ISSN 2007-9737

Optimizing Deep Neural Networks with Differential Evolution for COVID-19 Diagnosis

Néstor U. Hernández-Cortez[1], Arturo Téllez-Velázquez[1], Prometeo Cortés-Antonio[2], Patricia Melin[2], Raúl Cruz-Barbosa[1,*]

[1] Universidad Tecnológica de la Mixteca, Mexico

[2] Tijuana Institute of Technology, Mexico

nestorurielhc@gs.utm.mx, {prometeo.cortes, pmelin}@tectijuana.mx, {atellezv, rcruz}@mixteco.utm.mx

Abstract. Accurate detection of COVID-19 remains a significant challenge, particularly due to the inherent limitations in conventional diagnostic methods such as RT-PCR testing. This study introduces two deep learning-based classification strategies using the COVIDGR chest X-ray data set. The first approach leverages the ResNeXt50 architecture, enhanced through training hyperparameter optimization via the Differential Evolution algorithm. The second strategy also employs DE to design custom CNN architectures based on the Multi-scale Feature Learning model, optimizing structural hyperparameters. Experimental results show that both methods outperform their non-optimized counterparts as well as several state-of-the-art approaches, achieving an overall performance of 90.32%. These findings highlight the potential of DE as a powerful tool for improving automated diagnosis of respiratory diseases.

Keywords. Deep learning, differential evolution, COVID-19, COVIDGR chest X-ray data set.

1 Introduction

COVID-19 is caused by the Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2), a member of the coronavirus family. Its rapid spread is mainly attributed to a short incubation period, estimated between 1 and 14 days, with an average of 5.6 to 6.7 days [22]. Infected individuals become the primary source of transmission [2].

The disease primarily deteriorates the respiratory system and can also affect the cardiovascular system, increasing the risk of severe complications and death in patients with preexisting conditions [18]. By April, 2024, confirmed global cases exceeded 704 million and deaths exceeded 7 million, making COVID-19 one of the most significant pandemics in recent centuries [30].

The most common detection method is the Real-Time Polymerase Chain Reaction (RT-PCR) test, which, despite its reliability, presents some limitations [5]. Therefore, improving detection methods is essential for early diagnosis and containment. Computer-aided diagnostic systems can help experts by providing a second opinion and reducing uncertainty.

Although several years have passed since the emergence of COVID-19, its characterization remains challenging and the development of accurate classification systems is still under investigation.

Medical imaging, particularly chest X-ray (XR) and computed tomography (CT) images, plays a crucial role in these systems. CT images provide detailed information that can improve RT-PCR sensitivity, but are costly and require specialized equipment. On the other hand, XR images are more accessible and cost-effective, making them suitable for Computer-Assisted Diagnosis (CAD) [19]. These images reveal characteristic COVID-19

patterns such as consolidation, linear opacity, and ground-glass opacity [4].

CAD systems offer rapid and reliable support to radiologists [12]. In this sense, the COVID-19 classification has been approached using traditional methods and deep learning techniques [15]. Although traditional classifiers [10] and hybrid approaches combining them with Convolutional Neural Networks (CNN) [9] offer certain improvements, they remain less effective compared to fully deep approaches [3, 32].

CNN-based strategies have shown outstanding performance in image classification tasks [14], particularly when leveraging pre-trained models through Transfer Learning (TL) [8, 21, 17, 7, 13].

In addition, custom architectures such as DeTraC [1] and COVID-Net [29] have been introduced to address the specific challenges of COVID-19 detection. However, their success depends on the selection of optimal hyperparameters, which is complex and often requires automated optimization techniques [20]. This has led to the adoption of metaheuristic optimization techniques, resulting in notable performance gains [24, 23, 6].

For this reason, this study aims to optimize the training and structural hyperparameters of advanced CNN architectures, specifically ResNeXt50 and MFL_Net, using the Differential Evolution algorithm for the classification of COVID-19 from chest XR images.

The experiments use the COVIDGR dataset, notable for its selective construction by four experts and its mitigation of common biases in traditional datasets. Two optimization strategies are implemented: one that focuses on training hyperparameters for ResNeXt50 and another that emphasizes structural hyperparameters through a multi-scale block approach in MFL_Net.

Both optimized models are evaluated to assess their effectiveness in COVID-19 detection.

This paper is structured as follows: Section 2 presents the theoretical foundations that support this research. Section 3 describes the methodology employed, detailing how the Differential Evolution strategy is applied to optimize two innovative deep learning models. Section 4 outlines the experimental setup, specifies the conditions under which the experiments were conducted, and presents the results obtained after implementing the proposed methodology. Finally, Section 5 provides the conclusions obtained and discusses potential directions for future research.

2 Theoretical Background

This section reviews the foundational concepts and architectures relevant to the proposed approach.

First, it examines advanced CNN designs, such as ResNeXt, which builds upon ResNet by introducing cardinality as a key dimension to improve accuracy without increasing computational cost. Next, it explores lightweight models like the Multi-Scale Feature Learning Network, which takes advantage of the multi-scale feature extraction to enhance performance in medical imaging tasks. Finally, it outlines the Differential Evolution algorithm, an evolutionary optimization technique widely used for hyperparameter tuning due to its simplicity and effectiveness in high-dimensional search spaces. Together, these elements provide the theoretical basis for the integration of the evolutionary strategy proposed in this paper.

2.1 The ResNeXt

Residual Neural Network (ResNet) models are built by stacking blocks with the same topology, which promotes reusing of repetitive structures and significantly reduces the number of hyperparameters.

This feature also helps mitigate overfitting to a specific data set.

On the other hand, the ResNeXt architecture is built based on the residual blocks of ResNet to take advantage of the reduction in hyperparameters, incorporating modifications that improve classification performance without increasing computational complexity [31].

ResNeXt models adopt the principle of stacking residual blocks with the same topology and introduce two fundamental rules for their construction:

— When generating spatial maps of the same size, the blocks share the same hyperparameters (width and filter size).

Table 1. ResNet50 model architecture compared to ResNeXt50 model architecture [31]

Layer	Output	ResNet50		ResNeXt50_32x4d	
conv1	112×112	7×7, 64, stride 2		7×7, 64, stride 2	
conv2	56×56	3×3 MaxPool, stride 2		3×3 MaxPool, stride 2	
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}$	×3	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C=32 \\ 1\times1, 256 \end{bmatrix}$	×3
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}$	×4	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C=32 \\ 1\times1, 512 \end{bmatrix}$	×4
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}$	×6	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C=32 \\ 1\times1, 1024 \end{bmatrix}$	×6
conv5	7×7	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024 \\ 1\times1, 2048 \end{bmatrix}$	×3	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C=32 \\ 1\times1, 2048 \end{bmatrix}$	×3
	1×1	GlobalAveragePooling, 1000-d fc, softmax		GlobalAveragePooling, 1000-d fc, softmax	
# parameters		25.5×10^6		25.0×10^6	
FLOPs		4.1×10^9		4.2×10^9	

— Whenever the spatial map sample is reduced by a factor of two, the width of the blocks is doubled.

The first rule guarantees that the computational complexity and the number of hyperparameters remain stable, while the second ensures that the complexity, in terms of Floating-Point Operations (FLOPs), is similar across network blocks.

These blocks are also inspired by the *Inception* modules [27], which divide the input into embeddings using 1×1 convolutional layers. Each embedding is transformed with specialized convolutional layers of varying filter sizes and then merged by concatenation to produce a single output. This approach, known as *divide-transform-merge*, yields representations comparable to those of large, dense layers, but with lower computational complexity. However, the use of specialized layers significantly increases the number of hyperparameters and can make it difficult to adapt these architectures to tasks other than those for which they were designed.

The operation of these blocks is described in three stages:

1. Division: The block input is divided into a cardinality of low-dimensional embeddings using 1×1 convolutional layers. The dimension d of each embedding is calculated by dividing the original dimension of the convolutional layer by the cardinality of the block.

2. Transformation: Each embedding is subject to a transformation defined as a *bottleneck* block, where the first 1×1 layer generates the embedding and the last layer restores the original input dimension.

3. Aggregation: All embeddings are combined to produce a single output. This process can be interpreted as a modified version of the inner product of an artificial neuron, where the cardinality represents the number of inputs and each embedding acts as a transformed input. The operation is formalized in Equation 1 by the following residual function:

$$F(x) = \sum_{i=1}^{C} \tau_i(x), \tag{1}$$

where C is the cardinality, i denotes each embedding, and $\tau_i(x)$ represents the individual transformation of each input. By incorporating the identity mapping connection, the residual function is expressed as follows (see Equation 2):

$$y = x + \sum_{i=1}^{C} \tau_i(x), \tag{2}$$

where x is the input to the block and y is the output.

Table 1 compares the ResNeXt50 architecture with ResNet50, showing that the number of parameters and FLOPs remain similar.

The ResNeXt models have demonstrated that cardinality is a dimension as important as the width and depth of the network.

In fact, increasing the cardinality is a more effective strategy for improving accuracy than increasing depth or width, especially when these dimensions begin to cause performance issues.

2.2 The Multi-Scale Feature Learning Network

The Multi-Scale Feature Learning Network (MFL_Net) is a shallow and significantly lightweight CNN model implemented for COVID-19 classification problems in CT images. Being a small model, it benefits from a considerably low number of parameters and a reduced training time. Furthermore, it proves to be an efficient model thanks to the implementation of multi-scale learning [11].

CNN models that implement a linear stacking structure for the connection of their convolutional layers typically focus on feature extraction on a homogeneous scale. However, feature extraction at different scales has proven to be highly effective in classification tasks [16].

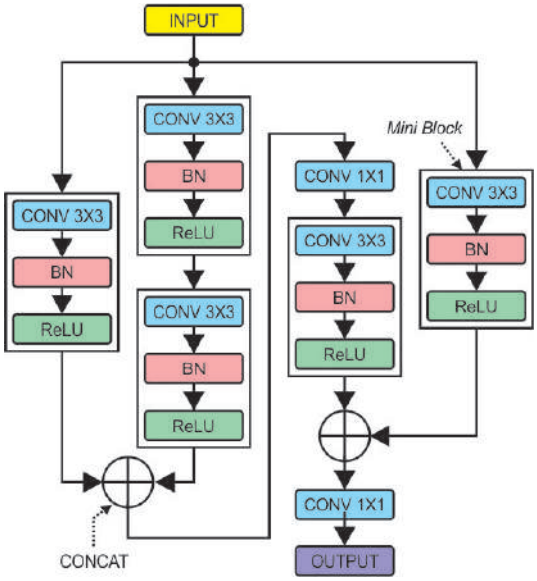

Fig. 1. MFL Blocks

2.2.1 MFL Blocks

The key design of MFL networks lies in the Multi-Scale Feature Learning (MFL) blocks, which work as a feature extraction module at different scales. These blocks aim to learn features at different levels of the receptive field. To do this, they implement the structure shown in Fig. 1.

The basic unit of MFL blocks are the so-called *Mini Block* structures, which consist of a sequence of a convolutional layer (CONV 3×3), a normalization layer (BN), and an activation layer (ReLU). MFL blocks are responsible for extracting features at two different scales simultaneously, which are:

— *Small scale*. It is obtained using 3×3 filters. These filters extract detailed features and are responsible for obtaining COVID-19 disease characteristics, such as ground-glass opacity patches and linear opacities.

— *Large scale*. It is represented by filters of size 5×5 and 7×7, which are responsible for obtaining general features such as the shape of the lungs.

One problem with using multi-sized filters is that the number of parameters increases significantly and proportionally as the filter size increases.

To address this, MFL blocks propose simulating large-scale convolutional layers by sequentially combining several *Mini Blocks*. Specifically, two Mini Blocks are used sequentially to simulate a 5×5 convolutional layer and three for a 7×7 convolutional layer; the resulting architecture is shown in Fig. 1. This implementation optimizes network performance by significantly reducing the number of parameters.

2.3 The Differential Evolution Algorithm

The Differential Evolution (DE) algorithm is an evolutionary algorithm that performs a multidimensional search through mutation and genetic recombination [26]. Similarly to other evolutionary approaches, DE operates on a population of candidate solutions (individuals), where each individual is represented as a vector that matches the dimensionality of the problem. Each gene within these vectors is initialized using a random number generator constrained to a predefined search range, as illustrated in Equation 3:

$$x_{i,t}^{G=1} = Random_t(LB_t, UB_t), \tag{3}$$

where $G = 1$ is the initial generation, t is the $t-$th chromosome of the $i-$th individual, LB_t and UB_t are the lower and upper bounds of the search range, respectively, and $Random_t(a, b)$ denotes a random number generator.

One of the main peculiarities of the DE algorithm is the way in which reproduction is performed.

Primarily, this algorithm does not use parental selection mechanisms; that is, each individual has the same opportunities to reproduce and generate offspring. For this purpose, each individual in the current population becomes a parent and is known as the primary parent. For each primary parent, three or more auxiliary parents are randomly selected to generate the mutated individual. To do this, the scalar difference between two of the auxiliary parents is calculated and added to the third. This operation is known as *differential mutation* and this is where this algorithm gets its name. The operation can be represented in Equation 4:

$$\overrightarrow{v}_i^G = \overrightarrow{r}_1 + F \times (\overrightarrow{r}_2 - \overrightarrow{r}_3), \tag{4}$$

where F is a positive real number known as the *amplification factor*, $\overrightarrow{r}_1$, $\overrightarrow{r}_2$, and $\overrightarrow{r}_3$ are the randomly selected *auxiliary* parents, and $\overrightarrow{v}_i^G$ is the resulting *mutated individual*.

The DE algorithm employs a crossover operator known as *discrete recombination*, where the primary parent is combined with a mutated individual to produce an *offspring*. This offspring is formed by randomly selecting genes from either parent. The gene selection process for the construction of the offspring is governed by Equation 5:

$$u_{i,t}^G = \begin{cases} v_{i,t}^G & [r(t) \leq CR] \vee t = rn(i), \\ x_{i,t}^G & else, \end{cases} \tag{5}$$

where each gene $u_{i,t}^G$ of the new individual $\overrightarrow{u}_i^G$ is obtained from $x_{i,t}^G$ and $v_{i,t}^G$ according to the *Crossover Rate* given by CR, and a randomly generated number $r(t)$, to ensure that at least one gene at index $rn(i)$ of the individual $v_{i,t}^G$ is inherited to the new individual.

Once an offspring has been generated, the DE algorithm proceeds to select the individuals for the next generation. This is done by conducting a one-to-one competition between each parent $\overrightarrow{x}_i^G$ and its corresponding offspring $\overrightarrow{u}_i^G$. The individual with lower fitness is discarded, ensuring that only the fitter of the two survives, as defined by Equation 6:

$$\overrightarrow{x}_i^{G+1} = \begin{cases} \overrightarrow{u}_i^G & f(\overrightarrow{u}_i^G) \leq f(\overrightarrow{x}_i^G), \\ \overrightarrow{x}_i^G & else, \end{cases} \tag{6}$$

where $\overrightarrow{x}_i^{G+1}$ is the individual who will be part of the next generation and $f(\cdots)$ denotes the score obtained by the individual after applying the objective function.

3 Methodology

This research is structured in three main stages:

— *Preprocessing*. This stage focuses on processing the COVIDGR data set. This process ensures that the data are suitable for subsequent analysis. Before starting with the preprocessing stage, the COVIDGR data set is partitioned as follows: 60% for the training set, 20% for the validation set, and 20% for the testing set. Once the data set is partitioned, the preprocessing is performed as follows:

1. For the training and validation sets.
   - Data augmentation. Data augmentation was chosen because of the limited number of images available in the COVIDGR data set. This technique generates new images from the originals by applying one or more transformations, thus increasing the number of elements per class when their size is small [25]. The implemented method follows the scheme proposed by Lopez-Betancur et al. [17], using a sequence of three transformations to generate the new images:
     * *RandomResizedCrop*: This involves performing a random crop on the image with variable area and aspect ratio, and then resizing the crop to a predetermined size.
     * *RandomHorizontalFlip*: This applies a horizontal flip to the image with a probability of 50%.
     * *RandomRotation*: This rotates the image by a random angle within a defined interval; in this work, this interval was set between 0° and 20°.

     Each input image is processed with this sequence of transformations, resulting in a single output image that can exhibit any possible combination of the above mentioned transformations. Fig. 2 shows examples of the effect of these transformations.

Fig. 2. Data augmentation examples

   - Resizing. In general terms, a CNN model requires that the input image have specific dimensions, which means that the height, width, and number of channels are predetermined. This makes image resizing an essential task in preprocessing. In this study, resizing is applied by reducing the dimensions of the image to 224×224 pixels. This choice is based on two main reasons: it is a widely used dimension in CNN architectures as it offers a suitable balance between visual quality and computational efficiency in convolution operations, and it corresponds to the input dimensions required by the architectures used in this research.
   - Normalization. The images in the COVIDGR data set are 8-bit and grayscale, so their intensity levels are represented by integer values in the range of 0 to 255. This scale differs significantly from that used by most deep learning models, which typically work with normalized values. For this reason, a normalization process was applied, adjusting the intensity range by multiplying each value by a factor of $1/255$.
2. For the testing set, the preprocessing applied is similar to that performed on the training and validation sets, with the only difference being that the data augmentation technique is not used. This is because the test set is used exclusively

for the final validation of the models and therefore does not require increasing its size or generating additional variations.

— *DE Optimization*. This stage proposes a DE strategy to adjust the hyperparameters of innovative CNN architectures. This optimization is applied to both training and structural hyperparameters, with the aim of maximizing performance in the subsequent classification task as follows:

  – Training hyperparameter optimization. In this stage, experiments are performed using the ResNeXt50_32x4d architecture, whose structure is detailed in Table 1. As is natural at this stage, the architecture remains fixed and only the training hyperparameters are modified: Learning Rate (LR), Batch Size (BS), and the number of epochs.
  – Structural hyperparameter optimization. On the other hand, the MFL_Net model [11] is used, which is organized into three main components:
    * Header: The MFL_Net structure uses the same header as the original MFL_Net version, consisting of an input layer for 224×224 pixel images. In addition, it includes an MFL *mini_block* and a *MaxPool2D* layer with kernel size of 2×2.
    * Network body: It consists of a specific number of MFL blocks, each with a defined number of filters, followed by a *MaxPool2D* layer of 2×2, except for the last block. Note that both the number of MFL blocks and the number of filters in each of those blocks are structural hyperparameters that should be optimized during this stage.
    * Classification: It is made up of a *GlobalAveragePooling2D* layer, followed by a dropout layer with a probability of 50%, and finally a *Dense* output layer with two neurons and *softmax* activation.

    Note that during structural search, the Stochastic Gradient Descent (SGD) optimizer with an LR of 0.001 and a momentum of 0.9 is used.

— *COVID-19 Classification*. This stage uses the preprocessed data from Stage 1, specifically the preprocessed testing set, and the optimized hyperparameters of the models from Stage 2. This process seeks to evaluate the feasibility of the proposed method as a tool to support the diagnosis of COVID-19.

To ensure satisfactory performance in the DE optimization process, three different search types were carried out, each with specific objectives and configurations. These are described in detail below:

— *Coarse Search (CS)*. This experiment consists of exploring a broad search space for all hyperparameters to be optimized. The optimization is run for 6 iterations, with a population of 96 individuals for the training hyperparameter optimization and 15 individuals for the structure hyperparameter optimization. The purpose of this stage is to identify promising initial configurations that can serve as a reference for more refined searches.

— *Fine Search (FS)*. The Fine Search acts as a refinement, reducing the search space around the values obtained in the CS. In this way, the search space depends entirely on the previous results, allowing the process to focus on more relevant configurations. This experiment uses the same population size and number of iterations as CS, i.e. 96 individuals for training hyperparameter optimization and 15 individuals for structural hyperparameter optimization. The goal is to adjust the parameters more precisely to improve performance without significantly increasing computational cost.

— *Extensive Search (ES)*. This experiment is independent of the previous ones and is performed on the same search space as CS, but incorporating a greater diversity of information. To achieve this, the number of iterations is increased to 10, and the population is expanded to 330 individuals for

the training hyperparameter optimization and 30 individuals for the structural hyperparameter optimization. It should be noted that, due to the high computational cost involved in evaluating complex models, an even larger population was not selected. The objective of this stage is to exhaustively explore the search space to identify optimal configurations that were not reached in previous experiments.

4 Results

Once the methodology used in this study was defined, the experimental setup was established to evaluate the performance of the proposed architectures. This setup included describing the data set, defining the software and hardware used, and applying DE optimization strategies.

Subsequently, the results obtained were analyzed, considering some of the main performance metrics for the classification task. This analysis aimed not only to determine the behavior of the models with the COVIDGR data set but also to discuss the viability of the proposal as a complementary tool in the diagnosis of COVID-19.

4.1 Experimental Settings

The experiments described in this study were developed using only open source libraries.

The different CNN architectures, as well as the training and evaluation processes, were implemented using the Keras and TensorFlow libraries. On the other hand, the OpenCV library was used to perform image pre-processing. In addition, the Imbalanced-learn (Imblearn) and Scikit-learn (Sklearn) libraries were used to visualize the results.

In terms of the execution environment, all experiments were performed on a server with the following hardware specifications: Intel(R) Xeon(R) E5-2630 32-core processor at 2.40 GHz, Ubuntu 18.04 LTS operating system, 120 GB solid state drive (SSD), 1 TB hard drive, 16 GB DDR4 RAM, and an NVIDIA Tesla K40c graphics processing unit.

Data set

Traditional COVID-19 data sets have several limitations, including bias toward more severe cases, the presence of labels and marks on images both within and outside the lung region, and a lack of data homogeneity.

To mitigate these drawbacks and improve the acceptability of the generated models, the COVIDGR data set [28] was chosen. This data set is characterized by its high quality, having been compiled exclusively from posteroanterior chest radiographs obtained at the San Cecilio University Clinical Hospital in Granada, Spain. The images do not contain marks or labels near the lung region, which contributes to cleanliness.

The images were evaluated by four specialists and a case was classified as positive for COVID-19 only when both the RT-PCR test and the radiologist's diagnosis coincided. Regarding its distribution, the set contains a total of 852 images, divided equally into two classes: 426 corresponding to normal cases and 426 to positive cases of COVID-19. The latter are categorized according to the severity level of the disease: severe, moderate, mild, and a subclass called Normal-PCR+, which includes images with diagnostic discrepancy, that is, those in which the radiologist does not detect signs of the disease, but the RT-PCR test yields a positive result. For this reason, elements of this subclass are not used in this study.

4.2 Analysis and Discussion

This section presents the results obtained after applying the methodology described in Stage 2 of the methodology (see Section 3). First, we present the results for a set of residual deep neural network architectures, evaluated without prior optimization.

Subsequently, the training hyperparameters are optimized using the DE algorithm, based on the ResNeXt50 architecture.

Finally, given that MFL_Net was originally designed to assist in COVID-19 diagnosis and is an exceptionally lightweight network with low computational and memory requirements compared to many CNN architectures pre-trained on ImageNet, this architecture was selected for structural hyperparameter optimization also using the DE algorithm.

Table 2. Classification results for non-optimized deep architectures using the COVIDGR data set. The suffix *TL* indicates pretraining with TL. The symbol '–' indicates that the information is not available

CNN model	ACC	PREC	SENS	SPEC	F1-score	IBA
MFL_Net [11]	72.07	72.19	72.25	70.85	72.07	50.74
ResNet50	69.03	70.20	69.03	70.21	69.05	48.04
ResNeXt50_32x4d	73.55	74.16	73.54	74.18	73.60	54.41
ResNet50 TL	82.58	82.56	82.58	81.87	82.54	67.52
ResNeXt50_32x4d TL	87.10	87.22	87.09	86.06	87.02	74.79
COVID-SDNet [28]	81.00	84.23	76.80	–	80.07	–

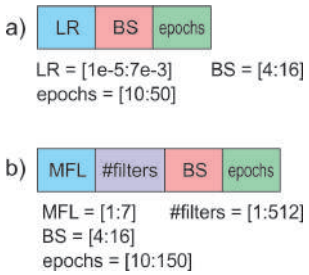

Fig. 3. Chromosomes for a) training hyperparameters and b) structural hyperparameters

4.2.1 Deep Architectures Without Optimization

To obtain fair comparisons between the results, residual models with similar depths and characteristics were selected. Specifically, the ResNet50 architecture was used for the ResNet models and the ResNeXt50_32x4d architecture for the ResNeXt models. For the MFL_Net model, the architecture proposed by Joshi and Nayak [11] was used.

The TL technique was applied to almost all the implemented architectures, which involves initializing the weights of each model using pre-trained weights. For this study, pre-trained weights were used from the ImageNet data set [14].

The architectures were trained using the same hyperparameters as adjusted in the work of Lopez-Betancur et al. [17], which include: a learning rate of 0.001, 50 training epochs, the SGD optimizer with a momentum of 0.9, and a batch size of 16.

For residual models, a performance comparison was performed between versions with and without TL. In the case of the MFL_Net network, the Adam optimizer was used, as specified in its original architecture.

The results obtained in this experiment are presented in Table 2. When considering only models that do not use TL, the performance of the classification task is comparable between the MFL_Net and ResNeXt50_32x4d models, the latter being slightly superior, showing a maximum difference of up to 2% in the evaluation metrics.

However, the overall performance of these models remains below 75%, highlighting the high complexity of classification in the COVIDGR data set.

The use of TL proves to be a determining factor in improving performance, thanks to the incorporation of pre-trained weights. This technique provides a significant increase in the performance of residual models. In contrast, because the MFL_Net architecture does not benefit from TL, it does not improve its performance.

The comparison allows us to conclude that the ResNeXt50_32x4d model is the best fit for the classification problem presented with this data set.

This is confirmed by observing that this model achieves the highest performance with and without TL, reaching values close to 87% in the evaluation metrics when the latter is used.

Furthermore, it should be noted that the performance of the ResNeXt50_32x4d model with TL even exceeds that of the COVID-SDNet model, whose results were reported in the original study of the COVIDGR data set [28]. However, the ResNet50 model is positioned as the least effective, exhibiting the lowest overall performance and only achieving acceptable results through the use of TL. This reinforces the relevance of TL, which allows residual models to outperform even the model originally proposed for this data set.

4.2.2 Training Hyperparameter Optimization

After establishing baseline performance for state-of-the-art comparison, the next step involves optimizing the training hyperparameters to enhance the accuracy of the models. This optimization is carried out using the DE algorithm, applied to the ResNeXt50_32x4d architecture. This architecture was selected due to its superior performance in initial evaluations, as evidenced in Table 2, making it the most suitable candidate for further refinement.

ISSN 2007-9737

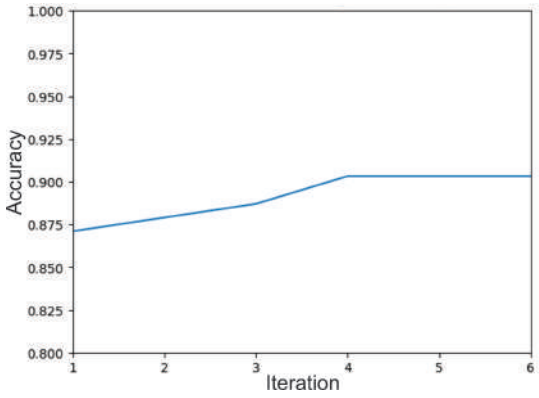

Fig. 4. ResNeXt50_32x4d evolution during CS

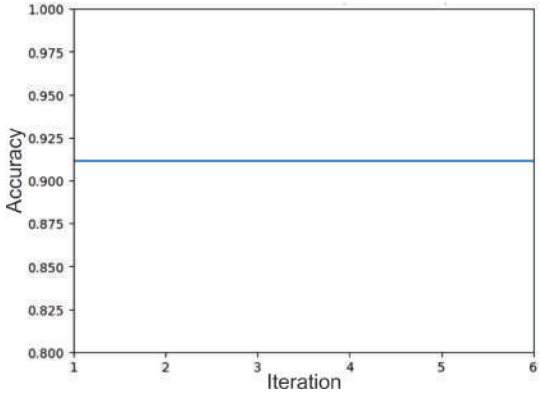

Fig. 5. ResNeXt50_32x4d evolution during FS

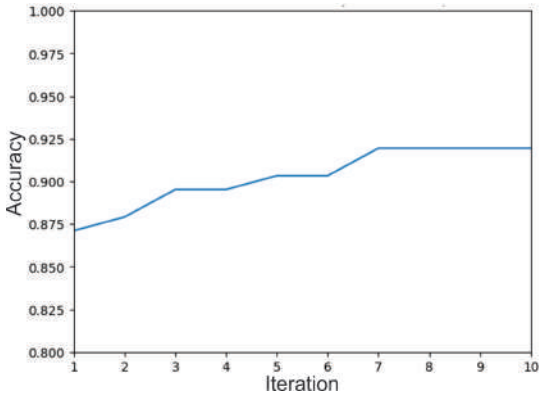

Fig. 6. ResNeXt50_32x4d evolution during ES

For training hyperparameter optimization, CS, FS, and ES were performed using the chromosome shown in Fig. 3a, considering the intervals defined for each of the categories.

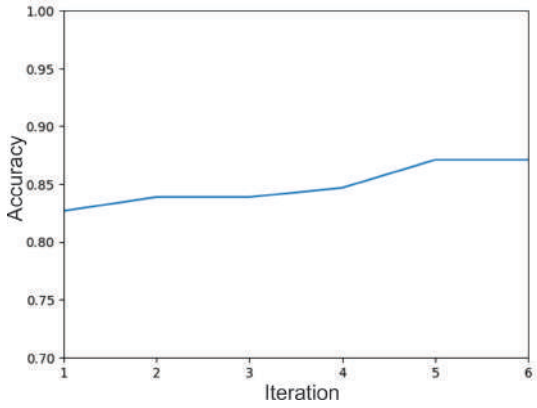

Fig. 7. MFL_Net evolution during CS

Table 3. Summary of the optimized values of the training hyperparameters of the ResNeXt50_32x4d network during CS, FS, and ES

Search type	LR	BS	#Epochs	ACC
CS	0.004	16	20	90.32 %
FS	0.003163	16	22	91.12 %
ES	0.007	16	17	91.93 %

In CS, the total execution time of the optimization process was 16 hours and 15 minutes, achieving a maximum accuracy of 90.32% in the validation set. Fig. 4 illustrates the evolution of performance during optimization for this network. It can be seen that, during the first iteration, an individual was obtained with an accuracy of 87.10%.

In the following three iterations, a progressive improvement in performance is observed, although with moderate increases: 87.90% in the second iteration and 88.70% in the third. The greatest increase occurs in the fourth iteration, where the accuracy reaches 90.32%, which is the best result obtained during CS. In the remaining two iterations, this performance remains without variations. The founded hyperparameters for CS are shown in the second raw of Table 3.

In the case of FS, the weights of each individual were initialized using the adjusted weights during CS. The total optimization run time was 22 hours and 1 minute, achieving a maximum performance of

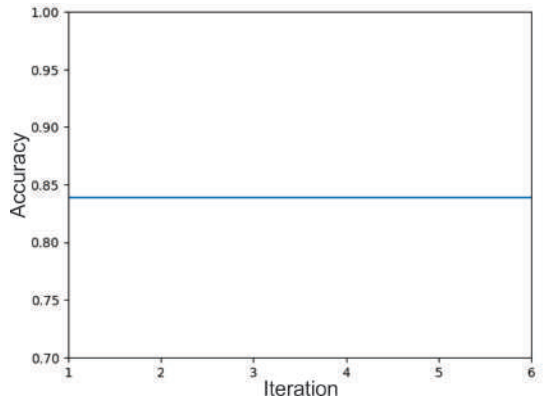

Fig. 8. MFL_Net evolution during FS

91.12% in the validation set. The hyperparameters associated with this result are presented in the third row of Table 3, where an adjustment in the LR values and the number of epochs can be observed, reflecting a slight improvement in classification performance.

As shown in Figure 5, initializing the individuals with the weights obtained in CS allows optimization to begin with high results, achieving an individual with an accuracy of 91.12% from the first iteration.

However, this value is not surpassed in subsequent iterations, as evidenced in the graph by the constant behavior throughout the execution.

Unlike the two previous searches, ES has a considerably longer execution time. This is because it uses a larger population and the algorithm runs for 10 iterations. The total execution time was 66 hours and 27 minutes, achieving a maximum accuracy of 91.93% in the validation set.

Fig. 6 shows that the process starts at the same point as CS, achieving an accuracy of 87.10% in the first iteration. Throughout the iterations, the performance shows a steady improvement: 87.90% in the second, 89.51% in the third, 90.32% in the fifth and a maximum of 91.93% in the seventh iteration. This value represents the best result achieved so far, as no further improvements were recorded in the remaining iterations.

Table 4. Optimized MFL_Net structure during CS

#Epochs	98	
BS	10	
Accuracy	87.09 %	
Layer name	Description	Output
Mini_Block	*mini_block*(16, 3×3)	224×224
Pooling	*MaxPool*(2×2)	112×112
BMFL_1	*MFL_Block*(8, 3×3)	112×112
Pooling_1	*MaxPool*(2×2)	56×56
BMFL_2	*MFL_Block*(128, 3×3)	56×56
Pooling_2	*MaxPool*(2×2)	28×28
BMFL_3	*MFL_Block*(324, 3×3)	28×28
Pooling_3	*MaxPool*(2×2)	14×14
BMFL_4	*MFL_Block*(222, 3×3)	14×14
GlobalAveragePooling		1×222
Dropout_1	*Dropout*(0.5)	1×222
Out	*Dense*(2, softmax)	1×2

4.2.3 Structure Hyperparameter Optimization

Similarly to the optimization of the ResNeXt50_32x4d training hyperparameters, the next stage focuses on optimizing the structural hyperparameters of the second architecture proposed in this study: MFL_Net. This step aims to refine the design of the network to improve performance while maintaining its lightweight nature as much as possible, using the same optimization strategy that was applied previously. In this case, optimization is performed with the aim of adjusting the number of MFL blocks, the number of filters assigned to each block, the BS, and the number of epochs. In this case, CS, FS, and ES were performed using the chromosome shown in Fig. 3b.

During CS, the total duration of the experiment was 120 hours and 57 minutes, significantly longer than the experiments described in Section 4.2.2. Fig. 7 shows its optimization behavior.

The results indicate that the process achieved a maximum accuracy of 87.09% in the validation set. During optimization, three improvement points were identified: the first in the second iteration, with an accuracy of 83.87%; the second, with a slight improvement to 84.67%; and finally, the most significant improvement in the fifth iteration, where the maximum performance was obtained.

The optimal model founded using CS is presented in Table 4. The network body comprises four MFL blocks with 8, 128, 324, and 222 filters, respectively, each followed by a *MaxPool2D*(2×2) layer, except

Table 5. Optimized MFL_Net structure during FS

#Epochs	88	
BS	15	
Accuracy	83.87 %	
Layer name	Description	Output
Mini_Block	*mini_block*(16, 3×3)	224×224
Pooling	*MaxPool*(2×2)	112×112
BMFL_1	*MFL_Block*(15, 3×3)	112×112
Pooling_1	*MaxPool*(2×2)	56×56
BMFL_2	*MFL_Block*(123, 3×3)	56×56
Pooling_2	*MaxPool*(2×2)	28×28
BMFL_3	*MFL_Block*(341, 3×3)	28×28
GlobalAveragePooling		1×341
Dropout_1	*Dropout*(0.5)	1×341
Out	*Dense*(2, softmax)	1×2

Table 6. Optimized MFL_Net structure during ES

#Epochs	130	
BS	6	
Accuracy	87.90 %	
Layer name	Description	Output
Mini_Block	*mini_block*(16, 3×3)	224×224
Pooling	*MaxPool*(2×2)	112×112
BMFL_1	*MFL_Block*(149, 3×3)	112×112
Pooling_1	*MaxPool*(2×2)	56×56
BMFL_2	*MFL_Block*(35, 3×3)	56×56
Pooling_2	*MaxPool*(2×2)	28×28
BMFL_3	*MFL_Block*(103, 3×3)	28×28
Pooling_3	*MaxPool*(2×2)	14×14
BMFL_4	*MFL_Block*(499, 3×3)	14×14
GlobalAveragePooling		1×499
Dropout_1	*Dropout*(0.5)	1×499
Out	*Dense*(2, softmax)	1×2

for the last block. For training, the model was tuned over 98 epochs, with a BS of 10.

In FS, optimization is performed on the basis of the results obtained during CS. However, it is not possible to reuse the tuned model in the previous stage to improve performance due to structural differences between the two architectures. However, the following search ranges for the hyperparameters were redefined:

- Number of MFL blocks: $[3 : 5]$
- Number of filters per block: $[1 : 58]$, $[78 : 178]$, $[274 : 374]$, $[172 : 272]$, and $[1 : 512]$, considering that the model obtained in CS has four MFL blocks, so the fifth block maintains the original boundaries
- Batch size (BS): $[5 : 15]$
- Number of epochs: $[88 : 108]$

In this case, the total execution time was 32 hours and 36 minutes, representing a significant reduction compared to CS. This decrease is due to the search space constraint, which rules out more complex and time-intensive architectures that do not offer substantial performance improvements.

Regarding optimization, Fig. 8 shows a consistent behavior. The optimization begins with a performance of 83.87%, higher than the starting point in CS; however, this value does not improve during the experiment, ending with a performance lower than that obtained in the previous stage.

The optimal model founded is presented in Table 5. The main difference between the CS and FS structures lies in the body of the network, which includes only three MFL blocks with 15, 123, and 341 filters (size 3×3), each accompanied by a *MaxPool2D*(2×2) layer, except for the last block.

Training was carried out over 88 epochs, with a BS of 15. Overall, the trained network exhibits fewer epochs, a larger BS, and fewer MFL blocks than the model obtained in CS. These features make the structure lighter, with fewer parameters and faster training; however, these advantages do not compensate for the significant reduction in performance.

The optimization of structural hyperparameters using ES achieves a total duration of the experiment of 267 hours and 57 minutes, significantly longer than any of the previous experiments. This increase is explained by the complexity of the MFL blocks and the large number of associated parameters, a consequence of working with the highest available filter values in the search space.

The optimization behavior is illustrated in Fig. 9. The optimization begins with a performance of 83.06%, approaching the best result obtained in FS from the beginning and slightly exceeding the initial point of CS, due to the increase in population size. In the second iteration, an accuracy of 84.67% is achieved, exceeding the FS performance. Subsequently, in the fourth iteration, another improvement is observed, reaching 86.29%, a value that is maintained for two consecutive

Table 7. Classification results for optimized deep networks using the final models of Stage 3 of the methodology. The symbol '–' indicates that the information is not available

CNN model	ACC	PREC	SENS	SPEC	F1-score	IBA
COVID-SDNet [28]	81.00	84.23	76.80	–	80.07	–
ResNeXt50_32x4d+DE	90.32	90.34	90.32	90.26	90.32	81.53
MFL_Net+DE	82.58	82.60	82.58	82.37	82.51	69.03

iterations. Finally, in the seventh iteration, the maximum performance obtained in this configuration is achieved, corresponding to 87.90%, slightly exceeding the CS result. The optimized structure during ES is shown in Table 6, where it presents similarities to the model obtained in CS, particularly in the number of MFL blocks in the network body. However, significant differences are observed in the number of filters assigned to each block, which are 149, 35, 103, and 499 filters (size 3×3). The model was trained over 130 epochs, with a BS of 6. The large number of filters contributes to the increased number of parameters, making this the heaviest structure obtained so far. Despite these characteristics, an increase in classification performance is achieved and the increase in the number of parameters does not represent a critical impact on the required computational resources.

4.2.4 Validation

This section corresponds to Stage 3 of the methodology (see Section 3) and presents the results obtained after validating the models generated in Stage 2. Specifically, the ResNeXt50_32x4d and MFL_Net architectures, both optimized using the DE algorithm, are evaluated.

For this validation, the training and validation sets are merged into a single training set and the final performance metrics are computed using the testing set. The objective of this evaluation is to analyze the final performance and assess the feasibility of the proposed approach for the COVID-19 classification.

The final classification results are presented in Table 7, which reflect the performance of the optimized models using the proposed strategy. These results show that the optimization of both training and structural hyperparameters yields

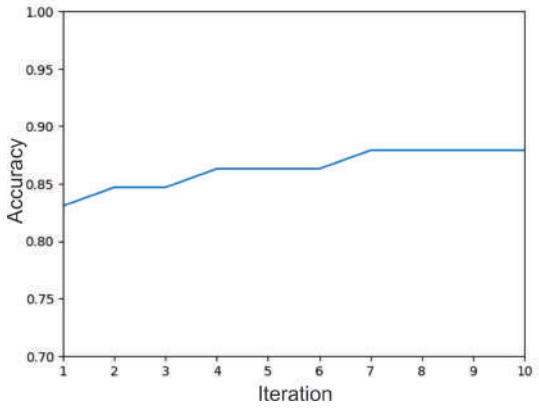

Fig. 9. MFL_Net evolution during ES

highly satisfactory results, showing a substantial improvement over the baseline model used in the COVIDGR data set.

Furthermore, the optimized models outperform other state-of-the-art approaches across nearly all performance metrics. This improvement becomes evident when comparing the values reported in Table 2, highlighting the positive impact of the proposed optimization strategy.

5 Conclusion and Future Work

In summary, this study demonstrates the effectiveness of metaheuristic optimization, specifically Differential Evolution, since it enhances the performance of deep neural networks for COVID-19 detection from X-ray images, i.e. the COVIDGR data set. By systematically tuning both training and structural hyperparameters in multiple search strategies, the optimized ResNeXt50_32x4d and MFL_Net models significantly outperformed established benchmarks.

Despite constraints in computational resources, the results emphasize the potential of automated hyperparameter optimization to reduce manual effort and accelerate model refinement.

Future work will focus on expanding the scope of optimization and exploring alternative metaheuristics, supported by more robust hardware to further advance diagnostic accuracy and efficiency.

Acknowledgements

This work was partially supported by the Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI). N. U. Hernández-Cortez also acknowledges SECIHTI for his MSc fellowship.

References

1. **Abbas, A., Abdelsamea, M. M., Gaber, M. M. (2021).** Classification of COVID-19 in chest X-ray images using DeTraC deep convolutional neural network. Appl. Intell., Vol. 51, No. 2, pp. 854–864.
2. **Calvo, C., López-Hortelano, M. G., de Carlos Vicente, J. C., Vázquez-Martínez, J. L., AEP (2020).** Recomendaciones sobre el manejo clínico de la infección por el nuevo coronavirus SARS-CoV2. Grupo de trabajo de la Asociación Española de Pediatría (AEP). An. Pediatr., Vol. 92, No. 4, pp. 1–11.
3. **Chandra, T. B., Verma, K., Singh, B. K., Jain, D., Netam, S. S. (2021).** Coronavirus disease (COVID-19) detection in chest X-ray images using majority voting based classifier ensemble. Expert Syst. Appl., Vol. 165, pp. 113909.
4. **Cleverley, J., Piper, J., Jones, M. M. (2020).** The role of chest radiography in confirming COVID-19 pneumonia. Br. Med. J., Vol. 370, pp. 1–9.
5. **Corman, V. M., Landt, O., Kaiser, M., Molenkamp, R., Meijer, A., Chu, D. K., Bleicker, T., Brünink, S., Schneider, J., Schmidt, M. L., Mulders, D. G., Haagmans, B. L., van-der Veer, B., van-den Brink, S., Wijsman, L., Goderski, G., Romette, J. L., Ellis, J., Zambon, M., Peiris, M., Goossens, H., Reusken, C., Koopmans, M. P., Drosten, C. (2020).** Detection of 2019 novel coronavirus (2019-nCoV) by real-time RT-PCR. Euro Surveill., Vol. 25, No. 3, pp. 1–8.
6. **Goel, T., Murugan, R., Mirjalili, S., Chakrabartty, D. K. (2021).** OptCoNet: an optimized convolutional neural network for an automatic diagnosis of COVID-19. Appl. Intell., Vol. 51, No. 3, pp. 1351–1366.
7. **Heidari, M., Mirniaharikandehei, S., Khuzani, A. Z., Danala, G., Qiu, Y., Zheng, B. (2020).** Improving the performance of CNN to predict the likelihood of COVID-19 using chest X-ray images with preprocessing algorithms. Int. J. Med. Inform., Vol. 144, pp. 1–9.
8. **Islam, M. M., Karray, F., Alhajj, R., Zeng, J. (2021).** A review on deep learning techniques for the diagnosis of novel coronavirus (COVID-19). IEEE Access, Vol. 9, pp. 30551–30572.
9. **Ismael, A. M., Şengür, A. (2021).** Deep learning approaches for COVID-19 detection based on chest X-ray images. Expert Syst. Appl., Vol. 164, pp. 114054.
10. **Iwendi, C., Mahboob, K., Khalid, Z., Javed, A. R., Rizwan, M. (2022).** Classification of COVID-19 individuals using adaptive neuro-fuzzy inference system. Multimed. Syst., Vol. 28, pp. 1223–1237.
11. **Joshi, A. M., Nayak, D. R. (2022).** MFL-Net: An efficient lightweight multi-scale feature learning cnn for covid-19 diagnosis from CT images. IEEE J. Biomed. Health Inform., Vol. 26, No. 11, pp. 5355–5363.
12. **Karar, M. E., Hemdan, E. E.-D., Shouman, M. A. (2021).** Cascaded deep learning classifiers for computer-aided diagnosis of COVID-19 and pneumonia diseases in X-ray scans. Complex Intell. Syst., Vol. 7, No. 1, pp. 235–247.
13. **Khan, I. U., Aslam, N. (2020).** A deep-learning-based framework for automated diagnosis of COVID-19 using X-ray images. Inf., Vol. 11, No. 9, pp. 1–13.
14. **Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012).** ImageNet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst., Vol. 25, pp. 1097–1105.
15. **LeCun, Y., Bengio, Y., Hinton, G. (2015).** Deep learning. Nat., Vol. 521, No. 7553, pp. 436–444.
16. **Li, S., Liu, Y., Sui, X., Chen, C., Tjio, G., Ting, D. S. W., Goh, R. S. M. (2019).** Multi-instance multi-scale CNN for medical

image classification. Proceedings of the 22nd International Conference on Medical Image Computing and Computer Assisted Intervention, pp. 531–539.

17. **Lopez-Betancur, D., Bosco Duran, R., Guerrero-Mendez, C., Zambrano-Rodríguez, R., Saucedo-Anaya, T. (2021).** Comparación de arquitecturas de redes neuronales convolucionales para el diagnóstico de COVID-19. Comput. Sist., Vol. 25, No. 3, pp. 601–615.
18. **López-Ponce de León, J. D., Cárdenas-Marín, P. A., Giraldo-González, G. C., Escandón, A. H. (2020).** Coronavirus - COVID-19: Más allá de la enfermedad pulmonar, qué es y qué sabemos del vínculo con el sistema cardiovascular. Rev. Colomb. Cardiol., Vol. 27, No. 3, pp. 142–152.
19. **Perez, D. P., Bustillos, R. S., Botto-Tobar, M., Mora, C. M. (2021).** Análisis de imágenes de rayos X por medio de redes neuronales artificiales. Ecuad. Sci. J., Vol. 5, No. 1, pp. 55–60.
20. **Probst, P., Boulesteix, A.-L., Bischl, B. (2019).** Tunability: importance of hyperparameters of machine learning algorithms. J. Mach. Learn. Res., Vol. 20, No. 1, pp. 1934–1965.
21. **Punn, N. S., Agarwal, S. (2021).** Automated diagnosis of COVID-19 with limited posteroanterior chest X-ray images using fine-tuned deep neural networks. Appl. Intell., Vol. 51, No. 5, pp. 2689–2702.
22. **Quesada, J. A., López-Pineda, A., Gil-Guillén, V. F., Arriero-Marín, J. M., Gutiérrez, F., Carratala-Munuera, C. (2021).** Período de incubación de la COVID-19: revisión sistemática y metaanálisis. Rev. Clin. Esp., Vol. 221, No. 2, pp. 109–117.
23. **Rodríguez-Hernández, L. J., Ochoa-Domínguez, H. d. J. (2021).** Métodos para el ajuste de los hiperparámetros de las redes convolucionales. Mem. Cienc. Tecnol., Vol. 1, No. 2, pp. 1–2.
24. **Serizawa, T., Fujita, H. (2020).** Optimization of convolutional neural network using the linearly decreasing weight particle swarm optimization. arXiv preprint arXiv:2001, Vol. 05670, pp. 1–10.
25. **Shorten, C., Khoshgoftaar, T. M. (2019).** A survey on image data augmentation for deep learning. J. Big Data, Vol. 6, No. 1, pp. 1–48.
26. **Storn, R., Price, K. (1997).** A simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim., Vol. 11, No. 4, pp. 341–359. DOI: 10.1023/A:1008202821328.
27. **Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. (2016).** Rethinking the Inception architecture for computer vision. Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826.
28. **Tabik, S., Gómez-Ríos, A., Martín-Rodríguez, J. L., Sevillano-García, I., Rey-Area, M., Charte, D., Guirado, E., Suárez, J.-L., Luengo, J., Valero-González, M., et al. (2020).** COVIDGR dataset and COVID-SDNet methodology for predicting COVID-19 based on chest X-ray images. IEEE J. Biomed. Health Inform., Vol. 24, No. 12, pp. 3595–3605.
29. **Wang, L., Lin, Z. Q., Wong, A. (2020).** COVID-Net: A tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images. Sci. Rep., Vol. 10, No. 1, pp. 1–12.
30. **Worldometers (2024).** COVID-19 coronavirus pandemic. Accessed November 9, 2025: https://www.worldometers.info/coronavirus/.
31. **Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K. (2017).** Aggregated residual transformations for deep neural networks. Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1492–1500.
32. **Yoo, S. H., Geng, H., Chiu, T. L., Yu, S. K., Cho, D. C., Heo, J., Choi, M. S., Choi, I. H., Van, C. C., Nhung, N. V., Min, B. J., Lee, H. (2020).** Deep learning-based decision-tree classifier for COVID-19 diagnosis from chest X-ray imaging. Front. Med., Vol. 7, pp. 427.

Article received on 31/10/2025; accepted on 15/12/2026.
**Corresponding author is Raúl Cruz-Barbosa.*