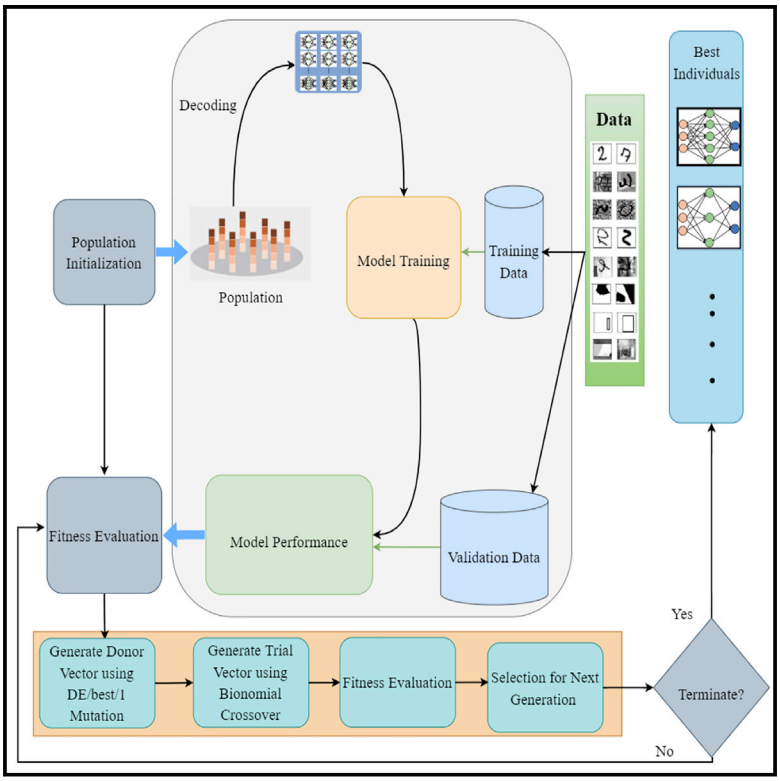
Article

Patterns

Designing optimal convolutional neural network architecture using differential evolution algorithm

Graphical abstract



Highlights

- Introduce DE algorithm to automatically design CNN architectures
- Variable-length encoding strategy is proposed to encode each CNN model
- For the DE framework, two CNN architectures undergo a refinement difference approach
- Design a heuristic mechanism for mutation operation to evolve CNN architectures

Authors

Arjun Ghosh, Nanda Dulal Jana, Saurav Mallik, Zhongming Zhao

Correspondence

zhongming.zhao@uth.tmc.edu

In brief

Designing an optimal convolutional neural network (CNN) for a particular problem is a challenging task requiring extensive expert knowledge and trial-error procedures. Moreover, designing different CNN architectures for several computer vision tasks is time-consuming. Therefore, it is essential to find an appropriate CNN model for a particular problem with the minimum available resources and human intervention. This article proposed the automatic design of CNN models using differential evolution meta-heuristic algorithm for several image classification problems.

Ghosh et al., 2022, Patterns *3*, 100567
September 9, 2022
https://doi.org/10.1016/j.patter.2022.100567

CellPress

Patterns

CellPress
OPEN ACCESS

Article

Designing optimal convolutional neural network architecture using differential evolution algorithm

Arjun Ghosh,[1] Nanda Dulal Jana,[1] Saurav Mallik,[2,3] and Zhongming Zhao[2,4,*]

[1]Department of Computer Science and Engineering, National Institute of Technology Durgapur, Durgapur, West Bengal 713209, India

[2]Center for Precision Health, School of Biomedical Informatics, The University of Texas Health Science Center at Houston, Houston, TX 77030, USA

[3]Molecular and Integrative Physiological Sciences, Department of Environmental Health, Harvard T. H. Chan School of Public Health, Boston, MA 02115, USA

[4]Lead contact

*Correspondence: zhongming.zhao@uth.tmc.edu

https://doi.org/10.1016/j.patter.2022.100567

THE BIGGER PICTURE Convolutional neural networks (CNNs) are a class of deep learning (DL) methods that have demonstrated improved performance in various computer vision tasks. With the growing popularity of CNNs, several CNN architectures have been introduced with a large number of design options that are problem dependent. In most situations, the constructed CNN model performs well on the dataset used to train it. There is no guarantee that the designed CNN model can achieve sufficient classification accuracy for other datasets. Designing an appropriate CNN model architecture for a particular problem requires human interaction and trial-and-error procedures, which are laborious and time consuming. This study uses an improved differential evolution of convolutional neural network (IDECNN) technique to automatically construct effective CNN architectures for several image classification problems, which mitigates the issues found with manually designed CNN models.

Development/Pre-production: Data science output has been rolled out/validated across multiple domains/problems

SUMMARY

Convolutional neural networks (CNNs) are deep learning models used widely for solving various tasks like computer vision and speech recognition. CNNs are developed manually based on problem-specific domain knowledge and tricky settings, which are laborious, time consuming, and challenging. To solve these, our study develops an improved differential evolution of convolutional neural network (IDECNN) algorithm to design CNN layer architectures for image classification. Variable-length encoding is utilized to represent the flexible layer architecture of a CNN model in IDECNN. An efficient heuristic mechanism is proposed in IDECNN to evolve CNN architecture through mutation and crossover to prevent premature convergence during the evolutionary process. Eight well-known imaging datasets were utilized. The results showed that IDECNN could design suitable architecture compared with 20 existing CNN models. Finally, CNN architectures are applied to pneumonia and coronavirus disease 2019 (COVID-19) X-ray biomedical image data. The results demonstrated the usefulness of the proposed approach to generate a suitable CNN model.

INTRODUCTION

Convolutional neural networks (CNNs),[1–3] one class of deep learning (DL) models, have become powerful tools for solving a variety of computer vision, speech recognition, text segmentation, cosmetic product recognition, and biomedical data-mining tasks.[4–9] Several CNN models, for example LeNet,[10] VGGNet,[11] AlexNet,[12] GoogleLeNet,[13] ResNet,[14] and many others, have been developed manually with increasing architectural depth and large numbers of parameters for solving different image

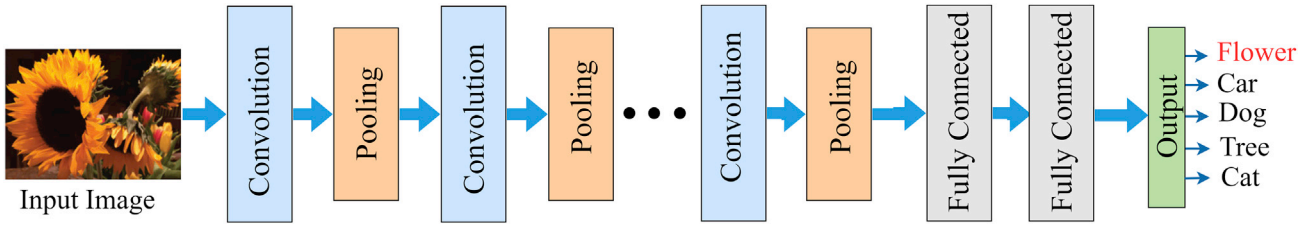


Figure 1. Conventional structure of a CNN model

classification tasks. These models are built on the basis of problem-specific domain knowledge expertise and trial-and-error procedures to select suitable architectures and parameters for a particular CNN model, which is labor intensive and time consuming. CNN model architecture size and its associated parameters directly influence its performance and complexity. Therefore, designing an optimal CNN from diverse architecture and parameters search space for a given problem is a challenging task without human participation.

Another issue is the use of substantial resources to find a network that can be transferred to datasets beyond the training set.[15–17] In most situations, the developed network performs well on the specific dataset that was used to train the network. There is no assurance that the constructed networks can achieve satisfactory classification accuracy for other datasets. Because of limited computational resources, it is practically impossible to design different networks for several datasets. Therefore, it is essential to find an appropriate model that helps non-DL researchers develop a DL model for a particular problem with minimum available resources. Given this goal, we were motivated to automate the optimal CNN architecture design for the image classification task.

Neural Architecture Search (NAS) is an efficient and effective approach for automatic architecture design that includes arrangement of layers and parameters that constitute a CNN model.[18] It has three components: search space, search strategy, and performance estimation strategy. The search space is responsible for representing architectures with some encoding mechanisms to achieve all possible combinations of architectures. The search strategy defines an efficient search technique for finding the best architecture from the search space, and performance evaluation strategy refers to the process of estimating the performance of the generated architectures to accelerate the search strategy and minimize the evaluation cost. Because of the nature of the three components, NAS can be treated as a bilevel, non-differential, non-convex optimization problem.[18,19]

Several research communities have focused their interest on different components of NAS to design architectures and parameters of CNN models for solving various image classification tasks.[20–26] However, more contributions can be made to the search space and the search strategy to attain an appropriate CNN model for a given problem with limited computational resources. Recently, meta-heuristic approaches have emerged as a powerful and popular search strategy to address the NAS approach compared with other conventional methods .[27] These meta-heuristics includes genetic algorithms (GAs),[28] genetic programming (GP),[29] ant colony optimization (ACO),[30] particle swarm optimization (PSO),[31] and differential evolution (DE),[32] to name a few. Generally, DE is a simple, robust, and mathematically sound approach with faster convergence for solving complex real-world optimization problems.[33,34] It is also easier to use because it has fewer parameters to configure and maintain exploration-exploitation trade-off with simple mutation and crossover operators during the search process.[35] To the best of our knowledge, only two studies of DE have been performed in the NAS perspective to design the optimal architecture of the CNN model for a classification task.[32,36] Therefore, there is a reason for using the DE algorithm to automate design of CNN architectures.

Wang et al.[32] introduced the DE method to automatically design an optimal CNN model for an image classification task. An Internet protocol (IP)-based encoding scheme was proposed to represent a CNN model in the search space. An extra second crossover operator was proposed with existing operators in DE for evolving CNN architectures. This strategy has some limitations; for example, layers are trimmed to adopt mutation operation to lead to loss of exploration in architecture search space. The added crossover operation bears extra computational cost. Awad et al.[36] proposed canonical DE to address NAS in the continuous search space. The proposed approach was investigated on cell-based architecture of a CNN model, which is very complicated and difficult to implement using limited computational resources. Continuous values are mapped to the NAS search space with a discretized architecture strategy to evaluate architectures. Mapping strategy is an important aspect because of loss of information during conversion of continuous to discrete domains. Therefore, an effective architecture representation strategy and efficient search mechanism are essential to explore the architectural search space fully and prevent premature convergence in meta-heuristic algorithm for architecture design of a CNN model.

In the paper, an improved DE-based approach is proposed to design layer-based CNN architecture for an image classification task. It is called the improved DE of CNN (IDECNN) algorithm. The proposed method introduced variable-length encoding and some efficient strategies in the original DE framework to enhance architecture search performance. The contributions to this work are as follows:

- Propose a direct encoding scheme for representing types and arrangements of layers of a CNN for easy conversion from genotype to phenotype during evaluation of CNN architecture. Each individual is encoded with a variable length to enhance the diversity in the depth of layer architectures. This encoding produces more flexibility and exploration within the architectural search space compared with fixed-length architecture.

Table 1. Summary of the related works and comparison with the proposed work

Model	Search method	Proposed work	Limitation	Dataset
GeNet[43]	GA	A fixed-length binary encoding strategy that represents CNN and GA was used to encode connections between layers of the CNN model.	Hyper-parameters of the associated layers were ignored.	CIFAR-10
EvoCNN[28]	GA	A variable-length encoding strategy was used to represent CNN, and GA was used to optimize both connections between layers and weights of the CNN model.	The best-generated architecture faced an over-fitting problem because of considering a large numbers of parameters.	MNIST, convex, rectangle
MA-NET[44]	Memetic	CNN was represented using a variable-length encoding strategy, and the memetic algorithm was used to optimize connections between layers of the CNN model.	Because of the large number of parameters considered, the best-generated architecture may have encountered an over-fitting problem.	MNIST, MNIST variation, convex, rectangle
DeepSwarm[30]	ACO	Pheromone information was used collectively to find the best CNN model. The authors used local and global pheromone update rules during method execution to balance exploration and exploitation.	Associated collective behavior made the approach computationally expensive.	MNIST, Fashion-MNIST, CIFAR-10
IPPSO[45]	PSO	A novel encoding scheme inspired by computer networking to represent a CNN architecture. PSO was used to optimize the layers and associated hyper-parameters of the CNN models.	Because of the architecture's fixed pre-defined length, the depth of the architectural search space was reduced.	MNIST, MNIST with noisy image, convex
psoCNN[31]	PSO	A variable-length encoding strategy to represent CNN architecture and layer type, such as Conv, Pool, and FC, was updated by copying the layers in a random fashion from the personal or global best solutions.	The architectural search space that might be less explored because a new particle was built from the global or personal best particle.	MNIST, MNIST variation, convex, rectangle
DECNN[32]	DE	An internet protocol (IP)-based encoding strategy represented a CNN architecture. DE mutation and crossover operations were used to evolve CNN models. An extra crossover operator was integrated to generate offspring from the parent individuals.	Trim operation before the mutation operation may have led to loss of exploration in the architectural search space, and two crossover operations made the approach very complicated and expensive.	MNIST, MNIST variation, convex, rectangle
DE-NAS[36]	DE	A cell-based encoding scheme was used to represent the CNN models. Continuous values were mapped to the NAS search space with a discretization strategy to evaluate architectures.	Converting from continuous to discrete space was expensive. Using a cell-based encoding strategy made the approach more costly and complicated.	CifarA, CifarB, Cifar C
IDECNN (proposed method)	DE	A variable-length direct encoding scheme is proposed to represent the depth, layer types, and arrangement of layers in CNN architectures; a simple difference mechanism between two architectures, followed by mutation and crossover operations to evolve each CNN through the original DE.	This only considered the layer-based CNN architecture design rather than cell- or block-based architecture design because of the limited computational resources on hand.	MNIST, MNIST variation, convex, rectangle

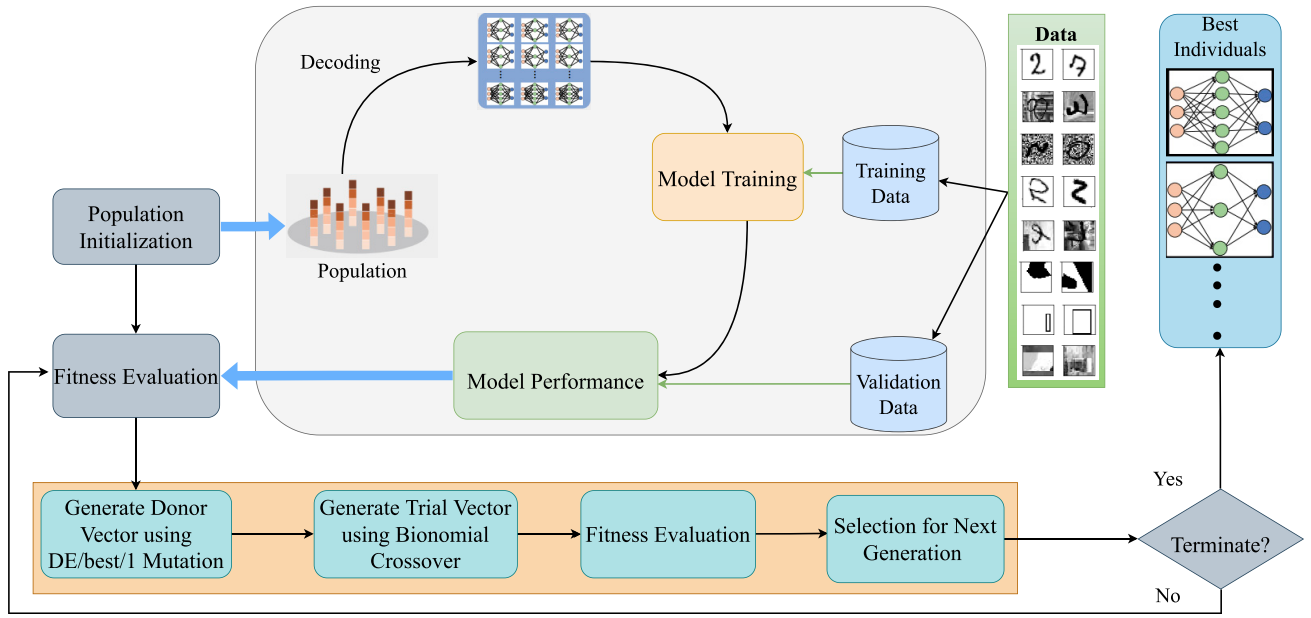


Figure 2. Framework of the proposed IDECNN algorithm

- Propose a refinement strategy to produce differences between two encoded architectures that enhance the exploration capability during a search of the optimal CNN model. This difference is very important for performing mutation for the original DE framework. The difference is produced based on the different layer type of each CNN model. This type of strategy produces the difference in a very transparent manner, which provides enough flexibility in the variable-length architecture search space.
- Design a heuristic mechanism for mutation operation to evolve CNN layer architectures to prevent getting stuck at local optima. This mechanism compares each layer type of the best architecture and the architecture obtained from the difference between the two architectures.
- experiment is performed on eight widely used benchmark image classification datasets to evaluate the effectiveness of the proposed model. Results are compared with 20 start-of-the-art CNN models that are hand crafted and evolution based.
- Perform ablation studies on the proposed method to investigate the impact of epoch numbers, generation numbers, population size, and parameter values of DE on the training accuracy of the best generated CNN architecture.
- Each best generated CNN architecture is applied to real-life pneumonia chest X-ray image classification for normal and pneumonia images, along with coronavirus disease 2019 X-ray images for COVID-19 and non-COVID-19 prediction.

The paper is organized as follows: First we provide background details and related works of CNN architecture searches for image classification problems. Then the proposed IDECNN

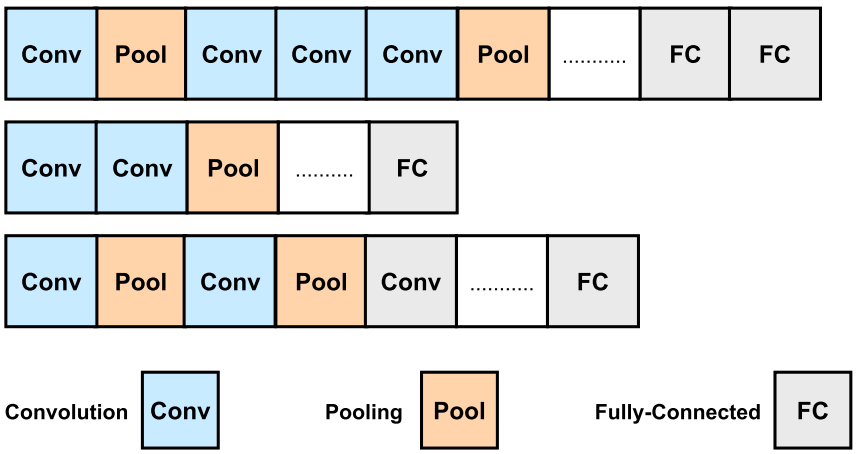


Figure 3. Individuals with different lengths in IDECNN

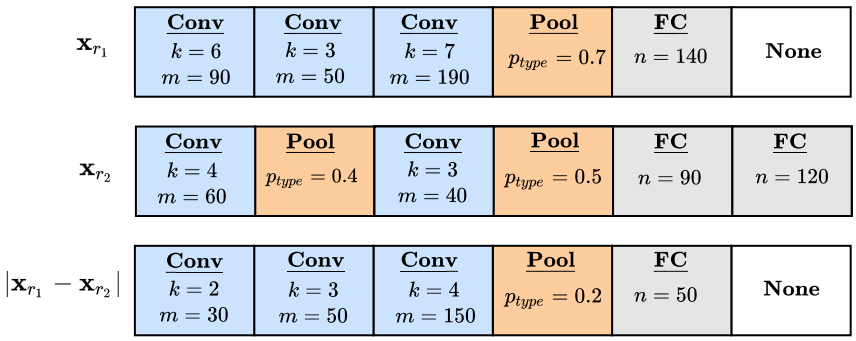


Figure 4. Difference calculation between two individuals

is described, and the experimental design of the proposed algorithm is presented. Next we provide a detailed analysis of results, discussion, and ablation studies. We also discuss the case study on the pneumonia and COVID-19 X-ray dataset. Finally, we draw conclusions for our study.

Preliminaries

Deep CNNs

CNNs are a class of DL models used extensively for analyzing visual images[37] and, recently, for analyzing many other types of complex data.[38,39] A conventional CNN has four different layers: convolution (Conv), pooling (Pool), fully connected (FC), and output. These layers are stacked to form a workable CNN model.[40] Figure 1 shows a conventional structure of a CNN model with different numbers of Conv, Pool, and FC and one output layer at the end.

The size of the output layer depends on the number of classes as given in a classification problem. The depth of a CNN model is strongly influenced by the number of Conv, Pool, and FC layers. These layers are responsible for extracting hierarchical features from raw input data.[41] These layers have various sets of parameters, such as kernel size, stride size, and number of feature maps for the Conv layer; kernel size, stride size, and pool type for the Pool layer; and total number of neurons for the FC layer. These parameters are known as hyper-parameters and must be adjusted or selected before training a CNN model for a classification task.

The majority of CNN architecture layers and the associated hyper-parameters are configured based on professional expertise and a trial-and-error process, which is labor intensive and time consuming. There are no pre-defined rules for which Conv and Pool layers can be arranged to design an appropriate CNN architecture to solve a particular task. Therefore, architectural design of layer types and the associated hyper-parameters of a CNN model remains a daunting task.

DE

DE is a population-based stochastic algorithm that aims to find global optimum solution for complex optimization problems.[42] Initialization, mutation, crossover, and selection are the main stages of the DE method.[42] First, individuals of the population are initialized randomly within the specified search space. Then, mutation operation is performed to generate donor vectors. A commonly used mutation scheme, $DE/best/1$, is performed according to the following equation:

$$\mathbf{v}_i^g = \mathbf{x}_{best}^g + F \times \left(\mathbf{x}_{r_1}^g - \mathbf{x}_{r_2}^g\right). \quad \text{(Equation 1)}$$

In Equation (1), $\mathbf{v}_i^g$ denotes i^{th} donor vector of the target vector $\mathbf{x}_i$ at generation g, $\mathbf{x}_{best}^g$ defines the best individual in the population according to the fitness value at the g^{th} generation, $\mathbf{x}_{r_1}^g$ and $\mathbf{x}_{r_2}^g$ are two randomly chosen mutually exclusive individuals of the population at generation g, and F is a scaling factor $\in (0,1)$ governing the rate of

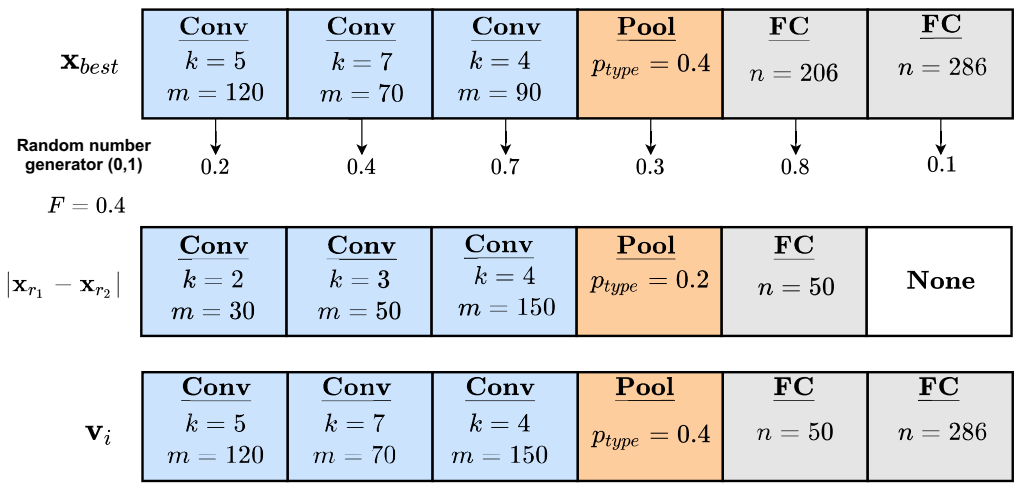


Figure 5. Donor vector generation using mutation operation

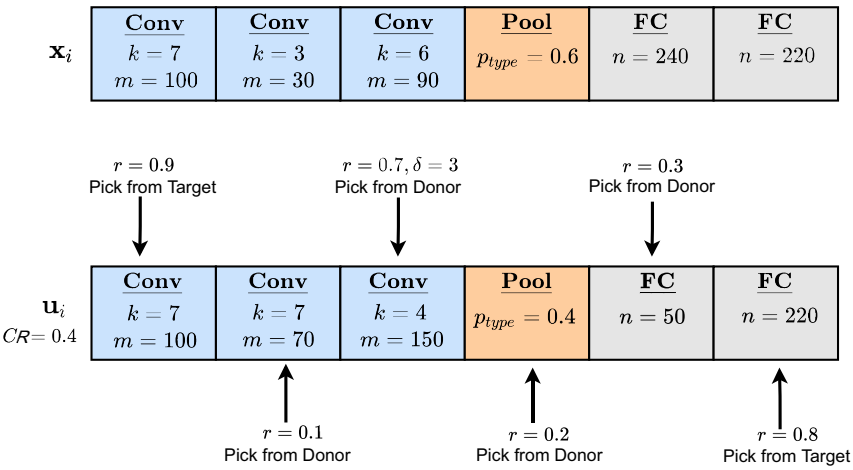


Figure 6. Trial vector generation using crossover operation

evolution. After donor vector generation, the crossover operation is performed as follows:

$$u_{j,i}^{g} = \begin{cases} v_{j,i}^{g} & \text{if } rand_j[0,1] \leq CR \text{ or } j = \delta \\ x_{j,i}^{g} & \text{Otherwise} \end{cases} \quad \text{(Equation 2)}$$

where $u_{j,i}^{g}$ defines the j^{th} dimension of the i^{th} individual at the g^{th} generation. In binomial crossover[42] of DE, at the beginning, a random δ number is generated for each individual. Another random number, $rand_j(0,1)$, is generated for each dimension of each individual and compared with the crossover rate CR and δ according to Equation (2) to determine whether the crossover will take place on that dimension. Finally, the generated trial vector $\mathbf{u}_i^g$ is compared with the target vector $\mathbf{x}_i^g$, and the best one is selected for the next generation according to its fitness value. This process repeats until the maximum number of generations or some stop criterion is achieved.

Related works

Because of emergence of CNNs in computer vision applications, the architecture of CNN models has become more complex and requires extensive human intervention. To reduce human intervention in building an appropriate architecture for a given problem is a very challenging task. Various population-based evolutionary approaches have been introduced to evolve the architecture of a CNN model for image classification problems.

Xie and Yuille[43] introduced a GA to automatically define CNN architecture, called GeNet. The authors proposed a fixed-length binary encoding strategy to represent CNN model architecture. In each generation, standard genetic operations are performed to generate competitive individuals and eliminate weak ones. Individuals are encoded as connections between layers of CNN architecture without considering hyper-parameters of the associated layers. GeNet model performance has been investigated only on the CIFAR-10 dataset. Sun et al.[28] proposed an algorithm called EvoCNN to evolve CNN models using GA. The authors introduced a variable-length encoding scheme instead of a fixed-length strategy for generating CNN architecture. Their proposed approach simultaneously optimizes architecture and connection weights of the CNN model. EvoCNN achieved significant results on various MNIST datasets as well as convex and rectangular datasets. However, the best generated architecture might be faced with over-fitting problems because of consideration of large numbers of parameters. Dong et al.[44] proposed a memetic algorithm-based automatic design of CNN architecture for image classification called MA-NET. The algorithmic framework of EvoCNN is employed in MA-NET, including a local search for generating optimal CNN architectures in each iteration. The efficacy of MA-NET is tested on the same datasets as used in EvoCNN.

In addition to GA, swarm intelligence (SI) has also been applied to find the best CNN models for image classification tasks.[30,31,45] Byla and Pang[30] introduced DeepSwarm, an approach based on ACO to evolve CNN architectures. In their proposed method, pheromone information is used collectively to find the best CNN model. The authors incorporated local and global pheromone update rules for balancing exploration and exploitation during execution of the DeepSwarm method. Their proposed method tested only three datasets: MNIST, Fashion-MNIST, and CIFAR-10. The pheromone information was updated based on the collective behavior of each ant, which makes the DeepSwarm approach computationally expensive. Wang et al.[45] used PSO, called IPPSO, to evolve CNN architectures. The authors proposed a novel encoding scheme inspired by computer networking to represent a CNN architecture. IPPSO has been validated on three image datasets: MNIST, MRDBI, and convex. Because of fixed pre-defined length in the architecture, it resulted in a loss of flexibility in the depth of architectural search space. Recently, Fernandes and Yen[31] proposed an algorithm based on conventional PSO called psoCNN. The authors used a variable-length encoding strategy to represent CNN architecture, and layer type, such as Conv, Pool, and FC, is updated by copying the layers in a random fashion from the personal or global best solutions. Their proposed algorithm was tested on MNIST along with variation of MNIST, convex, and rectangular datasets. The main weakness was seen in architectural search space, which might be less explored because each new particle is built from the global or personal best particle.

Recently, DE has shown enough exploring capability to search the optimal CNN architecture for image classification tasks. From this perspective, Wang et al.[32] first proposed a hybrid DE approach called DECNN to evolve CNN model architectures. The authors introduced an IP-based encoding strategy to represent a CNN architecture. A mutation and crossover operation is devised to evolve CNN models in DECNN. A second crossover operator is integrated to generate offspring from the parent individuals. DECNN has been used on MNIST, different variants of MNIST, and convex datasets. This work has some limitations.

Algorithm 1. The pseudocode of IDECNN

Input: N, population size; g_{max}, maximum generation number; CR, crossover rate; F, scaling factor; D_{train}, training dataset; D_{valid}, validation dataset; D_{test}, test dataset.
Output: Return best found individual x_{best} and its test error

$P = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \leftarrow$ population_initialization (N)
//Initialize population
for $i = 1$ to N **do**
  $\mathbf{x}_i \leftarrow$ random_configuration (layer_type, hyper-parameters, l_{max})
//randomly initialize Conv, Pool and FC layer along with their hyper-parameter and maximum length of individual
  $f(\mathbf{x}_i) \leftarrow$ compute_fitness($\mathbf{x}_i, D_{train}, D_{valid}$)
//Fitness evaluation
  $fitness(i) = f(\mathbf{x}_i)$
end
$\mathbf{x}_{best} \leftarrow$ min($fitness$); //best individual
While $g \leq g_{max}$ **do**
  for $i = 1$ to N **do**
    $\mathbf{v}_i^g \leftarrow$ compute_mutation($\mathbf{x}_i^g, \mathbf{x}_{r_1}, \mathbf{x}_{r_2}, \mathbf{x}_{best}, F$)
//Donor vector generation using mutation operation
    $\mathbf{u}_i^g \leftarrow$ compute_crossover($\mathbf{x}_i^g, \mathbf{v}_i^g, \delta, CR$)
//Trial vector generation using crossover operation
    $fitness(i) = f(\mathbf{u}_i^g)$
//Fitness evaluation of trial vector
    $\mathbf{x}_i^{g+1} \leftarrow$ compute_selection($f(\mathbf{x}_i^g), f(\mathbf{u}_i^g)$)//Selection for next generation
  end
  g $\leftarrow$ g + 1
end
$\mathbf{x}_{best} \leftarrow min($ fitness$)$
//Final best CNN architecture
for $e_1 \leq best\_trainepoch$ **do**
  $\mathbf{x}_{best}^{train} \leftarrow$ train ($\mathbf{x}_{best}, D_{train}, parameters$)
// $\mathbf{x}_{best}$ train with training data and parameters
end
for $e_2 \leq best\_testepoch$ **do**
  $f(\mathbf{x}_{best}) \leftarrow$ test ($\mathbf{x}_{best}^{train}$, D_{test})
end
Return $\mathbf{x}_{best}$ and its test error

First, the trim operation leads to loss of exploration in NAS space. Second, the proposed crossover operation in the DECNN algorithm may be complicated and expensive. Recently, Awad et al.[36] used DE for optimal CNN architecture in continuous search space and called it DE-NAS. In DE-NAS, a discretization method is proposed to map continuous to discrete searches to evaluate CNN model accuracy. Their proposed method was tested on various CIFAR datasets, including CifarA, CifarB, and CifarC. However, designing a discretization method for conversion from continuous to discrete space is very difficult and problem dependent.

To the best of our knowledge, only two studies have focused on DE for evolving architectures of CNN models for image classification problems. The proposed work introduces a simple difference mechanism between two architectures, followed by mutation and crossover operations, to evolve each CNN architecture through the original DE. This concept differs from the DECNN[32] model. Unlike the DE-NAS model,[36] a direct encoding scheme is proposed to represent the types and arrangements of CNN layer architecture for each individual in our study. Table 1 summarizes the main characteristics of each approach and the difference from the proposed work.

THE PROPOSED APPROACH (IDECNN)

This section explicitly describes the proposed optimal layered architecture generation of the CNN model for the image classification task. First, an overall framework of the proposed method is presented. Main components, such as encoding an individual, population initialization, fitness evaluation, mutation, crossover, and selection operation of DE concerning CNN model architecture evolvement are narrated in consecutive subsections.

Structure of IDECNN

The structure or framework of the proposed IDECNN algorithm is depicted in Figure 2.

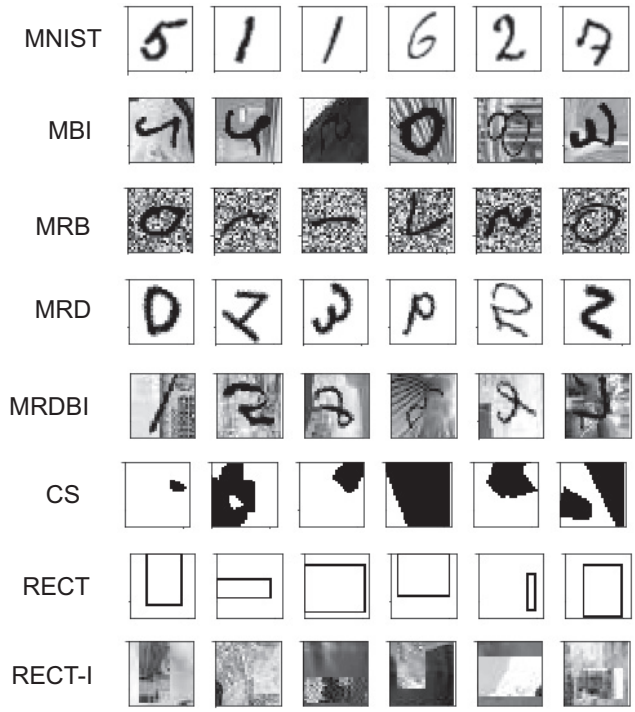


Figure 7. Sample pictures of each benchmark dataset used in the proposed work

The algorithm started with a randomly initialized population of N individuals. In the population, each individual stands for a workable CNN architecture, which was trained with training data (D_{train}) before being tested for fitness on the validation dataset (D_{valid}). We evaluated the fitness of each individual on their D_{valid} in terms of loss of classification error. The 10% of the training set was randomly extracted as the D_{valid} during the fitness evaluation process. After that, the main steps of the DE process were performed, where CNN models were evolved through the mutation and crossover operation of DE using our proposed unique strategy. When newly updated individuals were generated, their fitness was tested in the same way fitness was evaluated after the population initialization. This process continued until a stopping criterion was satisfied. Then, the fittest individuals were selected from each generation according to their fitness function, which was the minimum classification error in our proposed study. Finally, among the selected best individuals, the optimal CNN architecture was selected based on their lowest fitness value and tested on the test dataset (D_{test}) to determine the model's final performance.

Encoding strategy

Encoding, which is one of the most important key elements, is a very challenging task when designing an algorithm for efficient NAS.[20] It defines how an individual is encoded to represent a whole CNN architecture. In a NAS study, the architecture of any CNN model is encoded mostly in a layer-based or block-based encoding scheme.[46] In this study, a simple layer-based encoding scheme was used compared with complex block-based encoding, which demands huge computational resources. In this encoding scheme, each individual was composed of three types of layers sequentially that were selected randomly from a list consisting of Conv, Pool, and FC layers to build CNN architecture. To achieve a workable CNN model, the first and last components of individuals were always fixed with a Conv and FC layer, respectively. FC layers must be placed after all possible combinations of Conv and Pool layers in an encoded architecture. The length of individuals can be varied; this is generally known as variable-length encoding method. This strategy was considered in our IDECNN to achieve flexibility in CNN layer architecture. Figure 3 shows an example of three individuals with different lengths.

Table 2. Overview of the datasets used in the proposed IDECNN algorithm for experimental study

Dataset	Input size	Description	No. of training	No. of test	No. of classes
MNIST	28 × 28 × 1	handwritten digits	60,000	10,000	10
MBI	28 × 28 × 1	handwritten digits with background images	12,000	50,000	10
MRB	28 × 28 × 1	handwritten digits with random noise as background	12,000	50,000	10
MRD	28 × 28 × 1	handwritten rotated digits	12,000	50,000	10
MRDBI	28 × 28 × 1	handwritten rotated digits and background images	12,000	50,000	10
CS	28 × 28 × 1	convex shapes	8,000	50,000	2
RECT	28 × 28 × 1	rectangle border shapes	1,200	50,000	2
RECT-I	28 × 28 × 1	rectangle border shapes and image backgrounds	12,000	50,000	2

Initialization of population

Initialization of population plays a vital role in NAS. Individuals or architectures are initially distributed within the whole search space. Here, population P is a set of N individuals denoted as $P = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_N\}$. Each individual was initialized randomly over the Conv, Pool, and FC layers with some limitations. Limitations were imposed on the dimension or length and the position of each component for a particular individual. In this regard, the length was bounded by the minimum and maximum length, which were manually provided during initialization of each individual. In the case of position of component, each individual must be limited to the Conv and FC layers as the first and last component for a workable CNN architecture. Therefore, the ith individual is represented as $\mathbf{x}_i = (\text{Conv}, \ldots, \text{FC})$.

Each Conv, Pool, and FC layer had hyper-parameters that were selected randomly during the initialization $\mathbf{x}_i$. For instance, the hyper-parameters of Conv were kernel size, stride size, and number of feature maps; the Pool had kernel size, stride size, and pool type and number of neurons for the FC layer. In IDECNN, kernel size c_k and number of feature maps c_m for Conv were selected from a pre-defined range, and stride size c_s was fixed

Table 3. The hyper-parameter and parameter settings for IDECNN

Parameter name	Value
DE initialization	
Population size	20
# generation	20
F	0.6
CR	0.4
Hyper-parameters of CNN	
Conv kernel size	3–7
Conv stride size	1
# feature maps	3–256
Pool kernel size	3
Pool stride size	2
Pool type	average or max
No. of neurons in an FC layer	1–300
Length of CNN	3–10
Training of CNN	
Activation function	ReLu
Weight initialization	Xavier
Optimizer	Adam
Learning rate	0.001
Batch size	200
Dropout rate	0.5
No. of epochs for single individual evaluation	1
No. of epochs for final individual	100

at 1 × 1. Similarly, kernel size p_k and stride size p_s of the Pool layer were fixed to 3×3 and 2 × 2, respectively, in our study. Pool type p_{type} was set randomly for each component of the Pool layer as average or max Pool. The FC layer was associated with a number of neurons f_n, which was also pre-defined in the given range. The remaining hyper-parameters involved in their corresponding layers were considered based on various research studies. Therefore, each component of $\mathbf{x}_i$ was composed of layer type and hyper-parameters of such layers. Details of the associated parameter settings are provided in Table 3 for our proposed algorithm. Thus, the characteristic properties of the Conv, Pool, or FC layer can be varied within each individual of the population. *N* individuals of the population were initialized along with different architectures and hyper-parameter settings.

Fitness evaluation

Fitness function was used to determine the quality of every individual in the population *P*. The fitness of an individual represents how well it performs for a given task. In our proposed method, each individual represented one CNN architecture. Therefore, fitness was calculated with the help of encoded information for the corresponding individual. We evaluated the fitness of each individual on their D_{valid} in terms of loss of classification error. D_{valid} was considered for calculating the loss error to avoid the over-fitting problem.[47] Individuals were compared with their associated fitness value and the best selected on minimum fitness; i.e., loss of error from the population *P*.

Each individual (i.e. $\mathbf{x}_1$ to $\mathbf{x}_N$) of *P* was compiled into a CNN and trained with a number of training epochs on D_{train}. For training purposes, Xavier initialization[48] for weight initialization, rectified linear unit (ReLU)[49] as the activation function, and the Adam optimizer[50] for optimizing the model were used in this work. Then the fitness of each $\mathbf{x}_i$ was evaluated on D_{valid}. Here, fitness was scored based on classification loss error. Cross-entropy (CE) loss[51] was used as a fitness function for the proposed method because of its outstanding performance in terms of the loss error in multi-class classification. After evaluation of each individual in the population, the best was determined by the minimum loss of error function. Finally, the fitness of individuals was recorded along with their fitness values.

Mutation

In the context of the evolutionary computing paradigm, mutation can be viewed as a variation or a perturbation with a random element. In DE, a mutant vector (also called donor vector) $\mathbf{v}_i$ was obtained through different mutant operations for each target vector $\mathbf{x}_i$, which was a parent vector of current generation. In our study, we used the *DE*/*best*/1 mutation scheme to achieve simple implementation and provide more diversity in the best CNN architecture at each generation. A simple difference calculation method was proposed in the mutation step.

In the proposed IDECNN, two individuals ($\mathbf{x}_{r_1} \neq \mathbf{x}_{r_2}$) are selected randomly from the population *P* that are different from the target vector $\mathbf{x}_i$. Then the difference $(\mathbf{x}_{r_1}\text{-}\mathbf{x}_{r_2})$ was calculated based on the layer type of each individual's component (i.e., Conv, Pool, and FC). Figure 4 shows an example of the proposed difference calculation method.

If the j^{th} dimension of $\mathbf{x}_{r_1}$ and $\mathbf{x}_{r_2}$ has the same type of layers, then subtraction is done according to their associated hyper-parameters value. For example, in Figure 4, the first component of both individuals was the Conv layer. So the current value of kernel size *k* and number of feature maps *m* of $\mathbf{x}_{r_2}$ was subtracted from the corresponding values of $\mathbf{x}_{r_1}$ to represent their difference. The same difference mechanism is also applied for the Pool layer and FC layer. On the other hand, if the j^{th} dimension of $\mathbf{x}_{r_1}$ and $\mathbf{x}_{r_2}$ has different values in layer type, then it copies the j^{th} layer from $\mathbf{x}_{r_1}$ along with its corresponding hyper-parameters to represent the difference.

After the difference calculation, boundary checking was done for $(\mathbf{x}_{r_1}\text{-}\mathbf{x}_{r_2})$ as hyper-parameters of each layer constraint within a specified search range. Then IDECNN picked the best individual $\mathbf{x}_{best}$ from the population *P* according to their fitness value. The donor vector $\mathbf{v}_i$ was computed by using the uniformly generated random number *r* and selecting a layer from $\mathbf{x}_{best}$ or $(\mathbf{x}_{r_1}\text{-}\mathbf{x}_{r_2})$ based on the scaling factor *F*. If $r \leq F$, proposed mechanism select layer from $\mathbf{x}_{best}$. Otherwise, it was chosen from $\mathbf{x}_{r_2}$). Equation (3) defines the mutation operation, where $v_{j,i}$ defines the j^{th} dimension of the i^{th} individual in *P*. Finally, the mutant individual $\mathbf{v}_i$ (called the donor) was generated.

$$v_{j,i} = \begin{cases} \mathbf{x}_{j,best} & \textit{if } \mathrm{r} \leq \mathrm{F} \\ \left|\mathbf{x}_{r_1} - \mathbf{x}_{r_2}\right|_{j,i} & \textit{Otherwise} \end{cases} \qquad \text{(Equation 3)}$$

Table 4. Classification error results of IDECNN and state-of-the-art methods/models

Methods/models		MNIST	MBI	MRB	MRD	MRDBI	CS	RECT	RECT-I
LeNet-1[10]		1.70%	–	–	–	–	–	–	–
LeNet-4[10]		1.10%	–	–	–	–	–	–	–
LeNet-5[10]		0.95%	–	–	–	–	–	–	–
NNet[52]		4.69%	27.41%	20.04%	17.62%	42.17%	32.25%	7.16%	33.20%
SVM + Poly[52]		3.69%	24.01%	16.62%	13.61%	37.59%	19.82%	2.15%	24.05%
SVM + RBF[52]		3.03%	22.61%	14.58%	10.38%	32.62%	19.13%	2.15%	24.04%
DBN-1[52]		3.94%	16.15%	9.80%	12.11%	31.84%	19.92%	4.71%	23.69%
DBN-3[52]		3.11%	16.31%	6.73%	12.30%	28.51%	18.63%	2.60%	22.50%
SAA-3[52]		3.46%	23.00%	11.28%	11.43%	24.09%	18.41%	2.41%	24.05%
TIRBM[53]		–	–	–	–	35.50%	–	–	–
PGBM + DN-1[54]		–	–	36.76%	–	1.27%	–	–	–
RandNet-2[55]		11.65%	13.47%	8.47%	43.69%	5.45%	0.09%	17.00%	1.06%
PCANet-2[55]		11.55%	6.85%	8.52%	35.86%	4.19%		0.49%	13.39%
LDANet-2[55]		1.40%	12.42%	6.81%	4.52%	38.54%	7.22%	0.14%	16.20%
EvoCNN[28]	best	1.18%	4.53%	2.80%	5.22%	35.03%	4.82%	**0.01%**	5.03%
	mean	1.28%	4.62%	3.59%	5.46%	37.38%	5.39%	**0.01%**	5.97%
MA-NET[44]	best	–	3.56%	2.48%	3.33%	15.92%	–	–	–
	mean	–	–	–	–	–	–	–	–
DeepSwarm[30]	best	0.46%	–	–	–	–	–	–	–
	mean	0.39%	–	–	–	–	–	–	–
IPPSO[45]	best	1.13%	–	–	–	34.50%	8.48%	–	–
	mean	1.13%	–	–	–	33.00%	12.06%	–	–
	SD	0.10%	–	–	–	2.96%	2.25%	–	–
psoCNN[31]	best	0.32%	1.90%	1.79%	3.58%	14.28%	1.7%	0.03%	2.22%
	mean	0.44%	2.40%	2.53%	6.42%	20.98%	3.9%	0.34%	3.94%
DECNN[32]	best	1.03%	5.67%	3.46%	4.07%	32.85%	7.99%	–	–
	mean	1.46%	8.69%	5.56%	5.53%	37.55%	11.19%	–	–
	SD	0.11%	1.41%	1.71%	0.45%	**2.45%**	1.94%	–	–
IDECNN	best	**0.29%**	**1.01%**	**1.29%**	**3.02%**	**10.04%**	**1.36%**	0.08%	**1.62%**
	mean	**0.38%**	**2.29%**	**2.07%**	**4.16%**	**14.31%**	**2.96%**	0.75%	**2.66%**
	SD	**0.09%**	**1.09%**	**0.60%**	**0.45%**	4.11%	**1.14%**	**0.67%**	**0.92%**

An example of donor vector generation is shown in Figure 5 using the global best individual and the difference vector $(\mathbf{x}_{r_1}\text{-}\mathbf{x}_{r_2})$.

Crossover

To improve possible diversity in the population, a crossover operation occurred after generating the donor vector through mutation. The donor vector $\mathbf{v}_i$ exchanged its components with the target vector $\mathbf{x}_i$ through a crossover operation to form the trial vector $\mathbf{u}_i$. In DE, different kinds of methods are used in the crossover operation. For simplicity, the binomial crossover was used in this paper. In the binomial crossover, formation of the trial vector is done on the basis of crossover rate *CR* and a random number δ.

In IDECNN, we first counted the length of the donor vector $\mathbf{v}_i$. Then we assigned δ value randomly from the range of the length $\mathbf{v}_i$. Another random number, $rand_j(0,1)$, was generated for each dimension (*j*) of $\mathbf{u}_i$. If $r \leq CR$ or $j = \delta$, then it took the corresponding j^{th} value from donor $\mathbf{v}_i$ or otherwise from target vector $\mathbf{x}_i$. An example of the proposed method is given in Figure 6, where each dimension of the trial vector $\mathbf{u}_i$ was picked up from the target vector $\mathbf{x}_i$ or the donor vector $\mathbf{v}_i$ ($\mathbf{v}_i$ defined in Figure 5).

Selection

Selection stage was used to pick the target or trial vector based on the fitness value for the next generation to maintain a constant population size over successive generations. Each $\mathbf{x}_i$ of *P* evaluated fitness $f(\mathbf{x}_i)$ according to the fitness function in terms of classification loss error. IDECNN also calculated the fitness of $\mathbf{u}_i$, which is represented as $f(\mathbf{u}_i)$. Then it selected the better one between $\mathbf{x}_i$ and $\mathbf{u}_i$ based on their minimum loss error values for the next generation $(g+1)$ as follows:

$$\mathbf{x}_i^{g+1} = \begin{cases} \mathbf{u}_i^g & \text{if } f(\mathbf{u}_i^g) \leq f(\mathbf{x}_i^g) \\ \mathbf{x}_i^g & \text{Otherwise} \end{cases} \quad \text{(Equation 4)}$$

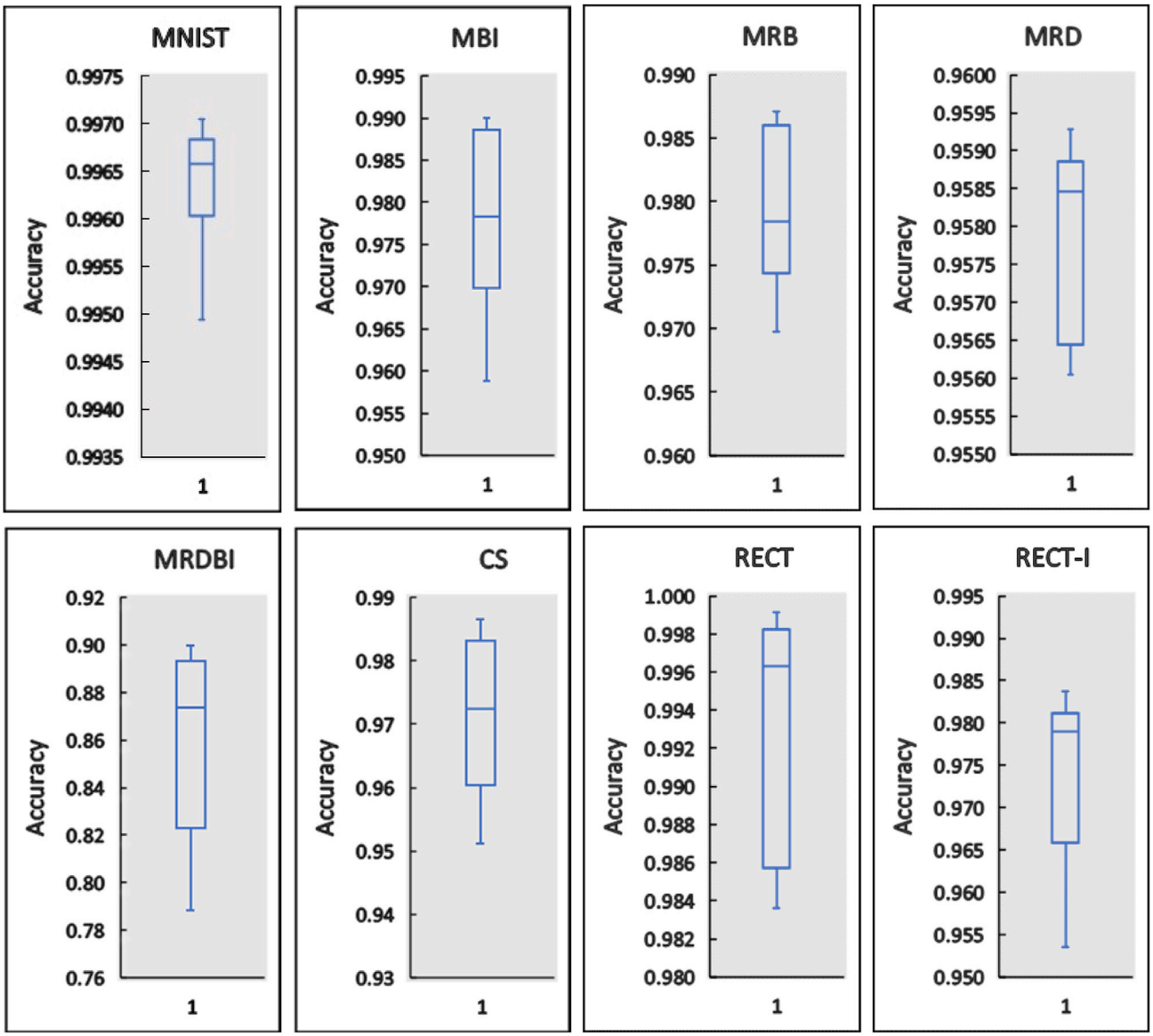


Figure 8. Test accuracy boxplots of the IDECNN algorithm for the MNIST, MBI, MRD, MRDBI, CS, RECT, and RECT-I datasets

Putting it all together for the IDECNN algorithm

The aforementioned subsections were assembled to build the overall algorithm of our proposed IDECNN method. The pseudo-code of IDECNN is presented in Algorithm 1.

Population P is initialized with population size N. In P, each individual $\mathbf{x}_i$ is randomly configured with the Conv, Pool, and FC layer along their hyper-parameter setting. We also set the length of each individual within the range of three to l_{max}. Then it computes the fitness of each $\mathbf{x}_i$ in terms of loss of classification error on D_{valid} after training with D_{train}. In IDECNN, 10% of the total training samples is used as a D_{valid} for fitness computation. After completion of N fitness evaluation, the best $\mathbf{x}_i$ is picked according to their minimum classification error and stored as $\mathbf{x}_{best}$. In each generation (g), during the mutation step, a donor vector ($\mathbf{v}_i$) is generated for each $\mathbf{x}_i$ by using two random individuals (i.e., $\mathbf{x}_{r_1}, \mathbf{x}_{r_2}$) and $\mathbf{x}_{best}$ from N along with scaling factor F. Again, in the crossover operation, trial vector ($\mathbf{u}_i$) is produced with the help of $\mathbf{x}_i$, $\mathbf{v}_i$, δ, and crossover rate CR. In the selection stage, the better vector is selected between $\mathbf{x}_i$ and $\mathbf{u}_i$ based on their minimum loss error for the next generation. At every generation, individuals are usually trained with a small number of epochs that are not good choices for any CNN

Table 5. Classification error results of psoCNN and IDECNN without BN and dropout on the CS image dataset

Model		CS
psoCNN-BN-Dropout[31]	best	5.53%
	mean	5.90%
IDECNN-BN-Dropout	best	**2.19%**
	mean	**4.32%**
	SD	**1.62%**

Table 6. Best CNN architectures evolved by IDECNN on eight image datasets

Dataset	CNN architecture with hyper-parameters									
MNIST	**Conv**	**Pool**	**Conv**	**Conv**	**Conv**	**Pool**	**Conv**	**Pool**	**FC**	**FC**
	$c_k = 3, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 6, c_s = 1$	$c_k = 4, c_s = 1$	$c_k = 3, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 3, c_s = 1$	$p_k = 3, p_s = 2$	$f_n = 128$	$f_n = 10$
	$c_m = 110$	*Max Pool*	$c_m = 132$	$c_m = 221$	$c_m = 194$	*Max Pool*	$c_m = 248$	*Max Pool*		
MBI	**Conv**	**Conv**	**Pool**	**Conv**	**Conv**	**Pool**	**FC**	**FC**		
	$c_k = 3, c_s = 1$	$c_k = 6, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 5, c_s = 1$	$c_k = 3, c_s = 1$	$p_k = 3, p_s = 2$	$f_n = 189$	$f_n = 10$		
	$c_m = 216$	$c_m = 196$	*Max Pool*	$c_m = 251$	$c_m = 240$	*Max Pool*				
MRB	**Conv**	**Pool**	**Conv**	**Conv**	**Pool**	**Conv**	**FC**	**FC**		
	$c_k = 6, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 4, c_s = 1$	$c_k = 5, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 3, c_s = 1$	$f_n = 128$	$f_n = 10$		
	$c_m = 176$	*Avg Pool*	$c_m = 192$	$c_m = 240$	*Max Pool*	$c_m = 248$				
MRD	**Conv**	**Pool**	**Conv**	**Pool**	**Conv**	**FC**	**FC**			
	$c_k = 5, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 3, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 6, c_s = 1$	$f_n = 106$	$f_n = 10$			
	$c_m = 196$	*Max Pool*	$c_m = 139$	*Max Pool*	$c_m = 232$					
MRDBI	**Conv**	**Conv**	**Conv**	**Conv**	**Pool**	**Conv**	**FC**	**FC**		
	$c_k = 4, c_s = 1$	$c_k = 5, c_s = 1$	$c_k = 5, c_s = 1$	$c_k = 6, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 6, c_s = 1$	$f_n = 107$	$f_n = 10$		
	$c_m = 243$	$c_m = 208$	$c_m = 139$	$c_m = 220$	*Avg Pool*	$c_m = 168$				
CS	**Conv**	**Pool**	**Conv**	**Conv**	**Conv**	**Conv**	**Pool**	**FC**	**FC**	
	$c_k = 5, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 4, c_s = 1$	$c_k = 6, c_s = 1$	$c_k = 4, c_s = 1$	$c_k = 6, c_s = 1$	$p_k = 3, p_s = 2$	$f_n = 286$	$f_n = 2$	
	$c_m = 192$	*Max Pool*	$c_m = 224$	$c_m = 170$	$c_m = 252$	$c_m = 238$	*Max Pool*			
RECT	**Conv**	**Conv**	**Pool**	**Conv**	**Conv**	**Conv**	**Pool**	**FC**		
	$c_k = 5, c_s = 1$	$c_k = 3, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 4, c_s = 1$	$c_k = 6, c_s = 1$	$c_k = 3, c_s = 1$	$p_k = 3, p_s = 2$	$f_n = 2$		
	$c_m = 96$	$c_m = 110$	*Avg Pool*	$c_m = 152$	$c_m = 253$	$c_m = 240$	*Max Pool*			
RECT-I	**Conv**	**Conv**	**Conv**	**Pool**	**Conv**	**Conv**	**FC**			
	$c_k = 6, c_s = 1$	$c_k = 3, c_s = 1$	$c_k = 3, c_s = 1$	$p_k = 3, p_s = 2$	$c_k = 5, c_s = 1$	$c_k = 3, c_s = 1$	$f_n = 2$			
	$c_m = 196$	$c_m = 110$	$c_m = 206$	*Max Pool*	$c_m = 246$	$c_m = 21$				

Conv, convolution; Pool, pooling; FC, fully connected; c_k, Conv kernel size; c_s, Conv stride size; c_m, number of feature maps; p_k, Pool kernel size; p_s, Pool stride size; f_n, number of neurons.

Table 7. Number of parameters used in each generated best CNN architecture

Optimal CNN architecture	No. of parameters
MNIST_CNN	4.32 million
MBI_CNN	12.41 million
MRB_CNN	9.40 million
MRD_CNN	5.58 million
MRDBI_CNN	6.14 million
CS_CNN	16.27 million
RECT_CNN	2.43 million
RECT-I_CNN	1.79 million

architecture in case of NAS implementation. Hence, our final CNN architecture needs more epochs to obtain a favorable result. For this purpose, the best generated CNN architecture (i.e., $\mathbf{x}_{best}$) at the end of the maximum generation (i.e., g_{max}) is again trained and tested using D_{train} and D_{test}. Finally, the testing error as the classification error of the best generated CNN model architecture is generated.

EXPERIMENTAL DESIGN

The experimental datasets, state-of-the-art methods, algorithm parameter settings, and experimental setup to evaluate the performance of the proposed IDECNN algorithm are described in this section.

Benchmark datasets

In this experiment, eight commonly used image datasets were selected to test the IDECNN algorithm. These were MNIST,[52] MNIST with background image (MBI),[52] MNIST including random noise as background (MRB),[52] MNIST with rotated digits (MRD),[52] MNIST including rotated digits and background image (MRDBI),[52] convex sets (CS),[52] rectangles[52] (RECT), and rectangles with image (RECT-I).[52] Some sample pictures of these datasets are presented in Figure 7, and details are provided in Table 2.

Because of our limited amount of computing power, the algorithm was tested only with datasets with a small input size.

The MNIST is the classification of 0–9 handwritten digits. The MNIST dataset is used extensively to test DL algorithms for the image classification task. It has 10 classes, contains 0–9 handwritten images in black and white with 60,000 training samples and 10,000 test samples. MBI, MRB, MRD, and MRDBI are the variants of the MNIST dataset. There are reasons to use MNIST variations besides only MNIST. First, although most algorithms give a minimum classification error for a basic MNIST dataset, additional noises are incorporated into these MNIST variations (e.g., random background images, random noise as background, rotated digits, or rotated digits and background images) to challenge the algorithm as it enhances the dataset complexity. Second, all four variants include 12,000 samples of training and 50,000 test images, which provide additional hurdles for the algorithm because significantly fewer images were available compared with testing.

On the other hand, CS includes black or white geometrical images in which the model is used to classify the convex shape. It consists of 8,000 training data and 50,000 test data. CS has a 2-class compared with the 10-class MNIST dataset. The RECT dataset comprises rectangle images that are classified in black and white images. Each rectangle is of a different size concerning width and height. This dataset includes 1,200 training instances and 50,000 test instances. A variation of the RECT dataset is the RECT-I dataset, which incorporated a rectangle border shape and image background together. It has more training images and the same testing images as the original RECT dataset: 12,000 and 50,000, respectively.

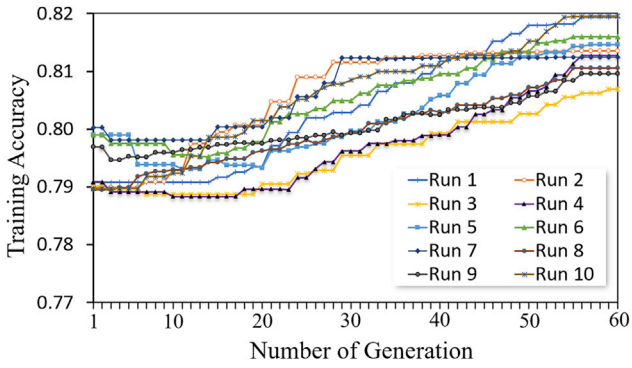


Figure 10. Training accuracy of the best individual for 10 runs on the CS dataset

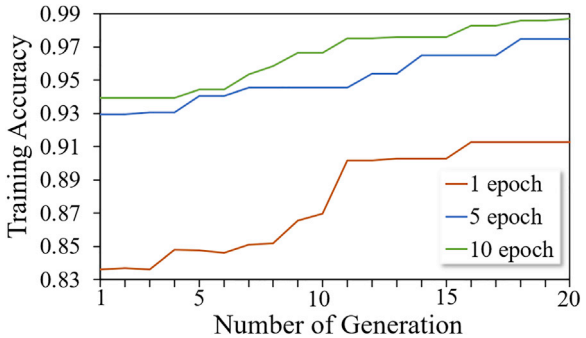


Figure 9. Effect of three epoch numbers (1, 5, and 10) on best model training accuracy during fitness evaluations on the CS dataset

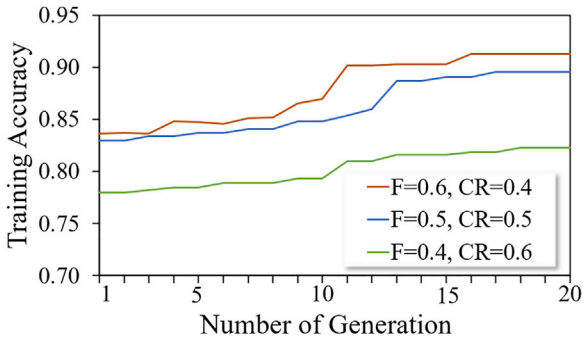


Figure 11. Training accuracy of the best individual with different *F* and *CR* settings for the CS dataset

Table 8. Overview of the chest X-ray dataset

Class name	Input size	No. of training	No. of validation	No. of test
Normal	180 × 180	1,082	267	234
Pneumonia	180 × 180	3,110	773	390

State-of-the-art models/methods

To test the performance of IDECNN, it was compared with 20 competitive DL methods. In this paper, selection of such competing algorithms has been done on similar datasets for the same task. Specifically, the first 14 models are designed manually for solving image classification problems. These models are LeNet-1, LeNet-4, LeNet-5,[10] NNet,[52] SVM + Poly,[52] SVM + RBF,[52] DBN-1,[52] DBN-3,[52] SAA-3,[52] TIRBM,[53] PGBM + DN1,[54] RandNet-2,[55] PCANet-2,[55] and LDANet-2.[55] We also used population-based methods to compare our results with the algorithms EvoCNN,[28] MA-NET,[44] DeepSwarm,[30] IPPSO,[45] psoCNN,[31]and DECNN.[32] These state-of-the-art models are closest to IDECNN, but there is a great difference in exploration strategy within the search space.

Parameter settings

In the proposed study, all parameters were set according to the conventional DE[56] and DL[57] communities. This is presented in Table 3.

In IDECNN, because of limited computational resources, safe computation time, and reduced search space complexity, we used 20 as the size of the population and 20 as the maximum number of the generation. In each generation, a population of N individuals was built, where each individual represents a CNN architecture in the constraint search space. In terms of DE, F and CR were fixed to 0.6 and 0.4, respectively.

The hyper-parameter settings are a very crucial and challenging task for any architecture-based CNN model design. These were selected in our study based on related research studies. Here, all the hyper-parameters information was encoded along with each individual at the time of population initialization. IDECNN used three kind of layers: Conv, Pool, and FC. The hyper-parameters of each layer were fixed within the range of some value. In Conv layer, the kernel size (c_k) ranged from 3 × 3 to 7 × 7 while stride size was permanently fixed at 1 × 1, respectively. The number of feature maps C_m for the Conv layer varied randomly from the range 3 to256. The proposed algorithm adds Pool with kernel size (p_k) 3×3 and stride size 2 × 2. On the other hand, the Pool type (p_{type}) was selected randomly: average Pool or max Pool. Here, p_{type} is defined as

$$p_{type} = \begin{cases} AvgPooling, & if\ 0 \le rand(0,1) \le 0.5, \\ MaxPooling, & Otherwise \end{cases} \quad \text{(Equation 5)}$$

Similarly, the number of neurons (f_n) in each FC layer was set randomly from 1 to 300. Finally, the length of each individual was bounded within the range of 3 to 10.

The widely used activation function ReLU[49] was considered to train generated CNN models in this study. For the purpose of evaluating the fitness of each CNN architecture, the model was trained with the popular Xavier[48] weight initialization and Adam[50] optimization algorithm. We fixed our learning rate to 0.001. We also introduced batch normalization (BN)[58] with batch size 200 and 50% of dropout[59] to accelerate the training process of CNN architecture. In IDECNN, 10% of the training samples were used as a D_{valid} at the time of fitness evaluation. The number of epochs was set as 1 for each individual's fitness evaluation. Finally, the best individual (or CNN architecture) was trained with the number of epochs before it was tested on D_{test}. In this study, such epoch numbers were set as 100.

Table 9. Comparison of the classification accuracy of psoCNN and IDECNN on pneumonia chest X-ray images using the best CNN architecture generated for each dataset

Optimal CNN model	Model	Accuracy
MNIST_CNN	**psoCNN**[31]	87.50%
	IDECNN	85.58%
MBI_CNN	psoCNN[31]	71.14%
	IDECNN	74.84%
MRB_CNN	**psoCNN**[31]	86.48%
	IDECNN	82.85%
MRD_CNN	psoCNN[31]	86.22%
	IDECNN	88.14%
MRDBI_CNN	psoCNN[31]	74.84%
	IDECNN	79.65%
CS_CNN	**psoCNN**[31]	83.49%
	IDECNN	82.53%
RECT_CNN	psoCNN[31]	72.60%
	IDECNN	76.60%
RECT-I_CNN	psoCNN[31]	74.68%
	IDECNN	79.49%

Experimental setup

The proposed IDECNN algorithm was implemented in Python Release 3.6.9 along with two libraries: Tensorflow 1.15.0 and Keras 2.3.1. Finally, the overall process was executed in a Dell Precision 7820 workstation configured with Ubuntu 18.04, 64-bit operating system, Intel Xeon Gold 5215, 2.5-GHz processor, 96-GB RAM, and Nvidia 16GB Quadro RTX5000 graphics.

Results

The proposed IDECNN method was tested over eight common image datasets, and respective results are shown in Table 4.

Results where the proposed IDECNN method outperformed other state-of-the-art methods are shown in italics. The classification error of the proposed method is presented in the three bottom rows of Table 4 with respect to the best, mean, and standard deviation (SD) errors, which were achieved from 20 independent runs. The symbol – indicates that results on the corresponding datasets were not reported in the original papers. IDECNN showed equal or better performance in terms of best, mean, and SD of classification error on six of eight datasets: MNIST, MBI, MRB, MRD, CS, and RECT-I.

The proposed IDECNN achieved best, mean, and SD error rates of 0.29%, 0.38%, and 0.09%, respectively for the MNIST dataset, producing better results than all other state-of-the-art models. For

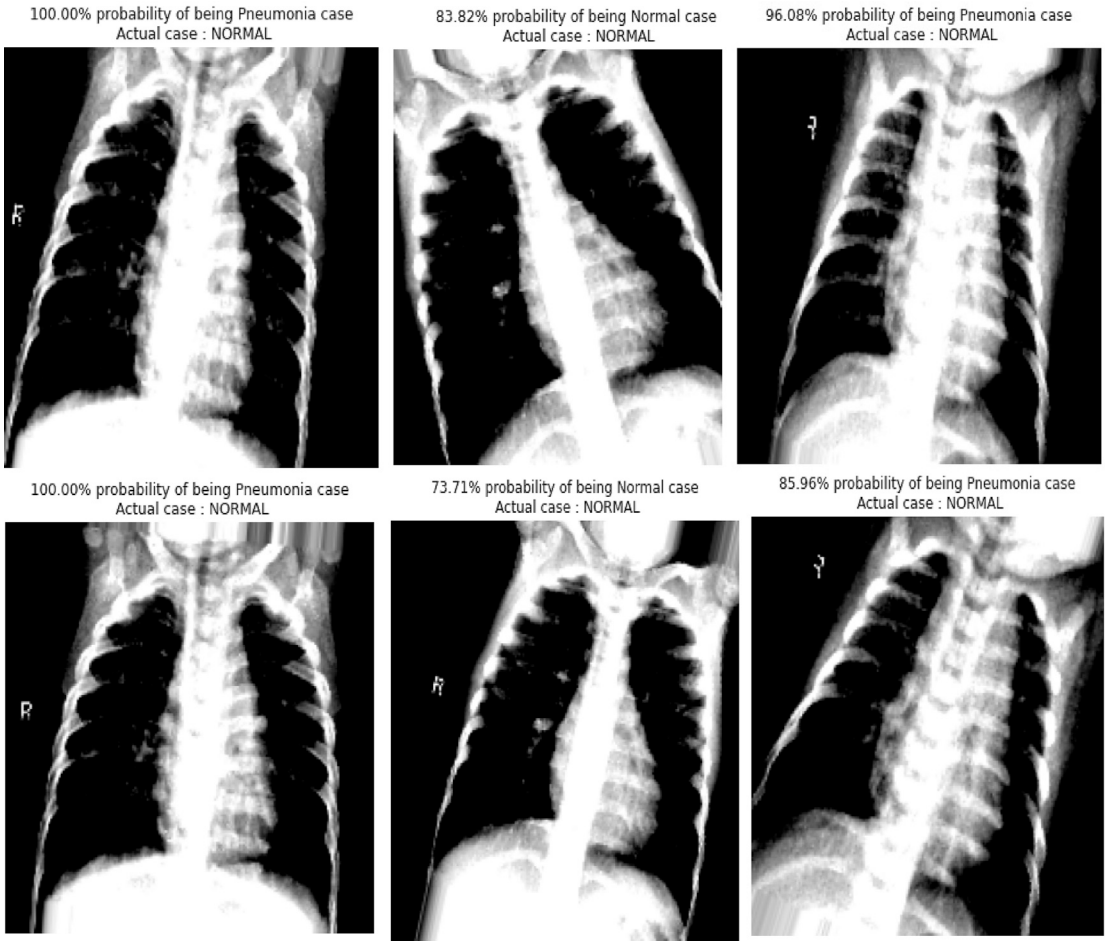


Figure 12. A sample of some of the predicted images with the percentage of predicted accuracy using the MNIST_CNN model in the case of psoCNN and the MRD_CNN model in the case of IDECNN

the MBI dataset, the proposed method generated a CNN architecture that achieved a best error rate of 1.01%, mean error rate of 2.29%, and SD error rate of 1.09%, and it stands in the first position against all other competitive models. The suggested method also outperformed other peer models for the MRB dataset. It provided a best, mean, and SD error of 1.29%, 2.07%, and 0.60%, respectively. Again, IDECNN produced better test results according to best, mean, and SD error, which were 3.02%, 4.16%, and 0.45%, respectively, for the MRD dataset. In the case of the MRDBI dataset, the proposed strategy performed better compared with other peer competitors with only best and mean error rate. It gave a best error rate of 10.04% and mean error rate of 14.31%. The suggested method took third place for SD error rate, which was 4.11% after DECNN and IPPSO. The proposed model responded better concerning best, mean, and SD error in the CS dataset than all other competitive models. It generated a best error rate of 1.36%, mean error rate of 2.96%, and corresponding SD error rate of 1.14%. The best, mean, and SD error for the RECT dataset were 0.08%, 0.75%, and 0.67%, respectively. In this case, IDECNN placed third with the best and mean error rate after EvoCNN and psoCNN. Finally, the suggested method outperformed in all cases of the best error rate of 1.62%, mean error rate of 2.66%, and SD error rate of 0.92% for the RECT-I dataset. The proposed IDECNN demonstrated a significant improvement over most datasets in terms of best, mean, and SD of classification error rate.

Test accuracy distributions of IDECNN using a boxplot graph are shown in Figure 8.

A Boxplot is a percentile-distributed graph divided into four groups known as quartile groups. Each group has 25% of the total score. In general, groups are labeled with numbers from 1–4 starting from the bottom. The distribution of test accuracy for MNIST data lies approximately between 0.995 and 0.997, and more variation can be observed in quartile group 1 compared with other groups. Similarly, for the MBI dataset, test accuracy scattered more in quartile group 1, whereas it is nearly identical for quartile groups 2 and 3. It is distributed within a small range from 0.959–0.990. In the MRB dataset, the variation of test accuracy is dispersed from nearly 0.970–0.987, and maximum variation is found in quartile group 3. The variation of test accuracy for the MRD dataset is maximal in quartile group 2 in a range of approximately 0.956–0.960. In the case of the MRDBI, the maximum is scattered in quartile group 2. It spreads in the range of roughly 0.788–0.90. In the CS dataset, all quartile groups have

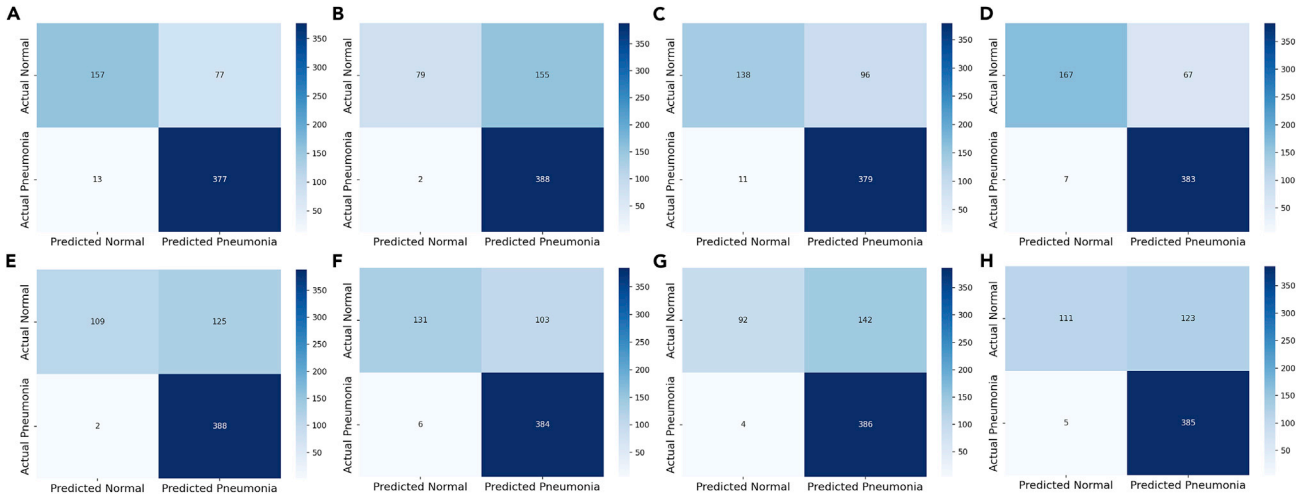


Figure 13. The obtained confusion matrices for the chest X-ray dataset using the eight best generated CNN architectures of the proposed IDECNN

(A–H) (A) MNIST_CNN, (B) MBI_CNN, (C) MRB_CNN, (D) MRD_CNN, (E) MRDBI_CNN, (F) CS_CNN, (G) RECT_CNN, and (H) RECT-I_CNN.

essentially identical distributions and ranges from approximately 0.950–0.985. In RECT, distribution is scattered in the range of approximately 0.983–1.0 with much scattering in the 2 quartile group. Finally, RECT-I ranges from nearly 0.955–0.985, and more variations are found in quartile group 2. Therefore, test accuracy distributions of IDECNN using the boxplot graph also show competitive performance for each dataset.

We also analyzed the performance of IDECNN with respect to classification errors such as best, mean, and SD error without BN and dropout, which is represented as IDECNN-BN-Dropout. The respective results are given in Table 5.

This paper considers only CS data for such experiments because of the limited computational resources. The results obtained are compared with the only available results in psoCNN.[31] The best, mean, and SD of classification error obtained by IDECNN-BN-Dropout are 2.19%, 4.32%, and 1.62%, respectively. These results substantially outperformed psoCNN .[31]

Discussion

In this work, the proposed IDECNN was used to find the optimal CNN architecture for image classification. In IDECNN, a refinement strategy was proposed to determine the difference between two CNN models and design a heuristic mechanism to perform the mutation and crossover operations to adapt the standard DE framework. The performance of IDECNN was examined through classification errors on eight popular image classification datasets and compared with 20 state-of-the-art models. The results demonstrated better performance on seven of eight datasets (MNIST, MBI, MRB, MRD, MRDBI, CS, and RECT-I) with respect to best and mean of classification errors. The best generated CNN architectures were robust in terms of SD values compared with other models on six datasets.

We further investigated the computation time of the proposed model compared with other popular competitive models. Because of the stochastic nature of DE, it does not allow us to compare the computational cost with other state-of-the-art methods. Different algorithms use different fitness functions, parameter settings, and system configurations, which makes it challenging to compare the proposed model in computation time. Generally, substantial time is needed for deep training each CNN architecture in each run. Sun et al.[28] showed that their algorithm (EvoCNN) took 2–3 days for each run of the same

Table 10. The obtained precision, recall, and F1 score of each class of chest X-ray dataset using the models MNIST_CNN, MBI_CNN, MRD_CNN, MRDBI_CNN, CS_CNN, RECT_CNN, and RECT-I_CNN

	Normal			Pneumonia		
Optimal CNN architecture	Precision	Recall	F1 score	Precision	Recall	F1 score
MNIST_CNN	0.92	0.67	0.78	0.83	0.97	0.89
MBI_CNN	0.98	0.34	0.50	0.71	0.99	0.83
MRB_CNN	0.93	0.59	0.72	0.80	0.97	0.88
MRD_CNN	0.96	0.71	0.82	0.85	0.98	0.91
MRDBI_CNN	0.98	0.47	0.63	0.76	0.99	0.86
CS_CNN	0.98	0.34	0.50	0.71	0.99	0.83
RECT_CNN	0.96	0.39	0.56	0.73	0.99	0.84
RECT-I_CNN	0.96	0.47	0.63	0.76	0.99	0.86

Table 11. Overview of the chest X-ray dataset with two times random splitting

No. of scenario	Class name	No. of training	No. of validation	No. of test
1	normal	1,108	316	159
	pneumonia	2,991	854	428
2	normal	791	474	318
	pneumonia	2,136	1,281	856

dataset we tested in this work. EvoCNN measured the running time of 10 runs using two GPU cards with the model number Nvedia GTX 1080. Wang et al.[45] took two and a half hours for each run with 30 independent runs for the datasets MBI, MRDBI, and CS by using two identical Nvidia GTX 1080 GPUs through their proposed model IPPSO. In this work, Nvidia RTX 5000 was used to measure the computational cost with 20 independent runs. We investigated the running time of the proposed IDECNN model concerning the best, average, and worst-case scenarios for the MNIST and CS datasets. For the MNIST dataset, the best, average, and worst running times were 28.12, 90.53, and 111.39 h, respectively, and 5.9, 15.7, and 27.7 h, respectively, for the CS dataset. Therefore, we observed the competitive performance of IDECNN in terms of computational cost compared with the well-known EvoCNN and IPPSO.

The best CNN architectures achieved through the IDECNN algorithm on each dataset are presented in Table 6.

The best generated CNN architectures for the datasets MNIST, MBI, MRB, MRD, and MRDBI consisted of a total number of layers of 10, 8, 8, 7, and 8, respectively. MNIST had more layers than others. It had 60,000 training samples, which is significantly larger than (only 12,000 training samples) for the MBI, MRB, MRD, and MRDBI datasets. Accordingly, rather than variations of MNIST, the CNN model for the MNIST dataset had a chance to train well with a large number of samples with more layers. In the MNIST dataset, five Conv, three max Pool, and two FC layers were sufficient to efficiently classify the dataset, whereas the MBI dataset required four Conv, two max Pool, and two FC layers in the respective CNN architecture. The CNN architectures for MRB consisted of four Conv, two Pool (average and max Pool), and two FC layers. For the MRD dataset, the Conv, Pool, and FC layers were three, two, and two, respectively, where both Pool types were max Pool. MRDBI had five Conv, one average Pool, and two FC layers. The best CNN architecture for the CS dataset took 9 layers to produce the final results. It consisted of five Conv, two max Pool, and two FC layers. For the RECT and RECT-I datasets, the final CNN architecture consisted of eight and seven total layers, respectively. In the RECT dataset, there are five Conv, two Pool (one average Pool and one max Pool), and one FC layers, whereas RECT-I worked with five Conv, one max Pool, and one FC layer.

In addition to the optimal architecture of CNN models, Table 7 shows the total number of parameters used in each designed CNN model.

The optimal CNN architecture for the MNIST dataset, MNIST_CNN, had 4.32 million parameters. On the other hand, the best CNN architecture for the MBI dataset, MBI_CNN, had 12.41 million parameters. In contrast, the parameters for the best CNN model of MRB, MRD, and MRDBI datasets (MRB_CNN, MRD_CNN, and MRDBI_CNN, respectively) were 9.40, 5.58, and 6.14 million, respectively. In the case of the CS dataset, the best generated CNN architecture, CS_CNN, included 16.27 million parameters, the maximum among all developed CNN architectures. Finally, for the datasets RECT and RECT-I, the best CNN models, RECT_CNN and RECT-I_CNN, had a total of 2.43 and 1.79 million parameters, respectively.

Ablation study

Ablation studies of the proposed model were performed to examine how to efficiently train the best generated architecture with respect to different epoch sizes, varying generation numbers, size of the population, and different values of *F* and *CR* on the CS

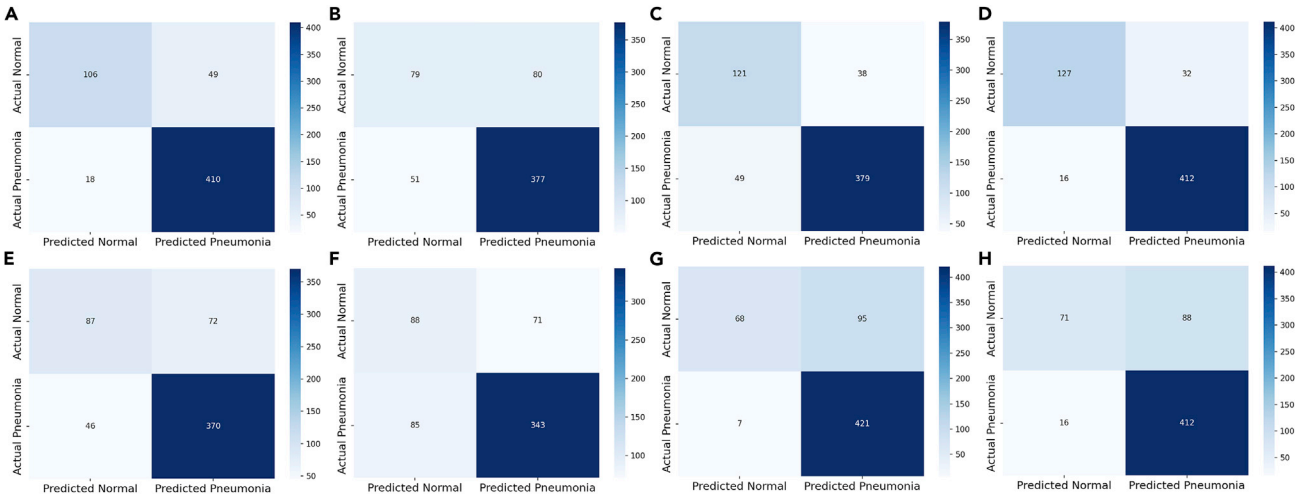


Figure 14. The obtained confusion matrices for the chest X-ray dataset with random splitting (scenario 1) using the eight best generated CNN architectures of the proposed IDECNN

(A–H) (A) MNIST_CNN, (B) MBI_CNN, (C) MRB_CNN, (D) MRD_CNN, (E) MRDBI_CNN, (F) CS_CNN, (G) RECT_CNN, and (H) RECT-I_CNN.

Table 12. The obtained precision, recall, F1 score, and model accuracy using the models MNIST_CNN, MBI_CNN, MRD_CNN, MRDBI_CNN, CS_CNN, RECT_CNN, and RECT-I_CNN for scenario 1

	Normal			Pneumonia			
Optimal CNN architecture	Precision	Recall	F1 score	Precision	Recall	F1 score	Model accuracy
MNIST_CNN	0.85	0.67	0.75	0.89	0.96	0.92	88.51%
MBI_CNN	0.61	0.49	0.54	0.82	0.88	0.85	77.68%
MRB_CNN	0.71	0.76	0.73	0.91	0.86	0.88	85.18%
MRD_CNN	0.89	0.79	0.84	0.93	0.96	0.94	91.82%
MRDBI_CNN	0.65	0.55	0.60	0.84	0.86	0.85	79.48%
CS_CNN	0.51	0.55	0.53	0.83	0.80	0.81	73.42%
RECT_CNN	0.91	0.42	0.57	0.82	0.98	0.89	82.74%
RECT-I_CNN	0.82	0.45	0.58	0.82	0.96	0.88	82.28%

dataset. To begin the process, different epoch sizes, such as 1, 5, and 10, were used to test the effectiveness of training accuracy of the best individual in each generation on the CS dataset. Figure 9 displays the corresponding test results.

The corresponding figure shows that the training accuracy rate increased exponentially with the number of epoch sizes. As a result, with more epochs during training, the model would reach a higher performance accuracy rate. More epochs take more time to evaluate the model because each individual needs to be trained with the number of training sets before evaluation on D_{valid}. We decided to fix the epoch size in the proposed method according to available computational resources.

We analyzed the model with a larger population size and generation numbers. Figure 10 shows the effect of training accuracy on the best individual for the CS dataset, including population size 30, generation number 60, and number of runs 10.

The result shows the improvement in training accuracy of the best individual at each run as the number of generations increased. Therefore, more generations and greater population size improved the performance of the proposed model. We set these values in our work based on the limited available resources.

In this paper, the performance of IDECNN was tested by using different *F* and *CR* values (0.6 and 0.4, respectively), based on the value use in conventional DE. We investigated the training accuracy of the best individual in each generation by fixing the values *F* and *CR* as 0.5 and 0.5, respectively, and the values *F* and *CR* as 0.4 and 0.6, respectively. The results of these investigations are shown in Figure 11.

The figure shows a consistent improvement in training accuracy with the corresponding *F* and *CR* values of 0.6 and 0.4, respectively. Therefore, according to the investigation, the setting of these values in our work is reasonable.

Case study on pneumonia and COVID-19 chest X-ray datasets

We investigated the effectiveness of the generated best CNN architectures for each dataset discussed in Table 6 through IDECNN on real-life application of pneumonia and COVID-19 chest X-ray images. For pneumonia, we used a chest X-ray dataset from the work of Kermany et al.[60] There are two classes in this dataset: normal and pneumonia. An overview of the corresponding classes is presented in Table 8.

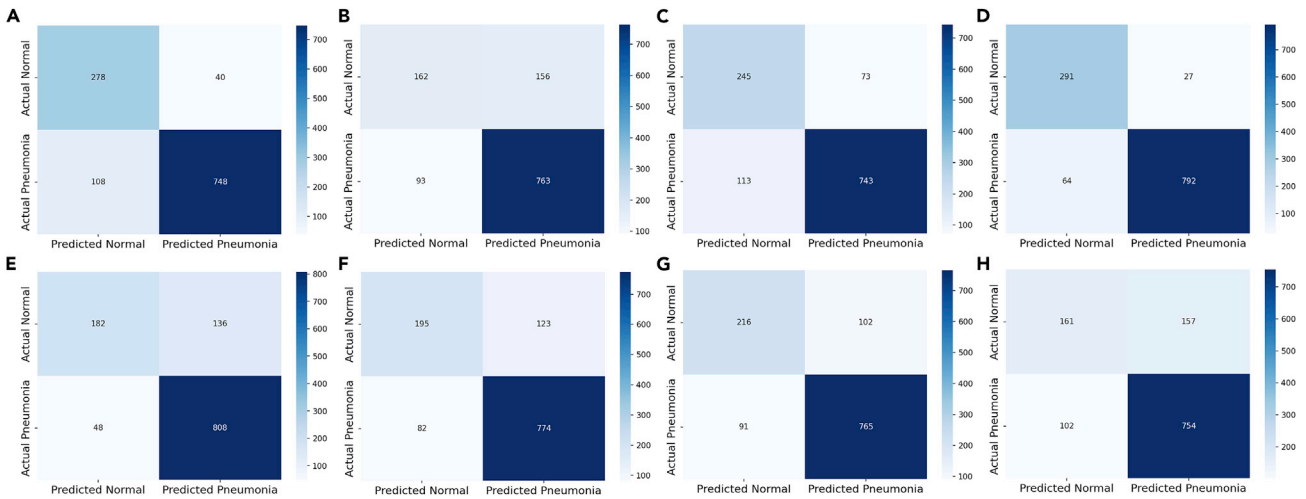


Figure 15. The obtained confusion matrices for the chest X-ray dataset with random splitting (scenario 2) using the eight best generated CNN architectures of the proposed IDECNN

(A–H) (A) MNIST_CNN, (B) MBI_CNN, (C) MRB_CNN, (D) MRD_CNN, (E) MRDBI_CNN, (F) CS_CNN, (G) RECT_CNN, and (H) RECT-I_CNN.

Table 13. The obtained precision, recall, F1 score, and model accuracy using the models MNIST_CNN, MBI_CNN, MRD_CNN, MRDBI_CNN, CS_CNN, RECT_CNN, and RECT-I_CNN for scenario 2

	Normal			Pneumonia			
Optimal CNN architecture	Precision	Recall	F1 score	Precision	Recall	F1 score	Model accuracy
MNIST_CNN	0.72	0.87	0.79	0.95	0.87	0.91	87.39%
MBI_CNN	0.64	0.51	0.57	0.83	0.89	0.86	78.79%
MRB_CNN	0.68	0.77	0.72	0.91	0.87	0.89	84.16%
MRD_CNN	0.82	0.92	0.87	0.97	0.93	0.95	92.25%
MRDBI_CNN	0.79	0.57	0.66	0.86	0.95	0.90	84.33%
CS_CNN	0.70	0.61	0.65	0.86	0.90	0.88	82.54%
RECT_CNN	0.70	0.68	0.69	0.88	0.89	0.88	83.56%
RECT-I_CNN	0.61	0.51	0.56	0.83	0.88	0.85	77.94%

There are a total of 5,856 chest X-ray images in the pneumonia dataset: 1,583 normal cases and 4,273 pneumonia case. For normal X-ray images, the training, validation, and testing datasets are 1,082, 267, and 234, respectively, whereas for pneumonia X-ray images, the training, validation, and testing datasets are 3,110, 773, and 390, respectively. The input size of all images in the normal and pneumonia cases is 180 × 180. We trained and then tested the class chest X-ray images (normal and pneumonia) using the best CNN architectures discussed in Table 6 and compared them with the psoCNN[31] model in terms of classification accuracy. In this experiment, we choose psoCNN because of its better performance compared with other state-of-the-art population-based methods. The authors of psoCNN reported the best CNN architectures for all datasets in their original paper, which are used here to evaluate the corresponding pneumonia dataset. In the experiment, binary_crossentropy[61] was used as a classification loss function because of the binary nature of the classification problem. Therefore, the last layer of all CNN models used a sigmoid[62] activation function, which is used widely for binary classification problems. We also fixed the batch size and number of epochs to 16 and 20, respectively. All other required parameters, such as weight initialization, optimization function, learning rate, batch size, dropout rate, and epoch numbers, were used as presented in Table 3.

In terms of classification accuracy, Table 9 shows the experimental findings for the psoCNN and our proposed approach.

The best CNN architecture for the MNIST dataset in the psoCNN model performs better for pneumonia chest X-ray images than the architecture generated by IDECNN for the same dataset. In this case, psoCNN and IDECNN have an accuracy of 87.50% and 85.58%, respectively. For the optimal CNN architecture MBI_CNN, the proposed IDECNN produced a higher classification accuracy, 74.84%, compared with 71.14% for the psoCNN model. Again, psoCNN responded better than IDECNN with the MRB_CNN model. The psoCNN model had an accuracy rate of 86.4%, whereas IDECNN's was 82.85%. In the cases of MRD_CNN and MRDBI_CNN, the generated CNN architecture using the proposed IDECNN produced better results than the existing psoCNN model. IDECNN generated 88.14% and 79.65% accuracy, whereas psoCNN produced 86.22% and 74.84%, respectively. In contrast, the best CNN architecture for the CS dataset, CS_CNN, performed well when the psoCNN algorithm generated it. Compared with 82.53% in IDECNN, psoCNN provided 83.49% accuracy. The suggested method outperformed classification accuracy with the CNN model RECT_CNN and RECT-I_CNN. RECT_CNN gave 72.60% and 76.60% accuracy, respectively, for the psoCNN and IDECNN models. RECT-I_CNN, on the other hand, produced 74.68% classification accuracy for psoCNN versus 79.49% for IDECNN. Figure 12 depicts some of the sample test cases performed by MNIST_CNN in the psoCNN model and MRD_CNN in the IDECNN model, which provided the highest classification accuracy among all CNN architectures.

Table 14. Overview of the COVID-19 X-ray dataset

Class name	Input size	No. of training	No. of validation	No. of test
COVID-19	180 × 180	2,531	723	362
Non-COVID-19	180 × 180	7,134	2,038	1,020

The confusion matrices for the pneumonia chest X-ray dataset generated using best CNN architectures through our proposed IDECNN are shown in Figure 13.

In each confusion matrix in the figure, the top left shows the number of images correctly predicted as normal cases (true positive), and the bottom right shows the correctly predicted number of images as cases of pneumonia (true negative). On the other hand, the top right denotes normal cases incorrectly predicted as pneumonia (false positive). The bottom left indicates the number of incorrectly predicted images for normal cases, but in reality, they are images for pneumonia cases (false negative). In addition to showing the confusion matrices, we present the classification report in Table 10 in terms of precision, recall, and F1 score for normal and pneumonia classes of the chest X-ray dataset.

Precision, recall, and F1 score are calculated as follows:

$$\text{Precision} = \frac{TruePositive}{TruePositive + FalsePositive} \quad \text{(Equation 6)}$$

$$\text{Recall} = \frac{TruePositive}{TruePositive + FalseNegative} \quad \text{(Equation 7)}$$

$$\text{F1} - \text{score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad \text{(Equation 8)}$$

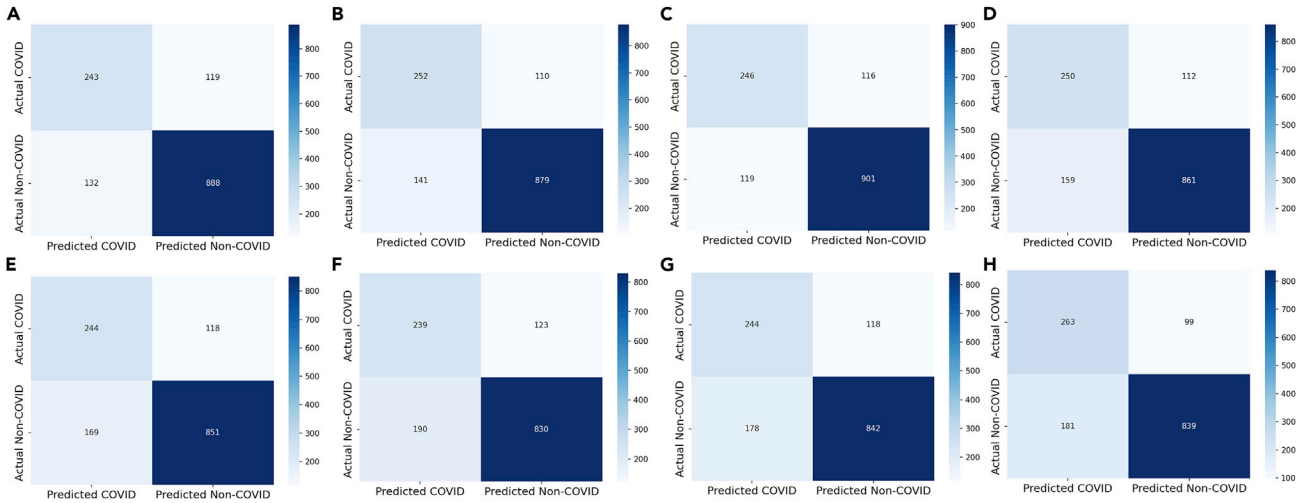


Figure 16. The obtained confusion matrices for the COVID-19 X-ray dataset using the eight best generated CNN architectures of the proposed IDECNN

(A–H) (A) MNIST_CNN, (B) MBI_CNN, (C) MRB_CNN, (D) MRD_CNN, (E) MRDBI_CNN, (F) CS_CNN, (G) RECT_CNN, and (H) RECT-I_CNN.

In addition, we randomly divided D_{train}, D_{valid}, and D_{test} into two more scenarios and analyzed the performance of the best generated CNN models on the pneumonia chest X-ray dataset, as shown in Table 11.

In scenario 1, we split D_{train}, D_{valid}, and D_{test} for normal X-ray images as 1,108, 316, and 159, respectively, whereas for pneumonia images, they were split as 2,991, 854, and 428, respectively. Similarly, we divided D_{train}, D_{valid}, and D_{test} into 791, 474, and 318 for normal X-ray images and 2,136, 1,281, and 856 for pneumonia images, respectively, for scenario 2. The confusion matrices, obtained precision, recall, F1 score, and model accuracy of all optimal CNN architectures for scenario 1 are shown in Figure 14 and Table 12, respectively.

Similarly, the same is presented in Figure 15 and Table 13, respectively, for scenario 2.

In both scenarios, the optimal CNN architecture MRD_CNN model had the highest classification accuracy compared with all other generated CNN models.

In addition to the pneumonia chest X-ray dataset, we also looked at how well the best CNN architecture generated by IDECNN for each dataset worked on another real-world application, the COVID-19 X-ray[63] dataset. An overview of this dataset is presented in Table 14.

There are two classes in the COVID-19 X-ray dataset: one class for chest X-ray images of individuals with COVID-19 and another class for non-COVID-19 individuals. This dataset has a total of 13,808 images with 9,665 training samples, 2,761 validation samples, and 1,382 test samples. The number of training, validation, and test samples for the COVID-19 class are 2,531, 723, and 362, respectively, whereas non-COVID-19 includes 7,134, 2,038, and 1,020 examples. The input size of images is 180 × 180. All other parameters used in this experiment were the same as those used previously in the chest X-ray experiment. The confusion matrices produced by each optimal CNN model are shown in Figure 16.

The classification reports in terms of precision, recall, and F1 score for COVID-19 and non-COVID-19 classes are presented in Table 15, including model classification accuracy.

In the case of classification accuracy, we can see that MNIST_CNN achieved an accuracy of 81.81%, which is close to the accuracy of MBI_CNN, 81.84%. MRD_CNN, on the other

Table 15. The obtained precision, recall, F1 score, and model accuracy using the models MNIST_CNN, MBI_CNN, MRD_CNN, MRDBI_CNN, CS_CNN, RECT_CNN, and RECT-I_CNN for the COVID-19 dataset

	COVID-19			Non-COVID-19			
Optimal CNN architecture	Precision	Recall	F1 score	Precision	Recall	F1 score	Model accuracy
MNIST_CNN	0.65	0.67	0.66	0.88	0.87	0.87	81.81%
MBI_CNN	0.64	0.70	0.69	0.89	0.86	0.87	81.84%
MRB_CNN	0.67	0.68	0.67	0.89	0.88	0.88	83.00%
MRD_CNN	0.61	0.69	0.65	0.88	0.84	0.86	83.39%
MRDBI_CNN	0.59	0.67	0.63	0.88	0.83	0.85	79.23%
CS_CNN	0.56	0.62	0.59	0.87	0.81	0.84	77.35%
RECT_CNN	0.58	0.67	0.62	0.88	0.83	0.85	78.58%
RECT-I_CNN	0.59	0.72	0.65	0.89	0.82	0.85	79.74%

hand, achieved an accuracy rate of 83.39%, slightly higher than MRB_CNN's 83%, and had the highest accuracy among all other optimal CNN models. In the case of the MRDBI_CNN architecture, it achieved a classification accuracy of 79.23%. Finally, CS_CNN had a model accuracy of 77.35%, and RECT_CNN and RECT-I_CNN had model accuracies of 78.58% and 79.74%, respectively. Therefore, this case study defines the number of acceptable CNN architectures that performed well in terms of classification in pneumonia and COVID-19 X-ray images.

Conclusions

The paper proposed an improved DE-based approach to design optimal CNN architectures, IDECNN, for classifying image datasets. Here, each individual served as an architecture consisting of layer types and arrangements of layers that constitute a CNN model. Individuals were encoded with a variable-length encoding scheme to achieve flexibility in architectural depth. A refinement strategy was proposed to calculate the difference between two CNN architectures and designing a heuristic mechanism to make the mutation and crossover operator in the framework of DE coherent. Each generated CNN model architecture was evaluated through classification error on eight widely used benchmark image datasets. The results obtained using the proposed IDECNN demonstrated superior performance compared with 20 state-of-the-art models, including handcrafted and evolution-based CNN models, on seven of eight datasets in terms of mean classification error.

An ablation study of the proposed method was performed in terms of scaling factor (*F*), crossover ratio (*CR*), number of generations, population size, and training epoch number on the CS dataset. The IDECNN method restricted ablation studies to the CS dataset because of limited computational resources. We transferred the best generated CNN model architectures of the eight datasets attained through IDECNN to classify normal and pneumonia chest X-ray images and compared the model accuracy with the existing psoCNN model. We also experimented with more random splitting of training, validation, and test samples for fair results on the same chest X-ray dataset. Finally, we transferred the same generated CNN models to another, more popular real-life application, the COVID-19 X-ray medical image dataset, to check the effectiveness of the proposed CNN models. The results obtained from the pneumonia and COVID-19 datasets demonstrated the significant performance of CNN models designed through our proposed algorithm.

In future work, the proposed IDECNN can be implemented to design a block-based CNN model architecture and tested on the complex CIFAR dataset to investigate its effectiveness. This model can also be applied to more complex biomedical image datasets, such as breast cancer, skin cancer, and OASIS brain MRI, among many others, for classification purposes.

EXPERIMENTAL PROCEDURE

Resource availability

Lead contact

Requests for further information can be directed to the lead contact, Z.Z. (zhongming.zhao@uth.tmc.edu).

Materials availability

The study did not generate new unique reagents.

Data and code availability

The codes are publicly available on Zenodo (https://doi.org/10.5281/zenodo.6567750Ω).

ACKNOWLEDGMENTS

The authors thank Dr. Irmgard Willcockson for professional English editing services. Z.Z. was partially supported by the Cancer Prevention and Research Institute of Texas (CPRIT 180734). The funders did not participate in the study design, data analysis, decision to publish, or preparation of the manuscript.

AUTHOR CONTRIBUTIONS

Concept formation and writing of the original draft were conducted by A.G., N.D.J., and S.M. Overall design of the methodology and experiments were performed by A.G., N.D.J., and S.M. The final draft and revisions were made by S.M. and Z.Z.

DECLARATION OF INTERESTS

The authors declare no competing interests.

Received: April 10, 2022
Revised: June 4, 2022
Accepted: July 13, 2022
Published: August 24, 2022

REFERENCES

1. Gaur, L., Bhandari, M., Razdan, T., Mallik, S., and Zhao, Z. (2022). Explanation-driven deep learning model for prediction of brain tumour status using mri image data. Front. Genet. *13*, 822666.
2. Sharma, P., Balabantaray, B.K., Bora, K., Mallik, S., Kasugai, K., and Zhao, Z. (2022). An ensemble-based deep convolutional neural network for computer-aided polyps identification from colonoscopy. Front. Genet. *13*, 844391.
3. Karri, M., Annavarapu, C.S.R., Mallik, S., Zhao, Z., and Acharya, U.R. (2022). Multi-class nucleus detection and classification using deep convolutional neural network with enhanced high dimensional dissimilarity translation model on cervical cells. Biocybern. Biomed. Eng. *42*, 797–814.
4. Voulodimos, A., Doulamis, N., Doulamis, A., and Protopa- padakis, E. (2018). Deep learning for computer vision: a brief review. Comput. Intell. Neurosci. *2018*, 7068349–7068361.
5. Nassif, A.B., Shahin, I., Attili, I., Azzeh, M., and Shaalan, K. (2019). Speech recognition using deep neural networks: a systematic review. IEEE Access *7*, 19143–19165. https://doi.org/10.1109/ACCESS.2019.2896880.
6. Pei, G., Hu, R., Jia, P., and Zhao, Z. (2021). Deepfun: a deep learning sequence-based model to decipher non-coding variant effect in a tissue-and cell type-specific manner. Nucleic Acids Res. *49*, W131–W139.
7. Xu, H., Jia, P., and Zhao, Z. (2021). Deepvisp: deep learning for virus site integration prediction and motif discovery. Adv. Sci. *8*, 2004958.
8. Umer, S., Mondal, R., Pandey, H.M., and Rout, R.K. (2021). Deep features based convolutional neural network model for text and non-text region segmentation from document images. Appl. Soft Comput. *113*, 107917.
9. Umer, S., Mohanta, P.P., Rout, R.K., and Pandey, H.M. (2021). Machine learning method for cosmetic product recognition: a visual searching approach. Multimed. Tool. Appl. *80*, 34997–35023.
10. Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (2001). Gradient-based learning applied to document recognition. Proc. IEEE *86*, 2278–2324.
11. Simonyan, K., and Zisserman, A. (2015). Very deep convo- lutional networks for large-scale image recognition. Preprint at axRiv. https://doi.org/10.48550/arXiv.1409.1556.
12. Krizhevsky, A., and Hinton, G. (2009). Learning Multiple Layers of Features from Tiny Images (Citeseer).

13. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9.

14. He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778. https://doi.org/10.1109/CVPR.2016.90.

15. Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., and Wang, X. (2022). A comprehensive survey of neural architecture search: challenges and solutions. ACM Comput. Surv. *54*, 1–34.

16. Bandyopadhyay, S., Mallik, S., and Mukhopadhyay, A. (2014). A survey and comparative study of statistical tests for identifying differential expression from microarray data. IEEE ACM Trans. Comput. Biol. Bioinf *11*, 95–115.

17. Mallik, S., and Zhao, Z. (2020). Graph- and rule-based learning algorithms: a comprehensive review of their applications for cancer type classification and prognosis using genomic data. Briefings Bioinf. *21*, 368–394.

18. Elsken, T., Metzen, J.H., and Hutter, F. (2019). Neural architecture search: a survey. J. Mach. Learn. Res. *20*, 1–21.

19. Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G.G., and Tan, K.C. (2021). A survey on evolutionary neural architecture search. IEEE Transact. Neural Networks Learn. Syst. 1–21. https://doi.org/10.1109/TNNLS.2021.3100554.

20. White, C., Neiswanger, W., Nolen, S., and Savani, Y. (2020). A study on encodings for neural architecture search. Adv. Neural Inf. Process. Syst. *33*, 20309–20319.

21. Ahmad, M., Abdullah, M., and Han, D. (2019). A novel encoding scheme for complex neural architecture search. In 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), pp. 1–4. https://doi.org/10.1109/ITC-CSCC.2019.8793329.

22. Kang, D., and Ahn, C.W. (2019). Efficient neural network space with genetic search. In International Conference on Bio-Inspired Computing: Theories and Applications, pp. 638–646.

23. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In International Conference on Machine Learning, pp. 2902–2911.

24. Zheng, X., Ji, R., Wang, Q., Ye, Q., Li, Z., Tian, Y., and Tian, Q. (2020). Rethinking performance estimation in neural architecture search. In IEEE Conference on Computer Vision and Pattern Recognition, pp. 11356–11365.

25. Sun, Y., Sun, X., Fang, Y., Yen, G.G., and Liu, Y. (2021). A novel training protocol for performance predictors of evolutionary neural architecture search algorithms. IEEE Trans. Evol. Comput. *25*, 524–536. https://doi.org/10.1109/TEVC.2021.3055076.

26. Tan, H., Cheng, R., Huang, S., He, C., Qiu, C., Yang, F., and Luo, P. (2021). Relativenas: Relative neural architecture search via slow-fast learning. IEEE Transact. Neural Networks Learn. Syst. 1–15. https://doi.org/10.1109/TNNLS. 2021.3096658.

27. Real, E., Aggarwal, A., Huang, Y., and Le, Q.V. (2019). Regularized evolution for image classifier architecture search. AAAI Conference on Artificial Intelligence *33*, 4780–4789.

28. Sun, Y., Xue, B., Zhang, M., and Yen, G.G. (2020). Evolving deep convolutional neural networks for image classification. IEEE Trans. Evol. Comput. *24*, 394–407.

29. Suganuma, M., Shirakawa, S., and Nagao, T. (2018). A genetic programming approach to designing convolutional neural network architectures. In 27th International Joint Conference on Artificial Intelligence, pp. 5369–5373.

30. Byla, E., and Pang, W. (2019). Deepswarm: Optimising Convo- Lutional Neural Networks Using Swarm Intelligence (UK Workshop on Computational Intelligence), pp. 119–130.

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

32. Wang, B., Sun, Y., Xue, B., and Zhang, M. (2018). A hybrid differential evolution approach to designing deep convo- lutional neural networks for image classification. In Australasian Joint Conference on Artificial Intelligence, pp. 237–250.

33. Das, S., Mullick, S.S., and Suganthan, P.N. (2016). Recent advances in differential evolution-an updated survey. Swarm Evol. Comput. *27*, 1–30.

34. Al-Dabbagh, R.D., Neri, F., Idris, N., and Baba, M.S. (2018). Algorithmic design issues in adaptive differential evolution schemes: review and taxonomy. Swarm Evol. Comput. *43*, 284–311.

35. Segredo, E., Lalla-Ruiz, E., Hart, E., and Voß, S. (2020). A similarity-based neighbourhood search for enhancing the balance exploration-exploitation of differential evolution. Comput. Oper. Res. *117*, 104871.

36. Awad, N., Mallik, N., and Hutter, F. (2020). Differential evolution for neural architecture search. In 1st workshop on neural architecture search(@ICLR'20).

37. Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. (2021). A survey of convolutional neural networks: analysis, applications, and prospects. IEEE Transact. Neural Networks Learn. Syst. 1–21. https://doi.org/10.1109/TNNLS.2021. 3084827.

38. Pei, G., Hu, R., Dai, Y., Manuel, A.M., Zhao, Z., and Jia, P. (2021). Predicting regulatory variants using a dense epige- nomic mapped cnn model elucidated the molecular basis of trait-tissue associations. Nucleic Acids Res. *49*, 53–66.

39. Li, B., Pei, G., Yao, J., Ding, Q., Jia, P., and Zhao, Z. (2021). Cell-type deconvolution analysis identifies cancer-associated myofibroblast component as a poor prognostic factor in multiple cancer types. Oncogene *40*, 4686–4694.

40. Rawat, W., and Wang, Z. (2017). Deep convolutional neural networks for image classification: a comprehensive review. Neural Comput. *29*, 2352–2449.

41. Jogin, M., Madhulika, M.S., Divya, G.D., Meghana, R.K., and Apoorva, S. (2018). Feature extraction using convolution neural networks (cnn) and deep learning. In 2018 3rd IEEE International Conference on Recent Trends in Electronics (Information Communication Technology (RTEICT)), pp. 2319–2323. https://doi.org/10.1109/RTEICT42901.2018.9012507.

42. Das, S., and Suganthan, P.N. (2011). Differential evolution: a survey of the state-of-the-art. IEEE Trans. Evol. Comput. *15*, 4–31. https://doi.org/10.1109/TEVC. 2010.2059031.

43. Xie, L., and Yuille, A.L. (2017). Genetic cnn. In IEEE International Conference on Computer Vision (ICCV), pp. 1388–1397.

44. Dong, J., Zhang, L., Hou, B., and Feng, L. (2020). A memetic algorithm for evolving deep convolutional neural network in image classification. In 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 2663–2669. https://doi.org/10.1109/SSCI47803.2020.9308162.

45. Wang, B., Sun, Y., Xue, B., and Zhang, M. (2018). Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. https://doi.org/10.1109/CEC.2018.8477735.

46. Wistuba, M., Rawat, A., and Pedapati, T. (2019). A survey on neural architecture search. Preprint at arXiv. https://doi.org/10.48550/arXiv.1905.01392.

47. Xu, Y., and Goodacre, R. (2018). On splitting training and validation set: a comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. J. Anal. Test. *2*, 249–262.

48. Chang, O., Flokas, L., and Lipson, H. (2020). Principled weight initialization for hypernetworks. In International Conference on Learning Representations.

49. Agarap, A.F. (2018). Deep learning using rectified linear units (relu). Preprint at arXiv. https://doi.org/10.48550/arXiv.1803.08375.

50. Kingma, D.P., and Ba, J.L. (2015). Adam: a method for stochastic optimization. In International Conference on Learning Representations.

51. Zhou, Y., Wang, X., Zhang, M., Zhu, J., Zheng, R., and Wu, Q. (2019). Mpce: a maximum probability based cross entropy loss function for neural network classification. IEEE Access *7*, 146331–146341. https://doi.org/10.1109/ACCESS.2019. 2946264.

52. Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Ben- gio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In Proceedings of the 24th international conference on Machine learning, pp. 473–480.

53. Sohn, K., and Lee, H. (2012). Learning invariant representations with local transformations. In Proceedings of the 29th International Conference on Machine Learning, pp. 1339–1346.

54. Sohn, K., Zhou, G., Lee, C., and Lee, H. (2013). Learning and selecting features jointly with point-wise gated Boltzmann machines. In 30th International Conference on Machine Learning, pp. 217–225.

55. Chan, T.H., Jia, K., Gao, S., Lu, J., Zeng, Z., and Ma, Y. (2015). Pcanet: a simple deep learning baseline for image classification? IEEE Trans. Image Process. *24*, 5017–5032. https://doi.org/10.1109/TIP.2015.2475625.

56. Gamperle, R., Müller, S.D., and Koumoutsakos, P. (2002). A parameter study for differential evolution. Advances in intelligent systems, fuzzy systems, evolutionary computation *10*, 293–298.

57. Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., and Lew, M.S. (2016). Deep learning for visual understanding: a review. Neurocomputing *187*, 27–48.

58. Ioffe, S., and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pp. 448–456.

59. Guo, D., Wang, X., Gao, K., Jin, Y., Ding, J., and Chai, T. (2022). Evolutionary optimization of high-dimensional multi- objective and many-objective expensive problems assisted by a dropout neural network. IEEE Trans. Syst. Man Cybern. Syst. *52*, 2084–2097. https://doi.org/10.1109/TSMC.2020.3044418.

60. Kermany, D., Zhang, K., and Goldbaum, M. (2018). Labeled optical coherence tomography (oct) and chest x-ray images for classification. Mendeley data *2*.

61. Ruby, U., and Yendapalli, V. (2020). Binary cross entropy with deep learning technique for image classification. Int. J. Adv. Trends Comput. Sci. Eng. *9*, 5393–5397.

62. Pratiwi, H., Windarto, A.P., Susliansyah, S., Aria, R.R., Susilowati, S., Rahayu, L.K., Fitriani, Y., Merdekawati, A., and Rahadjeng, I.R. (2020). Sigmoid activation function in selecting the best model of artificial neural networks. J. Phys, Conf. Ser. *1471*, 012010.

63. Rahman, T., Khandakar, A., Qiblawey, Y., Tahir, A., Kiranyaz, S., Abul Kashem, S.B., Islam, M.T., Al Maadeed, S., Zughaier, S.M., Khan, M.S., and Chowdhury, M.E.H. (2021). Exploring the effect of image enhancement techniques on covid-19 detection using chest x- ray images. Comput. Biol. Med. *132*, 104319.