Expert Systems With Applications 224 (2023) 120032

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

An evolutionary framework for designing adaptive convolutional neural network

Vidyanand Mishra [*], Lalit Kane

School of Computer Science, University of Petroleum and Energy Studies (UPES), Energy Acres, Bidholi, Dehradun 248007, Uttarakhand, India

ARTICLE INFO

Keywords:
Convolutional neural networks
Evolutionary algorithms
Deep learning
Genetic algorithm
Particle swarm optimization
Convolutional neural network optimization

ABSTRACT

The Convolutional Neural Network (CNN) is a complex architecture that performs magnificently in image classification and segmentation problems. Still, selecting an effective architecture is typically hindered by several parameters. Empirically, evolutionary algorithms (EA) have been found adequate in parameter selection and automated neural network search. However, the huge computational requirements imposed by evolutionary search make its applicability unexplored. Consequently, the idea of a CNN architecture selection based on EA is challenging as comparing complex candidate architectures towards their fitness would involve massive computations. In this work, we propose a novel framework using an adapted Genetic Algorithm (GA) that automatically evolves an effective CNN architecture. We rectify the GA by devising an effective encoding scheme, an approach to initialize the input population, and a diversified offspring generation method. We also suggest an optimized fitness function that makes the convergence faster, avoiding the local optima. The method is validated with the benchmark MNIST, Fashion_MNIST, and CIFAR-10 datasets. The results are comparable to the best manual and automatic state-of-the-art architectures regarding accuracy, convergence rate, and consumed computation resources.

1. Introduction

Convolutional Neural Networks (CNN)'s performance has proven to be outstanding in computer vision and image classification problems (Krizhevsky, Sutskever, & Hinton, 2017); nevertheless, the performance of a CNN architecture largely depends on the complexity of the architecture, training data, and the hyperparameter selection technique (Simonyan & Zisserman, 2014). For comparative analysis, several manually designed state-of-the-art architectures, such as ResNet (Wu, Zhong, & Liu, 2018), DenseNet (Huang, Liu, Van Der Maaten, & Weinberger, 2017), and GoogleNet (Szegedy et al., 2015), have been tested on real-time image datasets. Although the performances of the said architectures are outstanding on the target datasets, a generic architecture that adapts to a new dataset is still awaited. Today, CNN architectures are being used in a variety of domains, including image processing (Ren, He, Girshick, & Sun, 2015), healthcare (Joshi, & Singh, 2020), agriculture (Joshi, Mishra, Srivastav, & Goel, 2021), cyber security (Ghillani, 2022), and automatic vehicle routing (Kuo, Lu, Lai, & Mara, 2022). Therefore, developing a novel, adaptable framework that can generate the desired CNN architecture with the minimum expert intervention is desirable.

Furthermore, parameter tuning in the CNN model is difficult for experts because of its complex architecture, numerous parameters, and datasets. To solve the above problems, a few recent articles based on Recurrent Neural Networks (RNN), such as Neural Architecture Search (NAS) (Zoph & Le, 2016), NasNet (Zoph, Vasudevan, Shlens, & Le, 2018), MetaQNN (Baker, Gupta, Naik, & Raskar, 2016), and BlockQNS (Zhong, Yan, & Liu, 2017), are introduced. Nevertheless, experimental evidence shows that RNN based model requires huge computational resources to train the model (Mishra & Kane, 2022), which restricts its popularity. For instance, to perform at a level comparable to the NAS approach, which required 800 GPUs over 28 days to find the most potential CNN design on the CIFAR10 dataset, the Genetic CNN (Xie & Yuille, 2017) required approximately 17 GPUs per day in the CIFAR-10 dataset. There is a strong preference for CNN architectural designs based on evolutionary algorithms because not all interested users can access expensive computational resources. Evolutionary Algorithm (EA) based methods such as Genetic CNN, AE-CNN (Sun, Xue, Zhang, & Yen, 2019), CGP-CNN (Suganuma, Kobayashi, Shirakawa, & Nagao, 2020), and several other recent methods are also used to design a suitable

* Corresponding author.
E-mail addresses: mishra.vidyanand28@gmail.com (V. Mishra), lalit.kane@ddn.upes.ac.in (L. Kane).

https://doi.org/10.1016/j.eswa.2023.120032
Received 25 January 2023; Received in revised form 1 April 2023; Accepted 1 April 2023
Available online 5 April 2023

architecture with reduced training parameters and resources. EAs are metaheuristic algorithms that work on the concept of survival of the fittest. Some EA algorithms, such as Genetic Algorithm (GA) (Mirjalili & Mirjalili, 2019), genetic programming (Shirani Faradonbeh, Monjezi, & Armaghani, 2016), evolutionary strategy (Hansen, Arnold, & Auger, 2015), Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995), and many more, are used to optimize the parameters as well as architecture selection. Out of which GA is most suitable in the architecture selection problems (Vargas-Hakim, Mezura-Montes, & Acosta-Mesa, 2021). Additionally, due to the lesser number of parameters in genetic algorithms, it results in convolving and locating the ideal solution more quickly in search space.

In this article, we propose a GA-based framework to design CNN architectures automatically. We present a GA-based approach to architecture discovery in which learning hyperparameters such as kernel number, kernel size, learning rate, activation function, dropout rate, batch size, and others are manually initialized. However, the number of pooling and convolutional layers and their interconnections, skip connections, depth, and width are chosen automatically. We used this approach to reduce the computation cost and minimize the search space. As parameter initialization is an important part of EA for a faster convergence rate, we employ a random initialization approach. A random approach will help initialize in the different positions and is more likely to convolve in global optima. In the proposed architecture, a dynamic encoding technique (Jiang et al., 2020) is used to initialize the population and to represent the CNN architecture in the form of chromosomes.

We define genetic operators compatible with the encoding scheme to repopulate the new generation. After selecting the architecture, we used backpropagation for training the dataset. The conceptual model is examined with the current state of the art regarding the faster convergence rate in early stages optimized to find closer to global optima. Additionally, we inspect the diversity of CNN algorithms in simple and complex datasets such as MNIST (Deng, 2012), Fashion_MNIST (Xiao, Rasul, & Vollgraf, 2017), and CIFAR-10 (Krizhevsky & Hinton, 2009), respectively.

The novelty of the proposed framework is its intuitive nature, which means users can use domain knowledge of CNNs to utilize it while still acquiring a good perspective CNN design for image input.

Here is a summary of the contributions made by the suggested algorithm:

1. GA frequently adopt fixed-length encoding because crossover and mutation operators are primitively built for chromosomes of similar lengths. As a result, the desired depth of the CNN architectures can be inaccurately computed. We presented an elementary variable-length encoding approach, which is simple to implement and has a rapid convergence rate by efficiently exploring the search space.
2. Most present methods available extensively use computational resources to expedite the automation of CNN architectures and give users a better experience in designing an ideal CNN architecture within a reasonable amount of time. In the proposed algorithm, we used an adaptive mechanism to identify and eliminate the non-performing architectures in a few epochs instead of complete training.
3. The experiments compare the proposed methodology's accuracy, convergence rate, and computation cost with several state-of-the-art architectures. We tested the model over benchmark datasets, including MNIST, Fashion_MNIST, and CIFAR-10. The adaptability of an algorithm is also analyzed based on different epoch sizes and generations.

The article is organized as follows: The second section describes the literature review. The suggested methodology and framework are discussed in the third part. Section 4 discusses the experimental design, Section 5 addresses the results and analysis, Section 6 discussions, and section seven concludes with recommendations for further research.

2. Literature work

The convolutional neural network's layered structure consists of a convolutional layer, a pooling layer, and a fully connected layer. We transmit raw pixel data from the input image to CNN, which enables feature extraction at various levels to aid in model learning. Weighted filters are used in the convolutional layer to extract the features from the input data, and the activation function is used to introduce nonlinearity. In the pooling layer, the redundant features of the convolutional layer are removed using min, max, or average operations. The output matrix is transformed into a one-dimensional vector and trained as a neural network in a fully connected layer.

Numerous hyperparameters must be altered and enhanced to improve the CNN model. The filter size, the number of filters, pooling function, learning rate, activation function, stride size, and many more hyperparameters were among them. Due to huge parameters, researchers working on generating a CNN architectural design need help choosing appropriate hyperparameters. The complexity of the CNN architecture is a critical factor while learning complicated features from training datasets. As an architecture's depth and interconnections increase, so does its parameters and complexity. We need methods to automatically discover the hyperparameters and CNN architecture to solve the problem.

The performance of CNN architecture is determined by accuracy, training cost, and parameter count. The accuracy is mostly determined by the training dataset (image size, quality, and distribution) and the complexity of the architecture. However, the training cost is mostly associated with parameters such as depth of architecture, size of kernels, number of kernels, learning rate, epoch, activation function and many more. Hence, selecting an accurate architecture is tedious, as it takes knowledge of the CNN domain and several trial-and-error combinations for hyperparameter tuning (Li, Zhan, Xu, Kwong, & Zhang, 2021). It also increases the computation cost. Most of the CNN architecture, such as VGG net (Simonyan & Zisserman, 2014), Resnet, and DenseNet, was initially developed manually to solve image classification tasks. However, its effectiveness is restricted due to complex architecture and huge parameters (depth till 1052 layers in ResNet and 22 M parameter in DenseNet).

In recent years, automatic architecture selection and parameter optimization algorithms have evolved to overcome the problem of choosing a complex CNN architecture with minimal experience. In addition, the evolution of CNN architecture designs can be subdivided into two distinct types: reinforcement learning-based (Sutton & Barto, 1998) and evolutionary algorithm-based (Sun, Xue, Zhang, & Yen, 2019). Reinforcement learning-based architectural selection demands expensive computations (22400 in NAS), limiting its usefulness. Accuracy-wise, the performance of existing RL-based architectures is commendable, but they incur massive computing costs. Existing methods require extensive training time and epochs to achieve equivalent accuracy. In contrast, evolutionary algorithm-based approaches (GA, PSO, GP) are gaining popularity due to their performance with a substantial reduction in compute resources while maintaining comparable accuracy. Among different EA methods, GA is most popular in the neuro-evolution domain because of its fewer variables and faster convergence rate (Vargas-Hakim, Mezura-Montes, & Acosta-Mesa, 2021).

GA is a metaheuristic algorithm that draws inspiration from biological evolution based on the crossover, mutation, and selection techniques. Choosing a group based on fitness values is more likely to choose fit chromosomes is referred to as a selection operation. Recombining two or more chromosomes to produce a new chromosome is called crossover. By creating a set of values around the chosen point, the mutation process adds diversity to the solution set. GA creates new solutions by making "random" modifications to existing ones. A fitness function determines the optimal parameters; the best solutions are those with the highest fitness values.

In addition, finding the best structures requires careful consideration of hyper-parameter selection and weight initialization (Sun, Xue, Zhang, Yen, & Lv, 2020). As a result, the training expense is decreased, and the convergence rate to the global optimum is rapid. Complex topologies contain many hyper-parameters. Therefore manually initializing them is difficult. Talathi (2015) developed a sequential model-based optimization technique (SMBO) for selecting deep CNN hyperparameters, wherein only a few factors were considered, such as the range of convolutional layers, number of kernels per layer, size and stride of the kernels, the pooling layer (max/min/avg- size, and stride), and their interconnections.

The researchers employed the PSO method for hyperparameter optimization with a given architecture but still needs to develop the architecture from scratch (Serizawa & Fujita, 2020; Wang, Zhang, & Zhang, 2019; Dufourq & Bassett, 2017). PSO's modest convergence rate makes it more ideal for tuning hyperparameters than for building CNN architecture. With equivalent outcomes, our algorithms construct architecture from scratch utilizing GA with no human interaction.

Automatically examining the CNN structure has been done previously using a few state-of-the-art methods. A Genetic CNN technique describes the architecture using a fixed-length binary encoding strategy. A fixed architecture is represented by encoded methods and is tested on CIFAR-10 and MNIST datasets for 50 generations. Similarly, GA is used to develop a CNN architecture autonomously in AE-CNN (Sun, Xue, Zhang, & Yen, 2019). The underlying components used in this approach are ResNet and DenseNet blocks.

Cartesian genetic programming (CGP-CNN) (Suganuma, Kobayashi, Shirakawa, & Nagao, 2020) is a concept that automatically designs CNN architecture using genetic programming. CGP-CNN uses direct encoding to represent CNN blocks and their connectivity. The flexibility in choosing the depth of architecture and ease of skip connection implementation are two advantages of this format. CGP-CNN optimized the CNN architecture (ResSet) in around 14 days. Similarly, the Evolving Convolutional Neural Network (EvoCNN) (Sun, Xue, Zhang, & Yen, 2019) creates an effective algorithm using genetic encoding. The comparison of this suggested method, EvoCNN, with the top 10 peer rivals reveals that it outperforms them all. The CNN-GA technique uses a variable-length encoding scheme to evaluate the complexity of the CNN topology (Bakhshi et al., 2019; Sun, Xue, Zhang, Yen, & Lv, 2020). But, the predetermined blocks limit the ability to investigate multiple possibilities. Using a skip link also increases the computation cost as possible possibilities grow exponentially. Using a transformed crossover operator is also advised to explore the search space and describe the encoding scheme efficiently. The CNN-GA was used to identify architecture, automatically increasing the computation cost. Using fixed length encoding in (Xie & Yuille, 2017; Sun, Xue, Zhang, & Yen, 2019) reduces the flexibility to explore the optimal depth, whereas using fixed block size as a building block in (Suganuma, Kobayashi, Shirakawa, & Nagao, 2020; Esfahanian & Akhavan, 2019) suffers from a restricted search space. In addition to being straightforward to implement, it has a quick convergence rate, effectively explores the search space, is less likely to get stuck in local optima, is easy to create genetic operators, and minimizes computation costs. Using the mutation operator, it can also automatically expand and contract the layers, allowing it to investigate the depth of architecture. Additionally, we eliminated the fully connected layer, drastically reducing the parameter and computational cost.

3. Methodology

In this section, we describe the framework of the proposed algorithm in Subsection 3.1, followed by its critical points in Subsections 3.2 through 3.4. In order to assist the reader in comprehending the proposed algorithm, we will not only record the specifics of each major step but also provide analysis for certain architectural-level designs.

3.1. *Algorithm overview*

The proposed algorithm's framework is shown in Algorithm 1, and the flow chart of the proposed framework is depicted in Fig. 1.

In this algorithm, we pass input datasets, and after a sequence of evolution, the framework automatically evolves to a suitable CNN architecture. A random population is initialized using a predetermined encoding and population size throughout evolution. Fig. 2 depicts an example of the variable length encoding system employed in the proposed study. This representation uses a 32 × 32 dimension colour image as the input to the convolutional layer. The number of filters in a convolutional layer is randomly selected using population initialization methods. The dimension of a filter is fixed to 3 × 3, and a stride of 1 × 1 is used to make it homogenous and reduce the computational cost.

In the pooling layer, the algorithm automatically selects avg pool or max pool operation with equal probability having kernel size of 2 × 2 and stride 2 × 2. The concatenated string of different layers represents the encoded representation of CNN architecture, as shown in Fig. 2. In pooling layer representation, it shows with the pooling operation either min, max, or average pooling along with kernel size 2 × 2 and stride 2 × 2. The concatenated string represents the encoded representation of CNN architecture.

The hyperparameters are manually chosen using the existing state-of-the-art model. Each individual's fitness, which encodes a specific CNN architecture, is assessed throughout evolution using the provided dataset. In the subsequent generation, parent individuals are selected according to their fitness, and new offspring are generated utilising genetic operators such as crossover and mutation. The newly created population is combined with the existing population to create a new list of offspring. The evolution proceeds until the counter exceeds the maximum generation, increasing the counter by one. Most existing frameworks are developed using fixed maximal generation, which could restrict resource management. This paper employed adaptive exit conditions that terminate automatically when the convergence rate is slow or near zero.

Algorithm 1 Framework of the proposed algorithm using EA

```
Input: A dataset of a set of CNN architectures represented by the variable length
    encoding technique.
Output: Identifies the best CNN architecture.
1.    Propose an encoding scheme to represent CNN architecture.
```

(*continued on next page*)

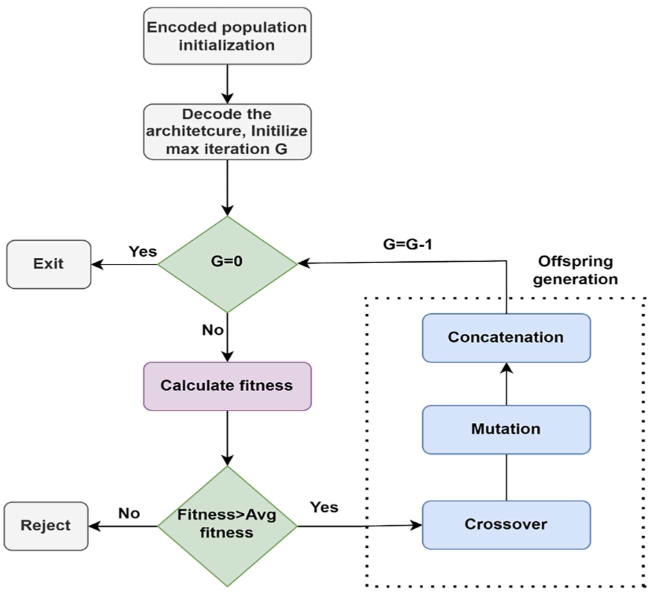

Fig. 1. Flow chart of evolutionary algorithms.

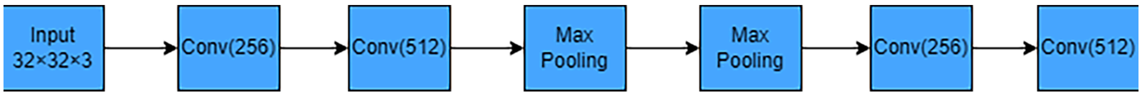

Fig. 2. Decoded architecture of encoding representation "256-512-max-max-512-256".

(*continued*)

```
2.   Initialize the population of N CNN architectures with the help of the proposed
     encoding method. Initialize max iteration G, the number of epochs for the fitness
     function, and the input dataset.
3.   Initialize the hyperparameter kernel size, loss function, learning rate, and stride
     size.
4.   While (G > 0)
       4.1 Calculate the fitness of each architecture.
       4.2. Select N/2 best architectures for reproduction using GA operators.
       4.3 Apply crossover and mutation operators to generate new offspring.
       4.4 Concatenate the new population with the existing best population to
     create a new pool of N architectures.
5.   G ← G-1
6    End
7.   Return the best CNN architecture
```

3.2. Population initialization

The basic components of a CNN are convolutional layers, pooling layers, and sometimes fully linked layers. The CNN's performance heavily depends on its parameters, which depend on the connection depth and width. The fully connected layer is discarded in this encoding as many parameters make it computationally inefficient. Initially, the number of population and the depth of each population is selected randomly. In the selected population, the first layer is fixed as a convolutional layer; then, convolutional and pooling layers are determined randomly with equal probability. The convolutional layer's filter count is randomly chosen in the range of $[2^5 - 2^9]$. All the selected population is organized in a list to evaluate the fitness value after initialization. The filter size and pooling operation range are selected manually based on a few standard architectures. The algorithm for population initialization is mentioned in Algorithm 2.

Algorithm 2 Population Initialization

```
Input: The number of initial population N.
Output: The list of N initialized architecture using encoding representation.
1.   P ← Ø
2.   While |P| < N
3.   Choose random integer D as depth.
4.   Generate a convolutional layer with the number of filters between [$2^5 - 2^9$] and
     filter size is 3 × 3.
5.   While (D > 0)
       5.1. Choose a random number between (0-1)
       5.2. If number <0.5
         5.2.1 Generate a convolutional layer with the number of filters are between
     [$2^5 - 2^9$] and a filter size is 3×3.
       5.3. Else
         5.3.1. Choose between max pool and avg pool randomly.
         5.3.2. Concatenate the selected layer with the existing architecture Pi.
       5.4. D–;
6.   P = P U Pi
7.   End
8.   Return P.
```

3.3. Fitness function

Algorithm 3 evaluates the fitness of all input populations using a given dataset. An individual's CNN is initially decoded using a predetermined set of hyperparameter parameters. CNN decoding is trained with training data, and accuracy is used to determine fitness. Because the training of CNN is a time-taking task, we used half of the dataset for initial training to make it efficient. After training the population, half of the population is eliminated based on fitness score. The best population is chosen for reproduction in the following offspring generation. If the model is showing good training accuracy, but validation accuracy is not increasing in respective of training in a few successive epochs, then architecture may suffer from overfitting (Gavrilov, Jordache, Vasdani, & Deng, 2018). We can eliminate the overfitted model to reduce the computation cost in the early stages.

Algorithm 3 Fitness function

```
Input: The selected population list of CNN architecture, input dataset, range of
  hyperparameters, optimizer, loss function, epoch, train data, and test data.
Output: Best CNN architecture with fitness value
1.   Divide the dataset into train and test data.
2.   $F_{best}$ ← 0
3.   For each population $P_i$ in population pool P do:
4.   Decode the architecture and calculate fitness accuracy using half of the
     population using backpropagation methods.
5.   Eliminate the architecture based on overfitting.
6.   Choose P best population, train using the complete dataset, and calculate fitness
     value F for each.
       6.1 If F > $F_{best}$
       6.2 $F_{best}$ = F
       6.3 End
7.   End
```

3.4. Offspring generation

Algorithm 4, consisting of two parts, illustrates the specifics of producing the offspring. Crossover is the first, and mutation is the second. Specifically, two parents are selected based on which of two randomly selected individuals is more suitable. We build a new set of populations with equal probability by utilising mutation and crossover processes. In a crossover operation, each parent is arbitrarily divided into two pieces, and the two pieces from each parent are exchanged to generate two offspring. We have chosen crossover probability 0.8 and mutation probability 0.2. Mutation operation helps define the architecture's exact depth, whereas the crossover operation increases the convergence rate. Both operations must be compatible with the encoding scheme. Newly generated offspring will be combined with the previous best architecture to create a new population pool.

Algorithm 4 Offspring generation

```
Input: Input population list P, with its fitness value, mutation, and crossover
  operation with their probability value.
Output: Newly generated population list $Q$.
1.   Q ← Ø
2.   While $|Q_t|\langle|P|$ do
       2.1. p1,p2 ← randomly select two population values from P
       2.2. r ←randomly generate number in range [0, 1].
       2.3. If (r < 0.5)
         2.3.1. Select mutation operations [add conv layer, add skip layer, add pool
     layer, remove layer of filters], change the value, and position (index value in
     offspring) randomly
       2.4. Else
         2.4.1. Choose the crossover point in p1 and p2.
         2.4.2. Apply crossover operation
       2.5. End
3.   Return $Q_t$
4.   End
```

4. Experiment design

Several tests have been done on image classification tasks to assess the performance of the proposed algorithm. The experiments are performed on A-100 Tensor Core GPU. In particular, Subsection 4.1

introduces the peer competitors identified to compare with the suggested solution. Following that, Subsection 4.2 describes the benchmark datasets that were employed. Finally, Subsection 4.3 displays the suggested algorithm's parameter settings.

A. **Peer Competitors:** To demonstrate the efficiency and effectiveness of the proposed algorithm among all currently available state-of-the-art CNNs, we present extensive comparisons to CNN architectures designed with various evolutionary algorithms, including encoding scheme, manual intervention required, and hyperparameter optimization. First, we select state-of-the-art where fixed architecture is selected, and hyperparameters are tuned with the help of EA. Specifically, Eden (Dufourq & Bassett, 2017), C-PSO-CNN (Wang, Zhang, & Zhang, 2019), and LDPSO (Serizawa & Fujita, 2020) are compared over the CIFAR-10 dataset regarding the accuracy and Epoch size. We also compared our methods with RL-based architecture, such as NAS (Zoph & Le, 2016), and EAS (Cai, Chen, Zhang, Yu, & Wang, 2018), regarding computation cost and epoch size. Genetic algorithm-based state-of-the-art such as Genetic CNN (Xie & Yuille, 2017), CGP-CNN (Suganuma, Kobayashi, Shirakawa, & Nagao, 2020), E-CNN-MP (Loussaief & Abdelkrim, 2018), CNN-GA (Bakhshi, Noman, Chen, Zamani, & Chalup, 2019), and DCNN (Ma, Li, Xia, & Zhang, 2020) is also used to show the effectiveness in computation power, accuracy and convergence rate. For a fair comparison, we used the same data set, CIFAR-10, MNIST, and Fashion_MNIST without any preprocessing techniques.
B. **Dataset:** The CIFAR10 dataset depicted in Fig. 3, in particular, serves as a benchmark for image classification, classifying ten different types of natural objects, including birds, horses, ships, deer, frogs, dogs, trucks, cats, vehicles, and aeroplanes. It comprises 60,000 RGB images, each 32 × 32 pixels in size. Furthermore, there are 50,000 images in the training set and 10,000 in the testing set. There are the same amount of images in every category. Similarly, the MNIST dataset, shown in Fig. 4 is a benchmark for image classification for identifying digits. It has 70,000 grayscale pictures in total, each measuring 28 × 28 pixels. Furthermore, the training set consists of 60,000 images, and the testing set contains 10,000 images. The amount of data in each category is the same. Also, we used Fashion_MNIST dataset (Fig. 5) as a benchmark for fashion image classification. It has 70,000 grayscale pictures in total, each measuring 28 × 28 pixels. It was created in 2017 and had ten classes. Furthermore, the training set consists of 60,000 images, and the testing set contains 10,000 images. The amount of data in each category is the same.

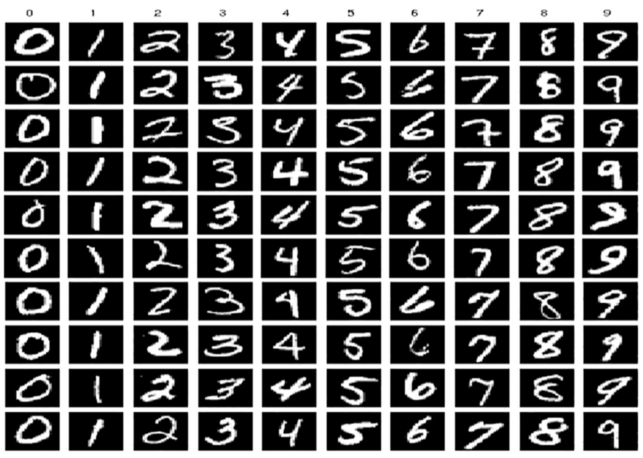

Fig. 4. Examples from MNIST data sets.

C. **Hyperparameter Tuning:** We used a 3 × 3 filter and a 1 × 1 stride throughout all convolutional layers for simplicity and homogeneity. Stride size is fixed at 2 × 2 in the pooling layer, where the max or average pooling operation is chosen randomly. We used the back-propagation method for training the architecture. Adam optimizer (Bock, Goppold, & Weiß, 2018) and sparse- categorical-cross-entropy (Zhang & Sabuncu, 2018) are used to calculate the loss function. We used TensorFlow deep learning framework and MNIST, Fashion_MNIST, and CIFAR-10 datasets to assess its performance. The input data is split into 80% train and 20% test sets. In training, the MNIST and Fashion_MNIST datasets use the value of epochs, and the population size is set to 10 and 5, respectively. At the same time, the number of generations is kept 10. In the CIFAR-10 dataset, we used ten generations and 40 epochs to train the model.

5. Experimental results and analysis

This section summarises the contrast between the suggested method and the findings of peer competitors. We compared our results to the most advanced approaches regarding classification precision, GPU days utilized, and architectural aspects. Specifically, the unit GPU day indicates that the algorithm has executed for one day on a single GPU, which measures the number of computational resources spent by these methods. Table 1 displays the outcomes of a comparison between the suggested algorithm and its peer rivals. The first column displays the list

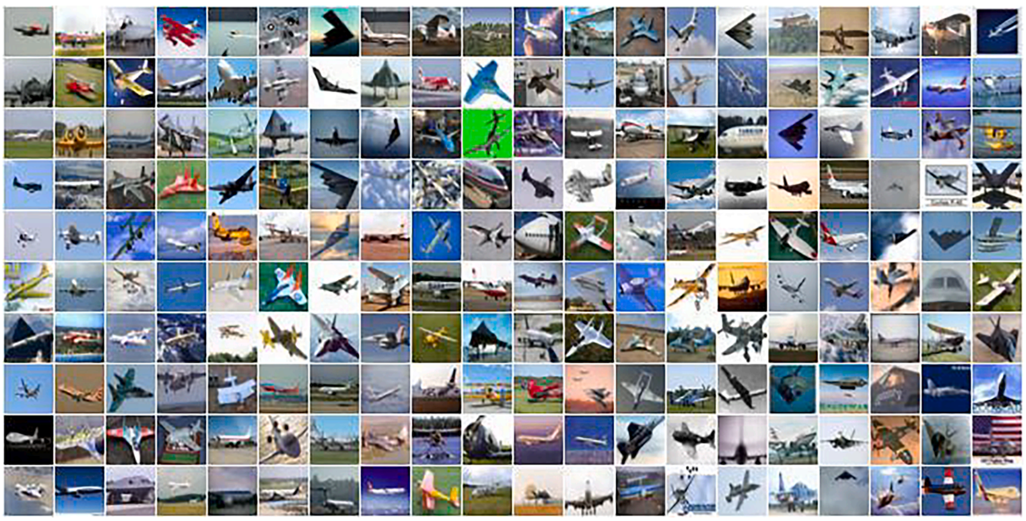
Fig. 3. Examples from CIFAR-10 data sets.

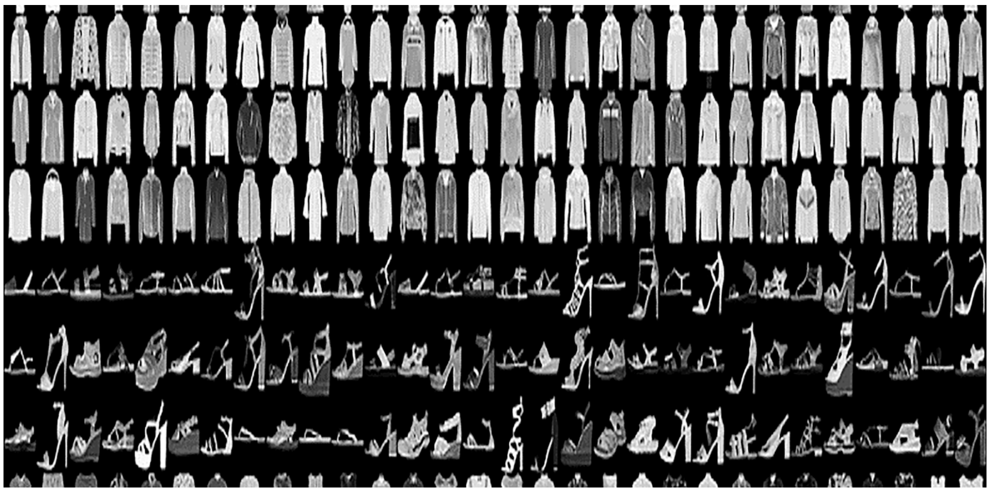

Fig. 5. Examples from Fashion_MNIST data sets.

Table 1
The classification accuracy comparison on the CIFAR-10 datasets between the proposed algorithm and the state-of-the-art peer competitors.

Reference	Architecture	Encoding	EA	Data set	Accuracy	Gen	Epoch	Para meters	GPU	Manual Assistance
(Dufourq & Bassett, 2017)	Eden	Fixed Length	PSO	CIFAR10 MNIST	74.5% 98.4%	10	13	1.8 M	12 h	Partially Required
(Wang, Zhang, & Zhang, 2019)	C-PSO-CNN (AlexNet)	Fixed Length	PSO	CIFAR10	89.99%	40	10	–	–	Partially Required
(Wang, Zhang, & Zhang, 2019)	C-PSO-CNN (VGG Net-16)	Fixed Length	PSO	CIFAR10	91.02%	40	10	–	–	Partially Required
(Serizawa & Fujita, 2020)	LDPSO	Fixed Length	PSO	CIFAR10	69.37%	–	10	–	2.37 h	Partially Required
(Xie & Yuille, 2017)	Genetic CNN	Fixed Length	GA	CIFAR10	77.06%	50	–	–	17 days	Partially Required
(Cai, Chen, Zhang, Yu, & Wang, 2018)	EAS	–	RL	CIFAR10	95.77%	–	300	23.4 M	10 days	Partially Required
(Suganuma, Kobayashi, Shirakawa, & Nagao, 2020)	CGP-CNN	Variable Length	GP	CIFAR10	94.02%	50	500	1.68 M	27 days	Not Required
(Zoph & Le, 2016)	NAS	–	LSTM	CIFAR10	93.99%	–	50	2.5 M	22,400 days	Not Required
(Loussaief & Abdelkrim, 2018)	E-CNN-MP	Variable Length	GA	MNIST	98.94%	5	–	–	–	Not Required
(Sun, Xue, Zhang, Yen, & Lv 2020)	CNN-GA	Variable Length	GA	CIFAR10	77.50% 95.22	5 20	350	– 2.9 M	– 30 days	Not Required
(Ma, Li, Xia, & Zhang, 2020)	DCNN	Variable Length	GA	CIFAR10 MNIST Fashion_MNIST	89.32 99.64 94.60	10 10 10	100 100 100	–	12 days 3 days 5 days	Not Required
	Proposed Methods	**Variable Length**	**GA**	**CIFAR10** **MNIST** **Fashion_MNIST**	**79.41%** **87.02%** **99.39%** **93.07%**	**5** **10** **10** **10**	**40** **40** **10** **10**	**1.2 M** **1.6 M** **2.7 M** **1.4 M**	**2.47 h** **6.38 h** **3.12 h** **3.27 h**	**Not Required**

of architecture classifications. The second column contains the names of the architectures. The third column represents the encoding methods; the fourth column represents the evolutionary algorithm used; the fifth column represents the datasets used; the sixth column represents the classification accuracy; the seventh column represents the number of generations; the eighth column represents training epochs, and ninth column represents the parameter count for the relevant CNN. Furthermore, the tenth column displays the number of GPU days utilized. All competitors' results in the table are extracted from the related publications; "–" denotes that the results have not been published.

6. Discussion

Table 1 displays the results of a comparison between the proposed algorithm and its peer competitors. Table 1 groups the peer competitors into two categories. In the first group, we compared architectures requiring manual aid in design selection or parameter adjustment. In this category, the computation cost is lower because half of the work is performed by professionals. In the second group, we compared architectures that evolved without human involvement. For the first category of peer competitors, our technique improves classification accuracy on the CIFAR10 dataset by 16.8 % for the Eden architecture and 25.4 % for the LDPSO design. In the second category, our technique improves classification accuracy on the CIFAR10 dataset by 2.46% for CNN-GA architecture and 12.8% for Genetic-CNN architecture. It also exhibits a 1.06% and 0.45% improvement on the MNIST dataset over the Eden and E-CNN-MP architectures, respectively. We further tested the efficacy of our approach using the Fashion_MNIST dataset, which demonstrates a 3.37% improvement over the capsuleNet (Sabour, Frosst, & Hinton, 2017) architecture. The classification accuracy is slightly less for EAS, CGP-CNN, NAS, CNN-GA, and DCNN architecture. However, in addition to classification, we also compare the effectiveness based on the number of parameters, epoch, and computation required in GPU. DCNN requires 100 epochs and 12 GPU days to train CIFAR-10 datasets, CNN-GA requires 30 GPU days and up to 350 epochs, CGP-CNN is trained in 500 epochs and requires 27 GPU days, EAS requires 10 GPU days with 300 epochs, and NAS requires 50 epoch and 22,400 GPU days. Our algorithm was trained in 10 generations, 10 epochs, and 6.38 h using the Nvidia A100 GPU configuration on the CIFAR 10 dataset, demonstrating significant improvement and a faster convergence rate. Our techniques provide competitive performance in terms of precision and the number of parameters while requiring less calculation time.

To demonstrate the efficacy of the proposed method for discovering the CNN architecture using the MNIST, and Fashion_MNIST datasets, we have displayed the evolution in Tables 2 and 3, respectively. We randomly chose the N population and initialized the value using the recommended encoding approach. In this experiment, the population size is N = 5, the number of epochs is 40, and the number of generations is 10. The architectures are trained using the backpropagation technique for forty epochs. In the training and validation sets, we utilized an 80/20 ratio. After training, the validation accuracy applied in the fitness function for decision-making of the proposed method is calculated. We eliminated weaker populations for the next generation using mode accuracy. The fitness function eliminates 50 % of the population in each generation, and the best 50 % are employed as parents. The selected population is repopulated using the genetic operator's mutation and crossover with probabilities of 0.2 for mutation and 0.80 for crossover. The existing fittest population is merged with the new population. Thus, using this method, the same number of chromosomes is available in each generation. We have described the efficacy of the proposed algorithm by using the minimum, mean, maximum, mode, standard deviation, and standard error of the mean (SEM). The standard error of the mean (SEM) measures how much discrepancy is likely in a sample's mean compared with the population mean. For each iteration, we have selected the most improved CNN architecture, which is represented in the final column of tables 2 and 3. The maximum accuracy indicates the best accuracy obtained by any CNN architecture at that generation. The standard deviation demonstrates the genetic algorithm's efficacy in terms of a quicker convergence rate. Initial standard deviation values are largely due to the random initialization of the population. However, its value decreases over successive generations, and the top accuracy rises, bringing the outcome close to the global optimum. If the standard deviation continues to decline, the subsequent few generations will see greater convergence. With this strategy, we reached a standard near the benchmark accuracy in 10 epochs and 10 generations, demonstrating a faster convergence rate with equivalent precision and less computational power. This technique yielded 99.39% top accuracy on the MNIST datasets and 93.07% on the Fashion_MNIST dataset, equivalent to the benchmark accuracy without user intervention and requiring less GPU days.

The proposed algorithm demonstrated an increase in convergence rate using the defined techniques. We have displayed the evolution in Tables 2 and 3 to help understand the efficacy of the suggested approach

Table 2
Evolution of CNN model using MNIST dataset with population size = 5, epoch 40, and generation = 10.

Generation	Min %	Avg %	Max %	Med %	Std-D	SME	Best CNN model
Gen 1	87.15	95.84	99.15	97.46	4.39	1.96	256-512-max-max-512-256
Gen 2	97.46	98.34	99.15	98.01	0.68	0.30	256-512-max-max-512-256
Gen 3	97.49	98.69	99.15	99.15	0.66	0.29	256-512-max-max-512-256
Gen 4	98.57	99.01	99.23	99.15	0.24	0.10	256-512-max-512-256-max-512-256
Gen 5	98.62	99.04	99.23	99.15	0.21	0.09	256-512-max-512-256-max-512-256
Gen 6	99.15	99.21	99.33	99.23	0.06	0.04	256-512-max-max-256-512-max-512-256-max-512-256
Gen 7	99.23	99.27	99.33	99.23	0.05	0.02	256-512-max-max-256-512-max-512-256-max-512-256
Gen 8	99.04	99.26	99.36	99.33	0.11	0.05	256-512-max-max-512-256-max-512-256
Gen 9	99.10	99.29	99.36	99.36	0.04	0.10	256-512-max-max-512-256-max-512-256
Gen 10	99.29	99.34	99.39	99.36	0.02	0.01	256-512-512-256-max-512-256-max-512-256

Table 3
Evolution of CNN model using Fashion_MNIST dataset with population size = 5, epoch 10, and generation = 10.

Generation	Min %	Avg %	Max %	Med %	Std-D	SME	Best CNN model
Gen 1	88.20	89.25	91.01	88.69	0.99	0.44	128-256-512-256-mean-mean
Gen 2	88.69	89.44	91.01	89.12	0.86	0.38	128-256-512-256-mean-mean
Gen 3	89.12	89.77	91.01	89.64	0.67	0.29	128-256-512-256-mean-mean
Gen 4	89.59	90.70	91.43	91.01	0.70	0.31	128-256-512-256-mean-mean-mean
Gen 5	91.01	91.35	91.60	91.40	0.19	0.08	128-256-512-256-512-256-mean-mean
Gen 6	91.01	91.35	91.60	91.40	0.19	0.08	128-256-512-256-512-256-mean-mean
Gen 7	91.40	91.86	93.07	91.60	0.24	0.11	128-256-128-256-mean-mean-mean
Gen 8	91.41	91.94	93.07	91.79	0.58	0.26	128-256-128-256-mean-mean-mean
Gen 9	91.60	91.98	93.07	91.79	0.55	0.24	128-256-128-256-mean-mean-mean
Gen 10	91.64	92.11	93.07	91.82	0.51	0.23	128-256-128-256-mean-mean-mean

for discovering CNN designs. We used the MNIST and Fashion_MNIST datasets with a population size of 5, where each architecture is trained for 40 and 10 epochs, respectively. The evolution of each generation is expressed using standard deviation and top accuracy. It shows the effectiveness of the proposed algorithm with its faster convergence rate in the initial iteration reaching toward global optima without being stuck in the local one. Additionally, it offers the diversity of the algorithm that works suitably in different datasets.

7. Conclusion and future work

This research aims to provide an automatic architecture design technique for CNNs based on the GA, which can identify the optimal CNN architecture for solving image classification tasks for users who lack knowledge in adjusting CNN structures. This objective was accomplished by proposing a novel encoding approach for the GA to encode arbitrary CNN depths. The proposed technique is evaluated and compared against 11 state-of-the-art peer competitors, including four partial tuning and seven automatic algorithms determining the architectures of CNNs. The experimental results on the MNIST, Fashion_-MNIST, and CIFAR10 datasets indicate that the proposed technique can automatically generate DCNN structures comparable to or even exceed state-of-the-art models. Further research could investigate the technique's efficacy in various evolutionary algorithms that can accelerate the CNN fitness measurement. A further attractive path for future research is the pursuit of lightweight CNN models with real-world restrictions using more efficient techniques.

Declaration of Competing Interest

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

Data availability

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

References

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM, 60*(6), 84–90. https://doi.org/10.1145/3065386

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. https://doi.org/10.48550/arXiv.1409.1556.

Wu, S., Zhong, S., & Liu, Y. (2018). Deep residual learning for image steganalysis. *Multimedia Tools and Applications, 77*, 10437–10453. https://doi.org/10.1007/s11042-017-4440-4

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700–4708).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems, 28*.

Joshi, D., & Singh, T. P. (2020). A survey of fracture detection techniques in bone X-ray images. *Artificial Intelligence Review, 53*(6), 4475–4517. https://doi.org/10.1007/s10462-019-09799-0

Joshi, D., Mishra, V., Srivastav, H., & Goel, D. (2021). Progressive transfer learning approach for identifying the leaf type by optimizing network parameters. *Neural Processing Letters, 53*(5), 3653–3676. https://doi.org/10.1007/s11063-021-10521-x

Ghillani, D. (2022). Deep learning and artificial intelligence framework to improve the cyber security. *Authorea Preprints*.

Kuo, R. J., Lu, S. H., Lai, P. Y., & Mara, S. T. W. (2022). Vehicle routing problem with drones considering time windows. *Expert Systems with Applications, 191*, Article 116264. https://doi.org/10.1016/j.eswa.2021.116264

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697–8710).

Baker, B., Gupta, O., Naik, N., & Raskar, R. (2016). Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.

Zhong, Z., Yan, J., & Liu, C. L. (2017). Practical network blocks design with q-learning. *arXiv preprint arXiv:1708.05552, 6*.

Mishra, V., & Kane, L. (2022). A survey of designing convolutional neural network using evolutionary algorithms. *Artificial Intelligence Review, 1–38*. https://doi.org/10.1007/s10462-022-10303-4

Xie, L., & Yuille, A. (2017). Genetic cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 1379–1388).

Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2019a). Completely automated CNN architecture design based on blocks. *IEEE Transactions on Neural Networks and Learning Systems, 31*(4), 1242–1254. https://doi.org/10.1109/TNNLS.2019.2919608

Suganuma, M., Kobayashi, M., Shirakawa, S., & Nagao, T. (2020). Evolution of deep convolutional neural networks using Cartesian genetic programming. *Evolutionary Computation, 28*(1), 141–163. https://doi.org/10.1162/evco_a_00253

Mirjalili, S., & Mirjalili, S. (2019). Genetic algorithm. *Evolutionary Algorithms and Neural Networks: Theory and Applications, 43–55*. https://doi.org/10.1007/978-3-319-93025-1_4

Shirani Faradonbeh, R., Monjezi, M., & Jahed Armaghani, D. (2016). Genetic programing and non-linear multiple regression techniques to predict backbreak in blasting operation. *Engineering with Computers, 32*, 123–133. https://doi.org/10.1007/s00366-015-0404-3

Hansen, N., Arnold, D. V., & Auger, A. (2015). Evolution strategies. *Springer Handbook of Computational Intelligence*, 871–898.

Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (Vol. 4, pp. 1942-1948). IEEE.

Vargas-Hakim, G. A., Mezura-Montes, E., & Acosta-Mesa, H. G. (2021). A review on convolutional neural network encodings for neuroevolution. *IEEE Transactions on Evolutionary Computation, 26*(1), 12–27. https://doi.org/10.1109/TEVC.2021.3088631

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. https://doi.org/10.1016/j.neunet.2019.12.005

[MNIST] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE signal processing magazine, 29*(6), 141–142. https://doi.org/10.1109/MSP.2012.2211477

[Fashion-mnist] Xiao, H., Rasul, K., & Vollgraf, R. (2017). A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*. https://doi.org/10.48550/arXiv.1708.07747.

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. http://www.cs.toronto.edu/kriz/cifar.html, 2009.

Li, J. Y., Zhan, Z. H., Xu, J., Kwong, S., & Zhang, J. (2021). Surrogate-assisted hybrid-model estimation of distribution algorithm for mixed-variable hyperparameters optimization in convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*. https://doi.org/10.1109/TNNLS.2021.3106399

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sutton, R. S., & Barto, A. G. (1998). Reinforcement Learning: An Introduction (3rd éd.).

Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2019b). Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation, 24*(2), 394–407. https://doi.org/10.1109/TEVC.2019.2916183

Bakhshi, A., Noman, N., Chen, Z., Zamani, M., & Chalup, S. (2019, June). Fast automatic optimisation of CNN architectures for image classification using genetic algorithm. In *2019 IEEE congress on evolutionary computation (CEC)* (pp. 1283-1290). IEEE.

Talathi, S. S. (2015, September). Hyper-parameter optimization of deep convolutional networks for object recognition. In *2015 IEEE international conference on image processing (ICIP)* (pp. 3982-3986). IEEE.

Serizawa, T., & Fujita, H. (2020). Optimization of convolutional neural network using the linearly decreasing weight particle swarm optimization. *arXiv preprint arXiv:2001.05670*.

Wang, Y., Zhang, H., & Zhang, G. (2019). cPSO-CNN: An efficient PSO-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. *Swarm and Evolutionary Computation, 49*, 114–123. https://doi.org/10.1016/j.swevo.2019.06.002

Dufourq, E., & Bassett, B. A. (2017, November). Eden: Evolutionary deep networks for efficient machine learning. In *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)* (pp. 110-115). IEEE.

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. https://doi.org/10.1109/TCYB.2020.2983860

Esfahanian, P., & Akhavan, M. (2019). Gacnn: Training deep convolutional neural networks with genetic algorithm. *arXiv preprint arXiv:1909.13354*.

Gavrilov, A. D., Jordache, A., Vasdani, M., & Deng, J. (2018). Preventing model overfitting and underfitting in convolutional neural networks. *International Journal of Software Science and Computational Intelligence (IJSSCI), 10*(4), 19–28. https://doi.org/10.4018/IJSSCI.2018100102

Cai, H., Chen, T., Zhang, W., Yu, Y., & Wang, J. (2018, April). Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 32, No. 1).

Loussaief, S., & Abdelkrim, A. (2018). Convolutional neural network hyper-parameters optimization based on genetic algorithms. *International Journal of Advanced Computer Science and Applications, 9*(10).

Ma, B., Li, X., Xia, Y., & Zhang, Y. (2020). Autonomous deep learning: A genetic DCNN designer for image classification. *Neurocomputing, 379*, 152–161. https://doi.org/10.1016/j.neucom.2019.10.007

Bock, S., Goppold, J., & Weiß, M. (2018). An improvement of the convergence proof of the ADAM-Optimizer. arXiv preprint arXiv:1804.10587.

Zhang, Z., & Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in Neural Information Processing Systems*, 31.

Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic routing between capsules. *Advances in Neural Information Processing Systems*, 30.