Artificial Intelligence In Medicine 142 (2023) 102571

Contents lists available at ScienceDirect

Artificial Intelligence In Medicine

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

Research paper

A genetic programming-based convolutional deep learning algorithm for identifying COVID-19 cases via X-ray images

Mohammad Hassan Tayarani Najaran

School of Computing Science, University of Hertfordshire, Hatfield, UK

ARTICLE INFO

Keywords:
Genetic programming
Deep learning
Optimization
Evolutionary algorithms
COVID-19
Convolutional Neural Networks

ABSTRACT

Evolutionary algorithms have been successfully employed to find the best structure for many learning algorithms including neural networks. Due to their flexibility and promising results, Convolutional Neural Networks (CNNs) have found their application in many image processing applications. The structure of CNNs greatly affects the performance of these algorithms both in terms of accuracy and computational cost, thus, finding the best architecture for these networks is a crucial task before they are employed. In this paper, we develop a genetic programming approach for the optimization of CNN structure in diagnosing COVID-19 cases via X-ray images. A graph representation for CNN architecture is proposed and evolutionary operators including crossover and mutation are specifically designed for the proposed representation. The proposed architecture of CNNs is defined by two sets of parameters, one is the skeleton which determines the arrangement of the convolutional and pooling operators and their connections and one is the numerical parameters of the operators which determine the properties of these operators like filter size and kernel size. The proposed algorithm in this paper optimizes the skeleton and the numerical parameters of the CNN architectures in a co-evolutionary scheme. The proposed algorithm is used to identify covid-19 cases via X-ray images.

1. Introduction

The spread of the Severe Acute Respiratory Syndrome CoronaVirus 2 (SARS-CoV-2) which causes CoronaVirus Disease 2019 (COVID-19) has caused grave concerns for governments around the world. Apart from the number of people who have succumb to death due to the disease, the pandemic has results in great economic impact on societies around the world. In this sense, identifying the COVID-19 cases is crucial for contact tracing and curbing the outbreaks. There are many testing methods used to identify the cases, and among the widely used ones is Reverse Transcription-Polymerase chain reaction (RT-PCR). This method uses reverse transcription to obtain DNA and then applies Polymerase Chain Reaction (PCT) to amplify the DNA for analysis. This method is accurate, but the is costly, requires time and medical resources. Therefore, developing low-cost and rapid methods for identifying the infected cases is a matter of importance.

Early in the epidemic, scientists discovered that analyzing CT images can be used for identifying covid-19 cases. It was shown that bilateral pulmonary parenchymal ground glass and consolidative pulmonary opacities is observed in the chest CT images of covid-19 patients.

The main objective of this paper is to develop a method for automatically detecting COVID-19 cases via X-ray images. To implement such an algorithm several sub-objectives have been identified. In the first step, a CNN algorithm was developed. Then, an evolutionary algorithm was proposed to optimize the architecture of the CNN. To optimize a CNN, two sets of parameters should be optimized, one is numerical and the other is the skeleton parameters of the algorithm. The proposed evolutionary algorithm aims at optimizing both sets of parameters. Because optimizing each set of parameters requires a specific set of evolutionary operators, the paper proposes two evolutionary processes that optimizes the architecture in a co-evolutionary scheme.

2. Convolutional neural networks

Convolutional Neural Networks (CNNs) have attracted the attention of many researchers due to their great potentials in image processing tasks. These networks have been widely used in a variety of tasks including face recognition [1], natural language processing [2], handwritten recognition [3], object recognition [4], radiology image processing [5] crowd counting [6], etc. Also, CNNs have been used in other machine learning tasks like transfer learning [7–9]. Transfer learning refers to storing knowledge about a solution to a problem and using it for solving a related problem. Inspired by convolutional model of cat visual cortex, CNNs were developed for processing video signals [10]. The first version of CNNs, called LetNet-5 used back-propagation algorithm [11] to optimize the connection weights. Since

E-mail address: m.tayaraninajaran@herts.ac.uk.

https://doi.org/10.1016/j.artmed.2023.102571
Received 15 September 2022; Received in revised form 7 March 2023; Accepted 27 April 2023
Available online 9 May 2023

then, a variety of different versions of CNNs are proposed including VGGNet [12] and ResNet [13]. Different versions of CNNs vary in architecture and weight connections, although majority of them utilize different approaches of gradient decent for the optimization process of the weights. The structure of the CNNs, namely convolutional operators, pooling operators and non-linear activation functions greatly affect the performance of the learning algorithm. This means that finding the optimal structure of CNNs is very important to achieve better results. Another set of parameters that affect the performance of the learning process are the initial weights of the connections. The performance of the gradient based algorithms depends on the initial weights of the connections as they determine how fast the gradient decent can reach the local optima, the quality of the local optima they find and the chance of the algorithm escaping from low fitness local optima. Thus finding the best architecture and the initial weights is a matter of importance in the learning process.

2.1. Previous work

Due to the importance of the structure of the deep learning algorithms, many research have been performed with the aim of optimizing the CNNs structure [14]. Mainly the design is performed by experts with rich knowledge about the field. However, there exist a need for automatically designing the best structure for these models. Thus, many research have attempted varies automatic methods to design the best structure for CNNs. Examples of these works include stacked auto-encoders [15,16], grid search, random search [17], Bayesian-based Gaussian process (BGP) [18,19], tree-structured Parzen estimators (TPE) [20], Reinforcement learning (RL) [21] and sequential model-based global optimization (SMBO) [22]. Each of these methods have their disadvantages. Since the grid search algorithm searches through all the possible combinations of the parameters, it guarantees the optimal set of parameters; however, the method is very computationally expensive and is not practical for many of real world applications including the structure design of CNNs. If the optimization parameters are continuous, then the grid search should descritize the search space which results in missing many areas in the search space. The random search is a blind search that usually does not reach promising results. The problem with BGP is the extra parameters, known as kernels that should be considered. In the optimization process of TPE, each parameter is optimized independently, which is unrealistic as there is huge dependency between the parameters.

A reinforcement learning scheme is proposed in [21] which performs a search on the whole CNN architecture. In this method, as a controller, a Recurrent Neural Network (RNN) is used to generate the architectural parameters of the learning algorithms. For an architecture, the performance of the CNN is evaluated and used as the reward to the RNN controller [23]. In [24], an ϵ-greedy strategy is used in Q-learning that exploits and explores the model space. Another approach in finding the optimized structure of the networks is proposed in [25], where the gradient based approach is used in the optimization process. In this method, first a structure is selected and then a gradient descent-based search algorithm is used to find the optimal architecture. Although effective, the problem with this approach is that the algorithm easily gets trapped in local optima. In [26], a method is proposed to transform the discrete architecture space into continuous to enable the gradient-based method. Because CNNs are computationally expensive, they cannot be efficiently used in wearable devices. In order to optimize these networks in terms of computational cost, a pruning algorithm is proposed in [27] that uses the energy consumption of CNNs to guide the pruning process.

Evolutionary algorithms have also been applied to the problem [28–30] and have reached reasonable results. An evolutionary paradigm is presented in [31] for optimizing hyper-parameters of deep learning algorithms. In [32], a tree growth algorithm which is a type of swarm intelligence algorithm is proposed for architecture design of CNNs. The genetic algorithms and grammatical evolution are used in [33] to optimize CNN topology. A genetic programming approach is presented in [34], in which the functional modules, such as convolutional blocks and tensor concatenation are used as node functions and the CNN structure is optimized. Among the most famous works in this field is large evolution for image classification (LEIC) performed by Google as an attempt to optimize the structure of CNNs [35]. LEIC implements on 250 high-end computers a genetic algorithm without the crossover operator which reaches competitive performance against the state of the art algorithms. In [36], a standard GA is used to evolve CNNs, in which the chromosomes have equal size.

While there are some works that have employed evolutionary algorithms to optimize the architecture of CNNs, these works mainly focus on optimizing the topology of the network. In these works, the learning weights are left to be optimized via gradient descent and the hyper-parameters are set manually. For example, AmoebaNet [37], which provides the first large scale evolutionary optimization of topology. A hierarchical representation is presented in [38] which allows a structure to evolve nonmodular layers to emerge. In [34] a Cartesian GP is used to create an architecture from modular blocks. Because the weighting and the hyper-parameters play a crucial role in the performance of CNNs, we believe that an evolutionary optimization of CNNs should be devised that automatically and simultaneously optimizes topology and weighting hyper-parameters. An evolutionary algorithm that uses the same phenotype to represent and optimize these two sets of parameters in the same process neglects the fact that each type of optimization problem requires its own set of representation and optimization procedure. Because the nature of numerical and topology optimization is different, there requires an algorithm that is specifically designed to optimize each optimization problem accordingly. In this paper, we propose a co-evolutionary scheme, in which the weighting parameters and the topological parameters of the CNN are optimized in two parallel procedures. Two different representations and optimization processes are devised in the algorithm, one is specifically designed for topology and the other for hyper-parameters.

In the previous works, the topology of CNNs is considered as an arrangement of operator blocks which are shuffled via evolutionary algorithms to find the best topology [33,39–42]. In [33], for example, the architectures are represented by a vector of blocks which limits the number of possible architectures generated by the phenotype and thus the search space of the architectures. In [33,36] the phenotype is an arrangement of operational blocks in a one dimensional array, which similarly limits the type of architectures that can be examined. In this paper, we propose a flexible graph-based representation for the solutions that is capable of generating and evaluating complex architectures. The graph-based representation proposed in this paper, unlike one or two dimensional representations in the literature, is capable of representing and examining a wide variety of architectures. The crossover and mutation operators presented in this paper are specifically designed for the graph representation to generate and explore all the possible architectures in this landscape.

2.2. CNN architecture optimization

As presented in Fig. 1, CNNs consist of a number of convolution and pooling layers stacked on top of each other. In this example, two convolutional and two pooling operators are used. The structure consists of four groups of feature maps and one fully connected layer at the tail. All the elements of the four groups of the feature maps are flattened and fed to the last layer. In general, CNNs consist of a combination of the convolutional and pooling operators at the beginning and a number of fully connected layers at the end of the pipeline. In CNNs, not only the parameters of each block should be optimized, but also the order of blocks determine the performance. Therefore, the optimization algorithm should optimize the parameters of the blocks and the order of the blocks simultaneously.

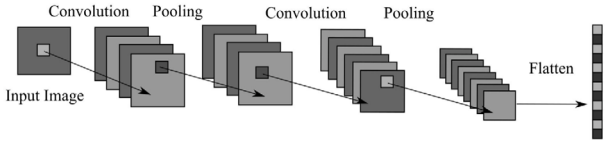

Fig. 1. The architecture of CNNs, consisting of a combination of convolution and pooling operators.

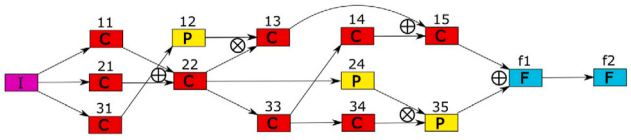

Fig. 2. The representation of solutions in a graph structure, where P represents the pooling, C represents the convolution operators and F represent the fully connected layers. When a node has more than one input, the inputs are aggregated via summation ⊕ or concatenate, ⊗ operators.

The input of the CNN is an n×n image that is processed in the pipeline. In order to generate a feature map via the pipeline, a filter should be defined for the convolution operator. The filter is a matrix and is usually randomly initialized with a predefined size (its width and height). The filter then slides over the image from the top left to the top right of the input image with the step size called stride. It then moves downward one step with the size equal to stride and traverses the image. This process continues until the whole image is covered. Each pixel in the feature map is then the sum of the products of the corresponding pixels in the filter and the image. The convolution operator here has some parameters that affect the performance of the feature extraction process. The parameters are *connection weight*, *convolution type*, *filter width*, *filter height*, *number of feature maps*, the *stride width* and the *stride height*.

The pooling operator is similar to the convolution in the way it traverses on the image to cover all the pixels. The difference is the operation it applied. The pooling operator performs via a predefined matrix called kernel some computations on the previous layers to collect the average or the maximum value of the elements in the previous layer. This operator is applied as layers to streamline the underlying processed information by reducing the spatial size of the representation to reduce the amount of computation, number of parameters and the required memory. By combining the output of a cluster of neurons at one layer into a single neuron in the next layer, the pooling operators reduce the dimensions of the data. The parameters of each pooling layer are the *pooling type*, *stride height*, *stride width*, *kernel height*, *kernel width* and *pooling type*.

The other set of parameters that should be optimized is the connection weight initialization. There are three main ways in which the connection weights can be initialized. (1) First is to initialize all the values connection weights with a constant value, like zero, one or any other number. The problem with this method is that the initialization is highly biased, so the performance of the search algorithms is significantly affected which usually leads to not very satisfying results. (2) The other way is to randomly initialize the values via a random number generator with particular distribution. Examples of this is to initialize the weights based on the Gaussian or uniform distribution. This approach resolves the problem associated with the first approach; however, there are some parameters in this method that should be tuned. For example, for the Gaussian distribution function, the mean and the standard deviation of all the connection weights should be set. To manage this, the Xavier initializer performs a sampling based on the neuron saturation before utilizing the sigmoid activation function [43]. Although successful, the Xavier initializer suffers from some deficiencies. One is that these values are much related to the structure of the CNN, and if the structure is not optimized properly, the initializer would not perform well either. The other problem is that Xavier initializer uses sigmoid activation, while most of approaches use ReLU as the activation function [13,44–46]. (3) The third way is to initialize the values based on some prior expert knowledge. For example, in [43], different initialization ways and their effect on the performance of gradient descent algorithms are studied and it is shown how prior knowledge can be used in the process.

In the optimization process of CNNs architecture consist of two main parts. One is to find the best number of operational layer and their combinatorial ordering, and the other is to optimize the numerical parameters of the CNNs, like the kernel and filter size, initialization weights, etc. To the best of our knowledge, all the works performed in the area try to optimize the parameters and the architecture in the same optimization process with the same genetic encoding. There is, however, a major deficiency with this approach. The optimization process of finding the best architecture for CNNs consists of two fitness landscapes: a) the combinatorial ordering of the layers and b) the numerical parameters of the CNN operators. The first fitness landscape is the combinatorial optimization problem of skeleton of the architecture and the second landscape is a numerical optimization problem of the operators' parameters. Although highly connected, these two landscape are different in nature and should be optimized in different processes. However, because these two are highly connected, the optimization process for both should be performed in cooperation. In order to overcome this, we propose a cooperation coevolution approach for the optimization of the structure of CNNs. In our proposed method, the optimization process consists of two threads that simultaneously perform search to optimize the architecture in both landscapes. One thread searches through the combinatorial ordering of the convolutional and pooling operators fitness landscape, and the other thread explores in the fitness landscape of the numerical parameters. These two threads should perform in connection with each other as they are highly related.

3. The proposed algorithm

In this section we introduce the proposed evolutionary algorithm for optimization of deep CNNs architecture for the identification of COVID-19 cases via X-ray images. In order to use genetic algorithms, first a coding for the solutions should be proposed. As mentioned before, there are two aspect of the CNNs architecture that should be optimized, one is the combinatorial ordering of the convolutional and pooling operators, and one is parameters of these operators. This means that a coding should be proposed that represents both aspects. In terms of the numerical parameters of the operators, a simple fixed-size numerical coding can provide a reasonable representation. In terms of the combinatorial optimization of the ordering of the operators, however, it is more complicated. In this landscape, not only the order of the operators should be optimized, but also the number of layers is a matter of importance. There is no straight forward way of knowing the optimal number of layers. Too few number of layers may result in not enough processing and so not good performance. Too many layers, on the other hand, will result in higher computational cost. It also may result in poor performance as there will be more number of learning parameters that the learning process needs to optimize. A consequence of a larger number of learning parameters is a larger search space for the learning algorithms which obviously makes the better solutions harder to find. In this sense, not only the ordering of the convolutional and pooling operators should be optimized, but also the number of these operators should be adjusted. Therefore, the proposed coding should be flexible enough to both search through the ordering of the operators and the number of operators. To manage this, we propose a graph coding for the solutions as presented in Fig. 2.

In this representation, each node in the graph represents an operator. This graph coding gives a flexible representation that enables

the algorithm to explore in the combinatorial fitness landscape of the skeleton of CNNs. At the same time, the representation is flexible enough to allow architectures with different number of layers, different combination of connections, different number of operators, etc. It is crucial in this problem to explore among CNNs with different number of operators, as the number of operators greatly affect the performance of the resulting CNN. There sits an operator in each node of this graph that can be convolutional or pooling operators or fully connected layers. Note that the fully connected layers are always at the tail of the CNNs.

The proposed representation is a directed graph, where the output of previous layers are only connected to the operators in the next layers. There are some parameters that determine the size of the CNN, which are as follows,

- H: The number of rows in the graph
- W: The number of columns
- O: The maximum number of outputs of each operator
- I: The maximum number of inputs to each operator
- L: The maximum number of fully connected layers
- l: The levels-back parameter

In a CNN, the input of each operator at level c comes from the output of the operators in previous layers from $c-l$ to $c-1$. Here l, which is called the level-back parameter, determines the maximum number of layers a connection can shortcut.

When a node receives more than one input, then the inputs should first be aggregated before they are fed to the operator. In this paper we use summation, denoted as $\oplus$ and concatenate, $\otimes$ operators. The concatenation operation concatenates two feature maps in the channel dimension. If the input feature maps are of different sizes, the larger image is downsampled via max pooling so the two become the same size. The summation operator here performs an element-wise sum of the input feature maps. If there are more than one channel, the operation is performed channel by channel, and if the inputs are of different sizes, the larger image is downsampled. If the input feature maps have different number of channels, then the smaller feature map is padded with zeros to increase the number of channels so the input feature maps are with the same number of channels. Having these operators makes the structure flexible to include shortcut connections and branching layers as in GoogleNet [46] and Residual Net [13].

The nodes in this graph are each a vector of numerical values representing the parameters of the operators. The operators are encoded as follows,

1- Convolutional layer: the stride width, the stride height, the filter width, the filter height, the number of feature maps, the convolutional type, the standard deviation and the mean value of the filter elements.

2- Pooling layer: the stride width, the stride height, the kernel width, the kernel height, the number of feature maps, the pooling type chosen from the average or maximum.

3- Fully connected layer: the mean and standard deviation of the connection weights and the number of neurons.

This representation allows the optimization algorithm to explore both in the skeleton landscape, by searching in different orderings of the operators and to search the fitness landscape of operator parameters by manipulating the parameter values in each node. The main body of the proposed algorithm is presented in algorithm 3.

At the beginning of the algorithm 3 the parameters of the structure of the CNNs should be set. Here, W and H determine the maximum size of the CNN. As discussed before, the size of CNN should be tuned carefully as the performance of the learning algorithm and the computational cost highly depend on the size of the neural network. The parameters of the algorithms, i.e. the population size n and mutation rate m should be set.

In step 1 of the algorithm 3, the population is initialized. In this step, n individuals X^i, $i = 1, \dots, n$ are generated randomly. In order to generate a random solution, a graph as presented in Fig. 2 is generated.

Algorithm 1 The Proposed Algorithm

```
begin
τ = 0
set the structure parameters, W, H, O, I, L, α and l
set the algorithm parameters, n, m
1. initialize the population Y_0,
2. while not termination condition do
   begin
3.   evaluates the individuals in Y_τ
4.   select the parent solutions via tournament selection
5.   if ⌊(τ/100)⌋ mod 2 = 0
     begin
6.     generate offsprings via skeleton-crossover and store
         them in O_τ
7.     perform skeleton-mutation on O_τ with probability m
     else
8.     generate offsprings via parameter-crossover and store
         them in O_τ
9.     perform parameter-mutation on O_τ with probability m
     end
10.  perform environmental selection on Y_τ ∪ O_τ
       and store it in Y_{τ+1}
     τ = τ + 1
   end
11return the best solution in Y_τ
end
```

For each node in the graph X_{wh} $(w = 1, \dots, W,\ h = 1, \dots, H)$, an operator is inserted with the probability of α. To do so, a random number $r = R(0,1)$ within is generated via a uniform random number generator, and if $r < \alpha$ an operator is placed in the node X_{wh}. Otherwise, the node is skipped and no operator is placed in the node. For example in Fig. 2, there is no operator at the nodes X_{32}, X_{23} and X_{25}. The parameter α determines the sparsity of the graph. At $\alpha = 1$ the graph will have operators at all its rows and columns. Note that α determines the sparsity at the initialization step, and the optimization process will start from this configuration. Then, during the optimization, the algorithm will evolve and optimize the sparsity, so in the end, a very sparse or dense graph may be achieved.

If it is determined to place an operator in X_{wh}, with the probability of half a convolution or a pooling operator is placed in the node. Because the pooling operators are adopted to streamline the underlying processed information, there is not much sense in putting a pooling operator at the first layer of the network where the input image is fed. Thus, we set all the nodes at the first layer of the network to be convolutional operators. Obviously, one can initialize the network to have pooling operators at the first layer and the optimization algorithm should have the ability to optimize the type of operators at each layer and if it is better to have convolutional operators at the first layer, the algorithm would automatically set them. However, some prior knowledge like this can help to limit the search space of the algorithm and accelerate the optimization.

When the operators are placed in the graph, the parameters of the operators are set at random. For example, for the convolutional operators the parameters *connection weight*, *convolution type*, *filter width*, *filter height*, *number of feature maps*, the *stride width* and the *stride height* are randomly set. The values of the filter or kernel matrices are also set at random. All these values are optimized during the optimization process. After placing the convolutional and pooling operators, the fully connected layers are generated at random. The maximum number of fully connected layers is O, thus, in order to generate these layers, a

random number $o = R(1, O)$ between 1 and O is generated using uniform random number generator and placed at the tail of the CNN. The parameters of the fully connected layers are the connection weights. Optimizing the connection weights individually is an arduous process as there may exist a large number of weights to be optimized, which is not possible to be performed in the evolutionary process. Therefore, we only optimize the mean and standard deviation of the Gaussian distribution functions that generate these weights. In the initialization step, the parameters of the fully connected layers, i.e. the mean and the standard deviation of the weights are initialized at random.

After placing the operators in the graph, the connections between the nodes should be set. Setting the connections is performed in two steps. In the first step, the input of all the nodes is determined. Because all the nodes must have at least one input, we first set the input of all the nodes to make sure no node remains without an input connection. Because all the nodes should receive an input from previous layers, the nodes at the first layer are connected to the input layer (input image). Also, for the last layer of convolutional and pooling operators, where the fully connected layers begins, the output of all convolutional and pooling operators should be connected to the fully connected operators. This is because the convolutional and pooling operators at the last layer should not remain without an output as this would make them redundant.

In the next step, the connections among the operators between the first and last layers should be set. For all the operators without an input, one of the operators from the previous layers is selected at random and a connection is created. There are two constraint parameters that should be taken into account in this process. First, is the levels-back parameter, l, which determines the maximum number of layers a connection can shortcut. Thus, the input of an operator at column c is chosen from the operators at levels $c-1$ to $c-l$. Because it is preferred to have connections to the closer layers, the operators at a closer layer are chosen with higher probability. The probability of choosing an operator from the column c' $(c - l \leq c' \leq c - 1)$ is found as,

$$p(c') = \frac{e^{c'-c}}{\sum_{i=c-l}^{c-1} e^{i-c}}. \tag{1}$$

This equation indicates that the closer columns have exponentially greater probability of being selected. Based on this probability, a column is chosen and an operator in the column is selected at random.

The other constraint parameter that should be taken into account is O which determines the maximum number of outputs an operator can have. If the number of outputs of the selected operator is equal to O, another operator is selected at random.

After the input of all the operators is determined, the algorithm should make sure that all the operators have at least one output, because an operator without an output is useless. If there is an operator without an output, one of the operators at the next columns is chosen at random and a connection is made. Here, similar to choosing the input connections, the operator is chosen from the columns $c + 1$ to $c + l$. Similarly, the closer columns to the current column have exponentially larger probability of being selected. The probability of choosing an operator from the column c' $(c + 1 \leq c' \leq c + 1)$ is calculated as,

$$p(c') = \frac{e^{c-c'}}{\sum_{i=c+1}^{c+l} e^{c-i}}. \tag{2}$$

There is a constraint here, I which determines the maximum input an operator can receive. If the selected operator has reached the maximum number of inputs, then it is skipped and another operator is selected via the same approach. The selection is performed until an operator is found with the number of inputs less than I.

The last step is to check the input of all the operators. If an operator receives more than one input, then the inputs should be merged. For all the operators with more than one input, a summation or concatenation operator is chosen at random and placed at the input. If there are three or more inputs to an operator, the same merger operator is used for all of them.

In step 3 of the algorithm 3, the solutions in the population are evaluated via fitness function. In order to evaluate the solutions, a CNN with the architecture proposed by the solution is generated, trained and tested on the data and the accuracy of the classification is taken as the fitness. In the training process, stochastic gradient descent algorithm is used in this paper. Note that there is a huge number of weights that should be optimized and gradient descent algorithms are usually better choice than evolutionary algorithms.

In step 4, the tournament selection strategy is used to select the parent solutions. In the problem of designing CNNs, there are two objectives that should be satisfied. The first objective is to find a structure that offers the best performance in terms of classification accuracy. At the same time, there also exists the objective of minimizing computational complexity of the algorithm. In many applications, for example the implementation of learning algorithms on wearable devices like smart phones or smart watches, there is always limitation on energy consumption and thus computational budget. In this sense, any selection algorithm devised for CNN architecture optimization should take both the objectives into account. However, the weight of each objective is different based on the application and the device on which the algorithm is to implemented. For some devices like smart watches, the limitation on computational complexity is huge, so the performance should be sacrificed to a great degree. In some cases there may exist powerful computers to process the task, so the performance of the learning algorithm, in terms of accuracy, is much more important than computational budget.

Algorithm 2 The Proposed Binary Tournament Selection

Input: Two individuals x_j and x_k, the performance and the cost of the solutions
begin
1. rank all the solutions in the population based on their accuracy and find $f_p(x_i)$
2. rank all the solutions in the population based on their computation cost and find $f_c(x_i)$
3. **for** all solutions calculate $f(x_i) = \beta f_p(x_i) + (1 - \beta) f_c(x_i)$
4. **if** $f(x_j) < f(x_k)$
5. return x_j
   else
6. return x_k

end

To mange this, we propose a selection mechanism presented in algorithm 2 in which both objectives are taken into account. In the proposed method, the solutions in the population are ranked based on the accuracy of the classification of the CNN and the rank of a solution, x is stored in $f_p(x)$. Then, the solutions are ranked based on computational complexity of the CNNs and the rank of a solution, x, is stored in $f_c(x)$. In the ranking process, the best solution is ranked first, so the aim is to use a selection mechanism that selects solutions with smaller $f_p(x)$ and $f_c(x)$. The proposed selection method first chooses two solutions from the population randomly. Then for both solutions calculates the value,

$$f(x) = \beta f_p(x) + (1 - \beta) f_c(x). \tag{3}$$

The parameter $0 \leq \beta \leq 1$ determines the weight of the two objectives. A larger β means the algorithm tends to optimize the solutions based on the performance of the CNN architecture in terms of accuracy and a smaller β takes more the computational cost into account. At $\beta = 0.5$

the algorithm treats both objectives with the same weight, at $\beta = 1$ only the accuracy and at $\beta = 0$ only the computational cost are optimized.

We can also use multi-objective optimization approaches, like SPEAII, that find the Pareto front. Such Pareto front consists of a variety of solutions that satisfy different objectives to different degrees, so the end user can choose from the Pareto front a solution that satisfies their requirements. The advantage of the proposed method is that the degree of the importance of each objective can be determined before the optimization and the algorithm will focus on finding only these solutions. For example, if the aim is to optimize the performance of CNNs in terms of accuracy, β is set to 1 and the evolutionary algorithm will focus only on the solutions that satisfy this objective. Or if accuracy and computational cost are of the same importance, β is set to 0.5 and the algorithm will focus on finding solutions that satisfy both objective to the same degree. In algorithms that find the Pareto front, however, the population always consists of solutions that satisfy different objectives. This means that the solutions in the population are scattered in the landscape around different local optima each satisfying different objectives in different degrees. Exploitation in these algorithms is sacrificed to achieve the diversity required for the Pareto front. Exploitation in the proposed algorithm, on the other hand, is performed better, because the population moves towards the area in the fitness landscape in which the objectives are satisfied to the required proportion. This way, the solutions in the population are all have the same objective so search around the same local optima.

In step 5 of the algorithm, it is determined if the skeleton optimization or the parameter optimization is performed. In CNNs, the skeleton, which is the topology of the graph and the parameters of the operators, which are numerical numbers should be optimized. Therefore the optimization problem can be decomposed into two fitness landscapes which are highly connected but of different natures. In this paper we propose a co-evolutionary process to optimize the CNN architectures in both fitness landscapes. In the proposed algorithm, the optimization process swaps between the two fitness landscapes in each 100 iterations. For 100 iterations, the optimization process is performed to optimize the skeleton of the CNNs while the parameters of operators are fixed. Then, for 100 iterations, the skeleton (the graph) of the solutions is fixed and the optimization is performed to optimize the parameters of the operators, i.e filter and kernel size, stride size, etc. The algorithm keeps switching between these two processes until the maximum number of iterations is reached.

In step 6, the skeleton-crossover is applied. The procedure of skeleton-crossover is presented in Fig. 3 where the two architectures at the top are crossed over and the architecture at the bottom is achieved. In order to apply the skeleton crossover between two architectures, a crossover point is randomly selected that splits the two graphs into two parts. The crossover point should be at the same place for both individuals. The resulting individual receives one part from the first individual and the other part from the second individual. When the crossover is applied, the connections that connect two nodes in the same part remain with no change. For example, in Fig. 3, the resulting individual receives its left part from the first and its right part from the second individual. All the connections between the nodes 11, 21, 31, 12, 22, 13 and 33 remain with no change in the resulting individual. The same applies to the right part. Then, the connections among the nodes that were connected to a node from a different part are set.

In order to set the connections at the border of the two parts, first the input of the nodes of the right part are set. Each node at the right part receives its input from the node at the same place it was connected to in the parent individual. For example, in Fig. 3, the node 25 will be connected to 12 and the node 24 gets connected to 33. Here, it does not matter if the type of the source node changes. If in this process, a node is to receive its input from a node that does not exist in the new architecture, the closest same type node is chosen as the source. For example, in the bottom parent, the node 34 is connected to the node 23, but in the resulting individual, there is no node at the place 23. In this case, the node 34 will receive its input from the closest node to 23 which is 33. Closest node means the node with the minimum Manhattan distance to the node. If there are more than one nodes with the same distance, the same type node has priority. That is if the source node in the parent architecture is a convolutional operator, a convolutional operator in the new architecture has priority. If two nodes are the at the same distance and are of the same type, the tie is broken by randomly choosing a node. Note that in the process, the constraint on the maximum number of outputs for a node should be satisfied. Thus, if a node is chosen to receive its output from a node that has reached its maximum outputs, another node should be chosen as the source node as described (the next closest node).

After the input of the nodes at the right part are set, some nodes at the left part may remain with no output. For example, the node 13 remains with no output. The nodes at the left part with no connection are connected to the node to which they were connected in the parent individual. In the case of Fig. 3, node 12 is connected to the node 25. If in the crossover process the number of inputs for a node changes from one to more than one, then a concatenation or summation operator is randomly selected and put at the input. Node 15 has one input in the parent and two inputs in the offspring individual, so a merging operator (summation in this case) is chosen and placed at the input.

In step 7 of the algorithm the skeleton-mutation is performed. The skeleton-mutation performs in the following ways.

1- Select randomly a node in the graph, and change its type (if it is convolutional, it is changed to pooling and vise versa). When changed, the common parameters between convolutional and pooling operators, namely filter/kernel size, stride width and stride height remain with no change. The parameters that are different are initialized randomly. By retaining the common parameters, the operator maintains the already optimized parameters. The convolutional operators extract features and the pooling operators reduce dimensionality of the generated information. This mutation operator is devised to explore in the dimension of convolutional to pooling operator ratio. A CNN with too many convolutional and too few pooling operators will produce too many features which usually results in high computational cost and poor performance. Similarly, a CNN with a small convolutional to pooling operator ratio does not extract enough features, leading to low performance. This mutation explores to find the best balance in the ratio of convolutional to pooling operators.

2- Randomly select two nodes in the graph and swap their operators. This operator explores in the combinatorial ordering of the existing operators in the graph. For a certain set of operators, different ordering of operators results in different performance. This mutation is devised to find the best ordering of the operators.

3- Randomly remove from or add to the graph a node. A slot in the graph is chosen randomly, if there is a node in the slot, the node and its connections are eliminated. When eliminating the node, some nodes that were connected to this node may lose all their input or output connections. If this happens, the procedure explained in the initialization is performed to set a new input or output for the node. If there is no node in the slot, a convolutional or pooling operator is placed in the slot. The operator is chosen at random. The input and output of the newly placed node are set using the process described in the initialization step. This mutation is devised to optimize the number of operators in the network.

4- Randomly remove or add a connection to the graph. If removing a connection leaves a node with no input or output, the process in the initialization is performed to set the input or output for the node. This mutation operator is devised to explore in the connection space.

5- Randomly select an operator with more than one inputs and swap the merging operator. That is, if the inputs are merged via summation, $\oplus$, change the operator to $\otimes$ and vice versa.

The proposed algorithm optimizes in a co-evolutionary paradigm, the skeleton and the parameters of the CNNs. The crossover operator proposed in Fig. 3 is adopted to optimize the skeleton of the networks.

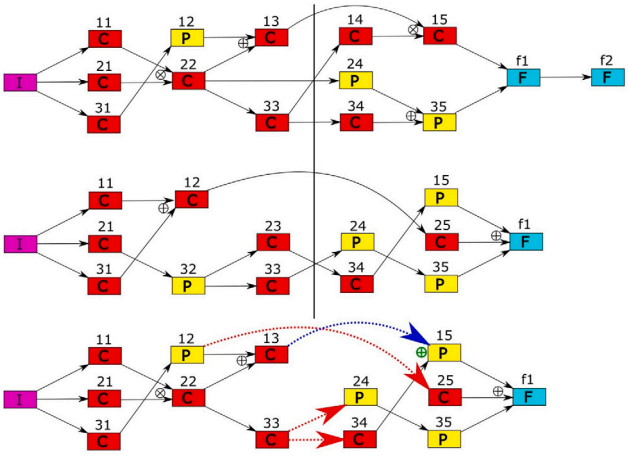

Fig. 3. Example of skeleton crossover that performs on the topology of the CNNs. The crossover is performed on the two CNNs at the top and the CNN at the bottom is achieved. The straight line between the column 3 and 4 show the crossover point.

Presented in Fig. 4, we propose a crossover to optimize the parameters of the network. Each node in the graph is an operator which is represented by its set of numerical parameters. In order to perform the parameter-crossover between the two individuals in Fig. 4, the convolutional, pooling and fully connected operators in each individual are aligned. Aligning is performed by selecting the nodes from top to bottom from the columns. The columns are selected from left to right. After aligning the operators, a crossover is performed between the operators. Note that because some places in the graph may be empty, or the operators at a place can be any of pooling or convolutional operators, the crossover is not necessarily applied between the operators at the same place in the graph. For example, in Fig. 4, the pooling operator at the place 12 in the top parent is matched with the pooling operator at 32 in the bottom one. Similarly is for the convolutional operator at 22 with 12. Also, some operators may not find a pair an so are passed to the offspring without undergoing the crossover operator. At step 9 of the algorithm, the parameter-mutation is performed. The parameter-mutation selects a node from the graph at random and sets the numerical parameters of the operator in the node randomly.

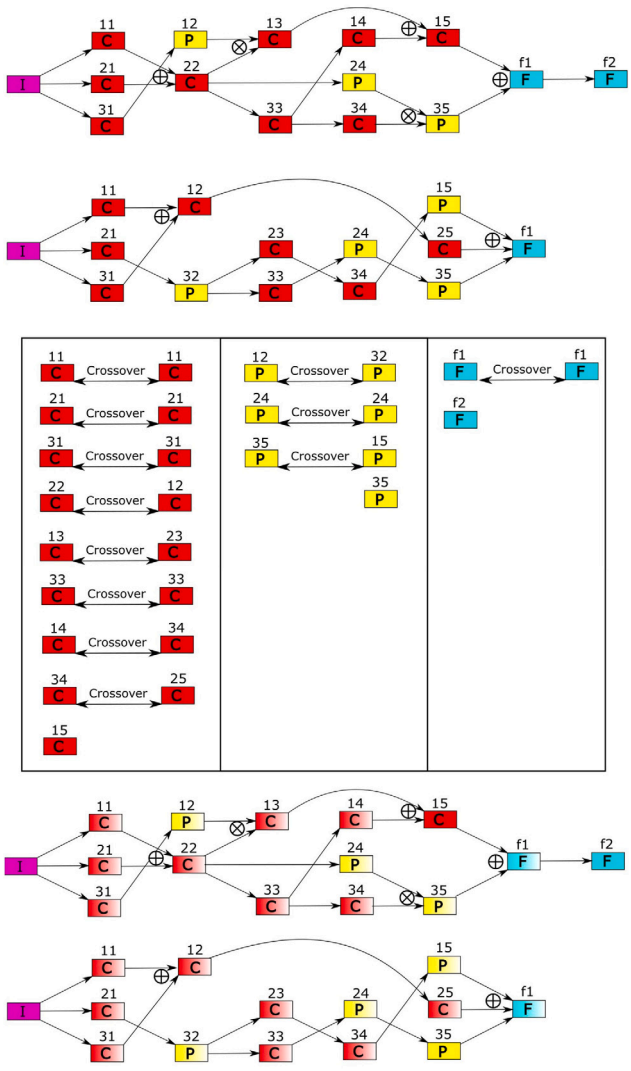

Fig. 4. Example of the parameter crossover between the two solutions at the top. After crossover, the solutions keep their skeleton and the crossover is only applied to the parameters. The operators of each skeleton are aligned and then the crossover is applied between the corresponding operators. If an operator remains with no pair, no crossover will be applied to it. The resulting architectures are at the bottom, where the operators that are affected by crossover are shown by gradient colors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4. Dataset

In this paper, we use three sets of publicly available COVID-19 data that are classified into normal, pneumonia and COVID-19 cases. The data are all X-ray images in jpg format. In order to have a large enough dataset, we combined two sets of data. The first set of data are prepared by Cohen et al. [47], which contain data about MERS, SARS and COVID-19. In this dataset, there are 76 COVID-19 cases that we use in this paper. The other dataset contains 219 X-ray images and is available in [48]. Combined, the two datasets contain 295 images. The third dataset contains images of 53 patients suffering from pneumonia [49]. We specifically use the third dataset because it contains pneumonia chest images that include viral and bacterial infections. This is important as it provide a good testing measure for distinguishing pneumonia caused by COVID-19 from other types of pneumonia. There are three classes in the resulting dataset: COVID-19, normal and pneumonia class. The covid-19 class contains 295 images, the normal class contains 65 images and the pneumonia class contains 98 images, so a total of 458 images are used in this paper.

4.1. Data preprocessing

In order for the learning algorithms to properly classify the images, some preprocessing on the images should be applied. The preprocessing step in this paper consists of two steps. The first step is to enhance the images. In this paper we use the technique presented in [50] to enhance the images. The second step is to further improve the image quality via a technique called stacking. The stacking technique uses different images captured at different focal distances and combines them to produce a better quality image. This method is usually used in order to improve the quality of a dataset by removing noise and other artefacts that affect images. Because the X-ray images are taken in different environmental parameters, there requires a preprocessing phase before they are prepared for training. We use the codes presented in [51] to perform the preprocessing. In this work, we first apply image enhancement to build the enhanced dataset. Then using the stacking technique, the original dataset is combined with the enhanced dataset to create the stacked dataset.

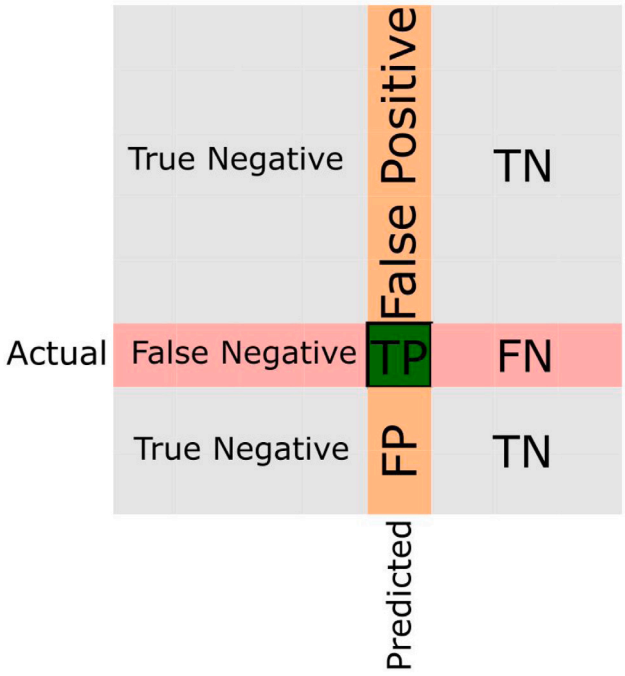

Fig. 5. The TP, TN, FP, and FN in the confusion matrix for multi-class classification problems. The rows show the actual class and the columns represent the predicted classes.

5. Experimental results

In order to test the algorithms the confusion matrix metrics are used in this paper. The metrics are based on True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) cases. These values are easy to grasp for binary classification. For multi-class classification, however, they are more complicated. For a given class k, TP is defined as the number of cases that have correctly been classified. The FN cases are the cases belonging to the class k that have been incorrectly classified. The FP cases are the cases that do not belong to the class k, but have incorrectly classified as k. The TN cases are all the cases that neither belong to the class k, nor have been classified as k. To clarify, Fig. 5 shows the cases in the confusion matrix.

In this paper, the following metrics have been used to compare the algorithms.

$$\mathrm{Se} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{4}$$

$$\mathrm{Sp} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}, \tag{5}$$

$$\mathrm{Pre} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{6}$$

$$\mathrm{Acc} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}. \tag{7}$$

The overall accuracy is the total number of correctly classified cases divided by the total number of data records.

In order to compare the proposed algorithm with state of the art algorithms, we use CAE-2 [52], TIRBM [53], PGBM+DN-1 [53], ScatNet-2 [54], RandNet-2 [55], LDANet-2 [55], SVM+RBF [56], SVM+Poly [56], NNet [56], SAAA-3 [56], SqweezNet (SQNet) [57], MobileNetV2 [58], DBN-3 [56], LSTM and the algorithms proposed in [33,39–42].

The parameters of these algorithms including the maximum length of each layer have been set based on the conventions in the community [59]. For the evolutionary algorithm, the population size is set to 10, the number of iterations to 1000 (i.e. 10,000 fitness evaluations), the crossover-rate is 0.9 and the mutation rate is 0.1. The number of columns and rows are set to 15 and 3 respectively and the number of fully connected layers is set to 1. This size should be sufficient for the type of problem we are dealing with. For the fitness evaluation, 80% of data are used for training and 20% for testing. In order to simplify the search process, we have set the width and the height of the filter, stride and kernels to be the same (these are square matrices). The width and height of the stride in convolutional layer are set to one. For the pooling layers, the width and height of the stride are set to the same size as their kernel. It is common to set these values in the deep learning community. For the rival evolutionary optimization algorithms for CNN architecture design, we used the same set of parameters suggested in the reference papers.

In order to find the best deep training epoch and the epoch for catching the tendency of the performance, we randomly select some individuals and train them with a large number of epoch. We then record the classification errors at each epoch and then choose the best number of epoch which makes a trade-off between the computational cost and performance. In all experiments the deep training epoch is set to 100 and the epoch for catching the tendency of the performance is set to 10. We use Tensorflow [60] to implement our proposed algorithm. All the experiments are averaged over 30 runs.

Table 1 shows the experimental results on different CNNs. In terms of overal accuracy, as the data suggest, the best performance among the algorithms is achieved by the proposed evolutionary optimization of the CNNs architecture. After the proposed algorithms are the algorithms proposed in [39,40]. In terms of sensitivity of detecting COVID-19 cases, the best performance is achieved by the proposed algorithm followed by the algorithms proposed in [41,42]. The algorithms that optimize the architecture perform better than preset architectures. This suggests that tuning the architecture for a problem results in better performance. Overall, all the algorithms perform similar in detecting the COVID-19 and pneumonia cases, while their performance is worse when it comes to detecting normal cases.

Here, we present some extra experiments on the proposed algorithms. As explained before, we perform two steps of preprocessing on the images to prepare them for classification. In this section, we perform the classification without applying the preprocessing to show the effect of the process on the performance of the algorithms.

Table 2 shows the experimental results when only enhancement is performed on the data before the classification phase.

Table 3 shows the experimental results when only stacking is performed on the data before the classification phase.

Table 4 summarizes the statistical analysis on the data in Tables 1–3. Here, "Prop Alg". represents the proposed algorithm, "Enhn" represents the results when only enhancement is applied and "stacking" represents when stacking is applied. The ANOVA and Kruskal–Wallis are applied to test the significance of the difference between the results achieved by the set of algorithms used in this paper for comparison. These are for the data corresponding to COVID-19 results. In these tests, 'SS' represents the sum of squares of each source, 'df' shows the degree of freedom associated with each source, 'MS' is the mean squared (ie. the ratio SS/df) and 'Chi-square' is the ratio of mean squares. In this table, the p-values are very small, so the null-hypothesis that the samples are taken from the same mean is rejected with a high probability significance level.

Fig. 6, shows the progress of the evolutionary algorithm in finding the optimal architecture of CNN. These are for five metrics Se., Sp., Pre. Acc and overall accuracy. As mentioned, the number of iterations in the evolutionary process is 1000. The proposed algorithm performs in a co-evolutionary scheme, where after every 100 iterations, the evolution swaps between the optimization of numerical parameters and the skeleton. This behavior can be observed in Fig. 6. The vertical lines in this figure show the points at which the evolutionary process swaps between the two optimization schemes. As the graph suggests, quickly after the evolutionary process swaps (just after the vertical lines), the performance improves more rapidly, until it becomes slower

Table 1
The experimental results for different algorithms. The data are averaged over 30 runs. Junior is the method that is presented in [39], Ma is [40], Gottapu is [41], Sun is [42], and Baldominos is [33].

Alg.	Classes	Se.	Sp.	Pre.	Acc.	OA
LSTM	COVID-19	96.27	92.02	95.62	94.76	93.01
	Normal	86.15	97.71	86.15	96.07	
	Pneumonia	87.76	97.22	89.58	95.2	
SVM+RBF	COVID-19	97.63	93.25	96.32	96.07	93.23
	Normal	89.23	96.18	79.45	95.2	
	Pneumonia	82.65	98.61	94.19	95.2	
SVM+Ploy	COVID-19	97.29	92.64	95.99	95.63	93.89
	Normal	78.46	98.98	92.73	96.07	
	Pneumonia	93.88	96.67	88.46	96.07	
SAA-3	COVID-19	97.63	93.87	96.64	96.29	93.45
	Normal	78.46	97.96	86.44	95.2	
	Pneumonia	90.82	96.67	88.12	95.41	
NNet	COVID-19	97.29	92.64	95.99	95.63	93.23
	Normal	86.15	96.95	82.35	95.41	
	Pneumonia	85.71	98.06	92.31	95.41	
DBN-3	COVID-19	97.63	93.87	96.64	96.29	93.89
	Normal	86.15	97.46	84.85	95.85	
	Pneumonia	87.76	97.78	91.49	95.63	
CAE-2	COVID-19	95.59	94.48	96.91	95.2	94.32
	Normal	95.38	97.2	84.93	96.94	
	Pneumonia	89.8	98.33	93.62	96.51	
TIRBM	COVID-19	96.95	93.25	96.3	95.63	93.01
	Normal	81.54	97.71	85.48	95.41	
	Pneumonia	88.78	96.67	87.88	94.98	
PGBM	COVID-19	98.31	96.32	97.97	97.6	95.2
	Normal	90.77	96.95	83.1	96.07	
	Pneumonia	88.78	98.89	95.6	96.72	
ScatNet-2	COVID-19	97.97	95.09	97.31	96.94	95.85
	Normal	84.62	98.73	91.67	96.72	
	Pneumonia	96.94	98.33	94.06	98.03	
RandNet-2	COVID-19	98.31	94.48	96.99	96.94	95.63
	Normal	89.23	98.22	89.23	96.94	
	Pneumonia	91.84	98.89	95.74	97.38	
LDANet-2	COVID-19	97.29	95.71	97.62	96.72	96.07
	Normal	92.31	99.49	96.77	98.47	
	Pneumonia	94.9	97.5	91.18	96.94	
SQNet	COVID-19	96.61	95.09	97.27	96.07	94.32
	Normal	87.69	97.46	85.07	96.07	
	Pneumonia	91.84	97.78	91.84	96.51	
MBNet-2	COVID-19	97.29	95.09	97.29	96.51	95.63
	Normal	90.77	98.47	90.77	97.38	
	Pneumonia	93.88	98.33	93.88	97.38	
Junior	COVID-19	98.30	95.71	97.66	97.82	97.38
	Normal	96.92	99.49	96.92	99.13	
	Pneumonia	92.86	99.17	96.81	97.82	
MA	COVID-19	98.31	96.93	98.32	98.25	97.16
	Normal	92.31	98.98	93.75	98.03	
	Pneumonia	94.9	98.89	95.88	98.03	
Gottapu	COVID-19	98.64	96.93	98.31	98.03	96.51
	Normal	89.23	98.22	89.23	96.94	
	Pneumonia	94.9	98.89	95.88	98.03	
Sun	COVID-19	98.31	95.09	97.32	97.16	96.51
	Normal	92.31	98.98	93.75	98.03	
	Pneumonia	93.88	98.89	95.83	97.82	
Baldominos	COVID-19	96.27	98.77	99.3	97.16	96.29
	Normal	93.85	97.96	88.41	97.38	
	Pneumonia	97.96	98.06	93.2	98.03	
GPNet	COVID-19	98.98	98.16	98.98	98.25	98.25
	Normal	96.92	99.75	98.44	99.34	
	Pneumonia	98.31	98.89	96.04	98.91	

Table 2
The experimental results for different algorithms when only enhancement is applied to the input images. The data are averaged over 30 runs. Junior is the method that is presented in [39], Ma is [40], Gottapu is [41], Sun is [42], and Baldominos is [33].

Alg.	Classes	Se.	Sp.	Pre.	Acc.	OA
LSTM	COVID-19	94.92	92.02	95.56	93.89	91.92
	Normal	86.15	97.46	84.85	95.85	
	Pneumonia	86.73	96.11	85.86	94.1	
SVM+RBF	COVID-19	95.59	89.57	94.31	93.45	91.92
	Normal	86.15	97.71	86.15	96.07	
	Pneumonia	84.69	96.94	88.3	94.32	
SVM+Ploy	COVID-19	94.92	93.25	96.22	94.32	92.36
	Normal	87.69	96.95	82.61	95.63	
	Pneumonia	87.76	96.67	87.76	94.76	
SAA-3	COVID-19	95.93	93.87	96.59	95.2	92.14
	Normal	81.54	96.95	81.54	94.76	
	Pneumonia	87.76	96.11	86	94.32	
NNet	COVID-19	96.27	91.41	95.3	94.54	92.14
	Normal	76.92	97.2	81.97	94.32	
	Pneumonia	89.8	96.94	88.89	95.41	
DBN-3	COVID-19	96.61	91.41	95.32	94.76	92.36
	Normal	81.54	97.2	82.81	94.98	
	Pneumonia	86.73	97.22	89.47	94.98	
CAE-2	COVID-19	93.9	95.71	97.54	94.54	92.58
	Normal	86.15	95.42	75.68	94.1	
	Pneumonia	92.86	97.5	91	96.51	
TIRBM	COVID-19	95.59	96.32	97.92	95.85	91.92
	Normal	81.54	95.67	75.71	93.67	
	Pneumonia	87.76	96.11	86	94.32	
PGBM	COVID-19	95.25	92.64	95.9	94.32	92.79
	Normal	87.69	97.46	85.07	96.07	
	Pneumonia	88.78	96.94	88.78	95.2	
ScatNet-2	COVID-19	97.63	95.09	97.3	96.72	94.54
	Normal	89.23	97.46	85.29	96.29	
	Pneumonia	88.78	98.06	92.55	96.07	
RandNet-2	COVID-19	96.95	93.25	96.3	95.63	94.1
	Normal	87.69	98.47	90.48	96.94	
	Pneumonia	89.8	97.22	89.8	95.63	
LDANet-2	COVID-19	96.61	95.09	97.27	96.07	94.54
	Normal	89.23	98.47	90.63	97.16	
	Pneumonia	91.84	96.94	89.11	95.85	
SQNet	COVID-19	95.93	93.25	96.26	94.98	92.36
	Normal	87.69	96.95	82.61	95.63	
	Pneumonia	84.69	96.67	87.37	94.1	
MBNet-2	COVID-19	96.95	93.25	96.3	95.63	93.23
	Normal	83.08	97.2	83.08	95.2	
	Pneumonia	88.78	97.5	90.63	95.63	
Junior	COVID-19	98.31	97.55	98.64	98.03	96.51
	Normal	93.85	97.71	87.14	97.16	
	Pneumonia	92.86	99.17	96.81	97.82	
Ma	COVID-19	97.63	95.09	97.3	96.72	95.85
	Normal	87.69	99.49	96.61	97.82	
	Pneumonia	95.92	97.5	91.26	97.16	
Gottapu	COVID-19	97.97	93.87	96.66	96.51	95.41
	Normal	86.15	98.22	88.89	96.51	
	Pneumonia	93.88	98.89	95.83	97.82	
Sun	COVID-19	98.31	96.32	97.97	97.6	95.2
	Normal	84.62	98.47	90.16	96.51	
	Pneumonia	92.86	97.22	90.1	96.29	
Baldominos	COVID-19	97.63	93.87	96.64	96.29	94.98
	Normal	90.77	97.71	86.76	96.72	
	Pneumonia	89.8	98.89	95.65	96.94	
GPNet	COVID-19	98.64	96.93	98.31	98.03	96.94
	Normal	93.85	98.47	91.04	97.82	
	Pneumonia	93.88	99.17	96.84	98.03	

later on (before reaching the next vertical line). This is because when the algorithm optimizes one set of parameters (skeleton or numerical parameters), after some iterations, it reaches a local optimum where improvements become harder. Then, when the optimization swaps to the other set of parameters, the improvements become quicker again to reach local optima. This is because when the algorithm optimizes one

Table 3
The experimental results for different algorithms when only stacking is applied to the input images. The data are averaged over 30 runs. Junior is the method that is presented in [39], Ma is [40], Gottapu is [41], Sun is [42], and Baldominos is [33].

Alg.	Classes	Se.	Sp.	Pre.	Acc.	OA
LSTM	COVID-19	95.93	92.02	95.61	94.54	91.48
	Normal	75.38	96.95	80.33	93.89	
	Pneumonia	88.78	96.11	86.14	94.54	
SVM+RBF	COVID-19	95.59	90.18	94.63	93.67	91.92
	Normal	81.54	97.46	84.13	95.2	
	Pneumonia	87.76	96.94	88.66	94.98	
SVM+Ploy	COVID-19	96.61	92.02	95.64	94.98	92.79
	Normal	84.62	96.95	82.09	95.2	
	Pneumonia	86.73	97.78	91.4	95.41	
SAA-3	COVID-19	97.29	92.64	95.99	95.63	92.36
	Normal	72.31	96.95	79.66	93.45	
	Pneumonia	90.82	96.94	89	95.63	
NNet	COVID-19	94.92	93.25	96.22	94.32	92.14
	Normal	80	97.46	83.87	94.98	
	Pneumonia	91.84	95.83	85.71	94.98	
DBN-3	COVID-19	95.93	93.25	96.26	94.98	92.58
	Normal	75.38	97.96	85.96	94.76	
	Pneumonia	93.88	95.83	85.98	95.41	
CAE-2	COVID-19	98.64	92.02	95.72	96.29	93.67
	Normal	76.92	97.96	86.21	94.98	
	Pneumonia	89.8	97.78	91.67	96.07	
TIRBM	COVID-19	96.27	92.02	95.62	94.76	91.48
	Normal	81.54	96.44	79.1	94.32	
	Pneumonia	83.67	96.67	87.23	93.89	
PGBM	COVID-19	97.97	93.87	96.66	96.51	94.54
	Normal	87.69	97.46	85.07	96.07	
	Pneumonia	88.78	98.61	94.57	96.51	
ScatNet-2	COVID-19	97.97	95.09	97.31	96.94	95.2
	Normal	89.23	97.71	86.57	96.51	
	Pneumonia	90.82	98.61	94.68	96.94	
RandNet-2	COVID-19	97.63	96.32	97.96	97.16	94.98
	Normal	86.15	97.96	87.5	96.29	
	Pneumonia	92.86	97.5	91	96.51	
LDANet-2	COVID-19	97.29	97.55	98.63	97.38	95.41
	Normal	89.23	98.47	90.63	97.16	
	Pneumonia	93.88	96.94	89.32	96.29	
SQNet	COVID-19	95.93	93.87	96.59	95.2	93.23
	Normal	86.15	97.46	84.85	95.85	
	Pneumonia	89.8	96.94	88.89	95.41	
MBNet-2	COVID-19	96.61	98.16	98.96	97.16	94.76
	Normal	87.69	97.2	83.82	95.85	
	Pneumonia	93.88	97.22	90.2	96.51	
Junior	COVID-19	99.66	91.41	95.45	96.72	96.07
	Normal	86.15	99.49	96.55	97.6	
	Pneumonia	91.84	99.44	97.83	97.82	
Ma	COVID-19	98.87	95.09	97.33	97.6	96.07
	Normal	90.77	98.98	93.65	97.82	
	Pneumonia	90.82	98.33	93.68	96.72	
Gottapu	COVID-19	98.64	94.48	97	97.16	95.85
	Normal	87.69	98.22	89.06	96.72	
	Pneumonia	92.86	99.17	96.81	97.82	
Sun	COVID-19	97.97	95.71	97.64	97.16	95.85
	Normal	87.69	98.22	89.06	96.72	
	Pneumonia	94.9	98.61	94.9	97.82	
Baldominos	COVID-19	98.60	93.87	96.69	97.16	95.63
	Normal	89.23	97.96	87.88	96.72	
	Pneumonia	89.8	99.44	97.78	97.38	
GPNet	COVID-19	98.98	98.16	98.98	98.25	96.51
	Normal	92.31	97.96	88.24	97.16	
	Pneumonia	93.88	98.61	94.85	97.6	

set of parameters (for example the skeleton), the local optima of the other set of parameters (numerical parameters) change slightly which opens room for improvement once it is the turn for the optimization of numerical parameters.

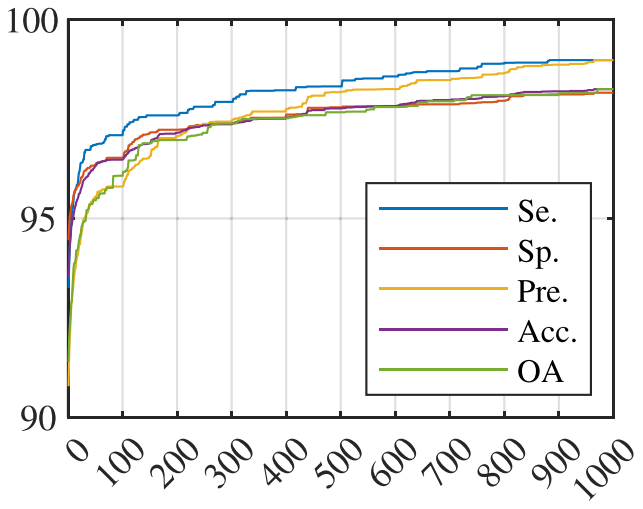

Fig. 6. The progress of the evolutionary algorithm in finding the best architecture for CNN.

6. Conclusion

The optimization process of CNN architecture consists of finding the best arrangement of convolutional and pooling operators, the filter and kernel size, stride size, etc. We believe that the architecture of CNNs consists of two distinct aspects, the skeleton that determines the arrangement of the operators and the numerical parameters of the operators. In the proposed algorithm, the optimization of the skeleton and the parameters are separated into distinct threads in a co-evolutionary paradigm. We propose a graph structure for the individuals in which each node is an operator and each edge represents a connection between two operators. We propose the crossover and mutation operators for the skeleton and the parameter optimization processes. The proposed crossover and mutation operators are flexible enough to generate any possible architecture for CNNs, so are able to explore the search space for all possible architecture for the learning algorithms. We set some constraints on the architecture, such as the number rows and columns or the connection among the nodes to limit the search space.

By separating the optimization of the skeleton from the optimization of numerical parameters into two threads, the algorithm performs the search process in each landscape separately. This helps the searching operators to explore the search space better as they need to deal will a smaller landscape at each thread. Because of the relationship between the two landscapes, the co-evolutionary paradigm is suggested so the search threads performing in the two landscapes are interconnected.

In this paper we propose a genetic programming algorithm for finding the best architecture of CNNs in detecting COVID-19 from normal and pneumonia cases via X-ray images. The best architecture found by the proposed algorithm in this paper is based on the data we used for classification. We believe that there is no optimal architecture that suits all applications and each application requires its own architectural tuning. The existing CNN architectures are usually fixed structures that are tuned beforehand and are used for a variety of applications. The advantage of the proposed algorithm is that it can be used to find the best architecture for any given application. For example, the architecture found by the proposed algorithm in this paper is specifically suitable for detecting COVID-19 cases and it is not necessarily the best architecture for, for example, detecting cars. Surely, such an architecture may be subject to over-fitting to the application it is used for (COVID-19 detection in this case). However, this is not

Table 4
ANOVA and Kruskal–Wallis test of the data in Tables 1–3. Here, "Prop Alg." represents the proposed algorithm, "Enhn" represents the results when only enhancement is applied and "stacking" represents when stacking is applied.

			Kruskal–Wallis					ANOVA				
		Source	SS	df	MS	F	Prob>F	SS	df	MS	Chi-sq	Prob>Chi-sq
Se.	Prop Alg.	Columns	5.02e−02	19	2.64e−03	1.88e+01	7.88e−49	7.86e+06	19	4.13e+05	2.61e+02	1.49e−44
		Error	8.15e−02	580	1.41e−04			1.01e+07	580	1.75e+04		
		Total	1.32e−01	599				1.80e+07	599			
	Enhn	Columns	1.01e−01	19	5.31e−03	3.40e+01	2.39e−81	1.02e+07	19	5.39e+05	3.41e+02	7.69e−61
		Error	9.07e−02	580	1.56e−04			7.76e+06	580	1.34e+04		
		Total	1.91e−01	599				1.80e+07	599			
	Stacking	Columns	8.63e−02	19	4.54e−03	6.61e+01	2.55e−131	1.27e+07	19	6.68e+05	4.22e+02	9.87e−78
		Error	3.98e−02	580	6.87e−05			5.31e+06	580	9.15e+03		
		Total	1.26e−01	599				1.80e+07	599			
Sp.	Prop Alg.	Columns	1.91e−01	19	1.01e−02	7.44e+01	1.24e−141	1.35e+07	19	7.11e+05	4.50e+02	1.88e−83
		Error	7.84e−02	580	1.35e−04			4.49e+06	580	7.73e+03		
		Total	2.70e−01	599				1.80e+07	599			
	Enhn	Columns	2.37e−01	19	1.25e−02	9.13e+01	3.12e−160	1.42e+07	19	7.46e+05	4.72e+02	4.74e−88
		Error	7.92e−02	580	1.36e−04			3.83e+06	580	6.59e+03		
		Total	3.16e−01	599				1.80e+07	599			
	Stacking	Columns	2.95e−01	19	1.55e−02	2.34e+02	5.05e−257	1.58e+07	19	8.31e+05	5.25e+02	2.67e−99
		Error	3.86e−02	580	6.65e−05			2.21e+06	580	3.82e+03		
		Total	3.34e−01	599				1.80e+07	599			
Pre.	Prop Alg.	Columns	4.71e−02	19	2.48e−03	3.01e+01	7.85e−74	1.03e+07	19	5.43e+05	3.43e+02	2.43e−61
		Error	4.78e−02	580	8.24e−05			7.68e+06	580	1.32e+04		
		Total	9.49e−02	599				1.80e+07	599			
	Enhn	Columns	7.94e−02	19	4.18e−03	2.82e+01	8.05e−70	1.03e+07	19	5.44e+05	3.44e+02	1.82e−61
		Error	8.60e−02	580	1.48e−04			7.66e+06	580	1.32e+04		
		Total	1.65e−01	599				1.80e+07	599			
	Stacking	Columns	9.89e−02	19	5.21e−03	4.21e+01	5.88e−96	1.19e+07	19	6.27e+05	3.97e+02	2.22e−72
		Error	7.17e−02	580	1.24e−04			6.08e+06	580	1.05e+04		
		Total	1.71e−01	599				1.80e+07	599			
Acc.	Prop Alg.	Columns	5.20e−02	19	2.73e−03	2.24e+01	2.83e−57	8.82e+06	19	4.64e+05	2.94e+02	3.99e−51
		Error	7.08e−02	580	1.22e−04			9.18e+06	580	1.58e+04		
		Total	1.23e−01	599				1.80e+07	599			
	Enhn	Columns	1.02e−01	19	5.36e−03	3.04e+01	2.34e−74	1.05e+07	19	5.51e+05	3.48e+02	2.11e−62
		Error	1.02e−01	580	1.76e−04			7.53e+06	580	1.30e+04		
		Total	2.04e−01	599				1.80e+07	599			
	Stacking	Columns	9.72e−02	19	5.11e−03	5.80e+01	1.82e−120	1.26e+07	19	6.61e+05	4.18e+02	7.94e−77
		Error	5.11e−02	580	8.82e−05			5.44e+06	580	9.38e+03		
		Total	1.48e−01	599				1.80e+07	599			
OA	Prop Alg.	Columns	1.40e−01	19	7.37e−03	6.04e+01	9.83e−124	1.26e+07	19	6.63e+05	4.19e+02	4.69e−77
		Error	7.08e−02	580	1.22e−04			5.41e+06	580	9.32e+03		
		Total	2.11e−01	599				1.80e+07	599			
	Enhn	Columns	1.56e−01	19	8.20e−03	6.37e+01	3.70e−128	1.30e+07	19	6.83e+05	4.32e+02	1.10e−79
		Error	7.47e−02	580	1.29e−04			5.03e+06	580	8.67e+03		
		Total	2.30e−01	599				1.80e+07	599			
	Stacking	Columns	1.71e−01	19	8.99e−03	8.90e+01	7.78e−158	1.41e+07	19	7.40e+05	4.68e+02	3.32e−87
		Error	5.86e−02	580	1.01e−04			3.95e+06	580	6.80e+03		
		Total	2.29e−01	599				1.80e+07	599			

a problem. First, because the approach helps the designers to find the optimal architecture for specific applications based on the application requirements. In real world problems, devices are designed for specific tasks and it arguably makes sense to specifically design them in a way that are particularly good at performing the task while not performing well on tasks outside the scope of the application requirements. For example, a device designed to detect cars on the roads does not need to perform optimally in distinguishing different types of fruits. Second, if an application requires an architecture suitable for variety of tasks, then the proposed algorithm is flexible enough to fulfill the need. In such a case, one should use a wide range of data when designing the CNNs architectures. This way, the algorithm will find an architecture that suits all these applications. This includes not only the performance in terms of accuracy, but also in terms of computational cost. Many applications may require to sacrifice accuracy to achieve lower computational/power cost.

The proposed algorithm shows success in diagnosing COVID-19 cases via chest X-ray images. This approach can be easily adopted in hospitals for quick diagnosis of these cases. The proposed algorithm can also be very beneficial as it provides information about how much of the lung has been affected by the virus and the level of damage inflicted on the respiratory system. This is something that cannot be diagnosed by existing test methods. Having such information helps doctors to provide the required care for patients that need special attention.

In this paper we distinguished two different fitness landscapes for the problem. To the best of our knowledge, there is no work that performs a study on the fitness landscape of this problem. Studying the fitness landscape of problems is very useful in understanding the problem nature and is enlightening in developing algorithms. We believe that studying the fitness landscape of the CNNs architectures should be targeted in future work [61–63].

While the proposed algorithm shows interesting results, the performance of these algorithms can still be further improved if they are combined with other test systems. For example, there are algorithms that diagnose cases via blood tests, cough signals, or other ways of test systems. Combining the advantages of these algorithms via ensemble approaches can result in more accurate algorithms. This remains for future research.

While the proposed algorithm achieves better performance compared to the existing algorithms, one of its limitations is the time it requires to perform the evolutionary optimization and find the best architecture. In traditional approaches, the best architecture is found

via trial and error and with some expert knowledge about the field. This process can be applied quicker than the proposed algorithm, so if time is a matter and the CNN should be designed quicker, it is better to use expert knowledge to create the CNN. However, it should be noted that designing the CNN architecture is performed only once, and after that the resulting CNN can perform prediction as quickly as the other approaches.

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.

References

[1] Lawrence S, Giles CL, Tsoi AC, Back AD. Face recognition: a convolutional neural-network approach. IEEE Trans Neural Netw 1997;8(1):98–113.

[2] Hu B, Lu Z, Li H, Chen Q. Convolutional neural network architectures for matching natural language sentences. In: Ghahramani Z, Welling M, Cortes C, Lawrence ND, Weinberger KQ, editors. Advances in neural information processing systems 27. Curran Associates, Inc.; 2014, p. 2042–50.

[3] Ciresan DC, Meier U, Gambardella LM, Schmidhuber J. Convolutional neural network committees for handwritten character classification. In: 2011 International Conference on Document Analysis and Recognition. 2011, p. 1135–9.

[4] Liang M, Hu X. Recurrent convolutional neural network for object recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR, 2015.

[5] Yamashita R, Nishio M, Do RKG, Togashi K. Convolutional neural networks: an overview and application in radiology. Insights Into Imaging 2018;9(4):611–29.

[6] Sam DB, Surya S, Babu RV. Switching convolutional neural network for crowd counting. In: 2017 IEEE conference on computer vision and pattern recognition. CVPR, 2017, p. 4031–9.

[7] Yosinski J, Clune J, Bengio Y, Lipson H. How transferable are features in deep neural networks? In: Advances in neural information processing systems. 2014, p. 3320–8.

[8] Oquab M, Bottou L, Laptev I, Sivic J. Learning and transferring mid-level image representations using convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2014, p. 1717–24.

[9] Shin H-C, Roth HR, Gao M, Lu L, Xu Z, Nogues I, et al. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE Trans Med Imaging 2016;35(5):1285–98.

[10] Hubel DH, Wiesel TN. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J Physiol 1962;160(1):106.

[11] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature 1986;323(6088):533–6.

[12] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. 2014, arXiv preprint arXiv:1409.1556.

[13] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, p. 770–8.

[14] Albelwi S, Mahmood A. A framework for designing the architectures of deep convolutional neural networks. Entropy 2017;19(6):242.

[15] Bourlard H, Kamp Y. Auto-association by multilayer perceptrons and singular value decomposition. Biol Cybernet 1988;59(4–5):291–4.

[16] Hinton GE, Zemel RS. Autoencoders, minimum description length and Helmholtz free energy. In: Advances in neural information processing systems. 1994, p. 3–10.

[17] Bergstra J, Bengio Y. Random search for hyper-parameter optimization. J Mach Learn Res 2012;13(1):281–305.

[18] Rasmussen CE. Gaussian processes in machine learning. In: Summer school on machine learning. Springer; 2003, p. 63–71.

[19] Močkus J. On Bayesian methods for seeking the extremum. In: Optimization techniques IFIP technical conference. Springer; 1975, p. 400–4.

[20] Bergstra JS, Bardenet R, Bengio Y, Kégl B. Algorithms for hyper-parameter optimization. In: Advances in neural information processing systems. 2011, p. 2546–54.

[21] Zoph B, Le QV. Neural architecture search with reinforcement learning. 2016, CoRR arXiv:1611.01578.

[22] Hutter F, Hoos HH, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In: International conference on learning and intelligent optimization. Springer; 2011, p. 507–23.

[23] Zoph B, Vasudevan V, Shlens J, Le QV. Learning transferable architectures for scalable image recognition. In: 2018 IEEE/CVF conference on computer vision and pattern recognition. 2018, p. 8697–710.

[24] Baker B, Gupta O, Naik N, Raskar R. Designing neural network architectures using reinforcement learning. 2016, arXiv preprint arXiv:1611.02167.

[25] Chen Y, Zhu K, Zhu L, He X, Ghamisi P, Benediktsson JA. Automatic design of convolutional neural network for hyperspectral image classification. IEEE Trans Geosci Remote Sens 2019;57(9):7048–66.

[26] Liu H, Simonyan K, Yang Y. DARTS: Differentiable architecture search. 2018, CoRR arXiv:1806.09055.

[27] Yang T-J, Chen Y-H, Sze V. Designing energy-efficient convolutional neural networks using energy-aware pruning. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, p. 5687–95.

[28] Real E, Moore S, Selle A, Saxena S, Suematsu YL, Le QV, et al. Large-scale evolution of image classifiers. 2017, CoRR arXiv:1703.01041.

[29] Real E, Aggarwal A, Huang Y, Le QV. Regularized evolution for image classifier architecture search. 2018, CoRR arXiv:1802.01548.

[30] Qolomany B, Maabreh M, Al-Fuqaha A, Gupta A, Benhaddou D. Parameters optimization of deep learning models using particle swarm optimization. In: 2017 13th international wireless communications and mobile computing conference. IWCMC, IEEE; 2017, p. 1285–90.

[31] Young SR, Rose DC, Karnowski TP, Lim S-H, Patton RM. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: Proceedings of the workshop on machine learning in high-performance computing environments. 2015, p. 1–5.

[32] Strumberger I, Tuba E, Bacanin N, Jovanovic R, Tuba M. Convolutional neural network architecture design by the tree growth algorithm framework. In: 2019 International joint conference on neural networks. IJCNN, IEEE; 2019, p. 1–8.

[33] Baldominos A, Saez Y, Isasi P. Evolutionary convolutional neural networks: An application to handwriting recognition. Neurocomputing 2018;283:38–52.

[34] Suganuma M, Shirakawa S, Nagao T. A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the genetic and evolutionary computation conference. 2017, p. 497–504.

[35] Real E, Moore S, Selle A, Saxena S, Suematsu YL, Tan J, et al. Large-scale evolution of image classifiers. 2017, arXiv preprint arXiv:1703.01041.

[36] Xie L, Yuille A. Genetic cnn. In: Proceedings of the IEEE international conference on computer vision. 2017, p. 1379–88.

[37] Real E, Aggarwal A, Huang Y, Le QV. Regularized evolution for image classifier architecture search. In: Proceedings of the Aaai conference on artificial intelligence. 33, (01):2019, p. 4780–9.

[38] Liu H, Simonyan K, Vinyals O, Fernando C, Kavukcuoglu K. Hierarchical representations for efficient architecture search. 2017, arXiv preprint arXiv:1711.00436.

[39] Junior FEF, Yen GG. Particle swarm optimization of deep neural networks architectures for image classification. Swarm Evol Comput 2019;49:62–74.

[40] Ma B, Li X, Xia Y, Zhang Y. Autonomous deep learning: a genetic DCNN designer for image classification. Neurocomputing 2020;379:152–61.

[41] Gottapu RD, Dagli CH. Efficient architecture search for deep neural networks. Procedia Comput Sci 2020;168:19–25.

[42] Sun Y, Xue B, Zhang M, Yen GG. A particle swarm optimization-based flexible convolutional autoencoder for image classification. IEEE Trans Neural Netw Learn Syst 2018;30(8):2295–309.

[43] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010, p. 249–56.

[44] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. 2012, p. 1097–105.

[45] Zeiler MD, Fergus R. Visualizing and understanding convolutional networks. In: European conference on computer vision. Springer; 2014, p. 818–33.

[46] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015, p. 1–9.

[47] Cohen JP, Morrison P, Dao L. COVID-19 image data collection, GitHub. 2020, arXiv:2003.11597 URL: https://github.com/ieee8023/covid-chestxray-dataset.

[48] Rahman T, Chowdhury M, Khandakar A. COVID-19 radiography database, Kaggle. 2020, URL: https://www.kaggle.com/tawsifurrahman/covid19-radiography-database/data.

[49] Ahmed A. Pneumonia sample X-Rays, GitHub. 2020, URL: https://www.kaggle.com/ahmedali2019/pneumonia-sample-xrays.

[50] Polesel A, Ramponi G, Mathews VJ. Image enhancement via adaptive unsharp masking. IEEE Trans Image Process 2000;9(3):505–10.

[51] Gingold Y. Image stack: simple code to load and process image stacks, GitHub. 2014, URL: https://github.com/yig/imagestack.

[52] Rifai S, Vincent P, Muller X, Glorot X, Bengio Y. Contractive auto-encoders: Explicit invariance during feature extraction. In: Icml. 2011.

[53] Sohn K, Lee H. Learning invariant representations with local transformations. 2012, arXiv:1206.6418.

[54] Bruna J, Mallat S. Invariant scattering Convolution Networks. IEEE Trans Pattern Anal Mach Intell 2013;35(8):1872–86.

[55] Chan T, Jia K, Gao S, Lu J, Zeng Z, Ma Y. PCANet: A simple deep learning baseline for image classification? IEEE Trans Image Process 2015;24(12):5017–32. http://dx.doi.org/10.1109/TIP.2015.2475625.

[56] Larochelle H, Erhan D, Courville A, Bergstra J, Bengio Y. An empirical evaluation of deep architectures on problems with many factors of variation. In: Proceedings of the 24th international conference on machine learning. 2007, p. 473–80.

[57] Iandola FN, Moskewicz MW, Ashraf K, Han S, Dally WJ, Keutzer K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1 MB model size. 2016, CoRR arXiv:1602.07360.
[58] Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C. MobileNetV2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR, 2018.
[59] Hinton GE. A practical guide to training restricted Boltzmann machines. In: Neural networks: Tricks of the trade. Springer; 2012, p. 599–619.
[60] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016, arXiv preprint arXiv:1603.04467.
[61] Tayarani-N. M, Prügel-Bennett A. On the landscape of combinatorial optimization problems. IEEE Trans Evol Comput 2014;18(3):420–34.
[62] Tayarani-N. M-H, Prügel-Bennett A. Anatomy of the fitness landscape for dense graph-colouring problem. Swarm Evol Comput 2015;22:47–65.
[63] Prugel-Bennett A, Tayarani-Najaran M. Maximum satisfiability: Anatomy of the fitness landscape for a hard combinatorial optimization problem. IEEE Trans Evol Comput 2012;16(3):319–38. http://dx.doi.org/10.1109/TEVC.2011.2163638.

Mohammad- H. Tayarani- N. received his Ph.D. degree from the University of Southampton, Southampton, U.K, in 2013. Then he worked as research fellow at the University of Birmingham, Birmingham, UK and University of Glasgow, Glasgow, UK. He is currently a senior lecturer at the University of Hertfordshire, Hatfield, UK.