The stages of the genetic algorithm are described below [242]:

– At random, generate a population (pop) of N chromosomes with a defined length and choose thiopropazate coding strategy for the provided problem.

$$pop_i(t), t = 1, i = 1,2, \cdots, N \tag{43}$$

– –Compute the fitness value of each chromosome $Tpop(t)$ inside the pop(t) population (t).

$$f_i = fitness(pop_i(t)) \tag{44}$$

– Assess if the convergence criteria are satisfied. If the output search results are satisfactory, go to the next stage.
– Another important aspect of genetic algorithms is the selection operation, which uses individual fitness to determine the probability of selection.

$$P_i = \frac{f_i}{\sum_{i=1}^{N}}, i = 1,2, \cdots, N \tag{45}$$

– The next generation of the $Tpop(t)$ population is formed by randomly selecting the aforementioned distribution of probabilities from the existing population.

$$newpop(t+1) = \{pop_j(t) | j = 1,2, \cdots, N\} \tag{46}$$

– The crossover procedure generated a set of chromosomes known as $tcrosspop(t+1)$ based on the matching probability Pc.
– Lastly, use a reduced probability Pm in the mutation operation to stimulate chromosomal gene mutations. A new population, denoted as $tmutpop(t+1)$, is formed. We call $Pop(t) = mutpop(t+1)$, he most recent population to emerge from a genetic operation. Furthermore, it represents the ancestor of the genetic operation, dating back to the second generation.

One common use of genetic algorithms is the efficient optimization of convolutional neural network (CNN) hyperparameters. Using hyperparameters and network configurations as search spaces, recent research [1, 17–54] used GA to discover the perfect structure for a given input and found the best network design. The research also takes into account the activation function and model layers as hyperparameters for optimization. The technique is built by encoding the connections between convolutional layers in a binary form. The optimization method and activation function are separated by bars and encoded into bits. To determine how many bits are required for the n-hyperparameters, one may use the following formula:

$$number\ of\ bits = \log_2(n) + 1 \tag{47}$$

The initialization process involves creating a binary-encoded population of convolutional neural network (CNN) models. Here, a Bernoulli distribution is used to separately examine each chromosomal bit. The algorithm is programmed to repeatedly establish the ideal fitness score at the beginning of the operation. Following the process of natural selection, the surviving chromosomes undergo a uniform crossover to generate new individuals. Finally, the mutation affects the chromosomal survivors. The construction of optimum network topologies resulted in a mean classification accuracy of 81.74%.

6.10 Bare Bones Fireworks Algorithm (BBFWA)

In 2010, Tan and Zhu were the ones who first suggested the fireworks algorithm [243]. The algorithm's popularity led to constant refinement and improvement. Regular intervals saw the release of newer, better versions, such as the bare-bones fireworks algorithm (BBFWA). In 2018, Li and Tan put forward the BBFWA [244]. The Bare Bones Fireworks Algorithm (BBFWA) is a streamlined variant of the original Fireworks Algorithm (FWA). Below are a few critical points of comparison [243, 244]:

– Simplicity: The BBFWA preserves just the fundamental explosive action from the original FWA, thereby enhanc-

ing its simplicity and ease of implementation. This simple methodology facilitates expedited execution and reduces code complexity.
- Efficiency: The design of the BBFWA prioritizes rapidity and efficacy. Experimental findings demonstrate that it exhibits competitive performance on benchmark functions and real-world challenges, highlighting its efficacy despite its simplicity.
- The convergence: The BBFWA establishes adequate conditions for local convergence, essential for the algorithm's efficacy in identifying optimum solutions.
- Exploration vs. Exploitation: In the original FWA, the formation of explosive sparks maintains a balance between exploration (investigating new territories) and exploitation (enhancing familiar advantageous locations). The BBFWA streamlines this process by concentrating mostly on the explosion operation, which might influence the equilibrium between exploration and exploitation.
- Performance: Although the original FWA incorporates other operators (explosion, mutation, and selection), the BBFWA's performance remains competitive, suggesting that the fundamental explosion operation may be sufficient for numerous optimization jobs.

The Bare Bones Fireworks (BBFWA) algorithm offers several advantages, especially in the context of tuning hyperparameters for convolutional neural networks (CNNs). We will explain some of these important advantages as follows:

- Efficient fitness function evaluations: The Bare Bones Fireworks algorithm significantly reduces the number of fitness function evaluations required to find the optimal hyperparameters. This is important because fitness function evaluation can be computationally expensive, especially in deep learning scenarios where training a CNN can be time-consuming.
- Improved classification accuracy: The Bare Bones Fireworks algorithm has shown an increase in classification accuracy on benchmark datasets. For example, the accuracy on the MNIST dataset improved from 99.25 to 99.34%, and on the CIFAR-10 dataset from 74.76 to 75.51%. This improvement indicates that the algorithm is tuning hyperparameters effectively to improve model performance.
- Solution stability: The algorithm allows for the identification of stable solutions over multiple epochs. The researchers found that after a certain number of epochs, the best solutions tend to stabilize, meaning that BBFWA can efficiently navigate the search space and find optimal hyperparameters that give consistent results.
- Flexibility in choosing hyperparameters: Researchers can modify BBFWA to adjust different hyperparameters for CNNs, making it a versatile tool for different architectures and datasets. This flexibility allows researchers and practitioners to apply the algorithm across different applications in image classification and beyond.

Tuba, Ira, et al. [245] introduced a simple modified fireworks technique to identify a specific subset of CNN hyperparameters and evaluated it on the MNIST and CIFAR-10 datasets. In comparison to the harmony search algorithm (HS technique) documented by Lee et al. in 2018 [230], the suggested method enhanced classification accuracy for both datasets while diminishing network complexity. The suggested BBFWA methodology requires many fewer assessments of the fitness function. The classification accuracy improved from 99.25 to 99.34% on the MNIST dataset and from 74.76 to 75.51% on the CIFAR-10 dataset.

These algorithms have been widely utilized for optimizing convolutional neural network (CNN) hyperparameters due to their balance of exploration and exploitation. Table 8 Summary of metaheuristic optimization algorithms used for CNN hyperparameter tuning, detailing initialization, processing steps, and termination conditions.

7 Convolutional Neural Network Architectures Optimized Via Metaheuristic Algorithms

One of the most important problems of deep learning (DL) is efficient fine-tuning of algorithm parameters. Convolutional Neural Networks (CNNs) with superior performance, in turn, usually require careful weight, hyperparameter, network structure, activation function, and learning algorithm tuning. Metaheuristic algorithms are excellent optimization methods that have been shown to optimize these factors for CNN performance improvement [246]. It has been demonstrated by various studies that CNNs can be improved with metaheuristic techniques. Hon et al. [247] used Genetic Algorithm (GA) to fine-tune CNN's parameters, and the result was a more accurate and error-free power price prediction. Reducing computation time and improving the performance of image classification tasks can be achieved through optimizing CNNs' hyperparameters with the Distributed Particle Swarm Optimization (DPSO) approach as suggested by Guo et al. [248].

The weights and biases of CNNs were optimized for intrusion detection in IoT networks by Kan et al. [249] with the Adaptive Particle Swarm Optimization (APSO). The recently introduced APSO-CNN model outperformed and worked well compared to the traditional methods. Likewise, Kanna and Santhi [250] optimised the weight using Black Widow Optimization (BWO) in CNN-LSTM, which significantly enhanced intrusion detection compared to baseline models. In contrast to previous works, Ragab

Table 8 Summary of metaheuristic optimization algorithms used for CNN hyperparameter tuning, detailing initialization, processing steps, and termination conditions

Algorithm	Initialization	Processing Steps	Termination Condition
Particle Swarm Optimization (PSO)	Initialize swarm size, positions, and velocities randomly	Evaluate positions → Update personal bests → Identify global best → Update velocities and positions → Repeat until stopping criterion is met	Maximum iterations or desired error level achieved
Ant Colony Optimization (ACO)	Initialize ants, pheromone trails, and relevant parameters	Ants explore CNN hyperparameter space → Select next move based on pheromone and heuristic → Update pheromones → Generate new ants → Increase search depth	Best ant determined by pheromone value
Firefly Algorithm	Randomly place fireflies within the CNN hyperparameter space	Evaluate solutions → Identify best solution → Sort by fitness → Execute exploration and exploitation → Update positions	Upon reaching maximum iterations
Whale Optimization Algorithm (WOA)	Randomly initialize whale population in solution space	Evaluate fitness → Identify the best whale → Update positions based on leader behavior → Adjust exploration/exploitation coefficients	Maximum iterations or convergence achieved
Gray Wolf Optimization (GWO)	Randomly initialize wolves and define boundaries	Evaluate fitness → Rank wolves (alpha, beta, delta) → Update positions → Balance exploration and exploitation over iterations	Best position found by alpha wolf
Harmony Search Algorithm (HSA)	Set parameters: Harmony Memory Size (HMS), HMCR, PAR	Initialize harmony memory → Generate harmony vectors based on CNN hyperparameters → Update HMCR and PAR → Replace worst solutions if new ones are better	After all iterations, return the best harmony
Sine Cosine Algorithm (SCA)	Generate random initial solutions	Explore using random solutions and sine/cosine functions → Exploit by fine-tuning around best solutions → Update positions	Maximum iterations, accuracy, or evaluation limit met
Differential Evolution (DE)	Randomly initialize parameter vectors	For each target vector: Generate mutant → Perform crossover → Select better solution → Repeat	Best fitness value reached
Genetic Algorithm (GA)	Randomly generate chromosomes (CNN models)	Evaluate fitness → Select parents → Apply crossover and mutation → Form next generation → Repeat	Predefined generations, fitness threshold, or convergence
Bare Bones Fireworks Algorithm (BBFWA)	Randomly initialize fireworks as candidate solutions	Generate sparks around good solutions for exploitation → Use a broader range for worse solutions → Select the best among the fireworks and sparks → Repeat	Satisfactory solution or iteration limit

Table 9 A summary of the developments in the methodology of mathematical algorithms for training and optimizing CNN architectures

Approach	Year	Optimized Hyperparameters	Dataset	Application
Hon et al. [247] Genetic algorithm (GA)	2020	Number of Convolutions, Filter Size for Layer 1, Kernel Size, Pool Size, Filter Size for Layer 2 and Layer 3, and Dropout Size	LMP spatiotemporal data were utilized	A technique for forecasting electricity prices over a 24-h period
Guo et al. [248] Distributed particle swarm optimization (DPSO)	2020	Convolution Kernel Size and Number, Pooling Type, Activation Functions, Number of Neurons, Dropout Rate, Learning Rate	MNIST is a popular image classification dataset	Optimizing CNN Hyperparameters
Kan et al. [249] Adaptive particle swarm optimization (APSO)	2021	Inertia weight factor	Public IoT data from nine devices	In the field of network security, reliably identify many types of Internet of Things (IoT) devices
Kanna and Santhi [250] Black widow optimization (BWO)	2022	Weights	NSL-KDD, ISCX-IDS, UNSW-NB15, and CSE-CIC-IDS2018	Intrusion Detection Systems
Ragab et al. [251] Enhanced gravitational search optimization (EGSO)	2022	The research does not define optimized hyperparameters, although popular ones in deep learning models like HCNN-GRUF include learning rate, layer and unit count, batch size, and dropout rate	Epidemiology data from Kaggle repository	Diagnosis of COVID-19
Adamu, Shamsuddeen, et al.[252] Manta Rays Foraging Optimizer (MRFO)	2023	Rotation angle, shifts (height and width), zoom level, shear transformation, flipping, loss function, training batch size, model dropout ratio, transfer learning ratio, and rotation angle	ISIC 2019 Dataset	Melanoma classification
Aguerchi, Khadija, et al. [253] Particle Swarm Optimization algorithm (PSO)	2024	stride, filter size (convolution layer), and filter number	DDSM and MIAS datasets	Detection of breast cancer through mammography
Purnomo et al. [254] (ACO), (GA), and (HS)	2024	Number of neurons in the first and second dense layers, first and second leakage layer values, and the number of batches. The type of activation function, the type of optimizer algorithm, and the type of loss function	MNIST	Optimizing CNN Hyperparameters
Nolia et al. [255] (GA), (PSO), Evolutionary Strategy (ES)	2024	Weights	Fruits360, and MNIST	Optimizing CNN Hyperparameters
Taji, et al. [46] (BDA), (ACO), Moth Flame Optimization algorithm (MFO)	2024	Pheromone Evaporation Rate, Pheromone Influence, Degree of Exploration, Feature Vector Size, Hybrid Feature Set, Optimization Algorithms	Plant Village	Classifying plant leaf diseases
Jovanovic et al. [259] Modified PSO + XGBoost/AdaBoost	2024	CNN architecture, XGBoost/AdaBoost parameters, spectrogram feature extraction	Clinical respiratory audio (mel spectrograms)	Binary/multi-class respiratory condition classification
Jovanovic et al. [260] Modified Crayfish (MSCHO) + CNN	2024	CNN architecture, sensor feature extraction, gait pattern thresholds	Wearable gyroscope data (Parkinson's patients vs control)	Parkinson's gait anomaly detection

Table 9 (continued)

Approach	Year	Optimized Hyperparameters	Dataset	Application
Salb et al. [261] Reptile Search Algorithm (RSA) + CNN-XGBoost	2024	CNN feature extraction layers, XGBoost hyperparameters	Public IoT intrusion dataset	Binary/multi-class intrusion detection
Al-Jahni and Mansoura et al. [256] Manta-ray foraging algorithm (MRFO)	2025	weights	Brian Tumor Dataset, BT Image Dataset	Classify brain tumour instances effectively
Papalkar et al. [257] The hybrid metaheuristic algorithm (WACSO), which combines (CSO) with (GWO)	2025	Number of filters in convolutional layers, learning rate, batch size, number of epochs, and dropout rate	MNIST, CIFAR-10	Optimizing CNN Hyperparameters
Gonzalez et al.[258] (GA), (PSO)	2025	Number of convolutional layers, filter size per layer, number of filters per layer, batch size	Alzheimer MRI Dataset	Classification of Alzheimer's disease
Ashraf et al. [262] Ant Colony Optimization (ACO) with NeuroNet57	2025	Feature subset selection for dimensionality reduction, CNN architecture configuration	ABIDE-I + II fMRI datasets (14,372 × 4096 and 16,168 × 4096 features) Pre-trained on the Brain Tumor dataset	Female autism spectrum disorder (ASD) classification from control behavior (92.21–93.49% accuracy)

et al. [251] achieved better performance when optimizing the parameters of CNN for COVID-19 diagnosis employing the Enhanced Gravitational Search Optimization (EGSO) strategy. Adamu et al. [252] used melanoma classification with DenseNet121-based CNN optimized using the Manta Ray Foraging Optimization (MRFO) method. Their method significantly enhanced the performance of the model in dermatology image classification.

Aguerchi et al. [253] introduced a PSO-based approach for CNN hyperparameter tuning to identify breast cancer through mammography images. Purnomo et al. [254] tested three metaheuristic algorithms—Ant Colony Optimization (ACO), GA, and Harmony Search (HS)—for CNN hyperparameter tuning, and they showed that ACO provided the best classification accuracy on the MNIST dataset. Nolia et al. [255] introduced novel metaheuristic methods for CNN weight optimization with a resulting 85.27% boost in accuracy while reducing computational complexity. Taji et al. [46] constructed a hybrid ensemble system that includes CNN-feature extracted features along with metaheuristic-optimized feature engineering using the algorithms of BDA, ACO, and MFO for a resulting accuracy of 99.8% in plant disease classification. Al-Jahni and Mansoura et al. [256] used the MRFO algorithm to optimize the topology of CNNs for MRI and X-ray image-based brain tumor detection. Their model's accuracy on MRI and X-ray data was 98.64 and 99.96%, respectively.

CNN accuracy in classification over unseen MNIST and CIFAR-10 datasets was greatly enhanced by the hybrid metaheuristic WACSO, introduced by Papalkar et al. [257], formed by the union of Crow Search Optimization (CSO) and Grey Wolf Optimizer (GWO). Gonzalez et al. [258] investigated GA and PSO for CNN optimization with a view to Alzheimer's disease classification with higher accuracy and lesser parameter tuning. A comprehensive summary of these studies and their contributions to CNN optimization is presented in Table 9.

8 Conclusions

Deep learning remains a force behind innovation in various fields due to its remarkable feature extraction and classification capabilities, achieved especially through convolutional neural networks (CNNs). The efficacy of CNNs is significantly affected by the fine-tuning of hyperparameters like layer depth, filter size, learning rate, and batch size. This work acknowledges the growing use of metaheuristic algorithms as efficient, general-purpose solutions to optimize CNN architectures. In comparison to exhaustive techniques like grid search, which are generally computationally infeasible, metaheuristic methods achieve a good balance between exploration and exploitation and thus facilitate efficient search in high-dimensional spaces at much less computational expense. By conducting a comprehensive survey of existing literature, we noted that a broad range of metaheuristic algorithms—from traditional techniques such as Particle Swarm Optimization and Genetic Algorithms to emerging or hybrid variants such as the Manta Ray Foraging Optimizer and Wolf-Crow Search—routinely improve CNN performance in numerous fields. These fields stretch from medical diagnosis to image classification, plant disease detection, and cybersecurity. In light of these observations, we suggest that future research concentrate on the formulation and enhancement of hybrid metaheuristic models with problem-specific requirements. Furthermore, testing of these algorithms on larger real-world data sets will determine their applicability and generalizability in real-world problems. Future research directions in this field include designing adaptive metaheuristic algorithms that are capable of changing their strategy dynamically during training and merging them with new paradigms such as transformer-based architecture and Neural Architecture Search (NAS). Another focal area is energy-efficient optimization algorithms that can be efficiently applied to real-time scenarios and on edge devices with limited resources. Solving these challenges will set the stage for deeper, smarter, more automated, and greener deep learning systems that are not just powerful but also practical for real-world applications.

Author Contributions M.Q.I. and N.K.H. conceptualized the study and performed the literature review. M.Q.I. wrote the main manuscript text with supervision from N.K.H. D.G. and M.Q. provided critical feedback, edited the manuscript, and contributed to the theoretical framework. D.G. and M.Q. supervised the overall project and finalized the manuscript. All authors reviewed and approved the final version of the manuscript.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Lecun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444. https://doi.org/10.1038/nature14539
2. Dongare AD, Kharde RR, Kachare AD (2012) Introduction to artificial neural network. Int J Eng Innov Technol 2:189–194
3. Gaspar A, Oliva D, Cuevas E et al (2021) Hyperparameter optimization in a convolutional neural network using metaheuristic algorithms. Metaheuristics in machine learning: theory and applications. Springer, pp 37–59
4. Aihara K, Takabe T, Toyoda M (1990) Chaotic neural networks. Phys Lett A 144:333–340
5. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86:2278–2324
6. Specht DF (1991) A general regression neural network. IEEE Trans Neural Networks 2:568–576
7. Galanis N-I, Vafiadis P, Mirzaev K-G, Papakostas GA (2022) Convolutional neural networks: a roundup and benchmark of their pooling layer variants. Algorithms 15:391
8. Jie HJ, Wanda P (2020) RunPool: a dynamic pooling layer for convolution neural network. Int J Comput Intell Syst 13:66–76
9. Zeiler MD, Fergus R (2014) Visualizing and understanding convolutional networks. In: Computer vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, Sept 6–12, 2014, Proceedings, Part I 13. Springer, pp 818–833
10. Yang C, Yang Z, Hou J, Su Y (2021) A lightweight full homomorphic encryption scheme on fully-connected layer for CNN hardware accelerator achieving security inference. 2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS). IEEE, pp 1–4
11. Sharma S, Sharma S, Athaiya A (2017) Activation functions in neural networks. Towar Data Sci 6:310–316
12. Koutsoukas A, Monaghan KJ, Li X, Huan J (2017) Deep-learning: investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data. J Cheminform 9:1–13
13. Snoek J, Larochelle H, Adams RP (2012) Practical bayesian optimization of machine learning algorithms. Adv Neural Inf Process Syst 25
14. Injadat M, Moubayed A, Nassif AB, Shami A (2020) Systematic ensemble model selection approach for educational data mining. Knowl-Based Syst 200:105992
15. Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13(1):281–305
16. Bengio Y (2000) Gradient-based optimization of hyperparameters. Neural Comput 12:1889–1900
17. Gogna A, Tayal A (2013) Metaheuristics: review and application. J Exp Theor Artif Intell 25:503–526
18. Lee W, Park S, Optik KS (2018) Optimal hyperparameter tuning of convolutional neural networks based on the parameter-setting-free harmony search algorithm. Elsevier
19. Acoustics ÖI-A (2023) CNN hyper-parameter optimization for environmental sound classification. Elsevier
20. Mirjalili S (2019) Genetic algorithm. Stud Comput Intell 780:43–55. https://doi.org/10.1007/978-3-319-93025-1_4
21. Wang D, Tan D, Liu L (2018) Particle swarm optimization algorithm: an overview. Soft Comput 22:387–408. https://doi.org/10.1007/S00500-016-2474-6
22. Dorigo M, Birattari M, Stutzle T (2007) Ant colony optimization. IEEE Comput Intell Mag 1(4):28–39
23. Pelikan M, Pelikan M (2005) Bayesian optimization algorithm. Hierarchical Bayesian optimization algorithm:toward a new generation of evolutionary algorithms, pp 31–48. https://doi.org/10.1007/978-3-540-32373-0_3
24. Li Z, Liu F, Yang W et al (2021). A survey of convolutional neural networks: analysis, applications, and prospects. IEEE Trans Neural Netw Learn Syst 33(12):6999–7019
25. Khalid R, Javaid N (2020) A survey on hyperparameters optimization algorithms of forecasting models in smart grid. Sustain Cities Soc 61:102275
26. Han F, Jiang J, Ling Q, et al (2019) A survey on metaheuristic optimization for random single-hidden layer feedforward neural network. Neurocomputing 335:261–273
27. Nematzadeh S, Kiani F, Torkamanian-Afshar M, Aydin N (2022) Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: a

bioinformatics study on biomedical andbiological cases. Comput Biol Chem 97:107619
28. Darwish A, Hassanien AE, Das S (2020) A survey of swarm and evolutionary computing approaches for deep learning. Artif Intell Rev 53:1767–1812. https://doi.org/10.1007/S10462-019-09719-2
29. Firat Kilincer I, Ertam F, Sengur A et al (2023) Automated detection of cybersecurity attacks in healthcare systems with recursive feature elimination and multilayer perceptron optimization. Biocybern Biomed Eng 43:30–41. https://doi.org/10.1016/J.BBE.2022.11.005
30. Muhajir D, Akbar M, Bagaskara A, Vinarti R (2022) Improving classification algorithm on education dataset using hyperparameter tuning. Procedia Comput Sci 197:538–544
31. Holly S, Hiessl T, Lakani SR et al (2022) Evaluation of hyperparameter-optimization approaches in an industrial federated learning system. Data Sci—Anal Appl. https://doi.org/10.1007/978-3-658-36295-9_1
32. Hoque KE, Aljamaan H (2021) Impact of hyperparameter tuning on machine learning models in stock price forecasting. IEEE Access 9:163815–163830. https://doi.org/10.1109/ACCESS.2021.3134138
33. Erkan U, Toktas A, Ustun D (2023) Hyperparameter optimization of deep CNN classifier for plant species identification using artificial bee colony algorithm. J Ambient Intell Humaniz Comput 14:8827–8838. https://doi.org/10.1007/S12652-021-03631-W
34. Irmawati CR, Basari GD (2022) Optimizing CNN hyperparameters for blastocyst quality assessment in small datasets. IEEE Access 10:88621–88631. https://doi.org/10.1109/ACCESS.2022.3196647
35. Leng X, Amidi E, Uddin KMS, Chapman Jr WC, Luo H, Kou S, Zhu Q (2021) Assessing rectal cancer treatment response using photoacoustic microscopy: deep learning CNN outperforms supervised machine learning model. In: Photons Plus Ultrasound: Imaging and Sensing 2021, Vol. 11642, p. 116420S, SPIE. https://doi.org/10.1117/12.2578017
36. Pranolo A, Mao Y, Wibawa AP et al (2023) Optimized three deep learning models based-PSO hyperparameters for Beijing PM2.5 prediction. Knowl Eng Data Sci 5:53. https://doi.org/10.17977/um018v5i12022p53-66
37. Belete DM, Huchaiah MD (2022) Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results. Int J Comput Appl 44:875–886. https://doi.org/10.1080/1206212X.2021.1974663
38. Du X, Xu H, Zhu F (2021) Understanding the effect of hyperparameter optimization on machine learning models for structure design problems. Comput Des 135:103013. https://doi.org/10.1016/J.CAD.2021.103013
39. Kiziloluk S, Sert E (2022) COVID-CCD-Net: COVID-19 and colon cancer diagnosis system with optimized CNN hyperparameters using gradient-based optimizer. Med Biol Eng Comput 60:1595–1612. https://doi.org/10.1007/S11517-022-02553-9/TABLES/7
40. Gonçalves CB, Souza JR, Fernandes H (2022) CNN architecture optimization using bio-inspired algorithms for breast cancer detection in infrared images. Comput Biol Med 142:105205. https://doi.org/10.1016/J.COMPBIOMED.2021.105205
41. Kilichev D, Kim W (2023) Hyperparameter optimization for 1D-CNN-based network intrusion detection using GA and PSO. Math 11:3724. https://doi.org/10.3390/MATH11173724
42. Kaveh M, Mesgari MS (2023) Application of meta-heuristic algorithms for training neural networks and deep learning architectures: a comprehensive review. Neural Process Lett 55:4519–4622. https://doi.org/10.1007/S11063-022-11055-6/TABLES/5
43. Raiaan MAK, Sakib S, Fahad NM et al (2024) A systematic review of hyperparameter optimization techniques in convolutional neural networks. Decis Anal J 11:100470. https://doi.org/10.1016/J.DAJOUR.2024.100470
44. Veeranjaneyulu K, Lakshmi M, Janakiraman S (2024) Swarm intelligent metaheuristic optimization algorithms-based artificial neural network models for breast cancer diagnosis: emerging trends, challenges and future research directions. Arch Comput Methods Eng 32:381–398. https://doi.org/10.1007/S11831-024-10142-2/METRICS
45. Brahim DB, Kobayashi M, Al AM et al (2024) Metaheuristic optimization algorithms: an overview. HCMCOU J Sci Adv Comput Struct 14:34–62. https://doi.org/10.46223/HCMCOUJS.ACS.EN.14.1.47.2024
46. Taji K, Sohail A, Shahzad T et al (2024) An ensemble hybrid framework: a comparative analysis of metaheuristic algorithms for ensemble hybrid CNN features for plants disease classification. IEEE Access 12:61886–61906. https://doi.org/10.1109/ACCESS.2024.3389648
47. Latif G, Alghazo J, Khan MA et al (2024) (2024) Deep convolutional neural network (CNN) model optimization techniques—review for medical imaging. AIMS Math 9:20539–20571. https://doi.org/10.3934/MATH.2024998
48. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436-444
49. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. Adv Neural Inf Process Syst 25
50. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXivpreprint arXiv:1409.1556
51. Vaswani A, Brain G, Shazeer N, et al (2017) Attention is all you need. Adv Neural Inf Process Syst 30
52. Brown TB, Mann B, Ryder N et al (2020) Language models are few-shot learners. Adv Neural Inf Process Syst 33:1877–1901
53. Herculano-Houzel S (2009) The human brain in numbers: a linearly scaled-up primate brain. Front Hum Neurosci 3:857. https://doi.org/10.3389/NEURO.09.031.2009/BIBTEX
54. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nat 323:533–536. https://doi.org/10.1038/323533a0
55. Apicella A, Donnarumma F, Isgrò F, Prevete R (2021) A survey on modern trainable activation functions. Neural Netw 138:14–32
56. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9:1735–1780. https://doi.org/10.1162/NECO.1997.9.8.1735
57. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86:2278–2323. https://doi.org/10.1109/5.726791
58. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp 807-814
59. Maas AL, Hannun AY, Ng AY (2013) Rectifier nonlinearities improve neural network acoustic models. Proc icml 30(1):3
60. He K, Zhang X, Ren S, et al (2015) Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision, pp 1026–1034
61. Han J, Moraga C (1995) The influence of the sigmoid function parameters on the speed of backpropagation learning. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics) 930:195–201. https://doi.org/10.1007/3-540-59497-3_175
62. LeCun Y, Bottou L, Orr GB, Müller K-R (2002) Efficient backprop. In Neural networks: Tricks of the trade (pp. 9–50). Springer, Berlin. https://doi.org/10.1007/3-540-49430-8_2

63. Rawat W, Wang Z (2017) Deep convolutional neural networks for image classification: a comprehensive review. Neural Comput 29:2352–2449
64. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65:386
65. Waibel A, Hanazawa T, Hinton G et al (2013) Phoneme recognition using time-delay neural networks. Backpropagation. Psychology Press, pp 35–61
66. LeCun Y, Boser B, Denker JS et al (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1:541–551
67. Ajmal H, Rehman S, Farooq U et al (2018) Convolutional neural network based image segmentation: a review. Pattern Recognit Track XXIX 10649:191–203
68. Hinton GE (2012) Improving neural networks by preventing co-adaptation of feature detectors. arXiv Prepr arXiv12070580
69. Wang G, Gong J (2019) Facial expression recognition based on improved LeNet-5 CNN. In; Proceedings of the 31st Chinese Control Decision Conference CCDC 2019, pp 5655–5660. https://doi.org/10.1109/CCDC.2019.8832535
70. You H, Yu L, Tian S et al (2021) MC-Net: multiple max-pooling integration module and cross multi-scale deconvolution network. Knowl-Based Syst 231:107456. https://doi.org/10.1016/J.KNOSYS.2021.107456
71. Lu C, Dubbelman G (2020) Semantic foreground inpainting from weak supervision. IEEE Robot Autom Lett 5:1334–1341. https://doi.org/10.1109/LRA.2020.2967712
72. de Souza BA, Vieira MB, de Andrade MLSC et al (2021) Combining max-pooling and wavelet pooling strategies for semantic image segmentation. Expert Syst Appl 183:115403. https://doi.org/10.1016/J.ESWA.2021.115403
73. Wang S, Jiang Y, Hou X et al (2017) Cerebral micro-bleed detection based on the convolution neural network with rank based average pooling. IEEE Access 5:16576–16583. https://doi.org/10.1109/ACCESS.2017.2736558
74. Li Y et al (2020) Modified convolutional neural network with global average pooling for intelligent fault diagnosis of industrial gearbox. Eksploatacja i Niezawodność 22(1):63–72
75. Kumaresan S, Aultrin KSJ, Kumar SS, Anand MD (2021) Transfer learning with CNN for classification of weld defect. IEEE Access 9:95097–95108. https://doi.org/10.1109/ACCESS.2021.3093487
76. Zhang Y, Pan S, Zhan X et al (2020) FLDNet: light dense CNN for fingerprint liveness detection. IEEE Access 8:84141–84152. https://doi.org/10.1109/ACCESS.2020.2990909
77. Shi Z, Ye Y, Wu Y (2016) Rank-based pooling for deep convolutional neural networks. Neural Netw 83:21–31. https://doi.org/10.1016/J.NEUNET.2016.07.003
78. Zhang YD, Satapathy SC, Wu D et al (2021) Improving ductal carcinoma in situ classification by convolutional neural network with exponential linear unit and rank-based weighted pooling. Complex Intell Syst 7:1295–1310. https://doi.org/10.1007/S40747-020-00218-4/FIGURES/7
79. Akhtar N, Ragavendran U (2020) Interpretation of intelligence in CNN-pooling processes: a methodological survey. Neural Comput Appl 32:879–898. https://doi.org/10.1007/S00521-019-04296-5/METRICS
80. Zhu Y, Su H, Tang S et al (2023) A novel fault diagnosis method based on SWT and VGG-LSTM model for hydraulic axial piston pump. J Mar Sci Eng 11:594
81. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv Prepr arXiv14091556
82. Krizhevsky A, Sutskever I, Hinton GE (2017) ImageNet classification with deep convolutional neural networks. Commun ACM 60:84–90
83. Paymode AS, Malode VB (2022) Transfer learning for multi-crop leaf disease image classification using convolutional neural network VGG. Artif Intell Agric 6:23–33
84. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
85. Zagoruyko S (2016) Wide residual networks. arXiv Prepr arXiv160507146
86. Huang G, Sun Y, Liu Z, et al (2016) Deep networks with stochastic depth. In: Computer vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14. Springer, pp 646–661
87. He K, Zhang X, Ren S, Sun J (2015) Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision, pp 1026–1034
88. Targ S, Almeida D, Lyman K (2016) Resnet in resnet: generalizing residual architectures. arXiv Prepr arXiv160308029
89. Szegedy C, Liu W, Jia Y, et al (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1–9
90. Ioffe S (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv Prepr arXiv150203167
91. Szegedy C, Vanhoucke V, Ioffe S, et al (2016) Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2818–2826
92. Szegedy C, Ioffe S, Vanhoucke V, Alemi A (2017) Inception-v4, inception-resnet and the impact of residual connections on learning. In: Proceedings of the AAAI conference on artificial intelligence
93. Zhang Z, Yang Y, Xia X et al (2021) Unveiling the mystery of API evolution in Deep Learning frameworks: a case study of TensorFlow 2. 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, pp 238–247
94. Pang B, Nijkamp E, Wu YN (2020) Deep learning with tensorflow: a review. J Educ Behav Stat 45:227–248
95. Abadi M, Barham P, Chen J, et al (2016) {TensorFlow}: a system for {Large-Scale} machine learning. 12th USENIX Symp Oper Syst Des Implement (OSDI 16)
96. Moolayil J, Moolayil J, John S (2019) Learn Keras for deep neural networks. Apress, Birmingham, pp 33–35
97. Lee H, Song J (2019) Introduction to convolutional neural network using Keras; an understanding from a statistician. Commun Stat Appl Methods 26:591–610
98. Wang J, Liu Y, Hu Y, et al (2021) Facex-zoo: a pytorch toolbox for face recognition. In: Proceedings of the 29th ACM international conference on Multimedia, pp 3779–3782
99. Fey M, Lenssen JE (2019) Fast graph representation learning with PyTorch Geometric. arXiv Prepr arXiv190302428
100. Chen T, Li M, Li Y, et al (2015) Mxnet: a flexible and efficient machine learning library for heterogeneous distributed systems. arXiv Prepr arXiv151201274
101. Li M, Wen K, Lin H et al (2019) Improving the performance of distributed mxnet with rdma. Int J Parallel Program 47:467–480
102. Bahrampour S, Ramakrishnan N, Schott L, Shah M (2016) Comparative study of caffe, neon, theano, and torch for deep learning.
103. Du Y, Yang R, Chen Z et al (2021) A deep learning network-assisted bladder tumour recognition under cystoscopy based on Caffe deep learning framework and EasyDL platform. Int J Med Robot Comput Assist Surg 17:1–8

104. Garea AS, Heras DB, Argüello F (2019) Caffe CNN-based classification of hyperspectral images on GPU. J Supercomput 75:1065–1077
105. Bourez C (2017) Deep learning with Theano. Packt Publishing Ltd
106. Ma H, Mao F, Taylor GW (2017) Theano-mpi: a theano-based distributed training framework. In: Euro-Par 2016: Parallel Processing Workshops: Euro-Par 2016 International Workshops, Grenoble, France, August 24–26, 2016, Revised Selected Papers 22. Springer, pp 800–813
107. Girosi F, Jones M, Poggio T (1995) Regularization theory and neural networks architectures. Neural Comput 7:219–269
108. Murugan P, Durairaj S (2017) Regularization and optimization strategies in deep convolutional neural network. arXiv Prepr arXiv171204711
109. Bengio Y, Lamblin P, Popovici D, Larochelle H (2006) Greedy layer-wise training of deep networks. Adv Neural Inf Process Syst 19
110. Zhang C, Zhou P, Li C, Liu L (2015) A convolutional neural network for leaves recognition using data augmentation. In: 2015 IEEE international conference on computer and information technology; ubiquitous computing and communications; dependable, autonomic and secure computing; pervasive intelligence and computing. IEEE, pp 2143–2150
111. Zhang Z (2018) Improved Adam optimizer for deep neural networks. In: 2018 IEEE/ACM 26th international symposium on quality of service (IWQoS). IEEE, pp 1–2
112. Kurbiel T, Khaleghian S (2017) Training of deep neural networks based on distance measures using RMSProp. arXiv Prepr arXiv170801911
113. Qian Q, Jin R, Yi J et al (2015) Efficient distance metric learning by adaptive sampling and mini-batch stochastic gradient descent (SGD). Mach Learn 99:353–372
114. Li XD, Wang JS, Hao WK et al (2022) Multi-layer perceptron classification method of medical data based on biogeography-based optimization algorithm with probability distributions. Appl Soft Comput 121:108766. https://doi.org/10.1016/J.ASOC.2022.108766
115. Yang Z (2024) Competing leaders grey wolf optimizer and its application for training multi-layer perceptron classifier. Expert Syst Appl 239:122349. https://doi.org/10.1016/J.ESWA.2023.122349
116. Anggoro DA, Mukti SS (2021) Performance comparison of grid search and random search methods for hyperparameter tuning in extreme gradient boosting algorithm to predict chronic kidney failure. Int J Intell Eng Syst 14(6)
117. Frazier PI (2018) Bayesian optimization. In: Recent advances in optimization and modeling of contemporary problems. Informs, pp 255–278
118. Zhang K, Zhang Z, Li Z, Qiao Y (2016) Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Process Lett 23:1499–1503
119. Bulat A, Tzimiropoulos G (2016) Human pose estimation via convolutional part heatmap regression. In: Computer vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14. Springer, pp 717–732
120. Wang X, Gao L, Song J, Shen H (2016) Beyond frame-level CNN: saliency-aware 3-D CNN with LSTM for video action recognition. IEEE Signal Process Lett 24:510–514
121. Wang H, Schmid C (2013) Action recognition with improved trajectories. In: Proceedings of the IEEE international conference on computer vision, pp 3551–3558
122. Ji S, Xu W, Yang M, Yu K (2012) 3D convolutional neural networks for human action recognition. IEEE Trans Pattern Anal Mach Intell 35:221–231
123. Kalchbrenner N, Grefenstette E, Blunsom P (2014) A convolutional neural network for modelling sentences. arXiv Prepr arXiv14042188
124. Dauphin YN, Fan A, Auli M, Grangier D (2017) Language modeling with gated convolutional networks. International conference on machine learning. PMLR, pp 933–941
125. Yin W, Schütze H (2016) Multichannel variable-size convolution for sentence classification. arXiv Prepr arXiv160304513
126. Collobert R, Weston J, Bottou L et al (2011) Natural language processing (almost) from scratch. J Mach Learn Res 12:2493–2537
127. Conneau A, Schwenk H, Barrault L, Lecun Y (2016) Very deep convolutional networks for natural language processing. arXiv Prepr arXiv160601781 2:
128. Huang K-Y, Wu C-H, Hong Q-B et al (2019) Speech emotion recognition using deep neural network considering verbal and nonverbal speech sounds. ICASSP 2019–2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp 5866–5870
129. Lu Z, Pu H, Wang F, et al (2017) The expressive power of neural networks: a view from the width. Adv Neural Inf Process Syst 30
130. Egmont-Petersen M, de Ridder D, Handels H (2002) Image processing with neural networks—a review. Pattern Recogn 35:2279–2301
131. Nogueira K, Penatti OAB, Dos Santos JA (2017) Towards better exploiting convolutional neural networks for remote sensing scene classification. Pattern Recognit 61:539–556
132. Lopes AT, De Aguiar E, De Souza AF, Oliveira-Santos T (2017) Facial expression recognition with convolutional neural networks: coping with few data and the training sample order. Pattern Recognit 61:610–628
133. Everingham M, Eslami SMA, Van Gool L et al (2015) The pascal visual object classes challenge: a retrospective. Int J Comput Vis 111:98–136
134. Abdel-Hamid O, Mohamed A, Jiang H, Penn G (2012) Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. 2012 IEEE international conference on Acoustics, speech and signal processing (ICASSP). IEEE, pp 4277–4280
135. Tang YY, Lee S-W, Suen CY (1996) Automatic document processing: a survey. Pattern Recognit 29:1931–1952
136. Vinciarelli A (2002) A survey on off-line cursive word recognition. Pattern Recognit 35:1433–1446
137. Jung K, Kim KI, Jain AK (2004) Text information extraction in images and video: a survey. Pattern Recognit 37:977–997
138. Eskenazi S, Gomez-Krämer P, Ogier J-M (2017) A comprehensive survey of mostly textual document segmentation algorithms since 2008. Pattern Recognit 64:1–14
139. Bai X, Shi B, Zhang C et al (2017) Text/non-text image classification in the wild with convolutional neural networks. Pattern Recognit 66:437–446
140. Gomez L, Nicolaou A, Karatzas D (2017) Improving patch-based scene text script identification with ensembles of conjoined networks. Pattern Recognit 67:85–96
141. Gers FA, Schmidhuber J, Cummins F (2000) Learning to forget: continual prediction with LSTM. Neural Comput 12:2451–2471
142. He P, Huang W, Qiao Y, et al (2016) Reading scene text in deep convolutional sequences. In: Proceedings of the AAAI conference on artificial intelligence. 30(1)
143. Shi B, Bai X, Yao C (2016) An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. IEEE Trans Pattern Anal Mach Intell 39:2298–2304
144. Wang T, Wu DJ, Coates A, Ng AY (2012) End-to-end text recognition with convolutional neural networks. Proceedings

of the 21st international conference on pattern recognition (ICPR2012). IEEE, pp 3304–3308

145. Jaderberg M, Vedaldi A, Zisserman A (2014) Deep features for text spotting. In: Computer vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part IV 13. Springer, pp 512–528
146. Jaderberg M, Simonyan K, Vedaldi A, Zisserman A (2016) Reading text in the wild with convolutional neural networks. Int J Comput Vis 116:1–20
147. Nguyen DT, Li W, Ogunbona PO (2016) Human detection from images and videos: a survey. Pattern Recognit 51:148–175
148. Li Y, Wang S, Tian Q, Ding X (2015) Feature representation for statistical-learning-based object detection: a review. Pattern Recognit 48:3542–3559
149. Pedersoli M, Vedaldi A, Gonzalez J, Roca X (2015) A coarse-to-fine approach for fast deformable object detection. Pattern Recognit 48:1844–1853
150. Platt J, Nowlan S (1995) A convolutional neural network hand tracker. Proc Adv Neural Inf Process Syst: 901–908
151. Girshick R, Iandola F, Darrell T, Malik J (2015) Deformable part models are convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 437–446
152. Vaillant R, Monrocq C, Le Cun Y (1994) Original approach for the localisation of objects in images. IEE Proceedings-Vision, Image Signal Process 141:245–250
153. Lin T-Y, Maire M, Belongie S, et al (2014) Microsoft coco: Common objects in context. In: Computer vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part V 13. Springer, pp 740–755
154. Uijlings JRR, Van De Sande KEA, Gevers T, Smeulders AWM (2013) Selective search for object recognition. Int J Comput Vis 104:154–171
155. Endres I, Hoiem D (2013) Category-independent object proposals with diverse ranking. IEEE Trans Pattern Anal Mach Intell 36:222–234
156. Gómez L, Karatzas D (2017) Textproposals: a text-specific selective search algorithm for word spotting in the wild. Pattern Recognit 70:60–74
157. Girshick R, Donahue J, Darrell T, Malik J (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 580–587
158. Simo-Serra E, Trulls E, Ferraz L, et al (2014) Fracking deep convolutional image descriptors. arXiv Prepr arXiv14126537
159. Shrivastava A, Gupta A, Girshick R (2016) Training region-based object detectors with online hard example mining. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 761–769
160. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 779–788
161. Liu W, Anguelov D, Erhan D, et al (2016) Ssd: Single shot multibox detector. In: Computer vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14. Springer, pp 21–37
162. Lu Y, Javidi T, Lazebnik S (2016) Adaptive object detection using adjacency and zoom prediction. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 2351–2359
163. Peng X, Hoffman J, Stella XY, Saenko K (2016) Fine-to-coarse knowledge transfer for low-res image classification. 2016 IEEE International Conference on Image Processing (ICIP). IEEE, pp 3683–3687
164. Kawashima T, Kawanishi Y, Ide I et al (2017) Action recognition from extremely low-resolution thermal image sequence. 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS). IEEE, pp 1–6
165. Lee S, Son K, Kim H, Park J (2017) Car plate recognition based on CNN using embedded system with GPU. 2017 10th International Conference on Human System Interactions (HSI). IEEE, pp 239–241
166. Zhang J, Li W, Ogunbona PO et al (2016) RGB-D-based action recognition datasets: a survey. Pattern Recognit 60:86–105
167. Wang L, Ge L, Li R, Fang Y (2017) Three-stream CNNs for action recognition. Pattern Recognit Lett 92:33–40
168. Tran D, Bourdev L, Fergus R, et al (2015) Learning spatiotemporal features with 3d convolutional networks. In: Proceedings of the IEEE international conference on computer vision, pp 4489–4497
169. Karpathy A, Toderici G, Shetty S, et al (2014) Large-scale video classification with convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1725–1732
170. Simonyan K, Zisserman A (2014) Two-stream convolutional networks for action recognition in videos. Adv Neural Inf Process Syst 27
171. Chéron G, Laptev I, Schmid C (2015) P-CNN: pose-based cnn features for action recognition. In: Proceedings of the IEEE international conference on computer vision, pp 3218–3226
172. Donahue J, Anne Hendricks L, Guadarrama S, et al (2015) Long-term recurrent convolutional networks for visual recognition and description. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2625–2634
173. Aszemi NM, Dominic PDD (2019) Hyperparameter optimization in convolutional neural network using genetic algorithms. Int J Adv Comput Sci Appl 10(6)
174. Injadat M, Moubayed A, Nassif AB, Shami A (2020) Multi-split optimized bagging ensemble model selection for multi-class educational data mining. Appl Intell 50:4506–4528
175. Hutter F, Kotthoff L, Vanschoren J (2019) Automated machine learning: methods, systems, challenges. Springer Nature
176. Zöller M-A, Huber MF (2021) Benchmark and survey of automated machine learning frameworks. J Artif Intell Res 70:409–472
177. Pérez-Pérez BD, Garcia Vazquez JP, Salomón-Torres R (2021) Evaluation of convolutional neural networks' hyperparameters with transfer learning to determine sorting of ripe medjool dates. Agriculture 11:115
178. Pezzano G, Ripoll VR, Radeva P (2021) CoLe-CNN: Context-learning convolutional neural network with adaptive loss function for lung nodule segmentation. Comput Methods Programs Biomed 198:105792
179. Georgakopoulos S V, Plagianakos VP (2017) A novel adaptive learning rate algorithm for convolutional neural network training. In: Engineering applications of neural networks: 18th international conference, EANN 2017, Athens, Greece, August 25–27, 2017, Proceedings. Springer, pp 327–336
180. Kandel I, Castelli M (2020) The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. ICT Express 6:312–315
181. Sharma S, Gupta S, Gupta D et al (2022) Performance evaluation of the deep learning based convolutional neural network approach for the recognition of chest X-ray images. Front Oncol 12:932496
182. Sinha S, Singh TN, Singh VK, Verma AK (2010) Epoch determination for neural network by self-organized map (SOM). Comput Geosci 14:199–206

183. Srivastava N, Hinton G, Krizhevsky A et al (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15:1929–1958
184. Bottou L (2012) Stochastic gradient descent tricks. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics) 7700 LECTU: 421–436. https://doi.org/10.1007/978-3-642-35289-8_25
185. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
186. Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Netw for Mach Learn 4(2):26
187. Kingma DP, Ba JL (2015) Adam: A method for stochastic optimization. 3rd Int Conf Learn Represent ICLR 2015—Conf Track Proc
188. Smith LN (2018) A disciplined approach to neural network hyper-parameters: Part 1—learning rate, batch size, momentum, and weight decay. arXiv preprint arXiv:1803.09820
189. Loshchilov I, Hutter F (2017) SGDR: Stochastic gradient descent with warm restarts. 5th Int Conf Learn Represent ICLR 2017—Conf Track Proc
190. Smith LN (2017) Cyclical learning rates for training neural networks. Proceed—2017 IEEE Winter Conf Appl Comput Vision, WACV 2017, pp 464–472. https://doi.org/10.1109/WACV.2017.58
191. Claesen M, Simm J, Popovic D, et al (2014) Easy hyperparameter search using optunity. arXiv Prepr arXiv14121114
192. Lorenzo PR, Nalepa J, Kawulok M, et al (2017) Particle swarm optimization for hyper-parameter selection in deep neural networks. In: Proceedings of the genetic and evolutionary computation conference, pp 481–488
193. Witt C (2005) Worst-case and average-case approximations by simple randomized search heuristics. In: Annual symposium on theoretical aspects of computer science. Springer, pp 44–56
194. Eggensperger K, Feurer M, Hutter F, et al (2013) Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In: NIPS workshop on Bayesian Optimization in Theory and Practice, pp 1–5
195. Abreu S (2019) Automated architecture design for deep neural networks. arXiv Prepr arXiv190810714
196. Olof SS (2018) A comparative study of black-box optimization algorithms for tuning of hyper-parameters in deep neural networks
197. Ilievski I, Akhtar T, Feng J, Shoemaker C (2017) Efficient hyperparameter optimization for deep learning algorithms using deterministic rbf surrogates. In: Proceedings of the AAAI Conference on Artificial Intelligence
198. Injadat M, Salo F, Nassif AB et al (2018) Bayesian optimization with machine learning algorithms towards anomaly detection. 2018 IEEE global communications conference (GLOBECOM). IEEE, pp 1–6
199. DeCastro-García N, Muñoz Castañeda ÁL, Escudero García D, Carriegos MV (2019) Effect of the sampling of a dataset in the hyperparameter optimization phase over the efficiency of a machine learning algorithm. Complexity 2019:6278908
200. Bergstra J, Bardenet R, Bengio Y, Kégl B (2011) Algorithms for hyper-parameter optimization. Adv Neural Inf Process Syst 24
201. Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Learning and intelligent optimization: 5th international conference, LION 5, Rome, Italy, January 17–21, 2011. Selected Papers 5. Springer, pp 507–523
202. Seeger M (2004) Gaussian processes for machine learning. Int J Neural Syst 14:69–106
203. Yang HH, Amari S (1998) Complexity issues in natural gradient descent method for training multilayer perceptrons. Neural Comput 10:2137–2157
204. Maclaurin D, Duvenaud D, Adams R (2015) Gradient-based hyperparameter optimization through reversible learning. International conference on machine learning. PMLR, pp 2113–2122
205. Claesen M, De Moor B (2015) Hyperparameter search in machine learning. arXiv Prepr arXiv150202127
206. Bottou L (2010) Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010: 19th international conference on computational statistics, Paris, France, August 22–27, 2010 Keynote, Invited and Contributed Papers. Springer, pp 177–186
207. Zhang S, Xu J, Huang E, Chen C-H (2016) A new optimal sampling rule for multi-fidelity optimization via ordinal transformation. 2016 IEEE International Conference on Automation Science and Engineering (CASE). IEEE, pp 670–674
208. Elshawi R, Maher M, Sakr S (2019) Automated machine learning: state-of-the-art and open challenges. arXiv Prepr arXiv190602287
209. Karnin Z, Koren T, Somekh O (2013) Almost optimal exploration in multi-armed Bandits. International conference on machine learning. PMLR, pp 1238–1246
210. Li L, Jamieson K, DeSalvo G et al (2018) Hyperband: a novel bandit-based approach to hyperparameter optimization. J Mach Learn Res 18:1–52
211. Falkner S, Klein A, Hutter F (2018) BOHB: Robust and efficient hyperparameter optimization at scale. International conference on machine learning. PMLR, pp 1437–1446
212. Ashraf A, Anwaar A, Haider Bangyal W, et al (2023) An improved fire hawks optimizer for function optimization. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics) 13968 LNCS: 68–79. https://doi.org/10.1007/978-3-031-36622-2_6
213. Bangyal WH, Ahmad J, Rauf HT, Pervaiz S (2018) An overview of mutation strategies in bat algorithm. Int J Adv Comput Sci Appl 9:523–534. https://doi.org/10.14569/IJACSA.2018.090866
214. Poli R, Kennedy J, Blackwell T (2007) Quantification & assessment of the chemical form of residual gadolinium in the brain.pdf. Swarm Intell 1:33–57. https://doi.org/10.1007/s11721-007-0002-0
215. Bangyal WH, Ahmad J, Rauf HT (2013) An overview of mutation strategies in particle swarm optimization. Procedia Eng 11:16–37. https://doi.org/10.4018/IJAMC.2020100102
216. Challapalli JR, Devarakonda N (2022) A novel approach for optimization of convolution neural network with hybrid particle swarm and Grey Wolf algorithm for classification of Indian classical dances. Knowl Inf Syst 64:2411–2434. https://doi.org/10.1007/s10115-022-01707-3
217. Singh P, Chaudhury S, Panigrahi BK (2021) Hybrid MPSO-CNN: multi-level particle swarm optimized hyperparameters of convolutional neural network. Swarm Evol Comput 63:100863
218. Shaikh MS, Raj S, Babu R et al (2023) A hybrid moth–flame algorithm with particle swarm optimization with application in power transmission and distribution. Decis Anal J 6:100182. https://doi.org/10.1016/J.DAJOUR.2023.100182
219. Dorigo M, MB S (2006) T.: Ant colony optimization-artificial ants as a computational intelligence technique. Universit Libre de Bruxelles. Tech. Rep. TR/IRIDIA/2006–023
220. Lankford S, Grimes D (2020) Neural architecture search using particle swarm and ant colony optimization. CEUR Workshop Proc 2771:229–240
221. Yang XS (2010) Nature-inspired metaheuristic algorithms, Luniver press
222. Bacanin N, Bezdan T, Tuba E et al (2020) Optimizing convolutional neural network hyperparameters by enhanced swarm

intelligence metaheuristics. Algorithms. https://doi.org/10.3390/a13030067
223. Aswanandini R, Deepa C (2021) Hyper-heuristic firefly algorithm based convolutional neural networks for big data cyber security. Indian J Sci Technol 14:2934–2945. https://doi.org/10.17485/ijst/v14i38.1401
224. Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67. https://doi.org/10.1016/j.advengsoft.2016.01.008
225. Sruthi K, Rajalaxmi RR, Thangarajan R, Roopa C (2023) Optimizing CNN architecture using whale optimization algorithm for lung cancer detection. Elsevier Inc
226. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey Wolf optimizer. Adv Eng Softw 69:46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007
227. Shukla SK, Koley E, Ghosh S (2020) Grey Wolf optimization-tuned convolutional neural network for transmission line protection with immunity against symmetrical and asymmetrical power swing. Neural Comput Appl 32:17059–17076. https://doi.org/10.1007/s00521-020-04938-z
228. Mohakud R, Dash R (2022) Designing a grey wolf optimization based hyper-parameter optimized convolutional neural network classifier for skin cancer detection. J King Saud Univ Inf Sci 34:6280–6291
229. Yoon JH, Geem ZW (2021) Empirical convergence theory of harmony search algorithm for box-constrained discrete optimization of convex function. Mathematics 9:1–12. https://doi.org/10.3390/math9050545
230. Lee WY, Park SM, Sim KB (2018) Optimal hyperparameter tuning of convolutional neural networks based on the parameter-setting-free harmony search algorithm. Optik (Stuttg) 172:359–367. https://doi.org/10.1016/j.ijleo.2018.07.044
231. Kim S-H, Geem ZW, Han G-T (2020) Hyperparameter optimization method based on harmony search algorithm to improve performance of 1D CNN human respiration pattern recognition system. Sensors 20:3697
232. Huang Y-F, Liu J-S (2020) Optimizing convolutional neural network architecture using a self-adaptive harmony search algorithm. Advances in natural computation, fuzzy systems and knowledge discovery, vol 1. Springer, pp 3–12
233. Mirjalili S (2016) SCA: A sine cosine algorithm for solving optimization problems. Knowl-Based Syst 96:120–133. https://doi.org/10.1016/J.KNOSYS.2015.12.022
234. Bacanin N, Zivkovic M, Al-Turjman F et al (2022) Hybridized sine cosine algorithm with convolutional neural networks dropout regularization application. Sci Rep 12:1–20. https://doi.org/10.1038/s41598-022-09744-2
235. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11:341–359. https://doi.org/10.1023/A:1008202821328
236. Das S, Suganthan PN (2010) Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 15:4–31
237. Abdelkader EM, Al-Sakkaf A, Alfalah G, Elshaboury N (2022) Hybrid differential evolution-based regression tree model for predicting downstream dam hazard potential. Sustainability. https://doi.org/10.3390/su14053013
238. Podgorelec V, Pečnik Š, Vrbančič G (2020) Classification of similar sports images using convolutional neural network with hyper-parameter optimization. Appl Sci 10:1–24. https://doi.org/10.3390/app10238494
239. Mininno E, Neri F (2010) A memetic differential evolution approach in noisy optimization. Memetic Comput 2:111–135. https://doi.org/10.1007/s12293-009-0029-4
240. Ghasemi M, Rahimnejad A, Gil M et al (2023) A self-competitive mutation strategy for differential evolution algorithms with applications to proportional–integral–derivative controllers and automatic voltage regulator systems. Decis Anal J 7:100205. https://doi.org/10.1016/j.dajour.2023.100205
241. Katoch S, Chauhan SS, Kumar V (2021) A review on genetic algorithm: past, present, and future. Multimedia Tools Appl 80:8091–8126
242. Zhi H, Liu S (2019) Face recognition based on genetic algorithm. J Vis Commun Image Represent 58:495–502. https://doi.org/10.1016/j.jvcir.2018.12.012
243. Tan Y, Zhu Y (2010) Fireworks algorithm for optimization. In: Advances in swarm intelligence: first international conference, ICSI 2010, Beijing, China, June 12–15, 2010, Proceedings, Part I 1. Springer, pp 355–364
244. Li J, Tan Y (2018) The bare bones fireworks algorithm: a minimalist global optimizer. Appl Soft Comput J 62:454–462. https://doi.org/10.1016/j.asoc.2017.10.046
245. Tuba I, Veinovic M, Tuba E et al (2022) Tuning convolutional neural network hyperparameters by bare bones fireworks algorithm. Stud informatics Control 31:25–35
246. Ojha VK, Abraham A, Snášel V (2017) Metaheuristic design of feedforward neural networks: a review of two decades of research. Eng Appl Artif Intell 60:97–116. https://doi.org/10.1016/J.ENGAPPAI.2017.01.013
247. Hong Y-Y, Taylar JV, Fajardo AC (2020) Locational marginal price forecasting using deep learning network optimized by mapping-based genetic algorithm. Ieee Access 8:91975–91988
248. Guo Y, Li J-Y, Zhan Z-H (2020) Efficient hyperparameter optimization for convolution neural networks in deep learning: a distributed particle swarm optimization approach. Cybern Syst 52:36–57
249. Kan X, Fan Y, Fang Z et al (2021) A novel IoT network intrusion detection approach based on adaptive particle swarm optimization convolutional neural network. Inf Sci 568:147–162
250. Kanna PR, Santhi P (2022) Hybrid intrusion detection using mapreduce based black widow optimized convolutional long short-term memory neural networks. Expert Syst Appl 194:116545
251. Ragab M, Choudhry H, H. Asseri A, et al (2022) Enhanced gravitational search optimization with hybrid deep learning model for COVID-19 diagnosis on epidemiology data. In: Healthcare. MDPI, p 1339
252. Adamu S, Alhussian H, Aziz N et al (2023) Optimizing hyperparameters for improved melanoma classification using metaheuristic algorithm. Int J Adv Comput Sci Appl 14:531–540
253. Aguerchi K, Jabrane Y, Habba M, El Hassani AH (2024) A CNN hyperparameters optimization based on particle swarm optimization for mammography breast cancer classification. J Imaging 10:30
254. Purnomo H, Gonsalves T, Mailoa E et al (2024) Metaheuristics approach for hyperparameter tuning of convolutional neural network. J RESTI (Rekayasa Sist dan Teknol Informasi) 8:340–345
255. Naulia PS, Watada J, Aziz IA (2024) A mathematically inspired meta-heuristic approach to parameter (weight) optimization of deep convolution neural network. IEEE Access
256. Aljohani M, Bahgat WM, Balaha HM et al (2024) An automated metaheuristic-optimized approach for diagnosing and classifying brain tumors based on a convolutional neural network. Results Eng 23:102459
257. Papalkar RR, Jadhav J, Pattewar T et al (2025) WACSO: Wolf crow search optimizer for convolutional neural network hyperparameter optimization. Neural Process Lett 57:1–22
258. Gonzalez CI (2025) Designing optimal CNNs architectures using metaheuristic algorithms applied to the classification of Alzheimer's disease. Comput y Sist 29:179–189
259. Purkovic S, Jovanovic L, Zivkovic M et al (2024) Audio analysis with convolutional neural networks and boosting algorithms

tuned by metaheuristics for respiratory condition classification. J King Saud Univ—Comput Inf Sci 36:102261. https://doi.org/10.1016/J.JKSUCI.2024.102261

260. Jovanovic L, Damaševičius R, Matic R et al (2024) Detecting Parkinson's disease from shoe-mounted accelerometer sensors using convolutional neural networks optimized with modified metaheuristics. PeerJ Comput Sci 10:e2031. https://doi.org/10.7717/PEERJ-CS.2031/SUPP-2
261. Salb M, Jovanovic L, Bacanin N et al (2023) Enhancing Internet of Things network security using hybrid CNN and XGBoost model tuned via modified reptile search algorithm. Appl Sci 13:12687. https://doi.org/10.3390/APP132312687
262. Ashraf A, Zhao Q, Bangyal WH et al (2025) Female autism categorization using CNN based NeuroNet57 and ant colony optimization. Comput Biol Med 189:109926. https://doi.org/10.1016/J.COMPBIOMED.2025.109926