Neural Computing and Applications (2022) 34:15007–15029
https://doi.org/10.1007/s00521-022-07331-0

ORIGINAL ARTICLE

Joint design and compression of convolutional neural networks as a Bi-level optimization problem

Hassen Louati[1,2] · Slim Bechikh[2] · Ali Louati[1,2] · Abdulaziz Aldaej[1] · Lamjed Ben Said[2]

Received: 2 January 2022 / Accepted: 18 April 2022 / Published online: 17 May 2022

Abstract
Over the last decade, deep neural networks have shown great success in the fields of machine learning and computer vision. Currently, the CNN (convolutional neural network) is one of the most successful networks, having been applied in a wide variety of application domains, including pattern recognition, medical diagnosis and signal processing. Despite CNNs' impressive performance, their architectural design remains a significant challenge for researchers and practitioners. The problem of selecting hyperparameters is extremely important for these networks. The reason for this is that the search space grows exponentially in size as the number of layers increases. In fact, all existing classical and evolutionary pruning methods take as input an already pre-trained or designed architecture. None of them take pruning into account during the design process. However, to evaluate the quality and possible compactness of any generated architecture, filter pruning should be applied before the communication with the data set to compute the classification error. For instance, a medium-quality architecture in terms of classification could become a very light and accurate architecture after pruning, and vice versa. Many cases are possible, and the number of possibilities is huge. This motivated us to frame the whole process as a bi-level optimization problem where: (1) architecture generation is done at the upper level (with minimum NB and NNB) while (2) its filter pruning optimization is done at the lower level. Motivated by evolutionary algorithms' (EAs) success in bi-level optimization, we use the newly suggested co-evolutionary migration-based algorithm (CEMBA) as a search engine in this research to address our bi-level architectural optimization problem. The performance of our suggested technique, called Bi-CNN-D-C (Bi-level convolution neural network design and compression), is evaluated using the widely used benchmark data sets for image classification, called CIFAR-10, CIFAR-100 and ImageNet. Our proposed approach is validated by means of a set of comparative experiments with respect to relevant state-of-the-art architectures.

Keywords Deep CNN architecture design · Deep CNN architecture compression · Evolutionary algorithms · Bi-level optimization

✉ Ali Louati
a.louati@psau.edu.sa

Hassen Louati
hassen.louati@stud.acs.upb.ro

Slim Bechikh
slim.bechikh@fsegn.rnu.tn

Abdulaziz Aldaej
a.aldaej@psau.edu.sa

Lamjed Ben Said
lamjed.bensaid@isg.rnu.tn

1 Department of Information Systems, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia

2 SMART Lab, University of Tunis,ISG, Tunis, Tunisia

1 Introduction

CNNs are currently among the most widely used machine learning models for object recognition and computer vision [1–3]. Despite the fact that CNN with several layers have been in use for a long time, they gained widespread interest in the scientific community in 2006 following the work of several researchers, such as Bengio et al. 2007 [4] and LeCun et al. 2015 [5]. In fact, when dealing with extremely complex classification problems or for specific purposes, CNN has become increasingly used, especially at a high level of precision. CNNs architecture is defined by a large number of hyperparameters, which should be fine-tuned to optimize the architecture. Previous works in the literature have been proposed with the goal of optimizing architectures such as ResNet [6] and VGGNet [7]. Unfortunately, the majority of these architectures are either defined manually by experts or automatically created using greedy induction techniques. Despite the impressive performance of the CNN design, experts in the disciplines of optimization and machine learning proposed that improved structures may be discovered using automated approaches. Evolutionary computation researchers proposed modeling this task as an optimization problem and then solving it with an appropriate search algorithm [8]. Indeed, selecting the blocks' number, nodes per block and the graph's topology within each CNN block is similar to solving a problem of optimization within a large search space. Due to the fact that EAs are capable of approximating the global optimum and thus avoiding local optimal solutions (architectures), authors in [9] proposed recently the use of such metaheuristic techniques to handle the challenge of optimizing the CNN architecture.

Efficient model designs [10, 11] focus on acceleration over compression through the use of optimized convolutional operations or network architectures. Recently, as a means of improving accuracy, deepening on CNN models has become a popular trend, as demonstrated by ResNet [6], VGGNet [7] and Xception [12]. Indeed, it is challenging to deploy these deep models on low-resource devices such as smartphones and mobile robots. However, billions of network parameters represent a significant storage overhead for embedded devices, such as the VGG16 deep learning model, which has over 138 million parameters and requires over 500MB of memory space to classify a 224 224 image. Obviously, such a large model cannot be directly deployed in on-board devices. Deep compression process is a critical technique for resizing a deep learning model by consolidating and removing inadequate components. However, compressing deep models without significant loss of precision is a critical issue. Several techniques for CNN pruning have been presented, including neurons, filters and channel pruning approaches [13], which reduce model weight by removing unimportant connections.

Due to the fact that EAs are capable of approximating the global optimum and thus avoiding locally optimal solutions, recent works [9, 14, 15] recommend that similar metaheuristic algorithms be employed to address the CNN architectural optimization challenge in the field of network compression. To do so successfully, the solution encoding, the fitness function and the variation operators must all be defined. In fact, all previous work focuses on compressing manual architectures and their nonexistent compression for automated CNN architecture. We notice that in our previous works, the problematic of compression is not tackled.

Motivated by recent survey papers [9, 14, 16] on deep neural networks pruning and the reported interesting results, we decided to tackle the problem of filter pruning. As any CNN architecture could be pruned in different ways, we framed the problem of "joint design and pruning" as a bi-level optimization problem. The upper-level goal is to search for good architectures, while the lower-level one is to apply filter pruning on the considered architecture. Indeed, the evaluation of an upper-level architecture requires sending this architecture to the lower level to execute the fitter pruning on it by deactivating some filters. The filters that should be deactivated could not be known before hand as the number of possibilities is huge and corresponds to a whole search space. For this reason, the filter pruning task is executed at the upper level as an evolutionary optimization (search) process. In this way, the fitness evaluation of each upper-level solution (architecture) requires the (near) optimal filter pruning decision (encoded as a binary vector where 0 means that the corresponding filter is deactivated) found at the lower level. By following such as a bi-level optimization process, the final output of our approach is an CNN architecture with minimum number of filters and optimized topology. Figure 1 illustrates an example of a Bi-CNN-D-C (bi-level convolution neural network design and compression) scenario. To our knowledge, this is the first study to model and solve the CNN architecture design and compression problem as bi-level method. Each upper-level solution necessitates solving a separate lower-level optimization problem; the computational cost intends to be prohibitively expensive. We address this issue by solving combinatorial BLOPs using CEMBA [17]. Indeed, each upper-level population collaborates with its corresponding lower-level population. This fact enables a significant reduction in the number of evaluations performed during the lower-level search process. The main contributions of our paper could be summarized as follows:

- For the first time, an evolutionary method that combines CNN architecture generation with filter pruning within the optimization process is developed. This is motivated by the fact that any generated architecture should be first pruned before evaluating its classification performance.
- The joint design and filter pruning is modeled as a bi-level optimization problem where architectures are generated through crossover and mutation at the upper level with minimum NB and NNB, while filter pruning of each architecture is applied at the lower level.
- The bi-level optimization modeling is solved using a bi-level co-evolutionary algorithm to ensure the effective collaboration between the architecture generation (at

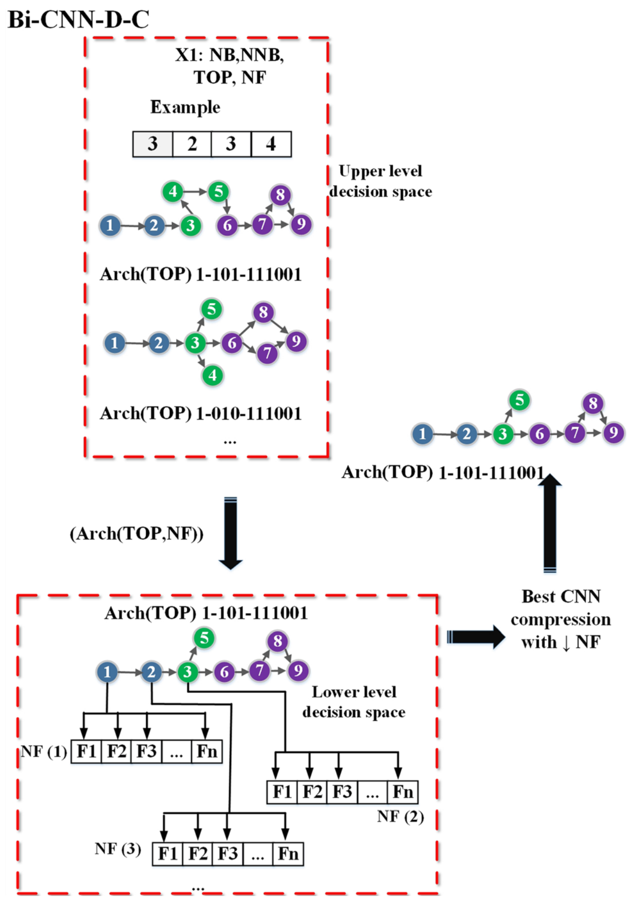


Fig. 1 Bi-CNN-D-C scenario

the upper level) and the filter pruning at (the lower level).

- Detailed experiments on CIFAR and ImageNet data sets in addition to a COVID-19 case study are conducted in comparison with several recent and prominent peer works. The merits of our proposed algorithm, Bi-CNN-D-C, are demonstrated based on several metrics including the classification error, the number of GPU-Days and the number of parameters.

The rest of this paper is structured as follows. Section 2 summarizes the review of the literature on CNN pruning. Section 3 details our proposed approach. Section 4 details the experimental design and performance analysis results. Finally, in Sect. 5, the paper is concluded and some future research directions are suggested.

2 Related work

2.1 CNN design based on evolutionary optimization

Recently, some researchers have taken an interest in EAs as a means of evolving deep neural network architectures. A survey on applications of swarm intelligence and evolutionary computing based optimization of deep learning models has been published by Darwish et al. [9]. Based on this survey, we selected the most representative:

- Cheung and Sable [18] optimized the architecture hyperparameters using a hybrid EA on the basis of the diagonal Levenberg Marquardt technique with rapid convergence and a low computing cost of fitness assessments number. They established the critical role of architectural choices in convolutional layer networks. Their findings demonstrate that even the simplest evolution strategies can yield significant gains. When variation effects are present, the employment of evolved parameters in combination with local contrast normalization preprocessing and absolute value across layers has proven a compulsive performance on the MNIST data sets [19].
- Fujino et al. [20] presented evolutionary Deep Learning, called evoDL, as a technique for discovering unique architectural designs. This technique is intended to be used to investigate the development of hyperparameters in deep convolutional neural networks, called DCNNs. Additionally, authors proposed AlexNet as a fundamental framework of the CNN and optimize both the parameters tuning and activation functions using evoDL.
- Real et al. [21] used the CIFAR-10 and CIFAR-100 data sets to develop the CNN structure in order to identify the classification model. They presented a mutation operator that may be used to avoid locally optimum models. They demonstrated that neuro-evolution is capable of constructing highly accurate networks.
- Xie et al. [22] maximized the recognition accuracy by representing the network topology as a binary string. The primary constraint was the high computing cost, which compelled the authors to conduct the tests on small-scale data sets.
- Mirjalili et al. [23] developed an adaption for solving bi-objective models, called NSGA-Net. The image classification and object alignment results obtained demonstrate that NSGA-Net is capable of providing the user with less than complicated correct designs.
- Alejandro et al. [24] developed EvoDEEP to optimize network characteristics by calculating the probability of

layer transitions based on the finite state machines concept. The goal was reducing classification error rates and preserving the layer sequence.

- Real et al. [25] provided a GA with an updated tournament selection operator that takes into account the age of the chromosomes while selecting youngest chromosomes. The architectures are described as small directed graphs with edges and vertices representing common network actions and hidden states. They developed novel mutation operators connecting the edges' origin to other vertices and rename the edges arbitrarily in order to cover the entire search space.
- Sun et al. [26] developed an evolutionary technique for improving convolutional neural network designs and initializing their weights for image classification problems. This aim was realized by developing a unique approach for initializing weight, a novel encoding variable-length chromosomes strategy, a slacked binary tournament selection methodology and an efficient fitness evaluation technique. Experiments indicated that the EvoCNN methodology surpasses clearly a wide number of existing approaches in terms of classification performance on practically all data sets investigated.
- Lu et al. [26] established a multi-objective modeling of the architectural search problem for the first time by minimizing two potentially conflicting objectives: classification error rate and computational complexity, as measured by the number of floating point operations (FLOPS). In order to execute a multi-objective EA, they updated the non-dominated sorting GA-II (NSGA-II) algorithm.
- Jing et al. [27] developed a multi-objective model aiming to maximize classification accuracy while keeping the tuning parameters to a minimum. The proposed model was solved based on a hybrid binary encoding representing component layers and network connections using multi-objective particle swarm optimization with Decomposition, called MOPSO/D. The architectures discovered are considered to be exceptionally competitive when compared to models created manually and automatically.

2.2 CNN compression

Deep network compression is one of the most significant strategies for resizing a deep learning model by combining the removal of ineffective components [14]. However, compressing deep models without considerable loss of precision is a key challenge. Recently, many studies have been focused on discovering new techniques to minimize the computational complexity of CNNs based on EAs while retaining their performance [14]. We divide the network compression techniques into three categories depending on the existing work: filter pruning [29–32], quantization [33–37] and Huffman encoding [38–40].

The convolutional operation in the CNN model integrates a large number of filters to improve its performance under various classification and prediction processes [41]. Recently, various pruning-based filter pruning techniques [29–32] have been suggested. The addition of filters enhances the defining features of the spatial characteristics generated by the CNN model [9, 42]. However, this increment results in a significant increase in the DNN model's FLOPs. As a result, removing superfluous filters is critical for reducing the computational requirements of the DCNN model. Figure 2 illustrates a scenario using filter-level pruning. We summarize the most important works on filter pruning currently available:

- Luo et al. [31] introduced an efficient framework named ThiNet for accelerating the operation of the CNN model through the use of compression during the training and testing phases. They implemented filter-level pruning, in which a filter that is no longer necessary is deleted based on statistical information generated from the following layer. The authors proposed pruning filters at the filter level as an optimization issue for determining which filters to prune. They solve the optimization problem with a greedy method which is defined as follows:

$$\begin{aligned} &arg \min_{E} \sum_{N}^{i=1} \left(y_i - \sum_{j\epsilon E} X_{ij} \right)^2 \\ &Subject\ to, |E| = k \times c_{rate} \\ &\quad E \subset \{1, 2, \ldots, k\}, \end{aligned} \tag{1}$$

  where N represents the training example number (X_i, Y_i), $|E|$ represents the subset element number, k represents the channel number within the CNN model and c_{rate} represents the channels number retrained after compression.
- Bhattacharya and Lane [43] developed a technique for CNN compression that removes sparsification in convolutional filters and the fully connected layer. The primary goal was to minimize the amount of storage required by devices throughout the training and inference processes. By utilizing layer separation and convolutional filters, the computational and spatial complexity of the DCNN model can be significantly expanded.
- Zhou et al. [44] suggested a multi-objective optimization problem for filter pruning, followed by a knee-guided approach. They proposed a trade-off between performance degradation and parameter count. The fundamental concept is to remove parameters that

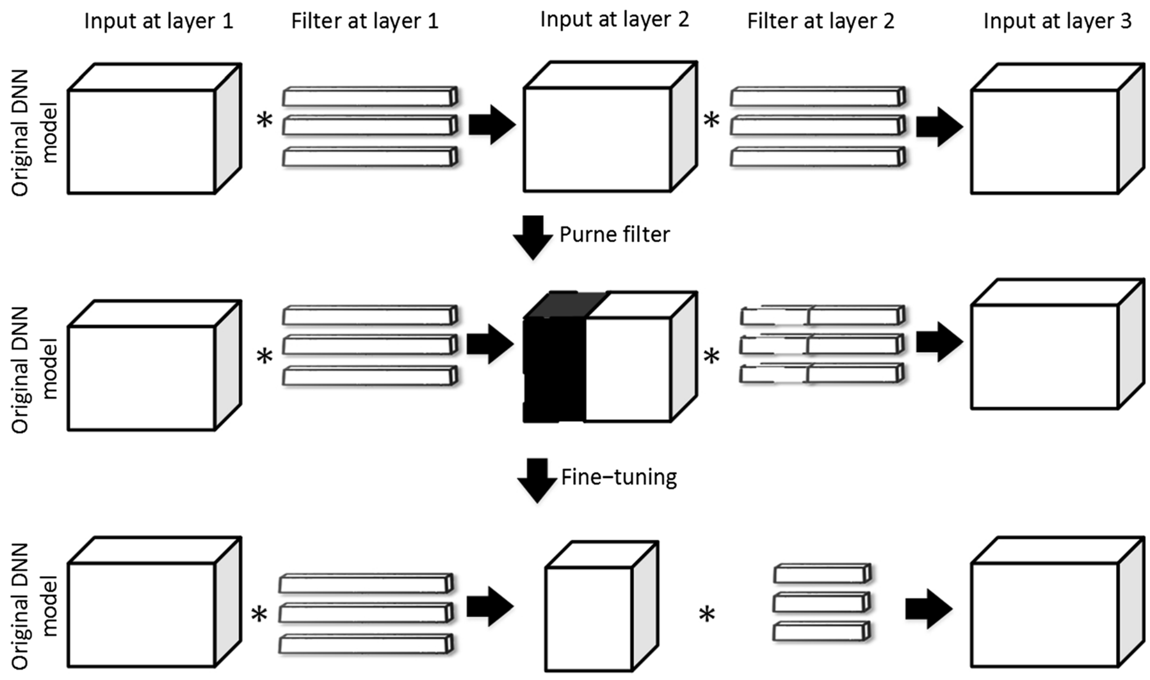


Fig. 2 An illustration of how filter-level pruning works [28]

contribute to performance degradation. They used the performance loss criteria to determine the significance of a parameter. To produce a tiny compressed model, the number of filters should be limited to a minimum while yet achieving a high degree of precision. The challenge can be handled by identifying a compact binary representation capable of pruning the maximum number of filters while maintaining a reasonable level of performance. This work has the advantage of lowering the number of parameters and processing overhead.

- Huynh et al. [45] presented the DeepMon approach for developing deep learning inference on mobile devices. They assert that they can do inference in a short period of time and with minimal power consumption by using the graphics processing unit on the mobile device. They presented a method for convolutional processes on mobile graphics processing units to be optimized. The technique repurposes the results by utilizing CNN's internal processing structure, which includes filters and a network of connections. Thus, deleting filters and superfluous connections demonstrates faster inference.
- Denton et al. [32] significantly reduce the time required to evaluate a large CNN model developed for object recognition. The authors used insignificant convolutional filters to develop approximations that significantly minimize the necessary computation. They began by compressing each convolutional layer using an appropriate low-rank approximation and then fine-tuning until prediction performance was recovered.

Weight quantization decreases both the storage and computing requirements of the CNNs model [33–37], Han et al. [34] suggested a weight quantization approach for compressing deep neural networks by reducing the number of bits needed to encode weight matrices. The authors attempt to decrease the number of weights which should be stored in memory. The identical weights are removed as a result, and numerous connections are derived from a single remaining weight. The authors used integer arithmetic for inference and floating point computations for training. Jacob et al. [37] presented a quantization technique based on integer arithmetic for inference. Integer arithmetic is more efficient than floating point arithmetic and requires fewer bits to represent. Additionally, the authors construct a training step that mitigates the accuracy penalty associated with the conversion of floating point operations to integer operations. As a result, the suggested technique eliminates the trade-off between on-device latency and accuracy degradation caused by integer operations. The authors performed inference using integer arithmetic and training using floating point operations. Quantization is a technique that creates an affine mapping between integers Q and real numbers R, i.e., of the type

$$R = W(Q - T) \tag{2}$$

where Eq. 2 denotes the quantization method with the parameters W and T. For instance, Q is set to 8 for 8-bit quantization. W is an arbitrary positive real number, and T has the same type as variable Q. The quantization strategy for compressing DNN models is explored in the current literature [33–37]. The strategies cover model reduction by arranging weight matrices optimally. However, the previous work does not address the negative repercussions of weight quantization or its estimation complexity.

A **Huffman encode** is a lossless data compression algorithm that is frequently used [46]. Schmidhuber et al. [39] utilized Huffman coding to compress text files generated by a neural prediction network. Han et al. [40] used a three-stage compression strategy to encode the quantized weights, which included pruning, quantization and finally Huffman coding [9]. Ge et al. [47] proposed a hybrid model compression technique based on Huffman coding to capture the sparse nature of trimmed weights. Huffman codes are superior to all other variable-length prefix codings. However, Elias and Golomb. 1975 encoding [48] can take advantage of various intriguing characteristics, such as the recurrence of specific sequences, to achieve greater average code lengths.

Despite the interesting findings of design and compression work on optimizing deep learning architectures, all researchers believed architecture optimization was a single-level problem. Therefore, We show that CNN design can be improved if two optimization levels are considered, where a search space is assigned to each level.

3 The proposed approach

3.1 Bi-CNN-efficient and compression overview

The two following questions motivate our bi-level model:

- How can we design a less complex architecture with the minimum possible convolution blocks (NB) and convolution nodes per block (NNB) while achieving high performance, which is highly dependent on the topologies of the convolution blocks' graphs?
- For any CNN architecture, there are a large number of filters per layer; how could we determine the optimal number of filters per layer?

For the following reasons, a bi-level modeling of the design and compression architecture is necessary to solve these two research problems. On the one hand, optimizing the design and compression of hyperparameters requires intelligent sampling of the entire high-level search space. On the other hand, in order to assess the upper-level quality solution (NB, avgNNB, NF, Err), we must pass the vector (TOP, NF) to the lower level as a fixed parameter, with the intention of finding the best selected filters (NF) from the lower-level search space. Once the lower-level process is completed, each architecture is passed through the process of quantization of 32-bit floating point values into 5-bit integer levels. This process is used to further reduce the stored size of the weights file. These strategies approached the problem as a bi-level optimization problem, evaluating each pair of hyperparameters independently. This observation demonstrates a significant inconvenience of present approaches and is the paper's key research gap. The bi-level modeling of the CNN architecture design and compression optimization problem illustrated in Fig. 3 demonstrates our approach.

In fact, the upper-level optimization process is concerned with optimizing the (NB, NNB) and determining the optimal topology sequence in terms of classification accuracy while the lower level focused on the CNN pruning filters. As we are in the case of bi-level optimization, we have two kinds of solutions: (1) an upper-level solution and (2) a lower-level one. Indeed, the upper-level solution is encoded as a vector containing two sub-vectors: (1) the first one contains integer values expressing the NB and the NNB and (2) the second one is a binary sequence expressing the topology (encoding adopted from Genetic-CNN [49]). This encoding is chosen to reduce as possible the chromosome length at the upper level. The lower-level is a sequence of sub-vectors each expressing the filter pruning decision of the corresponding convolution node. Modeling such a bi-level problem with the goal of finding better architectures with less complexity would be a better idea. It would be wiser to model such a bi-level problem with the goal of identifying more complex architectures.

3.2 Bi-CNN-D-C: adaptation of CEMBA to the bi-level model

To solve the proposed bi-level optimization model using CEMBA's adaptation, the following upper-level processes should be detailed:

3.3 Upper level: CNN design

– ***Upper-level solution encoding:*** It is constructed by concatenating the number of blocks NB with an integer sequence NNB representing the node numbers in every block and with a sequence of graph topology of the convolution layer. A possible directed graph is represented by this object.
– ***Upper-level fitness function:*** Aim to evaluate an upper-level solution, we must reduce the complexity of the CNN architecture as much as possible by optimizing the (NB,NNB) while achieving high performance. In order to accomplish this, we propose the following fitness function:

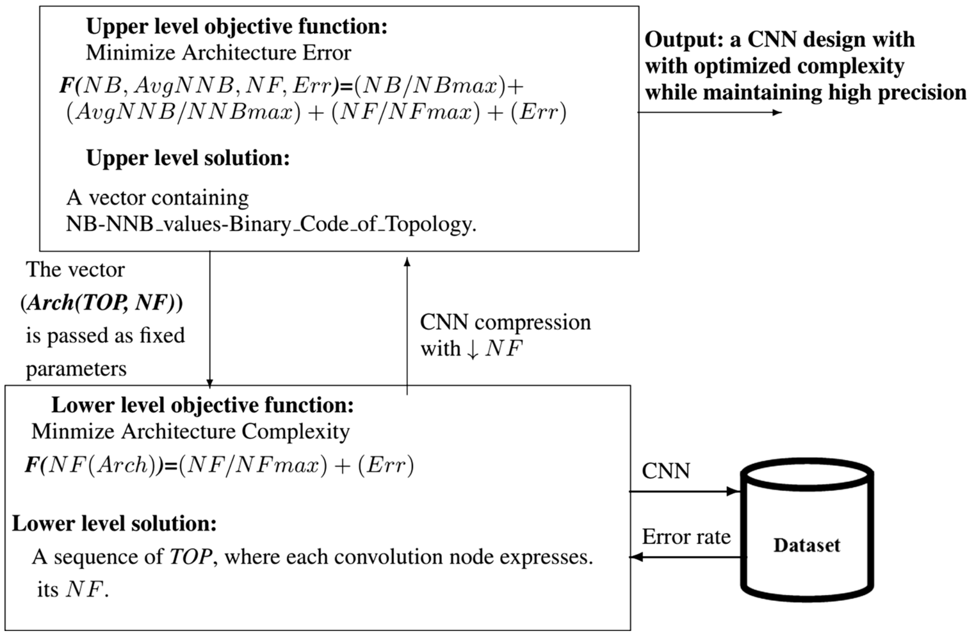

Upper level objective function:
Minimize Architecture Error
F(NB, AvgNNB, NF, Err)=(NB/NBmax)+ (AvgNNB/NNBmax) + (NF/NFmax) + (Err)
Upper level solution:
A vector containing NB-NNB_values-Binary_Code_of_Topology.
Output: a CNN design with with optimized complexity while maintaining high precision
The vector (Arch(TOP, NF)) is passed as fixed parameters
CNN compression with ↓ NF
Lower level objective function:
Minmize Architecture Complexity
F(NF(Arch))=(NF/NFmax) + (Err)
Lower level solution:
A sequence of TOP, where each convolution node expresses. its NF.
CNN
Error rate
Dataset

Fig. 3 Design and prune the CNN architecture using a bi-level model

$$
\begin{aligned}
F(NB, avgNNB, NF, ERR) = {} & (NB/NBmax) \\
& + (AvgNNB/NNBmax) \\
& + (NF/NFmax) + (Err)
\end{aligned}
$$

To differ the population at the upper level, the uniform crossover operator [50] has been considered, which allows for variation across all chromosomal segments. To guarantee the diversity of solution variation, every parent solution is converted into a binary sequence based on the Gray encoding [51]. This encoding technique is inspired by the fact that neighboring integer values vary by just one bit, which is not true for the conventional binary encoding [52]. This has been shown to help prevent premature convergence at so-called Hamming walls [53], where too many simultaneous mutations (or crossover events) are required to change the chromosome to a more advantageous solution. A uniform crossover procedure randomly selects a recombination mask from a uniform distribution. This mask represents a binary vector of 0 and 1. The first offspring is formed by extracting the bit from both parents in case that the corresponding mask bit is equal to 0 and from both parents if the corresponding mask bit is equal to 1. The second offspring is generated using the inverse mask. Finally, each offspring is encoded into an integer vector and the value of its fitness function is calculated. Due to the fact that the proposed solution represents a vector of integers, the length of the binary chromosome is a multiple of four (we mean that each integer is encoded using 4 bits). If this is not the case for a created offspring, the last bits are removed to maintain a multiple of four length. It is crucial to remember that the NB value of a created offspring may vary from the length of the NNB sequence. The offspring solution is rectified in this example by changing the sequence length value from NNB to NB. Then, in order to optimize accuracy, we must look for the optimum topologies. Then, in order to maximize accuracy, we must determine the optimal topologies. As seen in Fig. 4, the answer will be encoded as a squared binary matrices sequence, one for each conceivable directed network. A value of 1 indicates that the row node is the column node predecessor; a value of 0 indicates that there is no relationship between the two nodes. Due to the fact that this work is concerned with the CNN model, the following constraints must be respected:

- Each active convolution node should have a predecessor node. The latter may be a previous convolution node or the convolution node at the input.
- Each active convolution node should have a successor node. The latter may be a convolutional successor node or the output successor node.
- Any active convolution node should have predecessors in its preceding layers. For instance, node 4 may have predecessors in the form of nodes 3, 2, 1 and the input node.
- The initial convolutional node should have a single preceding node that acts as the input node.
- The last convolution node's output node should have only one successor node.

The goal of mutation is to inject abrupt changes within the population to ensure its diversity and thus its ability to explore other regions of the search space (e.g., non-visited

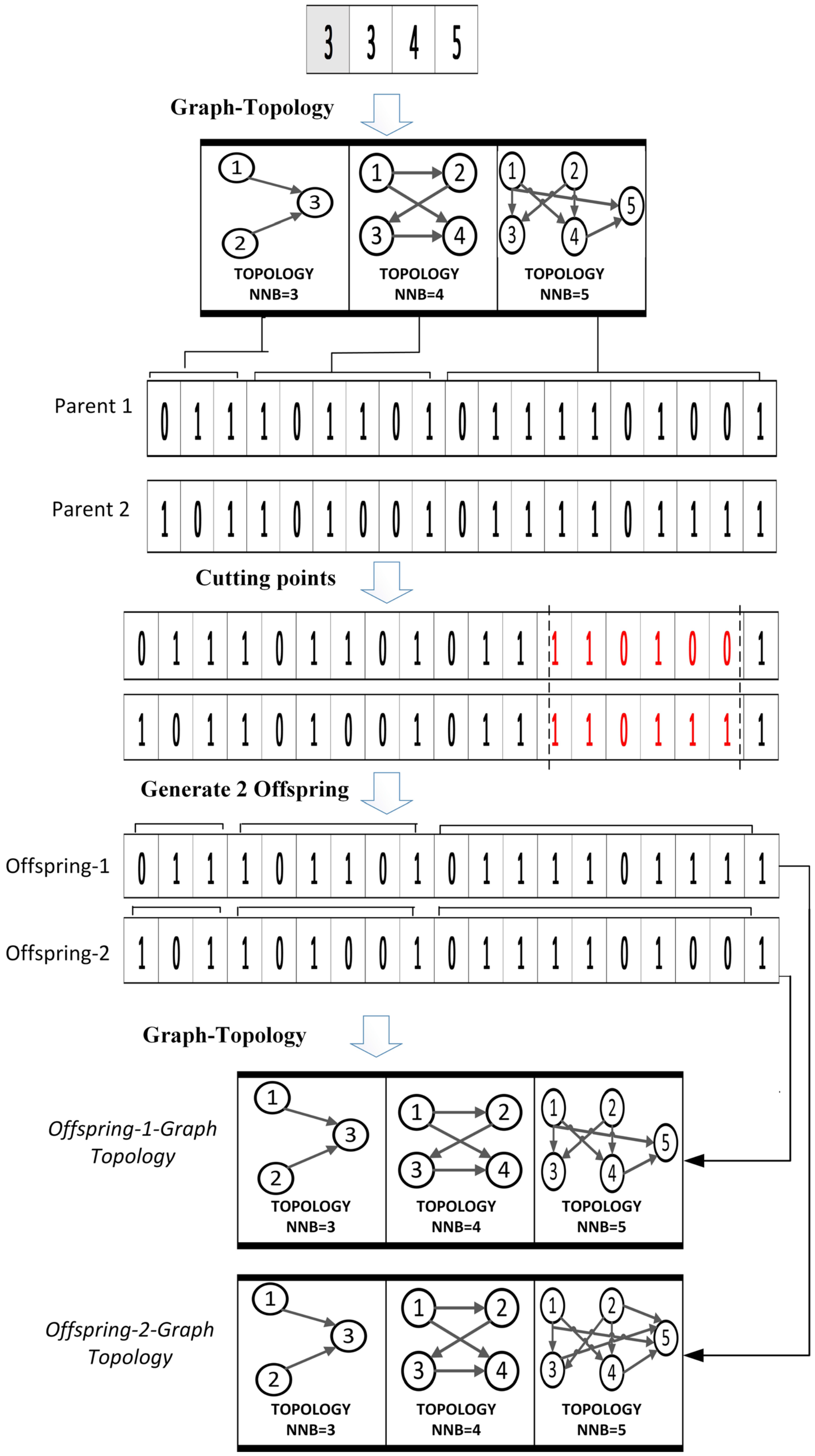


Fig. 4 Upper level: Crossover operator [4]

ones so far). Among these operators, we cite one-point mutation, random reset, inversion mutation, just to name a few. As we adopted binary encoding as both levels (NB integer, NNB integer, topology, filter activation decision vector), the one-point mutation allowed us a progressive change of the subject solution. In this way, diversity is slightly incorporated within the evolutionary process. As with the crossover operator, the solution of mutation operator is converted to a binary string using Gray encoding before applying the one-point mutation. Due to the possibility that the variation will alter the NB field, the consistency is achieved by using the following repair technique. (assumes LNS = length (NNB sequence)).

- If (NB< LNS), then delete the chromosome's final (LNS-(LNS-NB)) integers.
- If (NB > LNS), then at the end of the chromosome, add (NB-LNS) randomly generated integers. These two conditionals guarantee that NB will always equal LNS.

On the basis of prior work [1], we suppose that the quantity of NB must be in the interval [9, 11] while the quantity of NNB within the interval [32, 49] in this research. The Acc is computed using the holdout validation method [54] by 80% of data records are randomly selected for training and 20% for testing.

3.4 Lower level: CNN compression

- ***Encoding the solution of the Upper level:*** It resembles the selected filters number ***(NF)*** to be pruned in the convolutional layer. The filter subset of the binary vector represented by a bit sequence of 0, 1.
- ***Fitness function of the lower level :*** To assess the lower-level solutions, the complexity of the CNN architecture must be reduced by minimizing the NF while preserving or improving the high precision. To do so, we provide the following fitness function:

$$F(NF(Arch)) = (NF/NFmax) + (Err) \tag{3}$$

In the proposed lower-level deep pruning filters, a binary strings is adopted for representing the filters of a CNN model. It is essential to know that the suggested algorithm prunes convolution layers. DCNNs are constructed basically by stacking multiple convolution, pooling and fully connected layers. The goal of this paper is the automated joint design and filter pruning of CNN architectures. As filters are located only within convolution layers, only these latter are pruned. Our approach considers each bit as one single filter, e.g., if we are looking to represent two layers of convolution, 16 filters for one layer and 32 filters for the other one, we will require a string of 48-bit, while a bit with a zero assigned indicates the elimination of the corresponding filter. Furthermore, during pruning simple of CNN models, uniquely one bit string is needed, with every bit representing a model filter. Figure 5 shows the binary representation of a CNN before and after pruning. The two-point crossover operator is used to vary the population [50] since it enables chromosome parts to change. Each parent solution in this operation is a set of binary strings [51]. A couple of cutting points are chosen for each couple of parent in this process, after which the bits between the cuts will be exchanged to produce a couple of offspring solutions. In fact, the two-point crossover is adopted to allow the variation of all parts of the chromosomes. Indeed, if the one-point crossover is used, the extreme regions (extreme genes) of the chromosomes are likely to still unchanged. This could significantly reduce the exploitation ability of the crossover and the population diversity. To mitigate this issue, researchers proposed the use of two cut points instead of a single one to allow the variation of the entire chromosome.

Similar to crossover, the solution of mutation operator is encoded as a binary string, followed by a random mutation of one point. A point on the chromosomes of both parents is chosen at random and referred to as a "crossing point." The bits to the right of this point are exchanged between the two parent chromosomes.

A quantization of 32-bit floating point values into 5-bit integer levels is used to further reduce the stored size of the weights file. The quantization part are spread linearly between Wmin and Wmax because it produces higher accuracy results than density-based quantization; thus, even if a weight occurs with a low probability, it may have a high value and therefore a high influence, and if quantized to be less than its real value. This stage produces a compressed sparse row of quantized weights.

Due to the statistical characteristics of the quantization output, Huffman compression might be used to further reduce the weights file. However, this adds the additional hardware needs of a Huffman decompressor and a compressed sparse row to weights matrix converter.

The test error is computed using the holdout validation technique [55], which randomly selects 70% of the data records for training and 30% for testing. To deal with this the over-fitting issue, the training data (70%) is divided into 5 folds, and thus, fivefold cross-validation is applied during training. The classification performance is averaged over the 5 folds of the training partitions. Figure 6 illustrates the adopted validation strategy in this work [56]. Eventually, in the experiments, we report the classification error on the test data (30%).

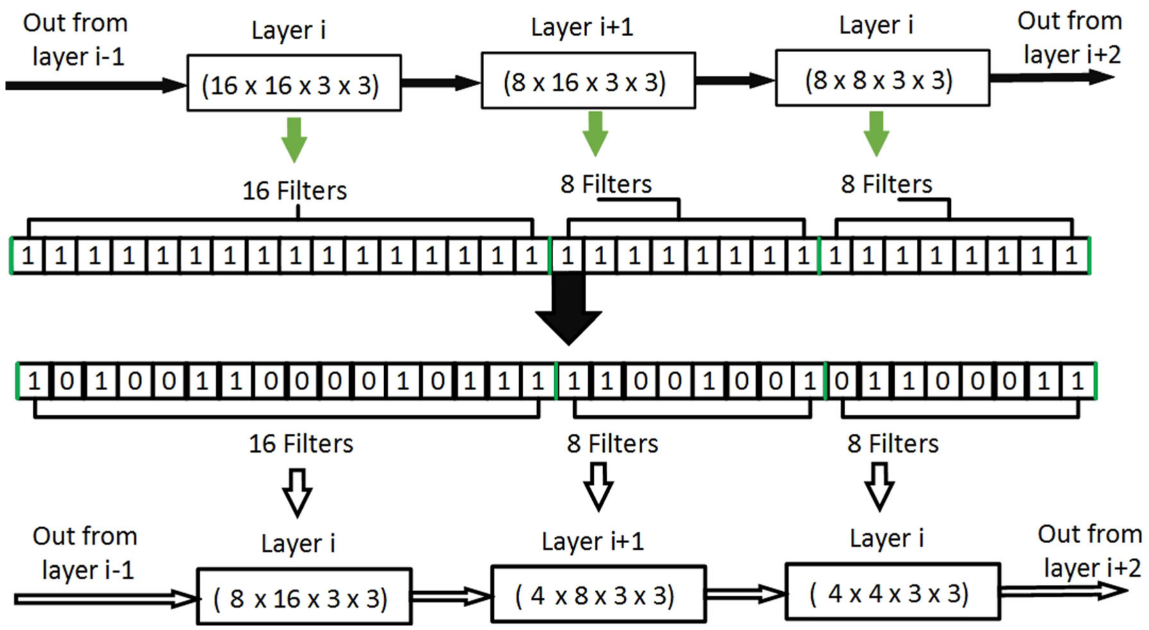
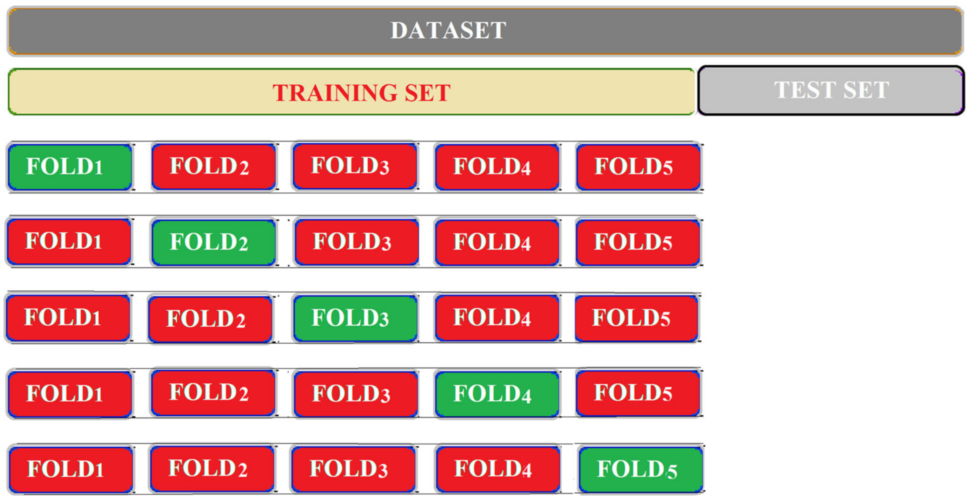


Fig. 5 CNN model's binary representation before and after pruning half of the filters from every layer

Fig. 6 Adopted nested validation strategy in our work



4 Experimental study

4.1 Benchmarks and research questions

We compared the performance of the suggested strategy to earlier work using the two frequently used benchmark data sets: CIFAR-10, CIFAR-100 and ImageNet. The first batch of data contains 60,000 32×32 RGB images that are grouped into ten groups of 6000 images each. Indeed, the test sample size is 10,000, but the training set sample size is 50,000. The other data set is similar to CIFAR-10, but differs in terms of class count; it comprises 100 classes and each class contains 600 images. Both data sets present significant challenges due to factors like as noise, image size and image rotation. The photographs are incremented during the processing stage to ensure that the comparisons are fair. Indeed, four zero pixels are added to each image, resulting in the modification of a 32×32 image. Following that, the clipped image is arbitrarily compressed with a probability of 0.5. This technique was influenced by [57]. Due to the enormous number of classes, the most of current studies do not conduct experiments using CIFAR 100 data sets. To demonstrate our Bi-CNN-D-C performance, we carry out a series of tests on CIFAR-10 and CIFAR-100. Finally, the ImageNet The third batch of collection contains 14,197,122 images that have been annotated using the WordNet hierarchy. Since 2010, the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has used the data set as a benchmark for image classification and object recognition. The freely available data set provides a collection of training images that have been hand labeled. Additionally, a set of test images is released without the associated manual annotations. Our examination study will address the following main questions:

- How do the architectures generated by Bi-CNN-D-C compare to previous work on CIFAR-10 image classification?
- Is it possible for Bi-CNN-D-C to maintain its efficacy on CIFAR-100 and ImageNet, that is, when the number of classes is increased to 100?
- Is Bi-CNN-D-C capable of producing high-quality designs in spite of its high computational cost?

To solve these RQs, we compare the best architecture developed by Bi-CNN-D-C to previously generated and current designs.

4.2 Performance indicators

According to previous work, the most used performance measures in image classification using DNN are error rate and floating point operations (FLOPs). Equation (8) gives the error rate (Test error), where FP stands for false positives, FN stands for false negatives and NE stands for the total number of samples.

$$test\ error = (FP + FN)/NE \tag{4}$$

#Params is the sum of the weights and biases in the convolution layer, and is given by Eq. (4), where Wc, Bc, pc and K are the weights, biases, parameters and the convolution layer size, respectively; N represents the kernels number; and C represents the channels number in the input image [1].

$$\begin{aligned} Pc &= Wc + Bc \\ Bc &= N \\ Wc &= K2 \times C \times N \end{aligned} \tag{5}$$

The *GPUDays* metric is the number of GPU day units where a unit means that the algorithm has performed one day on one GPU.

FLOPs represent the number of floating point operations per second, which is accredited as the computation speed, which is a measure of hardware performance, and is given by Eq. (7) where W, H and C_{in} represent, respectively, the width, height and number of channels of the input characteristics map. K is the core width, and C_{out} is the number of output channels.

Each method is executed 20 times, and then, the performance values of the best 20 outputted architectures of each method are averaged (for each metric).

$$FLOPs = 2HW(C_{in}k^2 + 1)C_{out} \tag{6}$$

4.3 Peer algorithms and parameters setup

The most representative previous studies from both categories of CNN generation methods are compared to our Bi-CNN-D-C methodology. From the evolutionary approach based on CNN Design, we selected BLOP-CNN, Genetic-CNN, LargeScale-Evo, AE-CNN, CNN-GA and NSGA-Net. From the pruning approaches, we selected DeepPruningES, Channel-Evo and Classical-Pruning [58] which are added to the experimental comparisons. The parameters of the compared algorithms are established on the basis of the commonly used trial-and-error strategy [59] in order to achieve as much impartiality as possible in the comparisons. The parameter settings used in our studies are summarized in Tables 1 and 2. The TensorFlow framework and Python (version 3.5) are used for implementation. In addition, 8 Nvidia 2080Ti GPU cards are used to assess the CNN architectures obtained from the testing data.

4.4 Comparative results

4.4.1 Comparisons to evolutionary design methods

Tables 3, 4 and 5 sum up the comparative results obtained for the various architectures generated by the various methods of CNN design on CIFAR-10, CIFAR-100 and ImageNet. We detail and clarify these results for each category and metric in the sections that follow. We detail and clarify these results for each category and metric in the sections that follow. In fact, evolutionary methods have the ability to escape local optima and cover the entire search space because of their global search capability. They also have the capability of accepting less performing architectures on a probabilistic basis through the use of the mating selection operator, which allows them to cover the entire search space. Indeed, The Acc of EA ranges from 92.90% to 94.12% for CIFAR-10 and from 70.97% to 88.42% percent for CIFAR-100. We noticed also that BLOP-CNN provided the highest performance with an *Acc* value of 98.12% for CIFAR-10 and 88.42% for CIFAR-100, where the *Acc* of Large-Scale-Evo, Genetic-CNN, CNN-GA, AE-CNN-NSGA-3, NSGA-4 and BLOP-CNN, are 94.60%, 92.90%, 95.22%, 95.70%, 97.78%, 97.98% and 98.12%, respectively. Always referring to Tables 3 and 4, we see that Bi-CNN-D-C is the optimal method, with a slight increase in terms of Acc value of 98.24% for CIFAR-10 and 88.83% for CIFAR-100. From the perspective of solution encoding methods, Large-Scale-Evo solution encoding and generation are constrained by a large number of constraints, whereas NSGA-Net, BLOP-CNN and Bi-CNN-D-C are not. This decrease the search space for

Table 1 Summary of peer algorithms' parameters settings

Search method	Method type	Parameters	Value
CNN-GA	EA	Epochs	350
		Learning rate	0.1
		Momentum	0.9
		Population size	300
		# Of generations	3000
		Crossover probability	0.9
		Mutation probability	0.2
NSGA-NET	EA	Batch size	128
		Weight decay	5.00E-04
		Epochs	36/600
		Learning rate	Cosine Annealing
		Population size	300
		# Of generations	3000
		Crossover probability	0.9
		Mutation probability	0.1
Genetic-CNN	EA	Epochs	20
		Learning rate	0.001
		Population size	300
		# Of generations	3000
		Crossover probability	0.2
		Mutation probability	0.005
Large-Scale Evo	EA	Learning rate	0.1
		SGD with momuntum	0.9
		Batch size	50
		Weight decay	0.0001
		Population size	300
		# Of generation	3000
DeepPruningES	EA	Epochs	50
		Learning rate	0.01
		Population size	300
		# Of generations	3000
		Mutation probability	0.1
		Offspring size	20
ChannelPruning	EA	Epochs	200
		Learning rate	0.1
		Batch size	128
		Crossover probability	0.1
		Mutation probability	0.1
		Population size	300
		# Of generation	3000
Classical-Pruning	EA	Epochs	200
		Learning rate	0.001
		Batch size	128
		Weight decay	0.9
BLOP-CNN	EA	**Upper Level :** # Of generation	50
		Population size	30
		Crossover probability	0.9
		Mutation probability	0.1
		Lower Level: Batch size	128
		Epochs	50/350
		SGD Learning rate	0.1
		Momentum	0.9
		# Of generation	30
		Population size	20
		Crossover probability	0.9
		Mutation probability	0.1

Large-Scale-Evo; however, it may have a detrimental effect on the diversity factor, which is a critical component of EAs. Therefore, NSGA-Net, BLOP-CNN and Bi-CNN-D-C outperforms Large-Scale-Evo on both data sets. Based on previous works, the NB and NNB values are predefined manually.

This practice helps to constrain the search space, as we can obtain more complex architectures with varying block counts and sizes. Indeed, the algorithm could produce a large number of architectures having multiple topologies for each pair NB, NNB. This behavior distinguishes BLOP-CNN and Bi-CNN-D-C as the first algorithms in the literature that simultaneously vary and optimize various components as bi-level design. However, efficient model designs such as BLOP-CNN focus more on acceleration than compression by optimizing convolution operations or network architectures. That is the reason, we are driving the compression research process more effectively and efficiently.

We will now proceed to the analysis of the #Params. These metrics are used to represent the number of parameters. Due to the reduction blocks and minimization of NB and NNB at the upper level, and compression convolution layer hyperparameters (FSi,Nbits of quantized weights) at the lower level, the #Params of Bi-CNN-D-C are less than those of NSGA-Net-4 and BLOP-CNN, as shown in Tables 3 and 4. Notably, the CIFAR-10 and CIFAR-100 images are identical in size (i.e., the images are identical in both data sets, except that the number of classes increases from ten (CIFAR-10) to one hundred (CIFAR-100)). The final fully connected layer for CIFAR-100 takes longer to compute theoretically, but this is irrelevant because it is so small in comparison with the rest of the model.

Table 2 Summary of Bi-CNN-D-C parameters settings

Categories	Parameters	Value
Upper-level	# Of generation	50
Search space	Population size	30
–	Crossover probability	0.9
	Mutation probability	0.1
	Batch size	128
	Epochs	50/350
	SGD Learning rate	0.1
	Momentum	0.9
	# Of generation	30
	Population size	20
Lower-level	# Of generation	30
Search space	Population size	30
–	Crossover probability	0.9
	Mutation probability	0.1
	Batch size	128
	Epochs	50/350
	SGD Learning rate	0.1
	Momentum	0.9
	# Of generation	30
	Population size	20

In what follows, we analyze the computation time of our algorithm in terms of the number of GPUDays, which mainly depends on the number of function evaluations and the population architectures' number of parameters (params). We agree that the bi-level optimization process computational cost is very important (31 GPUDays as shown in Tables 3, 4 and 5), but acceptable with respect to evolutionary NAS methods. This could be explained by the low population size and number of generations defined from the start, which are both equal to 30. Despite these low values, Bi-CNN-D-C is able to generate pruned architectures with minimum number of filters and thus much reduced number of parameters. It is worth noting that our algorithm is the first one that applies filter pruning to evolved architectures within the evolution process, because existing methods apply such pruning to existing architectures that are passed as inputs to the pruning method. The considerable reduction in terms of the NF makes the GPUDays metric decreasing from one generation to another, as the time required to compute the architecture classification error is dropping with the advance of the optimization process.

4.4.2 Evolution trajectory at the upper level

When evolutionary algorithms are employed to solve real-world issues, we typically looking to realize whether or not they have converged. In this section, the evolutionary trajectories of the suggested method in terms of the benchmark data sets are studied. We analyze the convergence behavior of BLOP-CNN on CIFAR-10

Over the evolutionary search process, the NB quantity is decreased from 15 to 9. Additionally, the interval extent is minimized over generations, resulting in architectures with similar NB values at the conclusion of the optimization process. Due to the fact that NB and NNB are mutually exclusive objectives, minimizing the NNB is not easy. Indeed, the AVG NNB's slope decrease is notably less than NB's. This fact could be explicated by the fact that these two quantities may have a conflicting connection. Indeed, reducing the number of blocks NB may result in an increase in the number of nodes per block NNB, with the goal of maximizing or preserving the classification Acc. We believe that the EA at the top is attempting to strike a favorable trade-off between NB and NNB. Moreover, We find that the upper level is progressively maximized from generation to generation with a degree of convergence toward the maximum attained value. The quantity of Acc is increased from 40 to 98%. The first 15 generations have a rather steep maximization slope in comparison with the latter 15 generations. This could be explained by the fact that the search space of the possible CNN architecture is well explored during the first phase of the evolution process. During this phase, the huge search space contains low-, medium- and high-quality architectures. Thus, during the first phase the population is distributed over the entire search space. For this reason, a high number of low- and

Table 3 Obtained results for *Err, #Params,* and *GPUDays* on CNN Design's CIFAR-10

Architecture	Search method	Err CIFAR-10	#Params	GPUDays
Large-Scale Evo	EA	5.40	5.4 M	2.750
Genetic-CNN	EA	7.10	–	17
CNN-GA	EA	4.78	2.9 M	39
NSGA-3	EA	2.22	2.2 M	27
NSGA-4	EA	2.02	4.0 M	27
BLOP-CNN	EA	1.88	4.1 M	29
Bi-CNN-D-C	EA	1.76	1.8 M	31

Table 4 Obtained results for *Err, #Params,* and *GPUDays* on CNN Design's CIFAR-100

Architecture	Search method	Err CIFAR-100	#Params	GPUDays
Large-Scale Evo	EA	23.00	40.4 M	2.750
Genetic-CNN	EA	29.03	6.2 M	17
CNN-GA	EA	22.03	4.1 M	40
NSGA-3	EA	17.23	2.2 M	27
NSGA-4	EA	14.38	4.1 M	27
BLOP-CNN	EA	11.58	4.1 M	29
Bi-CNN-D-C	EA	11.17	1.8 M	31

Table 5 Obtained results for *Err, #Params,* and *GPUDays* on CNN Design's ImageNet

Architecture	Search method	Err ImageNet	#Params	*GPUDays*
AmoebaNet-C	EA	7.60	6.4 M	3.150
AmoebaNet-A	EA	9.00	4.9 M	3.150
NSGA-2	EA	8.05	4.1 M	27
NSGA-3	EA	6.9	5.0 M	27
BLOP-CNN	EA	6.66	4.7 M	29
Bi-CNN-D-C	EA	7.17	2.1 M	31

medium-quality architectures are visited and even designated as best found architectures at that stage so far. After that the evolutionary process was able to focus the population on the promising regions of the search space, where most architectures have similar respectful classification performance values. This focus makes the algorithm sampling well-performing architectures that are similar in terms of classification accuracy, which explains the alleviation of the maximization slope. This phenomenon could be observed also in many other applications of genetic algorithms [49, 60].

The evolution trajectory analysis provides good interection between NB, NNB, NF and Err at the upper level, with the goal of developing effective CNN architectures with an optimum block and nodes per block size.

4.4.3 Comparisons to pruning methods

The most representative previous studies compressed the already-existing CNN manual architecture. There is no existing work that compresses an architecture that is generated automatically. For this reason, Bi-CNN-D-C is the first work to compress an evolving CNN architecture automatically. Table 10 summarizes the CNN manual architectures that were used to compare the proposed approach. In fact, the results using the CIFAR-10 and CIFAR-100 data sets are presented in Tables 6 and 7, respectively. For each DCNN architecture, the best test error and number of FLOPs are shown. Based on Table 6, the average test error of the DeepPruningES algorithm is between 7.43 and 8.91%, where for the VGG16 and VGG19, they obtain similar results of 8.21% with a 32.01% and 32.56% diminution in the FLOPs number, and for ResNet56, ResNet110, DenseNet50 and DenseNet100, they obtain values lying between 7.43% and 8.91% with a 16.72% and 32.56% reduction in the FLOPs number. In Table 6, the average test error of the Channel-Evo algorithm is between 5.85 and 7.91%, with the VGG16 and VGG19 achieving similar results of 7.26% with a respective 52% and 53.05% reduction in the number of FLOPs, and the ResNet56, ResNet110, DenseNet5 and DenseNet100 values lying between [5.85 and 7.91%] and a [16.02% and 17.35%] reduction in the number of FLOPs. Always on Table 6, the average test error of the auto-balanced filter is between 8.27 and 9.32%, whereas the VGG16, VGG19, ResNet56, ResNet110, DenseNet50 and DenseNet110 obtain values between 8.27% and 9.32% and a [36.5% and 57%] reduction in FLOPs. The average test error of Classical-Pruning is laying between [6.46 and 9.09%]. In fact, the proposed Bi-CNN-D-C algorithm provide 1.98 $\times$ 107 and 2.21 $\times$ 10^7 of #Flops, with 32.5%, 28.9% pruned percentages. Bi-CNN-D-C is capable of reducing FLOPs while maintaining acceptable test errors.

4.4.4 Further analysis and discussion

The main reason that could explain the outperformance of Bi-CNN-D-C over considered peer works corresponds to the principal motivation of this work. From the start of this paper, we have mentioned that the main shortcoming of existing pruning methods including evolutionary ones consists in the fact that these methods take as input an existing (already designed) architecture, and then, the algorithm searches for the best possible pruning decision. This drastically limits the performance of such kind of methods. From a metamorphic vision, we could say that such an algorithm remains paralyzed in a single point of the

Table 6 Compression results obtained on CIFAR10

Method	Architecture	Pruned (%)	Test error (%)	#FLOPs
DeepPruningES	VGG16	32.01	8.21	2.15×10^8
	VGG19	32.56	8.21	2.7×10^8
	ResNet56	21.31	8.11	1.01×10^8
	ResNet110	16.72	7.43	2.14×10^8
	DenseNet50	19.16	8.91	0.779×10^8
	DenseNet100	19.33	8.34	2.49×10^8
Channel-Evo	VGG16	52	7.26	2.74×10^8
	VGG19	53.05	7.26	3.13×10^8
	ResNet56	20	5.85	1.62×10^9
	Resnet110	21.01	7.72	1.02×10^9
	DenseNet50	16.02	7.91	0.86×10^8
	DenseNet100	17.35	7.88	2.97×10^8
Classical-Pruning	VGG16	34.2	6.75	3.82×10^8
	VGG19	34.9	7.05	3.45×10^8
	ResNet56	27.6	9.09	9.10×10^8
	ResNet110	38.6	6.46	1.20×10^8

Table 7 Compressed results obtained on CIFAR100

Method	Architecture	Pruned (%)	Test error (%)	#FLOPs
DeepPruningES	VGG16	19.93	32.94	2.531×10^8
	VGG19	19.09	33.11	3.246×10^8
	ResNet56	16.19	42.19	1.072×10^8
	ResNet110	17.73	50.97	2.117×10^8
	DenseNet50	19.05	41.59	7.569×10^7
	DenseNet100	19.05	35.344	2.469×10^8
Channel-Evo	VGG16	37	27.99	3.82×10^8
	VGG19	36.1	29.88	3.76×10^9
	ResNet56	16	25.90	1.74×10^9
	ResNet110	16.78	29.2	2.81×10^9
	DenseNet50	20	39.4	2.01×10^8
	DenseNet100	21	32.2	2.77×10^8

architecture search space, which is not the case for our algorithm.

By allowing the joint design and pruning of architectures, our algorithm is able not only to move from an architecture to another but also to (near) optimally prune each generated architecture. In this way, our algorithm is not still paralyzed in a single point of the architecture space (i.e., it has the freedom search space sampling). Moreover, the collaborative interaction of the two optimization levels (upper and lower) ensures the narrowing of the search process toward high-performing architectures with minimum number filters (and also minimum NB and NNB). To the best of the authors' knowledge, Bi-CNN-D-C is the first EA capable of compression automatically designing CNN architectures and providing bi-level interaction between convolutional layers and their hyperparameters.

4.5 Case study on COVID-19 diagnosis

4.5.1 Benchmarks

Chest X-ray and CT are two of the most commonly available radiological tests for the diagnosis of several lung diseases. Chest X-ray and CT are two of the most commonly available radiological tests for the diagnosis of several lung diseases. In our study, we acquired chest X-rays belonging to 50 COVID-19 patients from [61] by Dr. Joseph Cohen. In these data, a number of individuals having intense respiratory distress sickness, serious respiratory problem, pneumonia, COVID-19 have chest X-ray along with computed tomography images in this archive. We also choose normal chest X-ray photographs from the Kaggle library that have been labeled as chest X-ray images (pneumonia) https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia. This study uses a chest X-ray images database divided into 2 separated groups: COVID-19 patient images and normal patient images. We scaled all images within the data set into 224 by 224 pixels. Then, the data set is randomly separated into 2 distinct data sets where 80% is considered for training and 20% for testing. Figures 8 and 9 show uninfected and infected people's chest X-ray pictures, respectively.

4.5.2 Existing works

Many computational intelligence strategies for COVID-19 detection using computed tomography (CT) and X-ray images have been proposed recently [62–64]. Fei et al. [65] have proposed VB-Net, an interactive approach for CT images that is an extended version of V-Net. It is divided into two sections. The first is a contractual approach that uses downsampling and convolution to obtain the image's general features. The second is a broad approach that incorporates fine-grained image features through upsampling and convolution processes. The work's main distinctive feature is the addition of a human expert into the loop in order to lead the segmentation of the infection procedure. Prabira et al. [66] extracted a deep features set using nine pre-trained CNN models, which were then given to the support vector machine (SVM) classifier. In the comparison studies, manually constructed DCNNs architectures were used. Based on X-ray images, the suggested method was found to have superior detection accuracy. Xu et al. [67] developed an approach called in-depth screening strategy to identify pneumonia due to COVID-19 among both, viral Influenza-A healthy and cased of pneumonia. A 3D deep learning algorithm was used to segment candidate infection locations first. Individual images categorized as pneumonia due to Influenza-A viral, COVID-19, without being linked to infection categories, in addition to confidence ratings, through a classification model based on location attention. Eventually, the noisy or Bayesian function was used to calculate the kind of infection and the total confidence score. Shuai et al. [68] presented an Inception model based on transfer learning It is possible to divide the latter into two halves. To transform the input of the image and convert it into vectors of one-dimensional feature for the classification challenge, the first half used a pre-trained starter network, whereas a fully connected network is used by the second half used.

4.5.3 Experiment

The resulting test error findings on X-ray image an CT are summarized in Tables 8 and 9, respectively, using the identical computer configuration and implementation environment as the previous section's experiments (fourth). We notice that Bi-CNN-D-C outperforms the peer approaches in terms of results. In reality, it is worth mentioning that the suggested algorithm's primary purpose is to reduce architecture complexity. Manual architecture, from the standpoint of optimization, is a wide search space that must be carefully sampled in order to build an effective and efficient design. Bi-CNN-D-C might be justified in the same manner that images of both, CIFAR-100 and CIFAR-10 were. We recall the aim of our approach which is about focusing on the CNN design and compression. Bi-CNN-D-C shows its ability to provide end users successful designs capable to detect COVID-19 from X-ray and CT images, while taking into consideration the hierarchical structure of the CNN architecture design challenge. This case study, we believe, will encourage researchers and practitioners to use the scalable computational approach for X-ray image analysis utilizing DNN in the future.

5 Conclusions and future work

Deep neural networks have demonstrated outstanding performance in a wide range of machine learning tasks, including classification and clustering [69, 70], for real-life applications of soft computing techniques in different fields

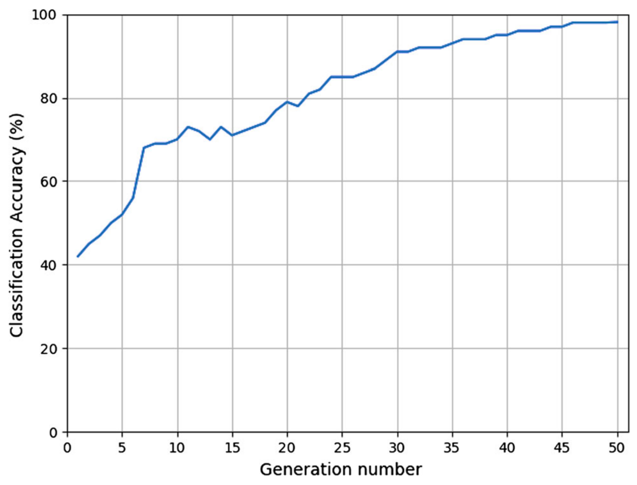


Fig. 7 Acc evolution of the best found NB and NNB values on CIFAR-10

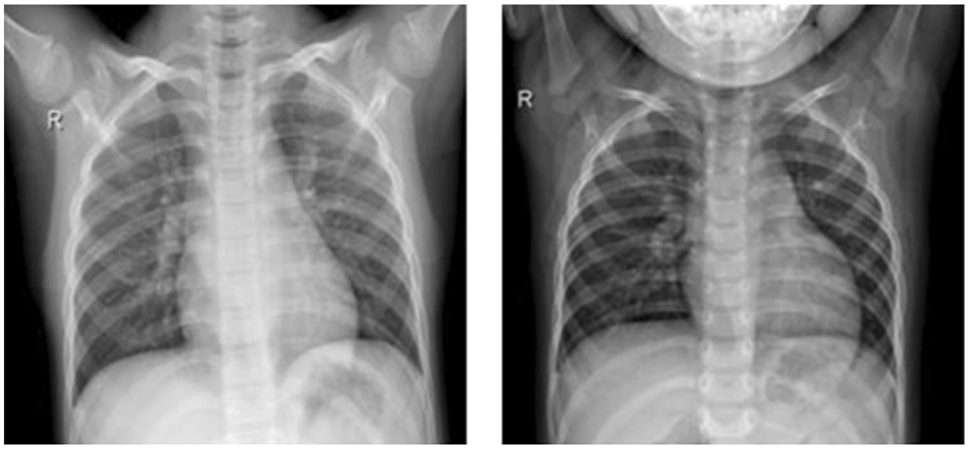


Fig. 8 Chest X-ray images appertaining to regular patients [1]

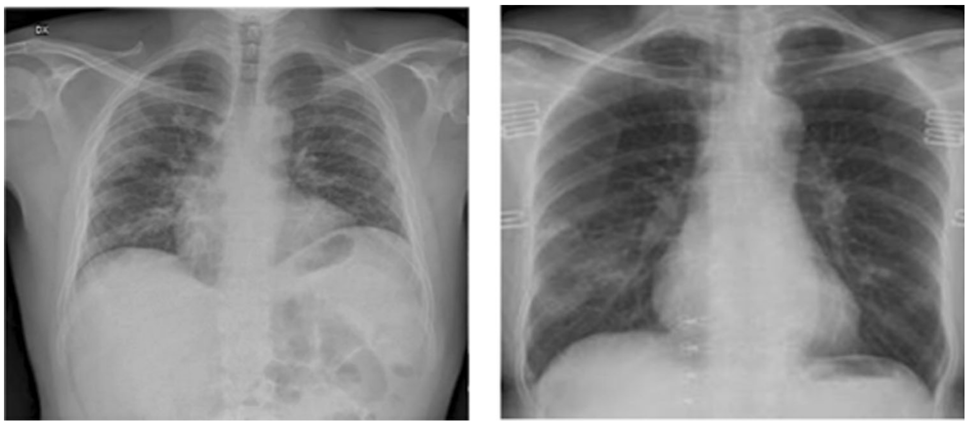


Fig. 9 Chest X-ray images appertaining to patients having COVID-19 [1]

[2, 71–73]. In fact, Designing an architecture for the Deep CNN is an extremely interesting, challenging and timely subject. Recently, several works have concentrated on developing novel methods for reducing the computational complexity of CNNs using EAs. Indeed, recent research [14] suggests that such metaheuristic algorithms could be used to solve the CNN architecture optimization problem in the field of network compression and acceleration. Nevertheless, none of the previous works took the bi-level nature of neural architecture design and compression into account. Following the CNN architecture design, there are an infinite number of alternative architectures with various network topologies for any set of blocks of various sizes. Following the CNN architecture compression, the search space expands exponentially in size as the number of layers increases, we must reduce network complexity and eliminate redundancy in order to achieve a small compressed model. Based on this observation, we suggest a bi-level model of the CNN architecture design and compression problem in this study, where the upper level seeks to minimize network complexity primarily in terms of the NB and NNB, and maximizes classification *Acc* with regard to

Table 8 Obtained *Acc* values on CT images

Detection method	Test Acc (%)	#Params	GPUDays
Transfer learning-based inception	87.0	138 M	4.125
Location attention-based ResNet	86.7	7.6 M	11
Genetic-CNN	96.31	9.0 M	12
NSGA-4	96.98	7.2 M	9
BLOP-CNN	97.12	5.0 M	7
Bi-CNN-D-C	96.81	2.1 M	2.5

Table 9 Obtained *Acc* values on chest X-ray images

Detection method	Test Acc (%)	#Params	GPUDays
ResNet101	89.26	11.7 M	14
Inceptionv3	91.08	24 M	12
VGG16	92.76	7.0 M	9
VGG19	92.91	7.0 M	9
XceptionNet	93.92	24 M	11
AlexNet	93.32	5.5 M	7
DenseNet201	93.88	6.2 M	13
Genetic-CNN	96.11	9.2 M	12
NSGA-4	96.98	7.5 M	9
BLOP-CNN	96.42	5.2 M	7
Bi-CNN-D-C	96.81	2.2 M	2.5

block network topologies, and the lower level minimizes the complexity of the network by minimizing the number of filter. CEMBA [17] is then used to solve the bi-level model. According to the results analysis, the suggested approach BI-CNN-D-C demonstrated its effectiveness and superior performance on the CIFAR-10 and -100 benchmark data sets when compared to several representative architectures as well as some additional ones generated by recent prominent from design and compression CNN architectures. Finally, we'd like to draw attention to some interesting perspectives. The first is directly related to our work and entails developing an interaction model that enables users to interact with Bi-CNN-D-C via the evolution process by examining some generated architectures, mining their common patterns and then making recommendations in the form of soft and/or hard constraints to generate CNN architectures that satisfy the expert's preferences and knowledge. The second objective is to extend our work to the field of real-time federated learning, in which multiple local devices collaborate to train a shared global model, while training data remains on edge devices. This enables the resolution of critical issues such as data privacy, data security, data access rights and access to heterogeneous data, all of which are critical in a wide variety of domains, including defense, telecommunications, the Internet of Things and pharmaceutics. The third one is directly related to future work. In fact, pure filter pruning can lead to the removal of important channels and could cause a considerable loss of interesting channels. Our future work will solve this issue by preserving important channels even if their corresponding filters contain a very small number of channels. Finally, it would be interesting to extend our approach to the case of dynamic environment (incremental learning) [74] and thus dynamic evolutionary algorithms could represent an interesting alternative to deal with such a challenge.

Appendix A Manually designed CNN architectures used for compression

The most evocative examples of manual architecture are VGG16, VGG19, ResNet50, DenseNet50 and ResNet110. The details of the CNN architectures used to compare the proposed approach are summarized in Table 10.

Appendix B Bi-level optimization's main definitions

In the academic and real-world optimization problems, the majority uses a single level of optimization. Multiples problems, however, are designed as two levels, referred to as BLOP [75]. We uncover a problem of optimization nested within external optimization constraints in such

Table 10 CNN architectures overview for comparing the suggested approach

Architecture	Description	# layers	Test error CIFAR10	Test error CIFAR100 (%)	%FLOPs
VGG16	Built in 2014, which has performed well in complex image classification data sets.	16	6.06%	32.33	3.15×10^8
VGG19		19	6.18%	32.57	4.01×10^8
DenseNet50	Proposed architectures provided by DenseNet authors that classifies the data set of CIFAR10.	50	6.92	41.25	0.93×10^8
DenseNet100		100	5.66%	33.06	3.05×10^8
ResNet56	Are the first ones proposed by He et al (2016) to be use with the data set of CIFAR 10.	50	6.90%	41.86	1.23×10^8
ResNet110		110	6.2%	50.47	2.57×10^8

scenarios. The upper-level problem, often known as the leader problem, is considered as an external task of optimization. The nested internal optimization work called also a follower problem or a lower level, and the two-level problem is referred to as a leader–follower problem or a Stackelberg game [76]. The follower problem looks like a constraint at the upper level; therefore, uniquely the best solution with regard to the follower optimization problem could be considered as a leader candidate.

Definition: Assuming $\Re^n \times \Re^n \rightarrow \Re$ to be the leader problem and $f : \Re^n \times \Re^n \rightarrow \Re$ to be the follower one, analytically, a BLOP could be stated as follows:

$$\mathrm{Min}_{x_u \in X_U, x_l \in X_L} L(x_u, x_l)$$
$$\text{subject to} \begin{cases} x_l \in \mathrm{ArgMin}\{f(x_u, x_l), g_j(x_u, x_l)) \leq 0, j = 1, \ldots, J\} \\ G_k(x_u, x_l) \leq 0, k = 1, \ldots, K. \end{cases} \tag{B1}$$

There are two types of variables in a BLOP: variables of the upper-level variables and variables of the lower level. The optimization work for the follower problem is done against the x_l variables, with the x_u variables acting as fixed parameters. As a result, each x_u represents a new follower issue, the optimal solution of which is a function of x_u and must be found. In the leader problem, all variables (x_u, x_l) are examined, with the exception of x_l. The formal definition of BLOP is provided below:

Single-level problems are intrinsically more complex to solve than BLOPs. It is not unexpected that the majority of previous research has focused on the more straightforward situations of BLOP, such as problems with good features like linear objectives, constraint functions, convexity or quadraticity [77]. Despite the fact that it was the first study on bi-level optimization originates from the 1970s, the utility of these mathematical programs in representing hierarchical decision-making engineering and processes challenges were not realized until the early 1980s. BLOPs have encouraged researchers to devote special attention to them. Kolstad [75] compiled the first bibliographic survey on the subject (1985) in the mid-1980s.

Existing BLOP-solving methods can be divided into two groups: (1) classical methods while (2) evolutionary methods. Number one family preserves extreme point-based approaches such as [78], branch-and-bound [79], penalty function methods [80], complementary pivoting [81] and methods of trust region [82]. These strategies have a disadvantage which is that they are significantly reliant to the mathematical properties of the BLOP in question. Metaheuristic algorithms, which are mostly evolutionary algorithms, belong to the second family (EA). Several EAs have recently proved their efficacy in addressing such problems due to their insensitivity to the mathematical properties of the problem, as well as their capacity to handle enormous problem instances by offering satisfactory answers in a fair amount of time. Here are a few examples of notable works [80, 81].

Appendix C Main principle of CEMBA

As for each upper-level architecture there exists a whole search space of possible filter pruning decisions, the joint neural architecture search and (filter) compression are framed as a bi-level optimization problem. Motivated by the recent emergence of the field of EBO (evolutionary bi-level optimization) and the related achieved interesting results in many application fields, we decided to use the evolutionary algorithm as a search engine to solve our bi-level optimization problem. The main difficulty faced in EBO is the important computational cost it needs. This is because each the fitness evaluation of each upper-level solution requires running a whole lower-level evolutionary process to approximate the optimal corresponding lower-level solution. Through the literature there exist many EBO algorithms [83], but most of them focus on problems with continuous variables (using approximation techniques and gradient information). The number of algorithms that were designed for the discrete case is much reduced with regard to the continuous case. Examples of discrete EBO algorithms are NBEA, BLMA, CODBA, CODCRO and CEMBA [84]. As the majority of algorithms deal with the continuous situation, CEMBA among the most effective bi-level EAs for dealing with discrete scenarios that has been demonstrated [17]. Every upper-level solution is evaluated using 2 stages: first, the variables of the upper-level are directed to the lower level; and second, the indicator-based multi-objective local search gets closer to the solution with the maximum marginal contribution in terms of multi-objective quality indicator and sends it to the next level to complete the evaluation of the quality of the consideration. To summarize how it works, we will go over the research processes of its upper and lower levels:

- **Upper and lower population initialization** Create the upper-level population and the lower-level population from scratch. Two starting populations were obtained by using the Discrete Space Decomposition Method twice. The goal of utilizing a decomposition approach is to produce uniform coverage of the decision space and, to the extent possible, a collection of solutions that are evenly dispersed over every level decision space.
- **Lower-level fitness assignment** To assess an upper-level solution in BLOP, a bi-level optimization problem necessitates to execute of an entire lower-level method, which is the BLOP's fundamental challenge. As a result, we dissect each problem level using 2

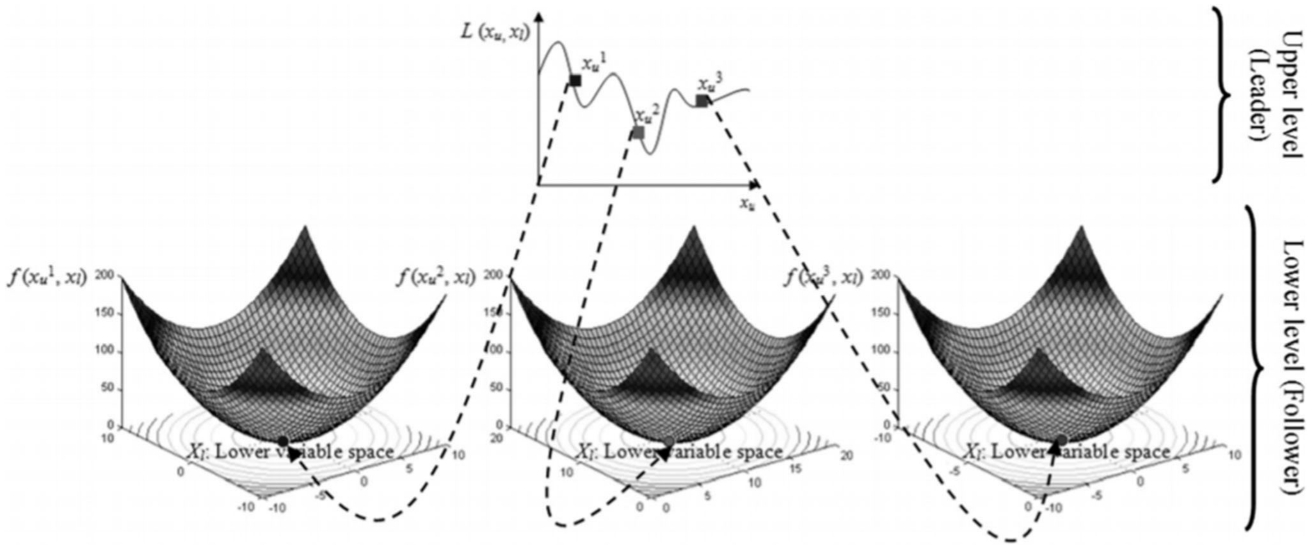


Fig. 10 An example of a single-objective BLOP with two levels is illustrated [1]

populations for solving it. The lower-level algorithm of each lower population employs the higher solutions of the matching upper population to evaluate the lower-level solutions.

- **Local search procedure** The local search is applied for each lower-level population using the IBMOLS principle for the lower-level method. In fact, we begin by calculating the objective function's normalized values. As a result, for each lower solution, we create a neighborhood and then calculate its fitness value using an indicator I and the objective function's normalized values. An update of the fitness values is then performed, the worst-case solution is removed, and the fitness values are updated again. It is worth noting that neighborhood generation comes to a halt in one of two situations: (1) whenever entire solution neighborhood has been examined, or (2) in case an adjacent solution that improves (with regard to I) is discovered. In case that all lower-level members have been visited, the entire local search procedure comes to an end.
- **Best indicator contribution lower-level solution determination** Because evaluating the upper-level population's leader solutions necessitates approximating each matching lower-level population's follower solution, the lower-level solutions are compared to the members of the upper-level.
- **Upper-level indicator-based procedure** Each higher-level population performs its algorithm based on the IBEA after obtaining the lower-level solutions with the best indicator contribution of the follower problem. This is because we find the person with the lowest fitness value, delete him or her and then update the fitness values of the remaining people until we reach the stop criterion. After that, mating selection and variation are used. We should remark that using IBEA aids the algorithm in approximating the best upper front.
- **Migration strategy (each α generations)** After a certain number of generations, use a migration strategy. As a result, we use the parameter β to select a set of solutions that includes β objective follower space

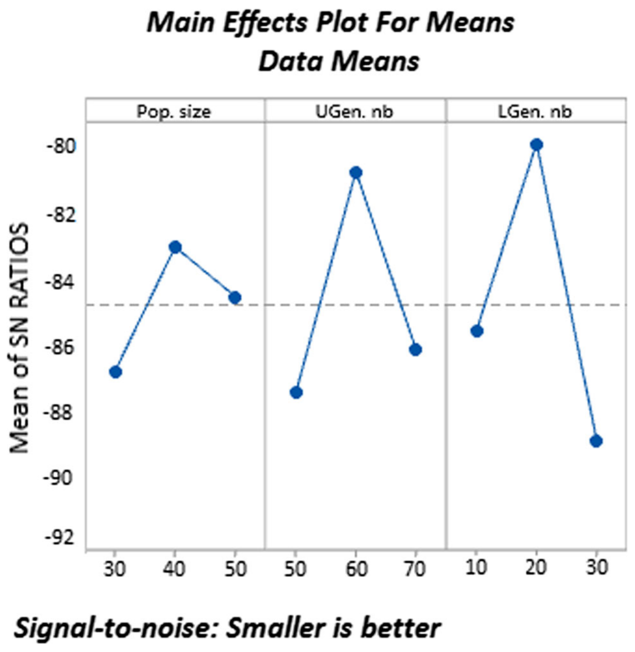


Fig. 11 Signal to noise

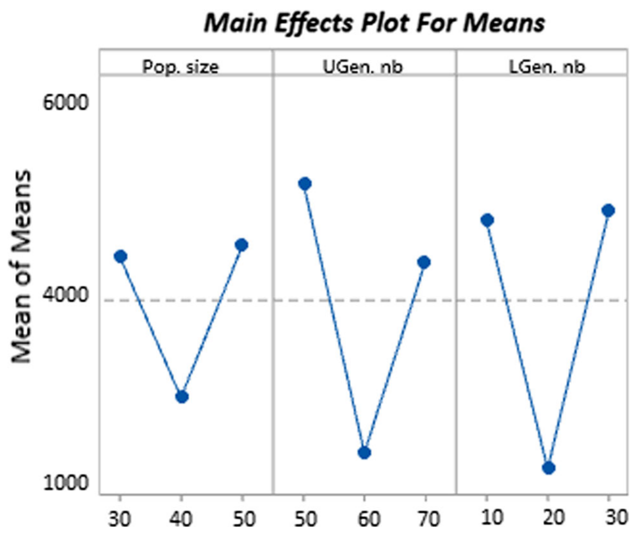


Fig. 12 Mean of means

solutions. The migration step employs this pre-selected collection of solutions.

Appendix D Tuning of parameters

The Taguchi method [85] is a sophisticated case of the trial-and-error one [86]. In order to clarify more and verify the proposed parameter tuning values, we have applied the Taguchi method in which the signal-to-noise ratio (SNR) parameter is calculated as follows:

$$\mathrm{SNR} = -log_{10}\left(\frac{1}{N}\sum_{i=1}^{N}(\text{objective function})_i^2\right) \tag{D2}$$

where N represents the number of performed runs. The SNR parameter reflects the variability and the mean of the experimental data. The used parameters for tuning are the following: (1) population size (Pop. size), (2) upper generation number (UGen. nb) and (3) lower generation number (LGen. nb). The considered levels for each parameter, while the corresponding orthogonal array (L27(33) where we have 27 experiments, 3 variables and 2 levels. Figure 11 displays the obtained SNR results for IB-CEMBA. Moreover, Fig. 12 displays computed results for Bi-CNN-D-C in terms of mean fitness values of the upper-level in Taguchi experimental analysis, which confirmed the achieved optimal levels using SNR parameter. In fact, the computed mean upper-level fitness values confirmed the achieved optimal level using SNR parameter.

Acknowledgements The authors thank the Deanship of Scientific Research at Prince Sattam bin Abdulaziz University for supporting this work.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Louati H, Bechikh S, Louati A, Hung C-C, Ben Said L (2021) Deep convolutional neural network architecture design as a bi-level optimization problem. Neurocomputing 439:44–62
2. Louati A (2020) A hybridization of deep learning techniques to predict and control traffic disturbances. Artif Intell Rev 53(8):5675–5704
3. Louati A, Louati H, Li Z (2021) Deep learning and case-based reasoning for predictive and adaptive traffic emergency management. J Supercomput 77(5):4389–4418
4. Bengio Y, Lamblin P, Popovici D, Larochelle H (2006) Greedy layerwise training of deep networks. In: Scholkopf B, Platt JC, Hofmann T (eds) Advances in neural information processing systems 19, Proceedings of the twentieth annual conference on neural information processing systems, pp 153–160
5. LeCun YY, Bengio H (2015) Deep learning. Neurocomputing 521:7553–436444
6. Zhen X, Chakraborty R, Singh V(2021) Simpler certified radius maximization by propagating covariances. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition, pp 770–778
7. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. CoRR, arXiv: abs/1409.1556
8. Lopez-Rincon A, Tonda A, Elati M, Schwander O, Piwowarski B, Gallinari P (2018) Evolutionary optimization of convolutional neural networks for cancer mirna biomarkers classification. Appl Soft Comput 65:91–100
9. Darwish A, Hassanien AE, Das S (2020) A survey of swarm and evolutionary computing approaches for deep learning. Artif Intell Rev 53(3):1767–1812
10. Chauhan J, Rajasegaran J, Seneviratne S, Misra A, Seneviratne A (2018) Performance characterization of deep learning models for breathing based authentication on resource-constrained devices. In: IMWUT, pp 1–24
11. Perenda E, Rajendran S, Bovet G, Pollin S, Zheleva M (2021) Evolutionary optimization of residual neural network architectures for modulation classification. IEEE Trans Cogn Commun Netw. https://doi.org/10.1109/TCCN.2021.3137519
12. Chollet F (2017) Xception: deep learning with depthwise separable convolutions. In: IEEE conference on computer vision and pattern recognition CVPR, pp 1251–1258
13. Hu H, Peng R, Tai Y-W, Tang C-K (2016) Network trimming: a datadriven neuron pruning approach towards efficient deep architectures. arXiv: 1607.03250 13(3):1–18
14. Mishra R, Gupta HP, Dutta T (2020) A survey on deep neural network compression: Challenges, overview, and solutions. arXiv:2010.03954
15. Abd Elaziz M, Dahou A, Abualigah L, Yu L, Alshinwan M, Khasawneh AM, Lu S (2021) Advanced metaheuristic optimization techniques in applications of deep neural networks: a review. Neural Comput Appl 33(21):14079–14099
16. Ünal HT, Başçiftçi F (2022) Evolutionary design of neural network architectures: a review of three decades of research. Artif Intell Rev 55:1723–1802
17. Said R, Bechikh S, Louati A, Aldaej A, Said LB (2020) Solving combinatorial multi-objective bi-level optimization problems using multiple populations and migration schemes. IEEE Access 8:141674–141695

18. Cheung B, Sable C (2011) Hybrid evolution of convolutional networks. In: In 2011 10th international conference on machine learning and applications and workshops, pp 293–297
19. Deng L (2012) The mnist database of handwritten digit images for machine learning research. IEEE Signal Process Mag 29:141–142
20. Fujino S, Mori N, Matsumoto K (2012) The mnist database of handwritten digit images for machine learning research. IEEE Signal Process Mag 29:141–142
21. Real E, Moore S, Selle A, Saxena S, Suematsu Y.L, Tan J, Le Q, Kurakin A (2017) Large-scale evolution of image classifiers. In: In 34th international conference on machine learning, pp 2902–2911
22. Xie S, Girshick R, Dollar P, Tu Z, He K (2017) Aggregated resid-'ual transformations for deep neural networks. In: In 34th international conference on machine learning, pp 1492–1500
23. Mirjalili S (2019) Evolutionary algorithms and neural networks. In: Studies in computational intelligence, ISBN:978-3-319-93025-1
24. Martín A, Lara-Cabrera R, Fuentes-Hurtado F, Naranjo V, Camacho D (2012) Evodeep: a new evolutionary approach for automatic deep neural networks parametrisation. J Parallel Distrib Comput 117:180–191
25. Real E, Aggarwal A, Huang Y, Le QV (2019) Regularized evolution for image classifier architecture search. In: In Aaai conference on artificial intelligence, pp 4780–4789
26. Sun Y, Xue B, Zhang M, Yen GG (2020) Completely automated CNN architecture design based on blocks. IEEE Trans Neural Netw Learn Syst 31(4):1242–1254
27. Liang J, Guo Q, Yue C, Qu B, Yu K (2018) A self-organizing multiobjective particle swarm optimization algorithm for multimodal multi-objective. In: In international conference on swarm intelligence, pp 550–560
28. Rahul M, Gupta HP, Dutta T (2020) A survey on deep neural network compression: challenges, overview, and solutions, arXiv: 2010.03954
29. Francisco E, Fernandes J, Yen GG (2021) Pruning deep convolutional neural networks architectures with evolution strategy. Inf Sci 552(4):29–47
30. Hao L, Kadav A, Durdanovic I, Samet H, Graf HP (2016)Pruning deep convolutional neural networks architectures with evolution strategy. Inform Sci, arXiv:1608.08710
31. Luo J, Wu J, Lin W (2017) Thinet: a filter level pruning method for deep neural network compression. In: ICCV, pp 5058–5066
32. Denton EL, Zaremba W, Bruna J, LeCun Y, Fergus R (2014) Exploiting linear structure within convolutional networks for efficient evaluation. In: NIPS, pp 1269–1277
33. Hu H, Peng R, Tai YW, Tang CK (2016) Network trimming: a datadriven neuron pruning approach towards efficient deep architectures. arXiv: 1607.03250
34. Han S, Mao H, Dally WJ (2015) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv:1510.00149
35. Qin Q, Ren J, Yu J, Wang H, Gao L, Zheng J, Feng Y, Fang J, Wang Z (2018) To compress, or not to compress: characterizing deep learning model compression for embedded inference. In: 2018 IEEE international conference on parallel, pp 729–736
36. Chauhan J, Rajasegaran J, Seneviratne S, Misra A, Seneviratne A, Lee Y (2018) Performance characterization of deep learning models for breathing-based authentication on resource-constrained devices. Proc ACM Interact Mob Wearable Ubiquitous Technol 2(4):1–24
37. Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, Adam H, Kalenichenko D (2018) Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: CVPR, pp 2704–2713
38. Han S, Mao H, Dally, WJ (2016) Deep compression: Compressing deep neural networks with pruning, trained quantization & huffman coding. In: ICLR
39. Schmidhuber J, Heil, S (1995) Predictive coding with neural nets: application to text compression. In: NeurIPS, pp 1047–1054
40. Han S, Mao H, Dally WJ (2015) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv:1510.00149
41. Louati A, Louati H, Nusir M, Hardjono B (2020) Multi-agent deep neural networks coupled with LQF-MWM algorithm for traffic control and emergency vehicles guidance. J Ambi Intell Humanized Comput 11(11):5611–5627
42. Liang F, Tian Z, Dong M, Cheng S, Sun L, Li H, Chen Y, Zhang G (2021) Efficient neural network using pointwise convolution kernels with linear phase constraint. Neurocomputing 423:572–579
43. Bhattacharya S, Lane ND (2016) Sparsification and separation of deep learning layers for constrained resource inference on wearables. In: SenSys, pp 176–189
44. Zhou Y, Yen GG, Yi Z (2021) A knee-guided evolutionary algorithm for compressing deep neural networks. IEEE Trans Cybern 51(3):1626–1638
45. Huynh LN, Lee Y, Balan RK (2017) Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In: SenSys, pp 82–95
46. Song Han HM, Dally, WJ (2016) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. In: ICLR
47. Elias P (1975) Universal codeword sets and representations of the integers. IEEE Trans Cybern 21(2):194–203
48. Gallager R, van Voorhis D (1975) Optimal source codes for geometrically distributed integer alphabets (corresp.). IEEE Trans Inf Theory 21(2):228–230
49. Xie L, Yuille A (2017) Genetic cnn. In: Proceedings of the IEEE international conference on computer vision, pp 1379–1388
50. Spears VM, Jong KAD (1991) On the virtues of parameterized uniform crossover. In: In fourth international conference on genetic algorithms, pp 230–236
51. Settle TF, Krauss TP, Ramaswamy K (2006) U.S. Patent No. 7,079,585. Washington, DC: U.S. Patent and Trademark Office
52. Chakraborty UK, Janikow CZ (2003) An analysis of gray versus binary encoding in genetic search. US Patent 156:253–269
53. Chakraborty UK, Janikow CZ (2003) An analysis of gray versus binary encoding in genetic search. Inf Sci 156(3–4):253–269
54. Lu Z, Whalen I, Dhebar Y, Deb K, Goodman E, Banzhaf W, Boddeti VN (2019) Multi-criterion evolutionary design of deep convolutional neural networks, arXiv:abs/1912.01369
55. Dwork C, Feldman V, Hardt M, Pitassi T, Reingold O, Roth A (2015) STATISTICS the reusable holdout: preserving validity in adaptive data analysis. Science 349(6248):636–638
56. Kohavi R, John GH (1997) Wrappers for feature subset selection. Artif Intell 97(1–2):273–324
57. Shinozaki T, Watanabe S (2015) Structure discovery of deep neural network based on evolutionary algorithms. In: 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp 4979–4983
58. Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2016) Pruning filters for efficient convnets. arXiv:1608.08710
59. Eiben AE, Smit SK (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol Comput 1(1):19–31
60. Lu Z, Whalen I, Dhebar Y, Deb K, Goodman E, Banzhaf W, Boddeti VN (2019) Nsga-net: neural architecture search using multi-objective genetic algorithm. In: In Genetic and evolutionary computation conference, pp 419–427

61. Cohen JP, Morrison P, Dao L, Roth K, Duong TQ, Ghassemi M (2020) Covid-19 image data collection: Prospective predictions are the future. journal of machine learning for biomedical imaging (melba). https://github.com/ieee8023/covid-chestxray-dataset
62. Canayaz M, Şehribanoğlu S, Özdağ R, Demir M (2022) COVID-19 diagnosis on CT images with Bayes optimization-based deep neural networks and machine learning algorithms. Neural Comput Appl 34(7):5349–5365
63. Louati H, Bechikh S, Louati A, Aldaej A, Said LB (2021) Evolutionary optimization of convolutional neural network architecture design for thoracic x-ray image classification. In: Advances and trends in artificial intelligence. Artificial Intelligence Practices, pp 121–132
64. Louati H, Bechikh S, Louati A, Aldaej A, Said LB (2022) Evolutionary optimization for cnn compression using thoracic x-ray image classification. In: the 34th international conference on industrial, engineering & other applications of applied intelligent systems
65. Shan F, Gao Y, Wang J, Shi W, Shi N, Han M, Xue Z, Shen D, Shi Y (2020) Lung infection quantification of COVID-19 in CT images with deep learning. arXiv:2003.04655
66. Sethy PK, Behera SK (2020) Detection of coronavirus disease (COVID-19) based on deep features. Int J Math Eng Manag Sci 5(4):643–651
67. Butt C, Gill J, Chun D, Babu BA (2020) Deep learning system to screen coronavirus disease 2019 pneumonia. Appl Intell. https://doi.org/10.1007/s10489-020-01714-3
68. Wang S, Kang B, Ma J, Zeng X, Xiao M, Guo J, Cai M, Yang J, Li Y, Meng X, Xu B (2021) A deep learning algorithm using CT images to screen for corona virus disease (COVID-19). Eur Radiol 31(8):6096–6104
69. Louati A, Lahyani R, Aldaej A, Aldumaykhi A, Otai S (2022) Price forecasting for real estate using machine learning: A case study on riyadh city. Concurr Comput Practice Exp 34(6):6748
70. Louati A, Masmoudi F, Lahyani R (2022) Traffic disturbance mining and feedforward neural network to enhance the immune network control performance. In: Proceedings of seventh international congress on information and communication technology
71. Banan A, Nasiri A, Taheri-Garavand A (2020) Deep learning-based appearance features extraction for automated carp species identification. Aquacult Eng 89:102053
72. Shamshirband S, Rabczuk T, Chau K-W (2019) A survey of deep learning techniques: application in wind and solar energy resources. IEEE Access 7:164650–164666
73. Fan Y, Xu K, Wu H, Zheng Y, Tao B (2020) Spatiotemporal modeling for nonlinear distributed thermal processes based on kl decomposition, mlp and lstm network. IEEE Access 8:25111–25121
74. Azzouz R, Bechikh S, Ben Said L (2014) A multiple reference point-based evolutionary algorithm for dynamic multi-objective optimization with undetectable changes. In: 2014 IEEE congress on evolutionary computation (CEC)
75. Kolstad CD (1985) A review of the literature on bi-level mathematical programming. Report Number: LA-10284-MS
76. Candler WV, Townsley R (1962) A study of the demand for butter in the united kingdom. Australian J Agricult Econom 6:36–48
77. Louati A, Lahyani R, Aldaej A, Mellouli R, Nusir M (2021) Mixed integer linear programming models to solve a real-life vehicle routing problem with pickup and delivery. Appl Sci 11(20):9551
78. Bard JF, Falk JE (1982) An explicit solution to the multi-level programming problem. Comput Oper Res 9(1):77–100
79. Shimizu K, Kobayashi Y, Muraoka K (1981) Midperipheral fundus involvement in diabetic retinopathy. Ophthalmology 88(7):601–612
80. Białas S, Garloff J (1985) Convex combinations of stable polynomials. J Franklin Inst 319(3):373–377
81. Sinha A, Malo P, Frantsev A, Deb K (2013) Multi-objective stackelberg game between a regulating authority and a mining company: a case study in environmental economics. In: 2013 IEEE congress on evolutionary computation, pp 478–485
82. Sinha A, Bedi S, Deb K (2018) Bilevel optimization based on kriging approximations of lower level optimal value function. In: 2018 IEEE congress on evolutionary computation (CEC), pp 1–8
83. Sinha A, Malo P, Deb K (2017) A review on bilevel optimization: from classical to evolutionary approaches and applications. IEEE Trans Evol Comput 22(2):276–295
84. Said R, Elarbi M, Bechikh S, Ben Said L (2021) Solving combinatorial bi-level optimization problems using multiple populations and migration schemes. Oper Res 1–39
85. Ross PJ (1996) Taguchi Techniques for Quality Engineering: Loss Function. Orthogonal Experiments, Parameter and Tolerance Design
86. Eiben AE, Smit SK (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol Comput 1(1):19–31