Intelligent Systems with Applications 22 (2024) 200349

Contents lists available at ScienceDirect

Intelligent Systems with Applications

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

Modified symbiotic organisms search optimization for automatic construction of convolutional neural network architectures

Fatsuma Jauro[a,b,**], Abdulsalam Ya'u Gital[b], Usman Ali Abdullahi[c], Aminu Onimisi Abdulsalami[a,f], Mohammed Abdullahi[a], Adamu Abubakar Ibrahim[d], Haruna Chiroma[e,*]

[a] *Department of Computer Science, Ahmadu Bello University, Zaria, Nigeria*
[b] *Department of Mathematical Science, Abubakar Tafawa Balewa University Bauchi, Nigeria*
[c] *Department of Computer Science, Federal College of Education, Technical, Gombe, Tanzania*
[d] *Department of Computer Science, International Islamic University, Malaysia*
[e] *College of Computer Science and Engineering, University of Hafr Al Batin, Saudi Arabia*
[f] *School of Computer Science and Artificial Intelligence, Wuhan University of Technology, PR China*

ARTICLE INFO

Keywords:
Convolutional neural network
Neural architecture search
Symbiotic organism search
And deep learning

ABSTRACT

Convolutional Neural Networks (ConvNets) have demonstrated impressive capabilities in image classification; however, the manual creation of these models is a labor-intensive and time-consuming endeavor due to their inherent complexity. This research introduces an innovative approach to Convolutional Neural Network (ConvNet) architecture generation through the utilization of the Symbiotic Organism Search ConvNet (SOS_ConvNet) algorithm. Leveraging the Symbiotic Organism Search optimization technique, SOS_ConvNet evolves ConvNet architectures tailored for diverse image classification tasks. The algorithm's distinctive feature lies in its ability to perform non-numeric computations, rendering it adaptable to intricate deep learning problems. To assess the effectiveness of SOS_ConvNet, experiments were conducted on diverse datasets, including MNIST, Fashion-MNIST, CIFAR-10, and the Breast Cancer dataset. Comparative analysis against existing models showcased the superior performance of SOS_ConvNet in terms of accuracy, error rate, and parameter efficiency. Notably, on the MNIST dataset, SOS_ConvNet achieved an impressive 0.31 % error rate, while on Fashion-MNIST, it demonstrated a competitive 6.7 % error rate, coupled with unparalleled parameter efficiency of 0.24 million parameters. The model excelled on CIFAR-10 and BreakHis datasets, yielding accuracies of 82.78 % and 89.12 %, respectively. Remarkably, the algorithm achieves remarkable accuracy while maintaining moderate model size.

1. Introduction

Deep learning algorithms, in contrast to traditional machine learning algorithms, can independently learn the features of a given dataset without the need for human experts to conduct feature extraction (Wani et al., 2020). A prominent approach in deep learning is the Convolutional Neural Network (ConvNet), which has exhibited remarkable performance in diverse domains such as object detection, image classification, and robotics. ConvNet models like LeNet (Lecun & Bengio, 1995), AlexNet (Krizhevsky et al., 2012), ResNet (He et al., 2016), VGG (Simonyan & Zissermann, 2015), and MobileNet (Howard et al., 2012) have displayed exceptional proficiency in image classification. However, these models were manually crafted by domain experts with specific knowledge, and their optimality may not be universal for every problem and dataset. Furthermore, transferring models from one domain to another necessitates expert modifications, posing a challenge in developing ConvNets for new domains where locating relevant experts can be challenging. The manual design of ConvNets can be a laborious task due to the intricate nature of the networks and the critical hyperparameters that require careful adjustment, including network depth, number of filters, filter size, and number of neurons. These hyperparameters significantly impact the performance of ConvNets. The challenges associated with manual design make automating ConvNet design a necessity.

* Corresponding author.
** Corresponding author.
E-mail addresses: fatijauro@gmail.com (F. Jauro), freedonchi@yahoo.com (H. Chiroma).

https://doi.org/10.1016/j.iswa.2024.200349
Received 24 October 2023; Received in revised form 15 February 2024; Accepted 27 February 2024
Available online 5 March 2024

Neural Architecture Search (NAS) has emerged as a research frontier facilitating the automatic generation of deep learning models through optimization algorithms (Kyriakides & Margaritis, 2020). NAS algorithms harness metaheuristic optimization methods that emulate the behaviour and evolution of species to address NAS challenges. These methods not only explore neural architectures but also optimize the weights of these architectures, a process known as NeuroEvolution (NE). The automation of ConvNet design via NAS enhances efficiency and reduces dependence on domain experts, allowing for the exploration of tailored architectures for specific domains. Recent developments in NAS have explored the application of Evolutionary Computing (EC) and Swarm Intelligence (SI) methods to autonomously generate ConvNet models without extensive reliance on experts or domain knowledge (Darwish et al., 2020). For example, the PSO_CNN algorithm (Fernandes Junior & Yen, 2019) was introduced to automatically search for ConvNets using the particle swarm optimization algorithm. While demonstrating commendable performance, the PSO algorithm requires tuning multiple parameters, posing a formidable challenge. Inadequate parameter selection can detrimentally impact the algorithm's efficacy. An investigation by Jauro et al. (2021) revealed that metaheuristic algorithms with fewer parameter settings can outperform popular algorithms relying on more parameters. One such algorithm is the Symbiotic Organism Search Optimization (SOS) algorithm, which has exhibited competitive or superior performance compared to other algorithms (Jauro et al., 2021). The SOS_CNN approach proposed by Miao et al. (2021) utilized the SOS algorithm to generate ConvNet architectures. Although the algorithm yielded promising results, it depended on random number generation, potentially limiting diversity in the solution space and leading to suboptimal solutions.

In this study, we adopt the SOS algorithm to explore ConvNet models. Diverging from SOS_CNN, our proposed approach adheres to the foundational steps of the traditional SOS algorithm, integrating a simple arithmetic rule for computations. This simplicity not only renders the algorithm straightforward but also fosters ample diversity in the solution space, potentially enhancing the quality of the generated ConvNet architectures. The primary objective is to introduce an algorithm for the automatic generation of ConvNet architectures using the symbiotic organism search optimization algorithm. In contrast to traditional ConvNets with fixed architectures, our approach permits individual architectures to dynamically adjust in length, thereby elevating flexibility and adaptability. The key contributions of this research include:

1. Introduction of a novel mutual vector operator, facilitating the algorithm to compute the average between two organisms with distinct architectural structures.
2. Design of a modified difference operator tailored for comparing two structurally different individuals, ensuring optimal updates during both the mutualism and commensalism phases.
3. Proposal of a novel sum operator that applies a simple arithmetic rule to compute the sum of two individual architectures that may vary in terms of the number of layers and hyperparameters.
4. Introduction of an enhanced initialization method that systematically generates a population of individual architectures with fewer pooling layers. This innovative approach establishes a foundation for efficient convergence and improved algorithm initialization.
5. The development of a novel strategy for generating parasite vectors, wherein one among multiple mutation strategies is randomly selected to obtain the parasite individual. This approach fosters diversity and introduces potential architectural enhancements.
6. Notably, the proposed method seamlessly applies the basic SOS without modifications to its phases and steps, ensuring adaptability to other metaheuristic algorithms. This underscores the versatility and ease of integration into various optimization frameworks.

In summary, this study introduces a comprehensive and innovative approach to ConvNet architecture generation, employing the SOS algorithm along with several novel operators and strategies to achieve superior performance and enhanced optimization capabilities. The subsequent sections are organized as follows: Section 2 provides a review of related works in the field, with a focus on NAS and ConvNets. Section 3 outlines the proposed SOS_ConvNet algorithm, offering a detailed explanation of the method. Section 4 details the experimental setup, including the datasets used and training procedures, to evaluate the performance of SOS_ConvNet. Section 5 presents the results and analysis of the experiments, highlighting the effectiveness of SOS_ConvNet in comparison to existing methods. Finally, Section 6 concludes the study, discussing findings, contributions, and future improvements for SOS_ConvNet.

2. Background and related works

2.1. Convolutional neural networks

The Convolutional Neural Network (ConvNet), originally designed by Lecun and Bengio (1995), is a deep learning technique primarily crafted for processing visual data, including images and videos. However, its versatility has been widely acknowledged, enabling effective handling of diverse data types, such as text and audio. ConvNets exhibit exceptional performance in various image-related tasks, including classification, detection, recognition, segmentation, restoration, and enhancement (Khan et al., 2019). These networks leverage a fundamental mathematical operation known as convolution, denoted as (f * g), where f and g represent functions. The result of convolution for a specific domain 'n' is defined as (Wani et al., 2020):

$$(f * g)(n) = \sum_m f(m)g(n-m) \tag{1}$$

Convolution can be extended to multi-dimensional functions as well. For example, when dealing with a two-dimensional image represented as Z, a 2D filter of $m \times n$ denoted as K, and a 2D feature map represented as X, the convolution operation can be mathematically expressed as follows:

$$X(i,j) = (Z * K)(i,j)\sum_m \sum_n Z(m,n)K(i-m,\ j-n) \tag{2}$$

This operation is commutative and can thus be expressed as:

$$X(i,j) = (Z * K)(i,j)\sum_m \sum_n Z(i-m,\ j-n)K(m,n) \tag{3}$$

The commutative property holds because the kernel is flipped relative to the input. If the kernel is not flipped, the operation becomes a cross-correlation operation, as shown below:

$$X(i,j) = (Z * K)(i,j)\sum_m \sum_n Z(i+m,\ j+n)K(m,n) \tag{4}$$

ConvNets consist of various layers, each tasked with distinct functions: convolution layer, activation function layer, pooling layer, fully connected layer, and dropout layer. Convolutional layers serve as the foundational building blocks of ConvNets, applying the convolution operation to capture local features within an image and transform them into feature maps (Yamashita et al., 2018). Activation functions such as the commonly used Rectified Linear Unit (ReLU), introduce non-linearity to enhance the network's training speed. In addition to convolutional layers, ConvNets incorporate crucial layers like pooling layers for dimensionality reduction, fully connected layers for classification, and dropout layers to mitigate overfitting. This collaborative interplay among these layers empowers ConvNets to effectively learn and represent intricate patterns and structures within data, establishing them as valuable tools in modern machine learning and computer vision applications. ConvNets find applications across diverse sectors, including healthcare, where they prove instrumental in the classification of electrocardiogram (ECG) images. This is attributed to their capability to convert 1D ECG signals into 2D image representations, facilitating effective classification (Musa et al., 2023).

2.2. Symbiotic organism search algorithm

The Symbiotic Organism Search Algorithm (SOS) is a metaheuristic technique introduced by (M. Cheng & Prayogo, 2014) to emulate the interdependent relationships among various species within an ecological system. In ecosystems, numerous organisms depend on each other for survival, forming symbiotic relationships. These relationships fall into three main categories: Mutualism, Commensalism, and Parasitism. Mutualism involves two interacting organisms that both derive benefits from the relationship. Commensalism describes a scenario where two organisms interact, with one deriving benefit while the other remains unaffected. In contrast, parasitism characterizes relationships where one organism benefits at the expense of harm inflicted upon the other. In line with the principles common to various algorithms, SOS operates by employing a population of potential solutions to systematically explore and identify the optimal one. Initially, this population, commonly known as the ecosystem, is randomly generated within the search space of potential solutions, with each solution represented as organism. Each organism in the ecosystem is assigned a fitness value, serving as an indicator of its viability within the system. The solution space undergoes iterative modifications through a series of processes involving mutualism, commensalism, and parasitism, persisting until the predefined termination criteria are met. This dynamic approach allows SOS to efficiently navigate the solution landscape in its quest for optimal solutions.

2.2.1. Mutualism

In a given population comprising organisms, each organism X_i engages in interaction with another randomly selected organism X_j (where $(i \neq j)$) from the ecosystem. The purpose of this interaction is to improve their chances of coexisting harmoniously within the ecosystem. New solutions resulting from mutualism for X_i and X_j are updated using the following equations introduced by (M. Cheng & Prayogo, 2014):

$$X_{inew} = X_i + rand(0,\ 1) * \ (X_{best} - Mutual_{Vector} * BF_1) \tag{5}$$

$$X_{jnew} = X_j + rand(0,\ 1) * \ (X_{best} - Mutual\_Vector\ * BF_2) \tag{6}$$

$$Mutual_{Vector} = \frac{X_i + X_j}{2} \tag{7}$$

The term " $rand(0,\ 1)$" represents a vector containing random numbers falling within the specified range of 0 to 1. BF_1 and BF_2 denote benefit factors that are randomly chosen to be either 1 or 2. $Mutual_{Vector}$ is a vector representing the extent of mutuality between organisms X_i and X_j. X_{best} represents the organism with the highest fitness value. The term $(X_{best} - Mutual_{Vector} * BF_1)$ signifies the joint effort made by organisms to enhance their chances of survival. It's important to note that updates to solutions occur only if the new fitness values surpass the previous ones. When this condition is met, organisms X_{inew} and X_{jnew} replace X_i and X_j, respectively, resulting in an update to the ecosystem. Conversely, if X_{inew} and X_{jnew} do not meet this condition, X_i and X_j continue to exist while the new solutions are discarded.

2.2.2. Commensalism

In commensalism, a type of relationship where one organism benefits while the other is neither helped nor harmed, the interaction involves selecting a random organism, X_j, to interact with a given organism, X_i, in the ecosystem. In this scenario, X_i gains an advantage from the interaction, while X_j remains unaffected. The update for X_i in the ecosystem is carried out using the following equation:

$$X_{inew} = X_i + rand(-1,\ 1) * \ \left(X_{best} - X_j\right) \tag{8}$$

Here, $(X_{best} - X_j)$ denote the benefit derived by X_i from X_j. X_{inew} replaces X_i in the ecosystem only when X_{inew} demonstrates superior fitness value compared to X_i, otherwise, X_i remains in the ecosystem, and X_{inew} is discarded.

2.2.3. Parasitism

Parasitism is a form of relationship where one organism benefits at the expense of harming another. In this context, the benefiting organism is referred to as the parasite. Within an ecosystem, an organism X_i is chosen and replicated to form a parasite vector. Simultaneously, another organism X_j is randomly selected from the ecosystem to serve as the host for this parasite vector. If the fitness value of the parasite vector surpasses that of X_j, it takes over X_j's position by replacing it. However, if X_j exhibits a superior fitness value, it remains the fittest and the parasite vector does not survive.

2.3. Related works

Advancements in Neural Architecture Search (NAS) aim to develop neural architectures that achieve optimal performance with limited computing resources, minimizing human intervention (X. Cheng et al., 2020). Pioneering contributions by (Baker et al., 2016) and (Zoph & Le, 2016) stand out as seminal works in the NAS domain. These approaches are recognized for their innovative use of reinforcement learning (RL) techniques, demonstrating their effectiveness in achieving state-of-the-art accuracy in image classification tasks. Their success highlights the potential and practicality of automated neural architecture design.

RL-based NAS algorithms play a crucial role in automating the neural architecture design process. The introduction of the Neural Architecture Search with Reinforcement Learning (NAS_RL) algorithm by (Zoph & Le, 2016) represents a milestone in this approach. NAS_RL represents neural architecture as a variable-length string generated by a recurrent neural network (RNN) acting as a controller. This string serves as a blueprint for constructing the corresponding neural architecture. Subsequently, reinforcement learning is employed as the search strategy to optimize and adjust the neural architecture search process based on this blueprint. NAS_RL, demonstrating superiority over some manually generated architectures, showcases the potential of RL-based NAS. A pivotal study, "Large-scale Evolution" (Real et al., 2017), further substantiates the viability of NAS concepts. This research, employing evolutionary learning, achieves results comparable to RL-based approaches, emphasizing the adaptability and robustness of NAS methodologies. NAS has gained prominence in the deep learning community, leading to the proposal of various NAS algorithms, including DARTS (Liu et al., 2018), ENAS (Pham et al., 2018), and P-DARTS (Chen et al., 2019). These algorithms exhibit significant improvements across diverse tasks such as image classification, object detection, and natural language processing.

In addition to RL methods, alternative approaches employ metaheuristic techniques for ConvNet search or optimization of hyperparameters. For instance, (Fernandes Junior & Yen, 2019) introduced a Particle Swarm Optimization (PSO)-based algorithm for searching the best ConvNet architecture in image classification. The algorithm encompasses various procedures, including ConvNet representation, swarm initialization, particles' fitness evaluation, measurement of differences between particles, velocity estimation, and particle update to explore an optimal ConvNet architecture. Evaluation results demonstrated the algorithm's capability to discover an optimal architecture for any given dataset without requiring prior domain knowledge. However, due to limited computational power, the algorithm faced challenges in discovering more complex networks. Another study by (Rasdi et al., 2016) delved into the performance assessment of three different metaheuristic algorithms for ConvNet optimization. The algorithms considered in this research are Harmony Search (HS), Simulated Annealing (SA), and Differential Evolution (DE). These optimization algorithms operate in the last layer of the ConvNet, where the values of weights and

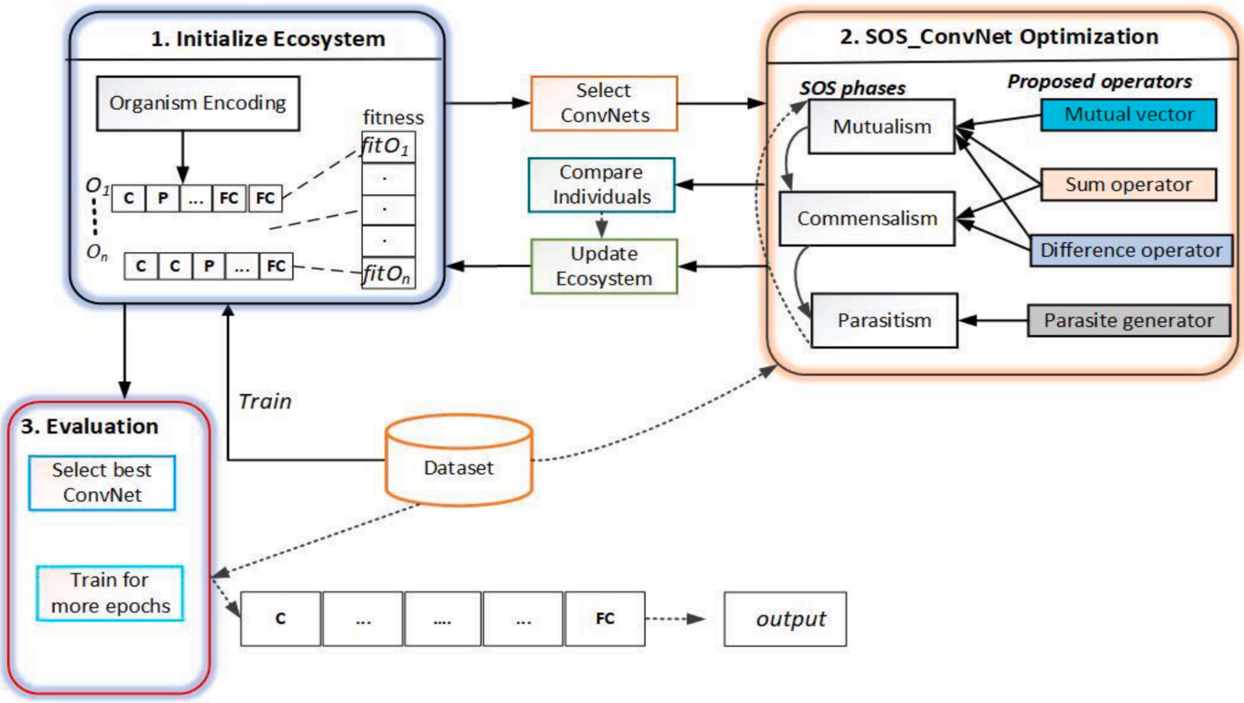


Fig. 1. General architecture of the proposed SOS_ConvNet.

biases form the solution vector for searching the optimal fitness function. The obtained results are then utilized to retrain the preceding layers. This comprehensive investigation contributes valuable insights into the effectiveness of various metaheuristic algorithms in optimizing ConvNet architectures. The study conducted by (Kabir Anaraki et al., 2019) explores the integration of ConvNets and Genetic Algorithm (GA) for the classification and grading of brain tumors based on magnetic resonance imaging (MRI) data. By employing ConvNets, the study facilitates the automatic extraction of intricate features from MRI images, enabling the classification of brain tumor grades. Genetic Algorithm is then employed to optimize the ConvNet architecture and hyperparameters, resulting in an overall enhancement of performance in terms of accuracy and efficiency. Another utilization of Genetic Algorithm for the automated ConvNet architecture generation was introduced by (Sun et al., 2020). This algorithm eliminates the need for users to possess ConvNet expertise to obtain effective architectures. An asynchronous component is incorporated to harness available computational resources, enhancing the efficiency of fitness evaluation. Additionally, the algorithm integrates skip connections to address the challenge of vanishing gradients in complex data. Further optimization in evaluation time is achieved through the inclusion of a cache component, reducing the time required for evaluating the entire population.

A multi-objective approach to neural architecture search was introduced by (Jiang et al., 2020), presenting an improved decomposition-based multi-objective PSO algorithm. This novel algorithm aims to optimize two conflicting objectives of neural networks, namely high accuracy and minimized learned parameters. Notably, the algorithm focuses on the search for ConvNet architectures, contributing to the field by addressing multiple objectives simultaneously and targeting the specific requirements of ConvNets. In a study by (Martín et al., 2020), a hybridized statistical coral-reef optimization algorithm was proposed to reduce the complexity of ConvNets without compromising their performance. The metaheuristic algorithm focuses on the reconstruction of the last layers, particularly the fully connected layers, in the ConvNet. A hybridization method incorporating backpropagation is included as the final stage to fine-tune the parameters of the network. This research presents an innovative approach to enhancing ConvNet architectures by addressing complexity issues in a targeted manner. Furthermore, the study by (Miao et al., 2021) proposed SOS-CNN for ConvNet architecture search. This research introduces three new non-numeric computational strategies—binary segmentation, slack gain, and dissimilar mutation. These strategies are seamlessly integrated with the Symbiotic Organism Search (SOS) algorithm, enriching its optimization capabilities for ConvNet architecture search. The algorithm demonstrated good performance in terms of accuracy, showcasing the effectiveness of the proposed non-numeric computational strategies in optimizing ConvNet architectures.

Wen et al. (2022) introduced the Evolutionary Neural Architecture Search algorithm with RepVGG nodes (EvoNAS-Rep), presenting a new encoding strategy that maps fixed-length encoding individuals to deep learning structures with variable depth. The algorithm leverages RepVGG nodes, and a Genetic Algorithm (GA) is employed to search for optimal individuals and their corresponding deep learning models. The iterative training process is designed to simultaneously evolve the GA and train the deep learning model. The research focuses on enhancing the efficiency and effectiveness of Neural Architecture Search (NAS) for image classification, utilizing GA and innovative encoding strategies to improve the search and optimization process. Additionally, (Kong et al., 2023) proposed a novel NAS framework specifically tailored for EEG-based sleep stage classification. The framework conducts architectural searches using a bilevel optimization approximation, refining the model through search space approximation and search space regularization while sharing parameters across cells. The study aims to enhance the accuracy and efficiency of sleep stage classification using EEG data. The NAS framework optimizes the neural architecture for this particular application, addressing the unique challenges and requirements of EEG-based sleep stage classification.

An algorithm called the Multi-Objective Evolutionary Algorithm with Probability Stack (MOEA-PS) for Neural Architecture Search (NAS) was proposed by (Xue et al., 2023). The study focuses on optimizing precision and time consumption as the primary objectives. The method utilizes an adjacency list to represent the internal structure of deep neural networks and incorporates a unique mechanism in the

Algorithm 1
The proposed SOS_ConvNet

Input: Eco size ($ecosize$), maximum number of layers (l_{max}), maximum number of filters (nf_{max}), maximum filter size (f_{max}), maximum number of neurons in FC layer (n_{max}), number of outputs (n_{op}), training data (X), epochs ($epochs$), maximum iteration (maxIt).
Output: Best SOS_ConvNet model architecture

```
1   $Eco = \{o_i, \dots, o_n\} \leftarrow \boldsymbol{InitializeEco}(ecosize, l_{max}, nf_{max}, f_{max}, n_{max}, n_{op})$
2   for ($i = 1\ to\ n$) do
3       $o_i.loss = \boldsymbol{ComputeLoss}(o_i, X, epoch)$
4   end
5   While ($it <$ maxIt)
6       for($i = 1\ to\ n$) do
7           select $o_{best}, o_i, o_j$ where $o_i \neq o_j$
8           //mutualism
9           $MutualVector = \boldsymbol{MV}(o_i, o_j)$
10          $o_{inew} = \boldsymbol{ConvNetSum}(o_i, \boldsymbol{ConvNet\ diff}(o_{best}, MutualVector))$
11          $o_{jnew} = \boldsymbol{ConvNetSum}(o_j, \boldsymbol{ConvNet\ diff}(o_{best}, MutualVector))$
12          if ($\boldsymbol{ComputeLoss}(o_{inew})) < o_i.loss$
13              Replace $o_i$ with $o_inew$
14          end if
15          if ($\boldsymbol{ComputeLoss}(o_{jnew})) < o_j.loss$
16              Replace $o_j$ with $o_jnew$
17          end if
18          //commensalism
19          select $o_j$ where $o_i \neq o_j$
20          $o_{inew} = sum\ (o_i, diff(o_{best}, o_j))$
21          if ($\boldsymbol{ComputeLoss}(o_{inew})) < o_i.loss$
22              Replace $o_i$ with $o_inew$
23          end if
24          //Parasitism
25          Select $o_i$
26          $o_{parasite} = o_i.copy$
27          If $o_{parasite}.length < 3$
28              If $rand() > rand()$
29                  $o_{parasite}.push(addConv)$
30              else
31                  $o_{parasite}.push(addPool)$
32              end if
33          else
34              $prob = rand()$
35              If $prob > 0.5$
36                  If $rand() > rand()$
37                      $o_{parasite}.push(addConv)$
38                  else
39                      $o_{parasite}.push(addPool)$
40                  end if
41              else
42                  $ridx = rand(2, o_{parasite}.length)$
43                  $o_{ridx} = null$
44              end if
45          end if
46          If ($\boldsymbol{ComputeLoss}(o_{parasite})) < o_i.loss$
47                  Replace $o_i$ with $o_{parasite}$
48          end if
49      end
```

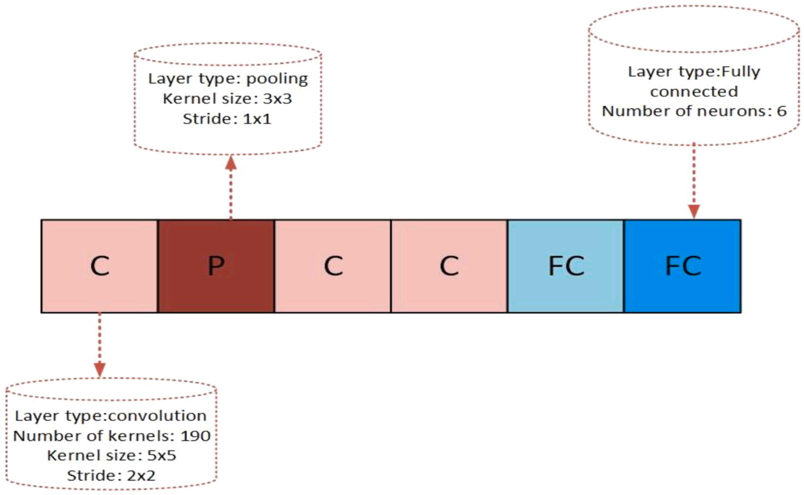


Fig. 2. Sample of a single organism representation.

multi-objective genetic algorithm to guide crossover and mutation processes during offspring generation. Additionally, structure blocks are stacked using a proxy model to generate deep neural networks. The approach aims to strike a balance between precision and computational efficiency, highlighting the potential of MOEA-PS in enhancing NAS for evolving optimal neural network architectures. An automated CNN design approach using Monarch Butterfly Optimization (MBO) was proposed by (Wang et al., 2023). The method focuses on creating a comprehensive Neural Function Unit (NFU)-based architecture representation, combining elements from GoogLeNet, ResNet, and DenseNet. This integration facilitates a joint exploration of both macro-architecture and depth in convolutional neural networks (CNNs). Additionally, a direct architecture encoding is implemented to leverage the fast convergence of MBO. This involves using evolutionary operators with minimal computational complexity to iteratively refine the architecture population through encoding optimization. Through extensive experiments the proposed method demonstrates consistently competitive performance with significantly reduced time and computational requirements. (Mishra & Kane, 2023)introduced a framework that employs a modified Genetic Algorithm (GA) to autonomously evolve a proficient Convolutional Neural Network (CNN) architecture for image classification. The GA is improved through the formulation of an effective encoding scheme, a method for initializing the input population, and a diverse approach for generating offspring. Furthermore, an optimized fitness function is suggested to accelerate convergence and mitigate the risk of becoming trapped in local optima. Experimental results affirm the efficacy of the approach, showcasing its performance on par with the top-performing manual and state-of-the-art automatic architectures in terms of accuracy, convergence rate, and computational resource utilization.

An optimization approach for residual networks by utilizing an improved Particle Swarm Optimization (PSO) algorithm was proposed by (Wang et al., 2024). The fundamental unit for architecture exploration is a low-complexity residual architecture block, enabling a more diverse investigation into network architectures while minimizing parameters. Additionally, a depth initialization strategy is employed to restrict the search space within a reasonable range, preventing unnecessary particle exploration. Furthermore, a unique method for calculating particle differences and updating velocity mechanisms is presented to enhance the exploration of updated trajectories. This approach significantly contributes to better utilization of the search space and increased particle diversity. Experimental results showcase that the algorithm can design lightweight networks with superior classification performance. In a recent work by (Sharif et al., 2024), a novel approach employing metaheuristic algorithms for brain tumor classification is introduced. The initial step involves enhancing contrast through a combination of hybrid division histogram equalization and an ant colony optimization approach. Subsequently, a newly designed nine-layered CNN model is trained on this preprocessed data. Feature extraction from the second fully connected layer is executed and optimized using both differential evolution and moth flame optimization. The outputs from these optimization methods are fused using a matrix length approach and fed into a multi-class support vector machine (MC-SVM). Comparative analysis with existing techniques highlights the superior performance of the proposed approach. Beyond its significant contributions to image classification and various optimization techniques for Convolutional Neural Networks (ConvNets), NAS has extended its impact into diverse domains. Particularly noteworthy are its applications in speech emotion recognition, as evidenced by the work of Wu et al. (2022), and driver emotion recognition, as showcased by Zaman et al. (2022). These additional domains underscore the versatility and adaptability of NAS, affirming its role in advancing not only computer vision and deep learning but also in enhancing our understanding of emotional cues in speech and drivers.

3. Proposed SOS_ConvNet

The fundamental architecture of SOS_ConvNet relies on a set of essential input hyperparameters, with the training data standing as a cornerstone within the framework. Beyond the training data, the algorithm's behaviour is intricately influenced by several key hyperparameters. These pivotal hyperparameters encompass the maximum number of layers during initialization, the count of organisms, the total number of iterations, the batch size, the number of epochs, the number of neurons, the number of outputs, the kernel count, and the kernel dimension. Each of these hyperparameters plays a critical role in shaping and defining the characteristics and performance of the SOS_ConvNet algorithm.

The SOS_ConvNet process for architecture creation and optimization revolves around three fundamental stages: initialization, optimization, and evaluation, as illustrated in Fig. 1. The optimization stage, a pivotal element of the process, encompasses three crucial phases; Mutualism, Commensalism, and Parasitism, similar to the basic SOS algorithm. To effectively apply these phases in the context of non-numeric neural architecture search, five distinct strategies have been developed:

1. Organism Encoding Strategy for Non-Numeric ConvNet Representation: This strategy defines how ConvNet architectures are non-numerically represented, offering a means to manipulate and evolve them within the algorithm.

Algorithm 2
Initialization of the ecosystem (InitializeEcosystem())

```
Input: Eco size ($n$), maximum number of layers ($l_{max}$), maximum number of filters ($nf_{max}$), maximum filter size ($f_{max}$), maximum number of neurons in F layer ($n_{max}$), number of outputs ($n_{op}$).
Output: A population of ($ecosize$) organisms $eco = \{o_1, \ldots, o_n\}$

1  For $i = i$ to $n$ do
2      $o_i.lenght = rand(3, l_{max})$;
3      for $j = 1$ to $o_i.lenght$ do
4          If $j == 1$ then
5              $list\_layers[j] \leftarrow InsertConv(f_{max}, nf_{max})$;
6          else if $j == o_i.lenght$ then
7              $list\_layers[j] \leftarrow InsertF(n_{op})$;
8          else if $list\_layers[j-1].type == "fully-connected"$ then
9              $list\_layers[j] \leftarrow InsertF(n_{max})$;
10         else
11             $layer\_type \leftarrow rand(1,3)$;
12             if $layer\_type == 1$ then
13                 $list\_layers[j] \leftarrow InsertConv(f_{max}, nf_{max})$;
14             else if $layer\_type == 2$ then
15                 $list\_layers[j] \leftarrow InsertPool()$;
16             else
17                 $list\_layers[j] \leftarrow InsertFC(n_{max})$;
18             end
19         end
20     end
21     $org_i.list\_layers \leftarrow list\_layers$;
22 end
23 return $eco = \{o_1, \ldots, o_n\}$
```

Algorithm 3
ConvNetDiff()

```
Input: two architectures $o1, o2$
Output: an architecture
1  If ($o_1.length > o_2.length$)
2      $O = o_1.length$
3  else if ($o_1.length < o_2.length$)
4      $O = o_2.length$
5  else
6      $O = o_1.length$
7  end if
8  $ConvNetDiff \leftarrow [\,]$
9  For $i = 1$ to $O$ do
10     If ($o1_{[i]}$ is not Empty and $o2_{[i]}$ is not Empty)
11         If($o1_{[i]}.type == o2_{[i]}.type$)
12             $o1_{[i]}.type = "keep"$
13             $ConvNetDiff \leftarrow o1_{[i]}$
14         else
15             $ConvNetDiff \leftarrow o1_{[i]}$
16     else if ($o1_{[i]}$ is not Empty and $o2_{[i]}$ is Empty)
17             $ConvNetDiff \leftarrow o1_{[i]}$
18     else if ($o1_{[i]}$ is Empty and $o2_{[i]}$ is not Empty)
19             $o2_{[i]}.type = "remove"$
20     end if
21 Return $ConvNetDiff$
```

2. Average Operator: Applied during the Mutualism phase, this operator generates mutual vectors by computing the average between two organisms, facilitating cooperative optimization.
3. Modified Difference Operator: Utilized in both Mutualism and Commensalism phases, this operator computes the difference between two organisms in a manner tailored to non-numeric representations, aiding in architecture refinement.
4. Novel Sum Operator for Non-Numeric Summation: In the Mutualism and Commensalism phases, this strategy defines how non-numeric summation between two organisms is carried out, promoting cooperative evolution.
5. Mutation Strategy for Parasite Vector Creation: Employed in the Parasitism phase, this strategy guides the creation of parasite organisms, contributing to diversity in the population.

During the optimization process, the architecture exhibiting the lowest loss value typically denotes the most optimal organism, with the loss value serving as the fitness metric. The overarching process is outlined in Algorithm 1. A crucial aspect of the proposed algorithm is that the best architecture, along with all its features, advances to the next phase without undergoing re-optimization. This strategic choice is pivotal in retaining and building upon the most promising architectures. The ensuing sections will delve into these methodologies in greater detail, offering a comprehensive understanding of their implementation and significance within the algorithm.

Input: Eco size (*ecosize*), maximum number of layers (l_{max}), maximum number of filters (nf_{max}), maximum filter size (f_{max}), maximum number of neurons in FC layer (n_{max}), number of outputs (n_{op}), training data (X), epochs (*epochs*), maximum iteration (maxIt).
Output: Best SOS_ConvNet model architecture

```
1  $Eco = \{o_i, \ldots, o_n\} \leftarrow$ InitializeEco(ecosize, $l_{max}, nf_{max}, f_{max}, n_{max}, n_{op}$)
```

(continued on next page)

(*continued*)

```
2   for ($i = 1$ to $n$) do
3   $o_i.loss = ComputeLoss(o_i, X, epoch)$
4   End
5   While ($it <$ maxIt)
6   for($i = 1$ to $n$) do
7   select $o_{best}$, $o_i$, $o_j$ where $o_i \neq o_j$
8   //mutualism
9   $MutualVector = MV(o_i, o_j)$
10  $o_{inew} = ConvNetSum(o_i, ConvNet\ diff(o_{best}, MutualVector))$
11  $o_{jnew} = ConvNetSum(o_j, ConvNet\ diff(o_{best}, MutualVector))$
12  if ($ComputeLoss(o_{inew})) < o_i.loss$
13  Replace $o_i$ with $o_inew$
14  end if
15  if ($ComputeLoss(o_{jnew})) < o_j.loss$
16  Replace $o_j$ with $o_jnew$
17  end if
18  //commensalism
19  select $o_j$ where $o_i \neq o_j$
20  $o_{inew} = sum\ (o_i, diff(o_{best}, o_j))$
21  if ($ComputeLoss(o_{inew})) < o_i.loss$
22  Replace $o_i$ with $o_inew$
23  end if
24  //Parasitism
25  Select $o_i$
26  $o_{parasite} = o_i.copy$
27  If $o_{parasite}.length < 3$
28  If $rand() > rand()$
29  $o_{parasite}.push(addConv)$
30  else
31  $o_{parasite}.push(addPool)$
32  end if
33  else
34  $prob = rand()$
35  If $prob > 0.5$
36  If $rand() > rand()$
37  $o_{parasite}.push(addConv)$
38  else
39  $o_{parasite}.push(addPool)$
40  end if
41  else
42  $ridx = rand(2, o_{parasite}.length)$
43  $o_{ridx} = null$
44  end if
45  end if
46  If ($ComputeLoss(o_{parasite})) < o_i.loss$
47  Replace $o_i$ with $o_{parasite}$
48  end if
49  end
```

3.1. Organism encoding strategy

A critical aspect of developing any population-based approach geared towards the evolution of ConvNets lies in the encoding or representation scheme. This work introduces a straightforward yet effective organism encoding strategy. Each organism in the population of SOS serves as a potential solution. In this encoding strategy, an organism represents a complete ConvNet architecture. It's crucial to emphasize that this proposed model does not optimize the weights of the network but focuses solely on defining the architectural components, such as layer types (convolution, pooling, and fully connected), and other pertinent hyperparameters. These hyperparameters include the number of kernels and kernel sizes for convolutional layers, pool window specifications for pooling layers, and the number of neurons for fully connected layers, all of which are essential for evolving architectures.

In a typical ConvNet architecture, three main hidden layers are involved: convolutional, pooling (which can be either max or average pooling), and fully connected layers. Within the proposed encoding strategy, an individual organism that represents a ConvNet includes all three layers. These layers are organized in the form of a list of dictionaries, where each dictionary entry corresponds to one layer. Each dictionary contains specific information related to the layer type and its respective hyperparameters. For example, a position in the list (a dictionary) stores details such as the type of layer, the number of kernels, kernel dimension, and stride for a convolution. In the case of a pooling layer, it includes information about the layer type, kernel dimensions, and stride value. A position representing a fully connected layer provides details about the layer type and the number of neurons. Fig. 2 illustrates this encoding strategy, where C, P, and FC respectively signify convolution, pooling, and fully connected layers. This encoding approach offers a concise yet comprehensive representation of ConvNet architectures, allowing the evolutionary algorithm to effectively explore and optimize these architectures.

3.2. Initialization of the ecosystem

The initial step in the proposed model involves the initialization of the Ecosystem. The function *initializeEcosystem()* is invoked to randomly generate N organisms, each representing distinct ConvNet models within the initial population. Adhering to ConvNet architectural conventions, it is a prerequisite for a ConvNet to consist of at least three layers, with convolution as the first layer and a fully connected layer as the final layer. Accordingly, a range between three to l_{max} is defined, within which the number of layers for each architecture is randomly generated. To ensure the validity of all generated architectures, two conditions are enforced. The first condition mandates that the first and last layers of every architecture must be convolution and fully connected layers, respectively. The second condition ensures that no other layer follows a fully connected layer except another fully connected layer. This adheres to the established convention in the literature on ConvNet design. Introducing a fully connected layer in-between other layers significantly increases the number of trainable parameters in a model, making it more complex. Pooling layers are strategically placed to reduce the number of output features by a factor of two, resulting in a more suitable input for the fully connected layer, where fewer neurons are required.

The algorithm's initialization process is executed through various functions as depicted in Algorithm 2. The *InsertConv()* function adds a convolution to the architecture, incorporating a random number of filters within the range $1\ to\ nf_{max}$. The filter size is also randomly selected within the range of $3\,X\,3$ to $f_{max}\,X\,f_{max}$, where f_{max} represents the maximum filter size, and the sliding value is set to 1. In this research, the activation function for all layers is consistently the rectified linear unit (ReLU), except for the softmax (output) layer. The *InsertPool()* function adds a max pooling layer randomly to the individual architecture, featuring a window size of $3\ X\ 3$ and a stride value of 1. To mitigate the risk of excessive inclusion of pooling layers, the probability of adding pooling layers is kept very low. This precaution is crucial to prevent over-pooling, which could lead to the loss of vital image details. The *InsertFC*() function adds a fully connected layer to the individual architecture, with the number of neurons ranging from 2 to a maximum of n_{max}. It's noteworthy that the initialization method is adapted from PSOCNN (Fernandes Junior & Yen, 2019) and has been modified to meet the specific requirements of the current research. Algorithm 2 orchestrates these functions to generate the initial population of ConvNet architectures.

Input: Eco size (n), maximum number of layers (l_{max}), maximum number of filters (nf_{max}), maximum filter size (f_{max}), maximum number of neurons in F layer (n_{max}), number of outputs (n_{op}).

Output: A population of ($ecosize$) organisms $eco = \{o_1, \ldots, o_n\}$

```
1 For $i = i$ to $n$ do
2   $o_i.lenght = rand(3, l_{max})$;
3   for $j = 1$ to $o_i.lenght$ do
4   If $j == 1$ then
5   $list\_layers[j] \leftarrow InsertConv(f_{max}, nf_{max})$;
6   else if $j == o_i.lenght$ then
7   $list\_layers[j] \leftarrow InsertF(n_{op})$;
8   else if $list\_layers[j-1].type == "fully-connected"$ then
9   $list\_layers[j] \leftarrow InsertF(n_{max})$;
```

(continued on next page)

(*continued*)

```
10  else
11  layer_type ← rand(1, 3);
12  if layer_type == 1 then
13  list_layers[j] ← InsertConv($f_{max}$, $nf_{max}$);
14  else if layer_type == 2 then
15  list_layers[j] ← InsertPool();
16  else
17  list_layers[j] ← InsertFC($n_{max}$);
18  end
19  end
20  end
21  $org_i$. list_layers ← list_layers;
22 end
23 return eco = {$o_1$, …, $o_n$}
```

3.3. Organism update operators

This section elaborates on the operators employed for updating individual architectures in alignment with the three phases of the SOS algorithm. The key operators utilized during the mutualism and commensalism phases are the sum and the difference operators, while the average operator plays a crucial role in generating mutual vectors at the Mutualism phase.

3.3.1. Difference operator

To measure the difference between two individual architectures, the approach outlined in Fernandes Junior and Yen (2019) was adopted and subsequently modified. First, each randomly selected individual is partitioned into two sections: one section comprises the convolution and pooling layers, and the other section encompasses the fully connected layers. This separation ensures that no fully connected layer is interposed between other layers, thereby preventing the formation of invalid architectures. This separation facilitates the computation of the difference between the convolution/pooling layers and the fully connected layers independently. The fully connected layer representing the output is isolated. The computation of the difference between two individuals, as outlined in algorithm 3, takes into consideration the layer types under the following conditions:

1. If both individuals have convolution layers at the current position, the difference is computed as 0, disregarding hyperparameters. This indicates to the update operator that the layer at this position should remain unchanged, preserving the hyperparameters of the corresponding layer in the first individual
2. If the layer types differ, the layer from the first individual is retained along with its hyperparameters
3. If the first individual has more layers than the second individual, an extra layer is appended to the final difference. The additional layer is randomly chosen from convolution, pooling, or fully connected layers. Conversely, if the first individual has fewer layers than the second individual, −1 is appended to the final result, signaling to the update operator to discard the layer at this position. The algorithm outlining the difference procedure is presented in Algorithm 3.

Input: two architectures $o1$, $o2$
Output: an architecture

```
1   If ($o_1$.length > $o_2$.length)
2   O = $o_1$.length
3   else if ($o_1$.length < $o_2$.length)
4   O = $o_2$.length
5   else
6   O = $o_1$.length
7   end if
8   ConvNetDiff ← []
9   For i = 1 to O do
10  If ($o1_{[i]}$ is not Empty and $o2_{[i]}$ is not Empty)
11  If($o1_{[i]}$.type = = $o2_{[i]}$.type)
```

(*continued on next column*)

(*continued*)

```
12  $o1_{[i]}$.type="keep"
13  ConvNetDiff ← $o1_{[i]}$
14  else
15  ConvNetDiff ← $o1_{[i]}$
16  else if ($o1_{[i]}$ is not Empty and $o2_{[i]}$ is Empty)
17  ConvNetDiff ← $o1_{[i]}$
18  else if ($o1_{[i]}$ is Empty and $o2_{[i]}$ is not Empty)
19  $o2_{[i]}$.type = "remove"
20  end if
21  Return ConvNetDiff
```

3.3.2. Average operator for mutual vector

The mutual vector operation in SOS_ConvNet, as outlined in Algorithm 4, is designed to calculate the mean of two chosen individuals. This involves implementing an average operator to compute the mean of hyperparameter values specifically within the convolution layers of the selected architectures. The exclusive application of the average operator to convolution layers is based on their paramount importance in a ConvNet. The average operation between two selected individuals involves calculating the mean of their kernel counts (number of kernels). It is crucial to emphasize that the previously discussed split operation is also applied at this stage. The average operation between layers can be formulated as follows:

$$MV(C_i, C_j) = \frac{KernelCnt(C_i) + KernelCnt(C_j)}{2} \tag{9}$$

Here, $KernelCnt(C_i)$ represents the number of kernels in the given convolution layer of the first selected architecture, and $KernelCnt(C_j)$ represents the number of kernels in the corresponding convolution layer of the second selected architecture. When the respective layers are not the same, the convolution layer with its features is chosen, given its high significance in feature extraction. In situations where one of the architectures has an empty layer while the other does not, the non-empty layer is added, and its features are retained.

Input: two architectures o_1, o_2
Output: Architecture representing the average between two architectures

```
1   If ($o_1$.length > $o_2$.length)
2   O = $o_1$.length
3   else If ($o_1$.length < $o_2$.length)
4   O = $o_2$.length
5   else
6   O = $o_1$.length
7   Endif
8   layersList ← []
9   For i = 1 to O do
10  If ($o1_{[i]}$.type is not Empty and $o2_{[i]}$.type is not Empty)
11  If($o1_{[i]}$.type == "C" and $o2_{[i]}$.type = = "C")
12  o1.kernelCount = ($o1_{[i]}$.kernelCount + $o2_{[i]}$.kernelCount)/2
13  layersList ← o1
14  Kerneldim.average
15  Strides.average
16  else if ($o1_{[i]}$.type == "C" and $o2_{[i]}$.type = = "P")
17  layersList ← $o1_{[i]}$
18  else if($o1_{[i]}$.type == "P" and $o2_{[i]}$.type = = "C")
19  layersList ← $o2_{[i]}$
20  else (tupleaverage.poolwindow o1, poolwindow o2
21  end if
22  else if ($o1_{[i]}$.type is not Empty and $o2_{[i]}$.type is Empty)
23  layersList ← $o1_{[i]}$
24  else if ($o1_{[i]}$.type is Empty and $o2_{[i]}$.type is not Empty)
25  layersList ← $o2_{[i]}$
26      endif
```

3.3.3. Novel sum operator

The addition operation is a fundamental operation extensively utilized in the SOS algorithm, applied during both the Mutualism and Commensalism phases. To implement the summation of individual ar-

Algorithm 4
The average operator algorithm

```
Input: two architectures $o_1, o_2$
Output: Architecture representing the average between two architectures
1   If ($o_1.length > o_2.length$)
2       $O = o_1.length$
3   else If ($o_1.length < o_2.length$)
4       $O = o_2.length$
5   else
6       $O = o_1.length$
7   Endif
8   $layersList \leftarrow [\,]$
9   For $i = 1$ to $O$ do
10      If ($o1_{[i]}.type$ is not Empty and $o2_{[i]}.type$ is not Empty)
11          If($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "C"$)
12              $o1.kernelCount = (o1_{[i]}.kernelCount + o2_{[i]}.kernelCount)/2$
13              $layersList \leftarrow o1$
14              Kerneldim.average
15              Strides.average
16          else if ($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "P"$)
17              $layersList \leftarrow o1_{[i]}$
18          else if($o1_{[i]}.type == "P"$ and $o2_{[i]}.type == "C"$)
19              $layersList \leftarrow o2_{[i]}$
20          else (tupleaverage.poolwindow $o1$, poolwindow $o2$
21          end if
22      else if ($o1_{[i]}.type$ is not Empty and $o2_{[i]}.type$ is Empty)
23          $layersList \leftarrow o1_{[i]}$
24      else if ($o1_{[i]}.type$ is Empty and $o2_{[i]}.type$ is not Empty)
25          $layersList \leftarrow o2_{[i]}$
26      endif
```

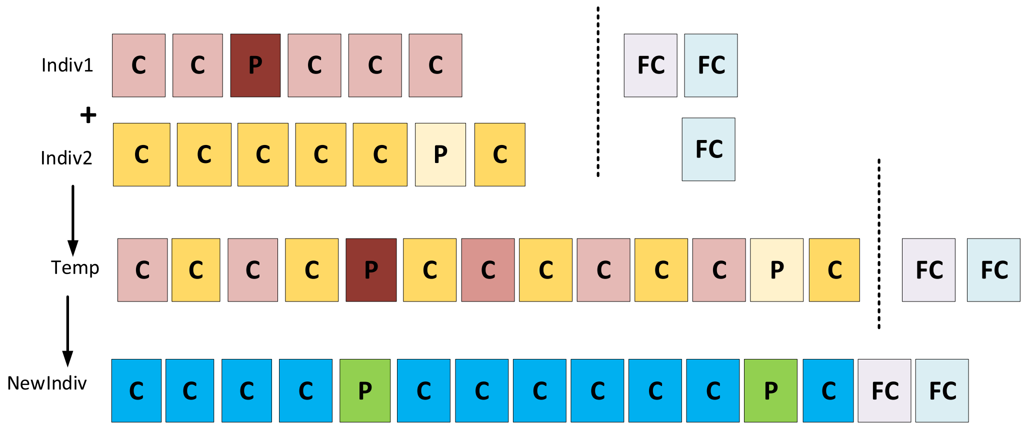


Fig. 3. Addition of two individuals.

chitectures, a simple arithmetic method is proposed. When two individuals are selected for summation, their layers are combined following the elementary arithmetic rule of $x + x = 2x$. This implies, for any two randomly selected architectures, the addition operator adds the respective layers of the two architectures. The length of the resulting architecture equals the total length of the two architectures. For instance, if two architectures with lengths of 5 and 4 are selected, the resultant architecture generated by the sum operator will have a length of 9, representing the total combined layers. In essence, this operation is analogous to the union operation in set theory. To maintain the validity and adherence to architectural conventions, the split operation is implemented at this stage. This process ensures that the resulting architecture remains structurally sound. A visual representation of the sum operation is illustrated in Fig. 3, and the operational process is detailed in Algorithm 5.

Input: two architectures o1, o2
Output: convNet

```
If ($o_1.length > o_2.length$)
$O = o_1.length$
else if ($o_1.length < o_2.length$)
$O = o_2.length$
else
$O = o_1.length$
end if
$ConvNetSum \leftarrow []$
For $i = 1$ to $O$ do
If ($o1_{[i]}.type$ is not Empty and $o2_{[i]}.type$ is not Empty)
If($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "C"$)
$ConvNetSum.extend(o1_{[i]}, o2_{[i]}$
else if ($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "keep"$)
$ConvNetSum.append(o1_{[i]})$
else if ($o1_{[i]}.type == "keep"$ and $o2_{[i]}.type == "C"$)
$ConvNetSum.append(o2_{[i]})$
else if ($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "remove"$)
Pass
else if ($o1_{[i]}.type == "remove"$ and $o2_{[i]}.type == "C"$)
Pass
else if ($o1_{[i]}.type == "P"$ and $o2_{[i]}.type == "C"$)
$ConvNetSum.extend(o1_{[i]}, o2_{[i]}$
else if ($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "P"$)
$ConvNetSum.extend(o1_{[i]}, o2_{[i]})$
else if ($o1_{[i]}.type == "P"$ and $o2_{[i]}.type == "P"$)
$ConvNetSum.extend(o1_{[i]}, o2_{[i]})$
else if ($o1_{[i]}.type == "P"$ and $o2_{[i]}.type == "keep"$)
$ConvNetSum.append(o1_{[i]})$
else if ($o1_{[i]}.type == "keep"$ and $o2_{[i]}.type == "P"$)
$ConvNetSum.append(o2_{[i]})$
else if ($o1_{[i]}.type == "P"$ and $o2_{[i]}.type == "remove"$)
Pass
else if ($o1_{[i]}.type == "remove"$ and $o2_{[i]}.type == "P"$)
Pass
end if
else if($o1_{[i]}.type$ is not Empty and $o2_{[i]}.type$ is Empty)
$ConvNetSum.append(o1_{[i]})$
else if($o1_{[i]}.type$ is Empty and $o2_{[i]}.type$ is not Empty)
if($o2_{[i]}.type == "keep"$)
pass
else if($o2_{[i]}.type == "remove"$)
pass
else
$ConvNetSum.append(o2_{[i]})$
end if
end if
Return $ConvNetSum$
```

3.3.4. *Mutation operator for parasite vector generation*

In the main SOS algorithm, a randomly selected organism within the ecosystem is subject to modification to generate the parasite vector. Subsequently, a second individual, also randomly chosen, is compared with the parasite vector. The outcome dictates the survival of the best individual, while the other ceases to exist in the ecosystem. To implement the creation of parasite ConvNet as the parasite vector, an individual architecture is randomly selected for modification. This modification encompasses actions such as adding or removing a layer, or altering the type of a layer. During the generation of the parasite, the architecture is split into two distinct sections: the Conv/Pool section, comprising convolution and pooling layers, and the FC section, housing the fully connected layers. The mutation of the selected architecture solely occurs within the Conv/Pool section. The process involves a randomized selection of one of the following actions: adding a layer, removing a layer, or changing the layer type. If the chosen action is to add a layer, either a pooling or convolution layer is automatically inserted at the end of the Conv/Pool section. Conversely, in the case of choosing to remove a layer, a layer is randomly selected from the architecture and discarded. Otherwise, if the selected option is to change a layer type, a layer is randomly selected and its type is modified (e.g., from *Conv to Pool, or Pool to Conv*). It is important to note that a higher probability is assigned to the removal or changing options, particularly for pooling layers. This emphasis on reducing the number of pooling layers aligns with the objective of minimizing the architecture's pooling layers, as mentioned earlier.

Algorithm 5
ConvNetSum()
Input: two architectures o1, o2
Output: convNet

```
If ($o_1.length > o_2.length$)
    $O = o_1.length$
else if ($o_1.length < o_2.length$)
    $O = o_2.length$
else
    $O = o_1.length$
end if
$ConvNetSum \leftarrow [\,]$
For $i = 1$ to $O$ do
    If ($o1_{[i]}.type$ is not Empty and $o2_{[i]}.type$ is not Empty)
        If($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "C"$)
            $ConvNetSum.extend(o1_{[i]}, o2_{[i]}$
        else if ($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "keep"$)
            $ConvNetSum.append(o1_{[i]})$
        else if ($o1_{[i]}.type == "keep"$ and $o2_{[i]}.type == "C"$)
            $ConvNetSum.append(o2_{[i]})$
        else if ($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "remove"$)
            Pass
        else if ($o1_{[i]}.type == "remove"$ and $o2_{[i]}.type == "C"$)
            Pass
        else if ($o1_{[i]}.type == "P"$ and $o2_{[i]}.type == "C"$)
            $ConvNetSum.extend(o1_{[i]}, o2_{[i]}$
        else if ($o1_{[i]}.type == "C"$ and $o2_{[i]}.type == "P"$)
            $ConvNetSum.extend(o1_{[i]}, o2_{[i]})$
        else if ($o1_{[i]}.type == "P"$ and $o2_{[i]}.type == "P"$)
            $ConvNetSum.extend(o1_{[i]}, o2_{[i]})$
        else if ($o1_{[i]}.type == "P"$ and $o2_{[i]}.type == "keep"$)
            $ConvNetSum.append(o1_{[i]})$
        else if ($o1_{[i]}.type == "keep"$ and $o2_{[i]}.type == "P"$)
            $ConvNetSum.append(o2_{[i]})$
        else if ($o1_{[i]}.type == "P"$ and $o2_{[i]}.type == "remove"$)
            Pass
        else if ($o1_{[i]}.type == "remove"$ and $o2_{[i]}.type == "P"$”)
            Pass
        end if
    else if($o1_{[i]}.type$ is not Empty and $o2_{[i]}.type$ is Empty)
        $ConvNetSum.append(o1_{[i]})$
    else if($o1_{[i]}.type$ is Empty and $o2_{[i]}.type$ is not Empty)
        if($o2_{[i]}.type == "keep"$)
            pass
        else if($o2_{[i]}.type == "remove"$)
            pass
        else
            $ConvNetSum.append(o2_{[i]})$
        end if
    end if
Return $ConvNetSum$
```

3.4. *Fitness evaluation*

In the process fitness evaluation, conventional metaheuristic algorithms often consider individual with the minimum objective function value as the best individual. In this study, the objective function is represented by the loss function, specifically the cross-entropy loss

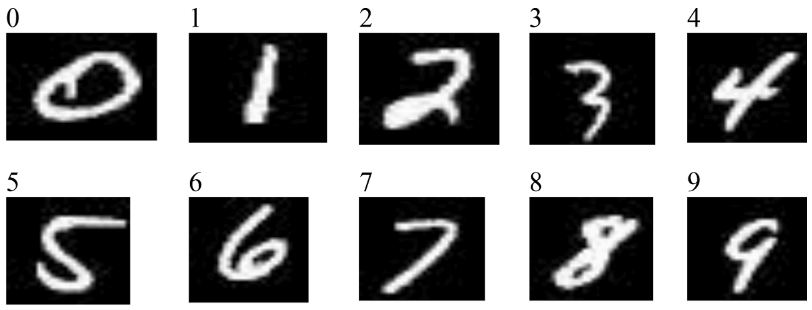


Fig. 4. Sample images from MNIST dataset.

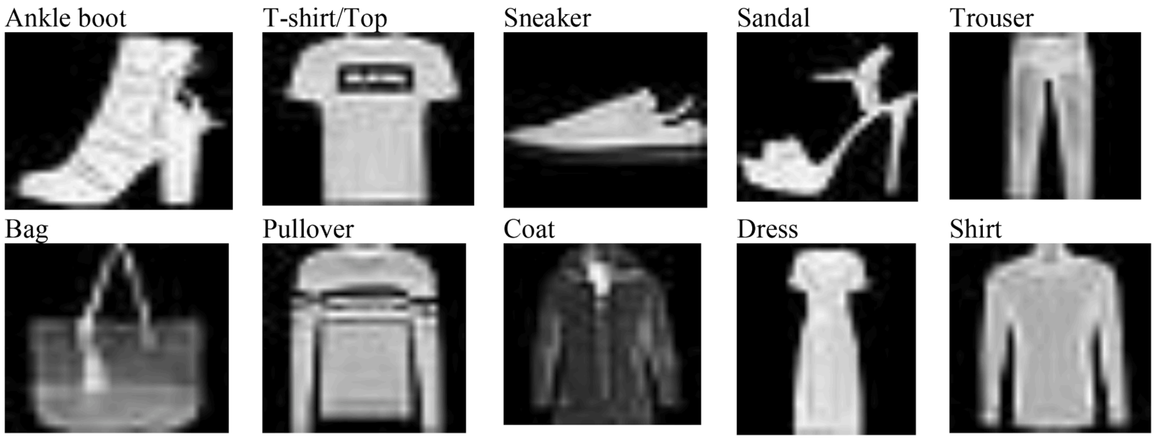


Fig. 5. Sample images from Fashion-MNIST dataset.

Fig. 6. Sample images from CIFAR-10 dataset.

function. Therefore, the fitness evaluation involves the comparison of the loss function for each individual based on the cross-entropy loss function. Consequently, the algorithm strives to identify an individual architecture with the lowest loss value. This is accomplished through a function called *evolve()*. At this stage, each individual architecture is compiled into a complete ConvNet and subjected to training using a specified data set for a predetermined number of epochs and iterations to obtain the corresponding the loss value. The training process employs the Adam optimizer and Xavier weight initializer. The choice of Adam optimizer and Xavier weight initializer is grounded in their proven ability to expedite training, and enhance the convergence properties of deep neural networks. These techniques have gained widespread adoption in the deep learning community due to their empirical success across various applications. However, a major challenge is that each and every individual architecture in the ecosystem has to be trained with the same dataset which is a computationally intensive task.

4. Experiment set up

The proposed algorithms for SOS_ConvNet were implemented using Python, utilizing the Google Colab platform for execution and experimentation. This study adhered to a meticulous approach, involving thorough testing and evaluation of SOS_ConvNet across diverse datasets. The subsequent subsections offer comprehensive insights into the datasets utilized and the specific parameter settings employed in the experimental procedures.

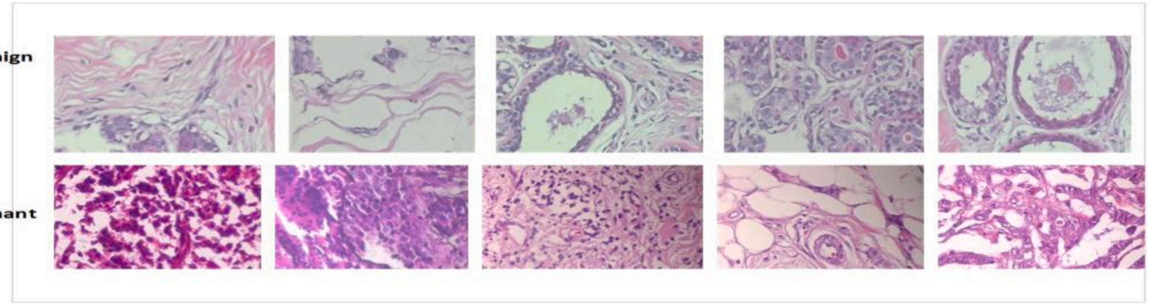


Fig. 7. Sample images from BreakHis dataset.

Table 1
Hyperparameter Settings.

SOS settings		ConvNet initialization settings		ConvNet training settings	
Eco size	20	Minimum number of layers	3	Individual optimization epochs	3
Number of iterations	10	Maximum number of layers	20	Training epochs for best ConvNet	300
		Minimum kernel size for Conv layers	3 × 3	Dropout rate	0.4
		Maximum kernel size f or Conv layers	7 × 7	Batch normalization layers	yes
		Minimum number of filters	3		
		Maximum number of filters	256		
		Minimum number of neurons in FC layer	1		
		Maximum number of neurons in FC layer	300		

4.1. Datasets

The study rigorously tested the proposed SOS_ConvNet algorithm across four distinct benchmark datasets: MNIST, Fashion-MNIST, BreakHis, and CIFAR-10. These datasets were carefully chosen to cover a range of image classification tasks. Through systematic experiments and evaluations on these datasets, the research validates and establishes the reliability of the proposed NAS algorithm. The obtained results underscore the algorithm's capacity to generate convolutional neural network architectures that exhibit outstanding performance across diverse image classification domains. **MNIST Dataset:** MNIST is a well-known dataset containing grayscale images of handwritten digits (0–9), each sized at 28 × 28 pixels. It stands as a cornerstone for image classification tasks, featuring 60,000 training samples and 10,000 testing samples. Exemplary samples from the MNIST dataset are illustrated in Fig. 4.

Fashion-MNIST Dataset: Fashion-MNIST, tailored for image classification tasks, stands as another prominent dataset in the study. It encompasses grayscale images featuring various fashion items, including clothing and accessories. Similar to MNIST, this dataset incorporates 60,000 training samples and 10,000 testing samples. Visual representations of sample images from the Fashion-MNIST dataset are depicted in Fig. 5.

CIFAR-10 Dataset: CIFAR-10, renowned for its complexity, features coloured images categorized into ten classes, including airplanes, automobiles, and various animals. Comprising 60,000 training images and 10,000 testing images, this dataset poses a significant challenge for image classification algorithms. Fig. 6 provides visual samples from the CIFAR-10 dataset

BreakHis Dataset: The BreakHis dataset is a specialized medical dataset tailored for breast cancer diagnosis. It encompasses microscopic breast histopathology images, meticulously classified into benign and malignant classes. This dataset serves as a pivotal benchmark for assessing the efficacy of SOS_ConvNet in the domain of medical image classification. Fig. 7 showcases representative samples from the BreakHis dataset.

4.2. Parameter settings

The success of SOS_ConvNet relies on carefully configured hyperparameters that govern various aspects of the SOS algorithm, ConvNet initialization, and training. Table 1 encapsulates the hyperparameter settings utilized in the experiments, providing insights into critical aspects of the optimization process. In the first column, parameters steering the behavior of the SOS algorithm are delineated. The basic SOS require few parameters setting that is the ecosystem size (eco size). Similarly, in the proposed SOS_ComvNet, the "eco size" determines the number of ConvNet architectures under consideration. Simultaneously, the "number of iterations" is required by all optimization algorithms to mark the termination point of the optimization.

The second group or column outlines the hyperparameters necessary for the initial generation of the organism population. These hyperparameters set boundaries for the random architectures within the initial population, establishing the foundation for subsequent evolutionary processes. The final group of parameters presented in the third column governs the training of the architectures. The "individual optimization epochs" hyperparameter dictates the number of epochs for which the architectures are trained during the optimization process. Additionally, the "training epochs for best ConvNet" hyperparameter specifies the number of epochs dedicated to training the globally recognized best architecture. Furthermore, batch normalization and dropout rates have been included as essential components to mitigate overfitting.

5. Results and discussion

In this section, the results obtained from the proposed SOS_ConvNet are presented and analyzed. The SOS_ConvNet model was evaluated using various datasets, and thus, the best model generated by SOS_ConvNet for each dataset is showcased. Additionally, a thorough comparison is conducted, comparing the performance of the SOS_ConvNet models against established models from the literature. The selection of compared models was driven by their prevalence in the NAS literature, representing some of the most common NAS models. Additionally, we included popular handcrafted models for a comprehensive comparison. This evaluation approach ensures a well-rounded analysis by considering both state-of-the-art NAS models and widely recognized handcrafted models, providing a comprehensive perspective on the performance of our proposed SOS_ConvNet. For each dataset, the evaluation results highlight the strengths and capabilities of the best SOS_ConvNet model. The evaluation results are discussed in detail, emphasizing the advantages and improvements achieved by the

Table 2
Description of the model generated on MNIST dataset.

Layer	Hyperparameters
Conv	Kernel size = 6 × 6, No. of kernels = 216
Conv	Kernel size = 5 × 5, No. of kernels = 25
Pooling	Pool window = 3 × 3
Pooling	Pool window = 3 × 3
Conv	Kernel size = 7 × 7, No. of kernels = 101
Conv	Kernel size = 7 × 7, No. of kernels = 11
Conv	Kernel size = 7 × 7, No. of kernels = 11
Pooling	Pool window = 3 × 3
FC	Number of neurons = 10

SOS_ConvNet models in terms of classification accuracy, model complexity, or other relevant factors.

5.1. Results on MNIST dataset

To evolve models using the MNIST dataset, the algorithm was trained for 3 epochs and 10 iterations. Subsequently, the global best architecture at the end of iterations was further trained for 300 epochs. The details of the best model obtained is shown in Table 2.

Fig. 8 illustrates the training and testing process of the SOS_ConvNet model over 300 epochs. The curves displayed in the figure demonstrate the model's performance throughout the training and testing phases. The absence of significant deviations or irregularities in these curves indicates that the proposed model does not suffer from overfitting or underfitting issues. This observation confirms that the generated ConvNet is capable of generalizing well to unseen data, ensuring reliable and consistent image classification.

The proposed SOS_ConvNet algorithm achieved remarkable results with the global best architecture, achieving an impressive accuracy of 0.9969, equivalent to 99.69 %. The corresponding error rate was calculated to be 0.0031, which translates to an error rate of 0.31 %. These results demonstrate the high precision and effectiveness of the SOS_ConvNet model in accurately classifying images.

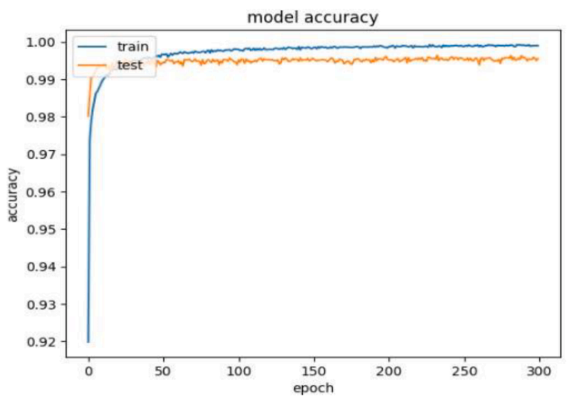
Table 3 provides a comprehensive comparison of the proposed model with other algorithms based on the reported error rates as percentages. The comparison aims to assess the performance of the proposed algorithm relative to the compared approaches. Notably, the compared algorithms consist of both manually generated models such as LeNet5, and automatically generated models using population-based algorithms, including IPPSO, PSO_CNN, and SOSCNN. The comparison is based on results reported in the literature. Based on the error rates reported in Table 3, the following analysis can be made: LeNet5 (LeCun et al., 1998) is a manually generated model with an error rate of 0.95 %. The proposed SOS_ConvNet algorithm outperforms LeNet5, achieving a lower error rate of 0.31 %. This indicates that the automatically generated SOS_ConvNet model is more effective in image classification compared to the manually designed LeNet5. The error rate reported for IPPSO (Wang et al., 2018) is 1.13 %. In comparison, the proposed SOS_ConvNet algorithm demonstrates superior performance with an error rate of 0.31 %. This shows that the SOS_ConvNet outperforms IPPSO in terms of classification accuracy. The PSO_CNN (Fernandes Junior & Yen, 2019) algorithm achieved an error rate of 0.32 %. The proposed SOS_ConvNet algorithm performs competitively, exhibiting a slightly lower error rate of 0.31 %. This suggests that both algorithms are effective in image classification, with the SOS_ConvNet demonstrating comparable performance to PSO_CNN. The SOSCNN (Miao et al., 2021) algorithm reported an error rate of 0.38 %. Once again, the proposed SOS_ConvNet algorithm outperforms SOSCNN achieving a lower error rate of 0.31 %. This indicates that the SOS_ConvNet approach is more successful in generating ConvNets with higher accuracy for image classification tasks. This comparison underscores the effectiveness of the SOS_ConvNet algorithm in generating ConvNets with superior accuracy for image classification tasks. The results showcase its competitive performance against both manually designed models and other automated algorithms based on population-based techniques. The high accuracy achieved by the global best architecture, coupled with the absence of overfitting or underfitting issues, reinforces the reliability and robustness of the proposed SOS_ConvNet model for image classification tasks.

Table 3
Comparison results of the proposed algorithm on MNIST dataset.

Model	Error (%)
LeNet5 (LeCun et al., 1998)	0.95
IPPSO (Wang et al., 2018)	1.13
PSO_CNN (Fernandes Junior & Yen, 2019)	0.32
SOSCNN (Miao et al., 2021)	0.38
Proposed SOS_ConvNet	**0.31**

Table 4
Layers and hyperparameters of the Global Best Model on Fashion-MNIST Dataset.

Layer	Hyperparameters
Conv	Kernel size = 4 × 4, No. of kernels = 161
Conv	Kernel size = 4 × 4, No. of kernels = 86
Pooling	Pool window = 3 × 3
Conv	Kernel size = 5 × 5, No. of kernels = 4
FC	No. of neurons = 512
$FC_{(output)}$	Number of neurons = 10



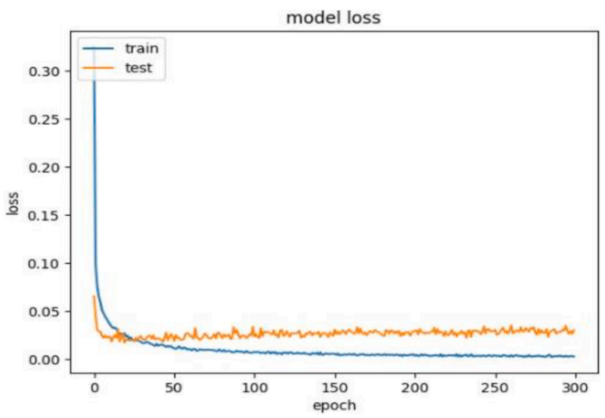


Fig. 8. Training curves of SOS_ConvNet on MNIST dataset.

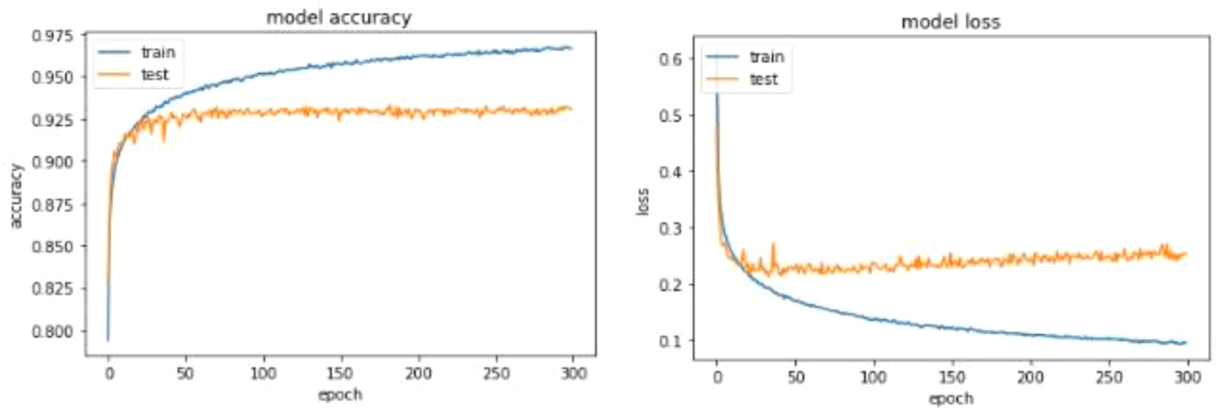


Fig. 9. Training and Testing Process of the Global Best Model on Fashion_MNIST dataset.

Table 5
Comparison results of the proposed algorithm on Fashion_ MNIST dataset.

Model	Error (%)	Parameters in millions
AlexNet*	10.1	62.3
Vgg_16*	6.5	26.0
MobileNet*	**5.0**	4.0
GoogleNet*	6.3	23.0
Evo_CNN (Sun et al., 2019)	5.47	6.68
PSO_CNN (Fernandes Junior & Yen, 2019)	5.53	2.32
SOSCNN (Miao et al., 2021)	5.68	2.30
Proposed SOS_ConvNet	6.7	**0.24**

* https://github.com/z alandoresearch/fashion-mnist.

Table 6
Layers and Hyperparameters of the Global Best Model on CIFAR-10 dataset.

Layer	Hyperparameters
Conv	Kernel size = 3 × 3, No. of kernels = 52
Conv	Kernel size = 3 × 3, No. of kernels = 95
Conv	Kernel size = 4 × 4, No. of kernels = 95
Pooling	Pool window = 2 × 2
Conv	Kernel size = 3 × 3, No. of kernels = 128
Conv	Kernel size = 3 × 3, No. of kernels = 128
Conv	Kernel size = 3 × 3, No. of kernels = 128
FC	Number of neurons = 512
$FC_{(output)}$	Number of neurons = 10

Table 7
Models evaluation results on CIFAR-10 Dataset.

Model	Train Accuracy (%)	Test Accuracy (%)	Time (minutes)
VGG16	97.48	67.62	4.99
VGG19	97.68	67.93	4.45
AlexNet	98.54	65.87	11.66
MobileNet	84.10	82.18	25.55
ResNet20	84.97	82.08	20.62
SOS_ConvNet	84.52	**82.78**	22.50

5.2. Results on fashion_ mnist dataset

The best-performing model generated by SOS_ConvNet showcased exceptional results, prompting a detailed comparison with published outcomes from other models based on neural architecture generation and manually designed models. The specifics of the top-performing model obtained through SOS_ConvNet are meticulously presented in Table 4. This comparison provides valuable insights into the efficacy of SOS_ConvNet in achieving remarkable performance relative to existing models in the domain of neural architecture search.

The training and testing curves, as depicted in Fig. 9, present the performance of the proposed SOS_ConvNet model on the Fashion_-MNIST dataset. These curves serve as valuable indicators of the model's learning dynamics and its capacity to generalize to previously unseen data. Analyzing the training curves reveals a consistent improvement in accuracy over the course of the training process. The training accuracy progressively rises to a commendable value of 0.9722, equivalent to 97.22 %. This signifies the model's adeptness in learning from the training data, effectively capturing intrinsic patterns and features crucial for accurate classification.

The test curve reveals the model's proficiency on previously unseen test data. The highest test accuracy achieved by the model is 0.9330, equivalent to an impressive accuracy rate of 93.30 %. This demonstrates that the model can successfully generalize its learnings to new and unseen examples, achieving a high level of accuracy in effectively classifying images from the Fashion_MNIST dataset. The error rate for the test accuracy is computed as 0.067, equivalent to a commendably low rate of 6.7 %. This signifies that the model exhibits a relatively low level of misclassification, further highlighting its efficacy in accurately predicting the class labels of the Fashion-MNIST images. Importantly, the minimal discrepancy between the training and testing accuracy suggests that the model does not suffer from overfitting. Overfitting occurs when a model performs exceptionally well on the training data but fails to generalize to new data. The results were benchmarked against PSO_CNN (Fernandes Junior & Yen, 2019), SOSCNN (Miao et al., 2021), Evo_CNN (Sun et al., 2019), AlexNet, VGG16, MobileNet, and GoogleNet.

Table 5 provides a comprehensive comparison between the proposed SOS_ConvNet model and other existing models on the Fashion_MNIST dataset. The comparison considers both the reported error rates and the number of parameters (the weights) in millions. An analysis of the results reveals the following insigths: AlexNet achieves an error rate of 10.1 % on the Fashion_MNIST dataset, with a relatively high parameter count of 62.3 million. AlexNet is outperformed by the proposed SOS_ConvNet, which demonstrates a lower error rate of 6.7 % and a significantly reduced parameter count of 0.24 million. Vgg_16 achieved an error rate of 6.5 % with a parameter count of 26.0 million. The proposed SOS_ConvNet model exhibits slightly higher error rate (6.7 %) but stands out for its efficiency, boasting a much smaller parameter count of 0.24 million. This indicates that the proposed model offers competitive performance while being more parameter-efficient. MobileNet achieves a better error rate of 5.0 %, the proposed SOS_ConvNet, with its higher error rate of 6.7 %, stands out for its remarkable parameter efficiency, featuring only 0.24 million parameters compared to MobileNet's 4.0 million. This emphasizes the compact and lightweight nature of the SOS_ConvNet architecture. GoogleNet achieved an error rate of 6.3 % with a parameter count of 23.0 million. The proposed SOS_ConvNet model exhibits a slightly higher error rate of 6.7 % but significantly reduced parameter counts of 0.24 million. This underscores the potential of the proposed model to achieve comparable performance

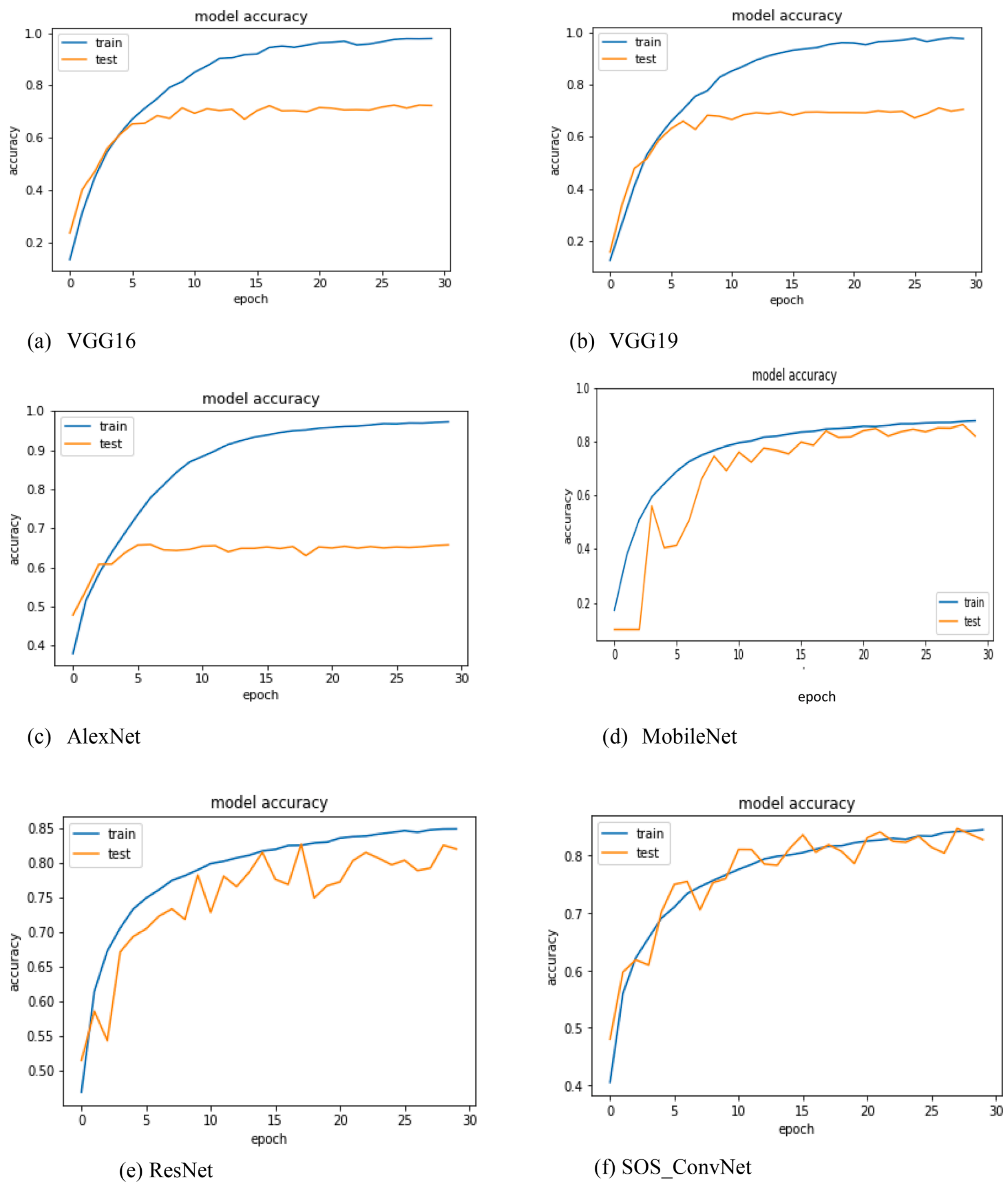


Fig. 10. Training and testing processes of the compared models on CIFAR-10 dataset.

to GoogleNet while featuring a simpler and more resource-efficient architecture.

Evo_CNN achieves a lower error rate of 5.47 %, the proposed SOS_ConvNet, with its 6.7 % error rate, stands out for its exceptional parameter efficiency, having only 0.24 million parameters compared to Evo_CNN's 6.68 million. This suggests that SOS_ConvNet can offer comparable performance with significantly fewer parameters, emphasizing its efficiency and suitability for resource-constrained scenarios. PSO_CNN achieves an error rate of 5.53 % with a parameter count of 2.32 million. The proposed SOS_ConvNet model with a marginally higher error rate of 6.7 %, showcases superior parameter efficiency, boasting only 0.24 million parameters. This emphasizes the compact architecture and resource efficiency of SOS_ConvNet, making it a favourable choice for scenarios where minimizing parameters is crucial. SOSCNN achieves an error rate of 5.68 % and a parameter count of 2.30 million, exhibiting a comparable performance profile to PSO_CNN. However, the proposed SOS_ConvNet, with its slightly higher error rate of 6.7 % and significantly fewer parameters (0.24 million), offers a more parameter-efficient alternative. This underscores SOS_ConvNet's ability to strike a balance between error rate and model simplicity. The

Table 8
Layers and Hyperparameters of the Global Best Model on BreakHis dataset.

Layer	Hyperparameters
Conv	Kernel size = 4 × 4, No. of kernels = 15
Pooling	Pool window = 3 × 3
Conv	Kernel size = 3 × 3, No. of kernels = 11
Pooling	Pool window = 3 × 3
Conv	Kernel size = 3 × 3, No. of kernels = 17
FC	Number of neurons = 17
$\text{FC}_{\text{(output)}}$	Number of neurons = 1

Table 9
Models evaluation results on BreakHis Dataset.

Model	Accuracy (%)	No. of Parameters	Training time
MobileNet	70.01	1103,345	20.10
EfficientNet	84.44	7267,317	21.33
VGG16	73.03	14,854,273	20.25
ResNet	85.25	24,097,601	22.22
SOS_ConvNet	**89.12**	**38,868**	**18.33**

comparison in Table 5 demonstrates that the proposed SOS_ConvNet model achieves competitive performance on the Fashion_MNIST dataset. Despite having a slightly higher error rate compared to some existing models, it stands out by offering a significantly smaller number of parameters. This indicates that the proposed model can achieve comparable accuracy while being more parameter-efficient and potentially more computationally efficient, making it an attractive solution for image classification tasks. Generally, the training and testing curves, along with the achieved accuracies and error rates, indicate that the proposed SOS_ConvNet model performs exceptionally well on the Fashion_MNIST dataset. Its high accuracy, minimal error rate, and absence of overfitting contribute to its reliability and effectiveness in accurately classifying images from the Fashion-MNIST dataset.

5.3. Results on CIFAR-10 dataset

The evaluation of the proposed SOS_ConvNet algorithm on the CIFAR-10 dataset involved a systematic experiment. Initially, architectures were generated using the dataset, and these architectures underwent optimization through a single epoch and 10 iterations to identify the global best architecture. Subsequently, the global best architecture was further trained on the CIFAR-10 dataset for a total of 30 epochs. The structural details of the resulting global best model are outlined in Table 6.

To conduct a comprehensive evaluation, well-known handcrafted models, including AlexNet, ResNet, VGG, and MobileNet, were trained on the same platform as the proposed model. This comprehensive evaluation considered key criteria, including both time efficiency and accuracy. The results obtained from this evaluation offer valuable insights into the effectiveness of the proposed model compared to these established models. The results of this evaluation are succinctly presented in Table 7.

From the results on Table 7; AlexNet, VGG19, and VGG16 achieved remarkable training accuracies of 98.54 %, 97.68 %, and 97.48 %, respectively. However, despite their impressive performance on the training data, these models paradoxically demonstrate lower accuracy when assessed on the test set. In fact, even the best-performing model, AlexNet, displayed the lowest overall test accuracy of 65.87 %. This consistent trend of high training accuracy coupled with low test accuracy strongly suggests that these models are suffering from overfitting when evaluated on the test data. MobileNet and ResNet, in contrast, demonstrated robust performance on both the training and testing datasets. MobileNet achieved training and test accuracies of 84.10 % and 82.18 %, respectively, while ResNet achieved training and test accuracies of 84.97 % and 82.08 %. Importantly, these models exhibited no apparent signs of overfitting, as indicated by the relatively small variation between their training and testing accuracies. This consistency in performance between the training and testing datasets suggests that MobileNet and ResNet have successfully learned to generalize from the training data to unseen examples. They appear to capture meaningful features and patterns that are applicable beyond the specific training set, resulting in reliable and consistent predictions on new data. The absence of overfitting is a positive indicator of these models' ability to generalize effectively.

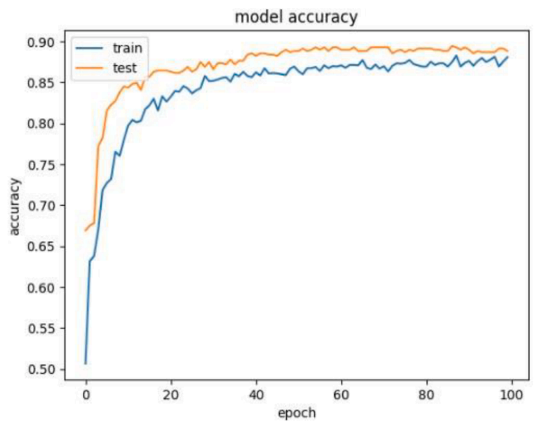
The model generated by the proposed SOS_ConvNet achieved an impressive test accuracy of 82.78 %, while its training accuracy stood at 84.52 %. Importantly, this performance indicates that no overfitting was observed in the SOS_ConvNet model. Furthermore, the training time required by SOS_ConvNet was reasonable, demonstrating that it can efficiently generate models with strong performance on a given dataset. These findings highlight the effectiveness of SOS_ConvNet in producing models that not only excel in terms of accuracy but also exhibit a balanced learning process. The absence of overfitting underscores the model's ability to generalize well to unseen data, while its efficient training time adds to its appeal as a practical and efficient approach for generating high-performing models for specific image classification tasks. In terms of training time, VGG19 emerged as the fastest model, requiring approximately 4 min and 45 s for training, followed closely by VGG16, which took approximately 5 min. In contrast, MobileNet had the longest training time, demanding approximately 25 min and 55 s. The proposed SOS_ConvNet model fell in between with a training time of 22.5 min. It's noteworthy that despite the moderate training time, the SOS_ConvNet model achieved a test accuracy comparable to that of MobileNet and ResNet20, even though its training accuracy was slightly lower. This balance between training time and performance is a notable advantage. In summary, the proposed SOS_ConvNet model effectively demonstrates its competence by achieving competitive test accuracy on the CIFAR-10 dataset while maintaining a reasonable training time. It performs on par with the MobileNet and ResNet20 models and surpasses the VGG16, VGG19, and AlexNet models in terms of test accuracy. The training and testing processes for each of these models are visually represented in Fig. 10, providing a comprehensive view of their learning curves and performance trends.

5.4. Results on breakhis dataset

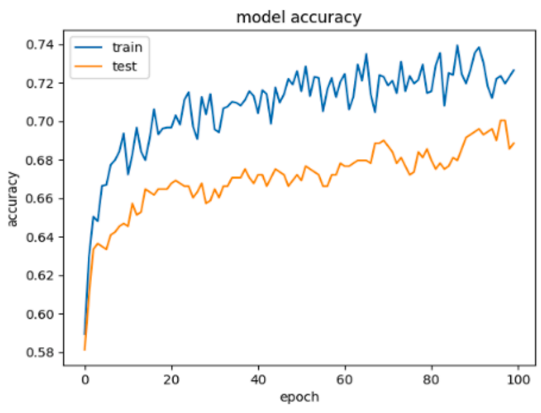
To evaluate the performance of the proposed SOS_ConvNet on the BreakHis dataset, a comprehensive experiment was conducted. Initially, architectures were generated using the BreakHis dataset, leveraging the capabilities of SOS_ConvNet's algorithm. These generated architectures were then optimized through a single epoch and 10 iterations to ultimately obtain the global best architecture, ensuring that it represents the most promising ConvNet structure for the given dataset. Subsequently, the global best architecture was subjected to further refinement through intensive training on the BreakHis dataset. This refinement involved training the global best architecture for a total of 100 epochs, allowing it to adapt and improve its performance on the dataset. The structure and configuration details of this global best model can be found in Table 8.

For the purpose of evaluation, popular handcrafted models; AlexNet (Krizhevsky et al., 2012), ResNet (He et al., 2016), VGG (Simonyan & Zissermann, 2015), EfficientNet (Tan & Le, 2019), and MobileNet (Howard et al., 2012) were trained on the same dataset and compared with the global best model. The results obtained are presented in Table 9.

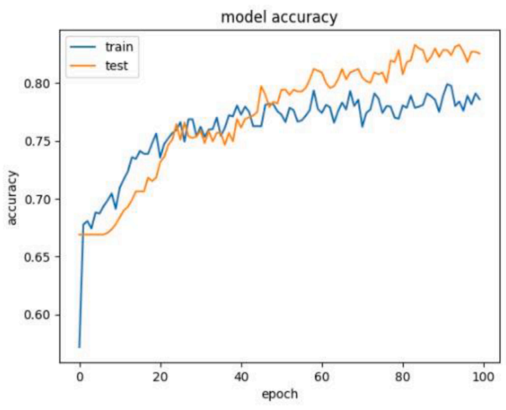
From the results presented in Table 9, SOS_ConvNet clearly stands out as the top-performing model on the BreakHis dataset, achieving the highest accuracy of 89.12 % while maintaining a compact model size of 38,868 parameters. MobileNet and EfficientNet, while having reasonable performance of 70.01 % and 84.44 % respectively, couldn't match



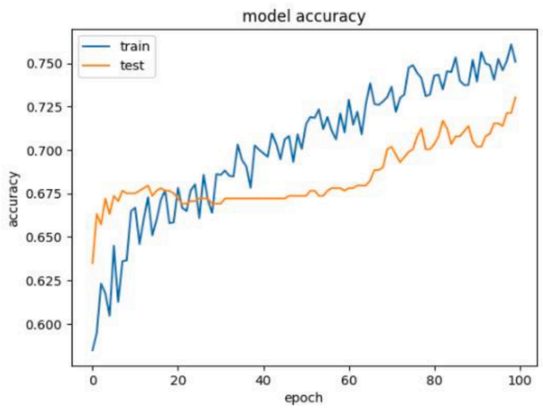
(a) SOS_ConvNet



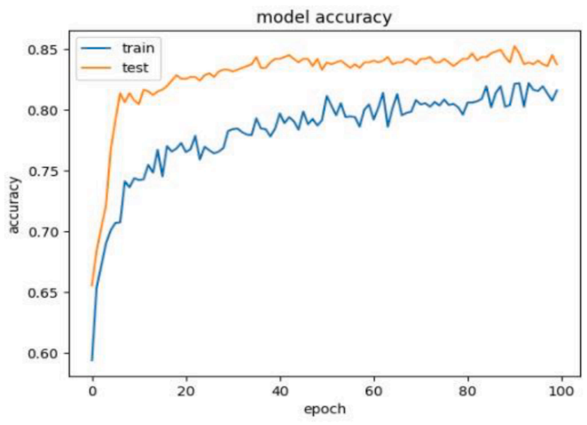
(b) MobileNet



(c) EfficientNet



(e) VGG16



(f) ResNet

Fig. 11. Training and testing process on BreakHis dataset.

the accuracy of SOS_ConvNet. Furthermore, EfficientNet had a substantially larger model size of 1103,345 parameters. VGG16 and ResNet, despite their higher parameter counts, achieved lower accuracy at 73.03 % and 85.25 % respectively, when compared to SOS_ConvNet. Significantly, SOS_ConvNet not only excelled in accuracy but also demonstrated efficiency by requiring less training time compared to the other models. This reinforces its effectiveness in generating high-performing and resource-efficient models for specific image classification tasks on the BreakHis dataset. In summary, SOS_ConvNet proves to be a compelling choice for the BreakHis dataset, offering an excellent balance of accuracy, model size, and training efficiency. This shows that many handcrafted models have a lot of unnecessary parameters despite their good performance. Having lighter models would save a lot of disk space, and computational power. The training curves of the models on BreakHis dataset are displayed on Fig.11.

6. Conclusion and future work

This research adopts a neural architecture search (NAS) methodology employing the innovative Symbiotic Organism Search (SOS) optimization algorithm to autonomously evolve Convolutional Neural Network (ConvNet) models. The study's approach involves evolving ConvNet architectures through SOS, selecting the best-performing models for further refinement and training. The proposed NAS algorithm's generalization capabilities were rigorously evaluated across diverse datasets, spanning domains such as handwritten digits (MNIST), fashion items (Fashion-MNIST), general image classification (CIFAR-10), and medical image analysis (Breast Cancer dataset). These datasets serve as a robust foundation for evolving optimal ConvNet models using the SOS_ConvNet approach. The generated architectures were subjected to comprehensive comparisons against existing models, encompassing both manually designed architectures and those derived from other NAS methodologies. The compelling comparative results underscore the superior performance of the models generated by the proposed SOS_ConvNet. This comprehensive analysis reaffirms the effectiveness and versatility of the SOS_ConvNet methodology in automatically generating ConvNet architectures that excel across diverse image classification tasks. By consistently outperforming established models, SOS_ConvNet demonstrates its potential to revolutionize neural architecture search, making significant contributions to the field of deep learning. The introduction of innovative operators, such as mutual vector, modified difference, parasite generator, and sum operators, enhances the efficiency of ConvNet generation. Additionally, the ability to achieve competitive accuracy with reduced model complexity emphasizes resource efficiency, crucial for applications with limited computational resources. SOS_ConvNet's versatility and adaptability, coupled with its symbiotic organism search framework, contribute to the broader landscape of NAS. For future improvements, the incorporation of more sophisticated operators could further enhance the optimization process. Additionally, exploring automated hyperparameter tuning techniques to optimize the hyperparameters of the generated ConvNet models holds the potential for achieving even better-performing architectures. These potential enhancements will continue to advance the state-of-the-art in NAS methodologies and ConvNet architectures.

CRediT authorship contribution statement

Fatsuma Jauro: Conceptualization, Methodology. **Abdulsalam Ya'u Gital:** Writing – review & editing, Supervision. **Usman Ali Abdullahi:** Writing – review & editing, Supervision. **Aminu Onimisi Abdulsalami:** Conceptualization, Methodology. **Mohammed Abdullahi:** Methodology. **Adamu Abubakar Ibrahim:** Writing – review & editing. **Haruna Chiroma:** Conceptualization, Methodology, Supervision.

Declaration of competing interest

The authors declare no conflict of interest

Data availability

The data used is available online.

References

Baker, B., Gupta, O., Naik, N., & Raskar, R. (2016). Designing neural network architectures using reinforcement learning. *ArXiv Preprint ArXiv:1611.02167*. https://doi.org/10.48550/arXiv.1611.02167

Chen, X., Xie, L., Wu, J., & Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 1294–1303).

Cheng, M., & Prayogo, D. (2014). Symbiotic Organisms Search: A new metaheuristic optimization algorithm. *Computers and Structures, 139*, 98–112.

Cheng, X., Zhong, Y., Harandi, M., Dai, Y., Chang, X., Li, H., et al. (2020). Hierarchical neural architecture search for deep stereo matching. *Advances in Neural Information Processing Systems, 33*, 22158–22169.

Darwish, A., Hassanien, A. E., & Das, S. (2020). A survey of swarm and evolutionary computing approaches for deep learning. *Artificial Intelligence Review, 53*(3), 1767–1812.

Fernandes Junior, F. E., & Yen, G. G. (2019). Particle swarm optimization of deep neural networks architectures for image classification. *Swarm and Evolutionary Computation, 49*, 62–74.

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).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2012). MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.. https://doi.org/10.48550/arXiv.1704.04861

Jauro, F., Gital, A. Y.u., Abdulsalami, A. O., Abdullahi, M., Zambuk, F. U., Jalo, H. S., et al. (2021). Investigating the influence of multiple parameter settings on selected metaheuristic algorithms. In *2021 IEEE Mysore Sub Section International Conference* (pp. 29–35). MysuruCon 2021.

Jiang, J., Han, F., Ling, Q., Wang, J., Li, T., & Han, H. (2020). Efficient network architecture search via multiobjective particle swarm optimization based on decomposition. *Neural Networks, 123*, 305–316.

Kabir Anaraki, A., Ayati, M., & Kazemi, F. (2019). Magnetic resonance imaging-based brain tumor grades classification and grading via convolutional neural networks and genetic algorithms. *Biocybernetics and Biomedical Engineering, 39*(1), 63–74.

Khan, M., Jan, B., Farman, H., Ahmad, J., Farman, H., & Jan, Z. (2019). Deep learning methods and applications. *Deep Learning: Convergence To Big Data Analytics*, 31–42. https://doi.org/10.1007/978-981-13-3459-7_3

Kong, G., Li, C., Peng, H., Han, Z., & Qiao, H. (2023). EEG-Based Sleep Stage Classification via Neural Architecture Search. *IEEE Transactions On Neural Systems And Rehabilitation Engineering, 31*, 1075–1085.

Krizhevsky, B. A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances In Neural Information Processing Systems, 25*. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Kyriakides, G., & Margaritis, K. (2020). An introduction to neural architecture search for convolutional networks. https://doi.org/10.48550/arXiv.2005.11074.

Lecun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time-series. *The Handbook of Brain Theory and Neural Networks, 3361*(10), 1995.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324.

Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. *ArXiv Preprint ArXiv:1806.09055*. https://doi.org/10.48550/arXiv.1806.09055

Martín, A., Manuel, V. V., Antonio, P. G., Camacho, D., & Hervas-Martnez, C. (2020). Optimising convolutional neural networks using a hybrid statistically-driven coral reef optimisation algorithm. *Applied Soft Computing Journal, 90*, Article 106144. https://doi.org/10.1016/j.asoc.2020.106144

Miao, F., Yao, L., & Zhao, X. (2021). Evolving convolutional neural networks by symbiotic organisms search algorithm for image classification. *Applied Soft Computing, 109*, Article 107537. https://doi.org/10.1016/j.asoc.2021.107537

Mishra, V., & Kane, L. (2023). An evolutionary framework for designing adaptive convolutional neural network. *Expert Systems with Applications, 224*, Article 120032. https://doi.org/10.1016/j.eswa.2023.120032

Musa, N., Gital, A. Y., Aljojo, N., Chiroma, H., Adewole, K. S., Mojeed, H. A., et al. (2023). A systematic review and Meta-data analysis on the applications of deep learning in electrocardiogram. *Journal of Ambient Intelligence and Humanized Computing, 14*(7), 9677–9750.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., & Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *In Proceedings of the 35th International conference on machine learning* (pp. 4095–4104). PMLR.

Rasdi Rere, L. M., Fanany, M. I., & Arymurthy, A. M. (2016). Metaheuristic algorithms for convolution neural network. *Computational Intelligence and Neuroscience*. https://doi.org/10.1155/2016/1537325. *2016*.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., et al. (2017). Large-scale evolution of image classifiers. *In Proceedings of the 34thInternational Conference on Machine Learning, 70*, 2902–2911. PMLR.

Sharif, M. I., Li, J. P., Khan, M. A., Kadry, S., & Tariq, U. (2024). M3BTCNet: Multi model brain tumor classification using metaheuristic deep neural network features optimization. *Neural Computing and Applications, 36*(1), 95–110.

Simonyan, K., & Zissermann, A. (2015). Very deep convolutional networks for large-scale image recognition. *1–14. arXiv preprint arXiv:1409.1556*. https://doi.org/10.48550/arXiv.1409.1556

Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2019). Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation, 24*(2), 394–407.

Sun, Y., Xue, B., Zhang, M., Yen, G. G., & Lv, J. (2020). Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics, 50*(9), 3840–3854.

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *In Proceedings of the 36th International Conference on Machine Learning, 97*, 6105–6114.

Wang, D., Zhai, L., Fang, J., Li, Y., & Xu, Z. (2024). psoResNet: An improved PSO-based residual network search algorithm. *Neural Networks, 172*, Article 106104. https://doi.org/10.1016/j.neunet.2024.106104

Wang, Y., Qiao, X., & Wang, G. G. (2023). Architecture evolution of convolutional neural network using monarch butterfly optimization. *Journal of Ambient Intelligence and Humanized Computing, 14*, 12257–12271, 2023.

Wani, A., Ahmad, F., Saduf, B., Asif, A., & Khan, I. (2020). *Advances in deep learning, 57*. Springer Nature.. ISBN: 978-981-13-6793-9.

Wen, L., Gao, L., Li, X., & Li, H. (2022). A new genetic algorithm based evolutionary neural architecture search for image classification. *Swarm and Evolutionary Computation*, 75, 101191. https://doi.org/10.1016/j.swevo.2022.101191.

Wu, X., Hu, S., Wu, Z., Liu, X., & Meng, H. (2022). Neural architecture search for speech emotion recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 6902–6906). Singapore pp.

Xue, Y., Chen, C., & Słowik, A. (2023). Neural architecture search based on a multi-objective evolutionary algorithm with probability stack. *IEEE Transactions on Evolutionary Computation, 27*(4), 778–786.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: An overview and application in radiology. *In Insights Into Imaging, 9*(4), 611–629.

Zaman, K., Zhaoyun, S., Shah, S. M., Shoaib, M., Lili, P., & Hussain, A. (2022). Driver emotions recognition based on improved faster R-CNN and neural architectural search network. *Symmetry*, (4), 14. https://doi.org/10.3390/sym14040687

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.015785*. https://doi.org/10.48550/arXiv.1611.01578