Evolution of Generative Adversarial Networks for the Synthesis of Biomedical Images

JUAN ANTONIO RODRÍGUEZ DE LA CRUZ

Biotechnology Engineer

Universidad Veracruzana

Director: Dr. Héctor Gabriel Acosta Mesa
Co-Director: Dr. Efrén Mezura Montes

A thesis submitted in fulfilment of
the requirements for the degree of
Master in Artificial Intelligence

Artificial Intelligence Research Institute
University of Veracruz
Xalapa-Enríquez, Veracruz, México

June 2021

Abstract

The use of biomedical images for the training of various Deep Learning (DL) systems oriented to health has reported a competitive performance. However, DL needs a large number of images for a correct generalization and, particularly in the case of biomedical images, these can be scarce. Generative Adversarial Networks (GANs) as Data Augmenting tools have reaped significant results to improve performance in tasks that involve the use of this kind of images. However, the architectural design of these generative models in the biomedical image area has been usually relegated to the expertise of researchers. Moreover, GANs are affected by training instability that may lead to poor quality results. One solution to these problems is neuroevolution, the use of evolutionary computation to create neural networks. This thesis presents two versions of a neuroevolution algorithm based on Particle Swarm Optimization for the design and training of GANs for the generation of biomedical Chest X-Ray (CXR) images. The proposed approach allows having a swarm of GANs topologies, where each one of them grows progressively while being trained at the same time. The fitness value is based on the Frechet Inception Distance (FID), a metric designed to measure the similarity in quality and diversity between sets of real and synthetic images. The proposed algorithm is able to obtain better FID results compared to handcrafted GANs for the synthesis of CXR images. It also allows to improve classification tasks through the use of synthetic images.

Acknowledgements

To the Mexican National Council of Science and Technology (CONACyT) for supporting me, with CVU number 1001447, through a scholarship to pursue postgraduate studies in the master's degree in Artificial Intelligence.

To the University of Veracruz (UV).

To the Artificial Intelligence Research Institute (IIIA).

To the Research Group on Computer Vision, Neural Networks, Evolutionary Computation and their Applications (COVNNEC-App) of which I have the honor to be a part.

To the UNAM-CViCom-COVID-19 system and the proyect PAPIIT IV100420 for supporting with the experimentation carried out.

To Ph.D. Héctor Gabriel Acosta Mesa and Ph.D. Efrén Mezura Montes for their support in conducting this research.

To Ph.D. Fernando Arámbula Cosío for their support in the research stay at IIMAS-UNAM.

To my family.

To Xitlali.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Overview

1.1.1 Deep Learning and Biomedical Imaging

In recent decades, the emergence and rise of machine learning (ML) implementations in the field of medicine, for purposes of diagnosis, processing of biological data and support in the work of medical specialists, has allowed optimizing the waiting times and accuracy rates for the detection of various diseases and disorders (Shukla et al. 2020). Among the main areas with countless approaches and proposed algorithms there is Computer Vision (CV), whose models can be used in biomedical imaging for applications ranging from segmentation of areas of interest, to classification for diagnosis purposes. The patient's information obtained from these images, improve the medical efforts for healthcare. Among these models, those based on Deep Learning (DL) stand out, which in recent years have shown significant improvements in tasks such as the aforementioned (Litjens et al. 2017, Bakator and Radosav 2018, Mohapatra et al. 2021) derived from the properties of this type of computing, being robust and capable of developing highly complex hypotheses.

However, an inherent characteristic of DL models, not only in their application in the field of computer vision, it is the need for a vast and diverse amount of training data, in order to perform an adequate generalization and avoid overfitting which is common is these complex models. Additionally, in the area of supervised machine learning, datasets with a high imbalance of instances between the various classes could result in models with high biases and low performances (Buda et al. 2018); this is even more present when it comes to biomedical images. This kind of data is scarce due to issues such as the privacy of patients, test's high costs, or even the patient's exposure to radiation (Guibas et al. 2017). Furthermore, the small available datasets have a marked imbalance of classes product of the procurement environment, since positive cases of diseases or disorders in patients are more common than healthy ones.

1.1.2 Data Augmentation

Data Augmentation (DA) is a set of techniques that artificially expand the size of a dataset for training ML models (Abdollahi et al. 2020). In recent years this concept has been focused on expanding images datasets for applications in DL models for computer vision tasks, improving the generalization ability of these models in the most difficult challenges. These techniques, which modify the images inside the original training set are applied under the assumption that more information can be extracted from the original dataset through augmentation procedures. These augmentations artificially enlarge the training dataset size by data warping or oversampling (Shorten and Khoshgoftaar 2019). The previously mentioned adversities in medical imaging have been overcome by DA approaches, demonstrating the improvement in DL models performance through this set of tools (Perez and Wang 2017). However, these classic techniques have been limited in improving models performance. This is because these approaches only edit original images with various angles, sizes, and filters. For this reason the efforts of the last decade have turned around developing more and better techniques to support model training with small datasets and unbalanced classes (Buda et al. 2018). One of the main branches of research has been the use of generative models for the synthesis of images that replicate the diversity and quality of the original sets and thereby enrich the amount of available data; one of these DL-based models with the greatest popularity and prestige in recent years are the Generative Adversarial Networks (GANs) (Shin et al. 2018).

1.1.3 Generative Adversarial Networks

The GANs family has its origin in (Goodfellow et al. 2014) and have demonstrated to be useful as generative models for various computer vision and digital imaging tasks (Pan et al. 2019). Its operation consists of two image processing models, generally Convolutional Neural Networks (CNN); one is in charge of generating images that are highly similar to the set of real images it was trained with, known as *Generator*; the second model, named *Discriminator*, performs the function of differentiating between the images that belong to the original set and those artificially created by the Generator. These models with opposing goals face each other during several training cycles, forcing both models towards continuous improvement and adaptation; the generator improves the quality of its creations and the discriminator improves its classification mechanisms (similar to reinforcement learning). At the end of the critical-creation cycles, the generator is ready to synthesize images that are highly similar to the real set (Huang et al. 2018). These new images are intended to enrich sets that have few instances or with an unbalance of classes, with the purpose to help improving the performance of other DL models and their applications.

Although there is no general metric to quantify the quality of the results of the GANs, the Fréchet Inception Score (FID) has been used in recent years as *state-of-the-art* metric to evaluate the similarity between the images of the real set and the synthetic images through the similarity of the features, extracted by a pre-trained CNN, of both sets of images (Lucic et al. 2018).

Despite the remarkable performances obtained by the GANs in the field of biomedical imaging (Yi et al. 2019) and many other CV areas (Wang et al. 2019), they suffer from their own problems that affect the training stability. These issues manifest very frequently and represent a great challenge in obtaining useful generators for various areas, leading to a reduction in quality and diversity of the generated images (Neyshabur et al. 2017). These common problems are:

- **Mode Collapse**: The situation in which the generator can only synthesize a small subset of images of the complete distribution since the training did not allow to generalize the richness of variants of the original images.
- **Vanishing Gradient**: Originated when the discriminator or the generator becomes powerful enough to cause an irreversible imbalance in training and by not using adequate cost functions that enable obtaining adequate learning gradients. This prevents the opposite network to improve its performances, causing a stalemate.

Therefore, in recent years, research has focused on solving these difficulties. Some of the approaches used to improve training stability have been the improvement of GAN architectures (Radford et al. 2015) or the use of alternative cost functions to avoid training imbalance problems (Arjovsky et al. 2017, Mao et al. 2017, Miyato et al. 2018). Moreover, Evolutionary Computing techniques have had a special growth in this field focused on training instability and the design and automatic training of GANs (Costa et al. 2020b).

1.1.4 Evolutionary Computing and GANs

The Evolutionary Computing (EC) bases its operation on the generation of a population of solutions in an environment that is constantly evaluated to drive potential solutions towards constant adaptation to their environment and continuous improvement (Eiben and Smith 2015). The use of these bio-inspired methods for the design and automatic training of artificial neural networks is known as neuroevolution. Evolutionary computing and neuroevolution techniques have been proposed in order to solve the training drawbacks for GANs (Costa et al. 2020b). These methods are based on the manipulation of populations of GANs that will be evolved, to choose those networks that are better at handling or avoiding common training problems.

1.1.5 Case Study: Chest X-ray Images.

Chest X-rays are a type of biomedical images with multiple uses in the medical area. Many of these applications have used Deep Learning. DL-based computer vision models are used to detect the attributes of these diseases and design appropriate models to support the biomedical area. Its most prominent applications are the diagnosis of respiratory or heart diseases, or the segmentation of areas of interest (Rajpurkar et al. 2017, Gordienko et al. 2018, Stirenko et al. 2018, Baltruschat et al. 2019, Bhandary et al. 2020, Jain et al. 2021, Ismael and Şengür 2021). For this reason, having quality and diverse imaging data sets of these diseases would improve the performance of DL-based models.

Inspired by the numerous successes of the evolutionary computing in neuroevolution on different areas of DL (Stanley et al. 2019) and specially in the recent efforts focused on GANs as well as the proven utility of GANs in increasing performance in applications in the biomedical imaging area (Shin et al. 2018, Sorin et al. 2020,Kazeminia et al. 2020), this work presents two designed versions of a neuroevolution algorithm based on Swarm Intelligence, a field of EC hitherto unexplored in the neuroevolution of GANs to evolve this type of networks that allow to synthesize chest X-ray images of different pathologies, as a case study of biomedical imaging. These algorithms allows to design and train GANs architectures automatically, aiding to prevent training instability problems to synthesize biomedical images.

1.2 Problem Definition

The design of Generative Adversarial Networks for the synthesis of data similar to real sets that increase the available amount of data is an research problem that requires recursive tuning and search by deep learning experts. Also, the common training instability problems that affect this kind of models could reduce the final quality of the new data.

That is the reason why it is necessary to improve the design and training mechanisms to obtain GANs that effectively increase the set of available data for various applications, including the amount of biomedical images for use in DL medical systems.

1.3 Research Proposal

The proposal of the present thesis is to design a neuroevolution algorithm to automatically design GANs architectures and also train them effectively to generate chest X-ray images.

The work will cover the following:

- **Search algorithm**: Employing, to the best of the author's knowledge for the first time a Swarm Intelligence algorithm for architecture search and training of GANs.
- **Biomedical imaging**: Applying evolved GANs models in biomedical images, to the best of the author's knowledge, for the first time, in order to support the growth of available datasets.
- **Low data regime**: Using a set of images with few instances (compared to large benchmark datasets previously used in GANs neuroevolution) that reflects the real conditions in which the needs of data augmentation are found.

1.4 Justification

The artificial augmentation of biomedical images is a field in full swing and exploration, the use of the images obtained through this Data Augmentation approach has allowed the improvement in the performance of multiple Deep Learning models and also helped the biomedical field in various activities of diagnosis and prevention e.g. CNN classification. Within this field, the use of Generative Adversarial Networks (GANs) as a generative model has allowed to increase the performances in DL systems that use their synthetic images to increase or balance datasets. However, GANs have problems inherent to their adversarial nature, often obtaining results that do not meet the necessary quality and diversity requirements that resemble actual image distributions. Furthermore, the design of this type of network is not standardized as it is based on empirical tests and on the experience of researchers, thus obtaining networks that generally cannot be exported to other domains or purposes.

Such problems have been partially overcome due to the implementation of evolutionary computing approaches in the design and training of GANs, allowing the obtaining of high-quality results. In addition, within this already small set of algorithms, few are responsible for the automatic design of the GANs topology, an approach that would save time and resources in empirical design tests and thus adapt to the type of images that they need to be synthesized. Within these previous algorithms, an unexplored field has been Swarm Intelligence, a branch of evolutionary computation, which provides algorithms with rapid convergence in the search for solutions, a feature that can help reduce search times, of great importance taking into account the high resources necessary to train potential solution networks.

Besides, there is a large gap between the previously mentioned neuroevolution approaches, none of which focuses on the generation of biomedical images, a field which would benefit from the automation of the design of GANs architectures and greater stability of the training that allows obtaining better qualities of results for the various uses of these images. Among the different branches of biomedical imaging, one type of image that is important, but which has few implementations for its synthesis, is chest X-ray (CXR) images. These have currently gained relevance due to the recent respiratory disease COVID-19 situation in which this type of images is used for diagnostic purposes using DL-based models but there are previous studies focused on other types of pulmonary and cardiac pathologies.

Due to the above, this thesis presents the use of CXR images as a case study, since in recent months the public datasets of CXR images of pneumonia by COVID-19 have increased, which enriches the quantity and diversity of this type of already existing images of healthy patients or with pneumonia. This provides for an appropriate real case to design and test a neuroevolution algorithm in the field of biomedical imaging that allows designing GANs architectures and at the same time allows an effective training through its progressive growth. This way, the instability training problems of these networks can be avoided or reduced, thus obtaining synthetic images with the same richness in quality and diversity as the real set.

1.5 Hypothesis

The Generative Adversarial Networks obtained by means of the designed neuroevolution algorithm will be able to:

- **Hypothesis 1**: Generate chest X-ray images with high similarity to the set of real images (measured by FID), preserving the quality and diversity of that, what would be an indicator of the avoidance or reduction of training instability problems.
- **Hypothesis 2**: Obtain better FID results than handcrafted GANs for the synthesis of CXR images of pneumonia by COVID-19.
- **Hypothesis 3**: Improve the classification, in terms of accuracy, of CXR images in CNN, with respect to unbalanced sets, using artificially balanced sets using synthetic images obtained from evolved GANs.

1.6 Overall Objective

To develop an neuroevolution algorithm based on Swarm Intelligence that allows the design and training of Generative Adversarial Networks (GANs) architectures for the synthesis of chest X-ray images with high similarity in quality and diversity than the real sets.

1.7 Specific Objectives

(1) To process chest X-ray images obtained from various public image datasets, in order to obtain images with high contrast and good resolution.

(2) To design a Swarm Intelligence (SI) algorithm that automatically designs the architecture of the GAN that will be used for the synthesis of biomedical images. In addition, through the same algorithm, the GANs will be trained at the same time and instability problems in training will be avoided or reduced.

(3) To generate and train various GANs using the designed SI algorithm and to record information from the performed experiments.

(4) To synthesize sets of images using the generators obtained with the designed algorithm and using the information collected from the experiments to corroborate the correct functioning and convergence of the algorithm.

(5) To evaluate the similarity of the synthesized images with the set of real images using quantitative and qualitative tests of the quality and diversity of the generated images.

(6) To compare the quality of the synthetic images obtained through the evolution of GANs with respect to the *state-of-the-art* works for synthesis of CXR images using handcrafted GANs.

(7) To evaluate the improvement in the classification task with Convolutional Neural Networks using the synthesized images to artificially balance the CXR image sets and thereby demonstrate the usefulness of the synthetic data generated.

1.8 Contributions

The contributions of the present thesis are:

- Designing for the first time, to the best of the author's knowledge, a Swarm Intelligence algorithm for the architecture design and training of GANs. Previous works of GANs neuroevolution do not use this approach.
- Using for the first time, to the best of the author's knowledge, a neuroevolution algorithm for GANs in the field of biomedical images.
- Use for the first time, to the best of the author's knowledge, an evolutionary progressive growth approach to GANs. Additionally, to being also the first time, to the best of the author's knowledge, of the use of progressive growth in the synthesis of CXR images.
- Using a low amount of data (compared to large benchmark datasets previously used in GANs neuroevolution) that represents the characteristic of dealing with limited biomedical imaging.

1.9 Thesis Content

The rest of the parts contained in the thesis are organized as follows:

Chapter 2 contains the theoretical framework of the main topics of the thesis, such as GANs, Evolutionary Computing and Biomedical Imaging.

Chapter 3 presents a review of the literature in the fields of progressive augmentation of GANs, neuroevolution of GANs and synthesis of CXR images using GANs.

Chapter 4 details the parts and operation of the proposed neuroevolution algorithm.

Chapter 5 shows the methodology of implementation and experimentation of the proposed algorithm, as well as the results of these experiments and their respective discussion.

Chapter 6 presents the conclusions reached, the limitations of the work and future guidelines for its improvement.

CHAPTER 2

Theoretical Framework

This chapter presents the theoretical foundations of this thesis. Three main topics are addressed: Generative Adversarial Networks, Evolutionary Computing and Biomedical Imaging.

2.1 Artificial Neural Networks

Neural Networks (NN) also named Artificial Neural Networks, are Machine Learning (ML) models that mimic the working principles of the human brain. NN consist of many simple information processing units called *neurons*.

2.1.1 Perceptron, Multi-Layer Perceptrons and Deep Learning.

A neuron, also called **Perceptron**, is the simplest NN, development in 1958 by Frank Rosenblatt (1958) using the ideas introduced by McCulloch and Pitts in (1943). The neuron, N_j, receives n input signals (x_i; $i=1,...,n$), through input connections, as the dendrites in biological neurons. Each connection has a weight w_{ij} to modulate the corresponding input x_i. The weighted sum of the inputs added with a bias term b_j (used to control the net input to the activation function) is mapped via an activation function $\varphi(\cdot)$ to generate the output y_j of the neuron, this is represented mathematically as follows:

$$y_j = \varphi(z_j) = \varphi(\sum_{i=1}^{n} x_i w_{ij} + b_j) \tag{2.1}$$

A visual representation of the Perceptron is found on the left in Figure 2.1.

A **Multi-Layer Perceptron** (MLP), also known as Multilayer Neural Network (MNN), is a set of neurons organized into layers. Each layer has one or more neurons. The leftmost and rightmost layer of the network are called the input layer and the output layer, respectively, and the intermediate layers are called the hidden layers. The later ones receive their name because the computations performed are not visible to the user. The neurons in the input layer do not perform any processing on the received data, they just pass it to the next layer as inputs. This new layer processes the inputs and generate the outputs and then passes them to the next layer of neurons to repeat the process. The architecture of MLPs is called *feed-forward* networks, because successive layers feed into one another in the forward direction from input to output. The *feed-forward* architecture assumes that all neurons in one layer are connected to those of the next layer (Aggarwal and C 2018). A MLP with one input layer, one hidden layer and one output layer is shown to the right of Figure 2.1.

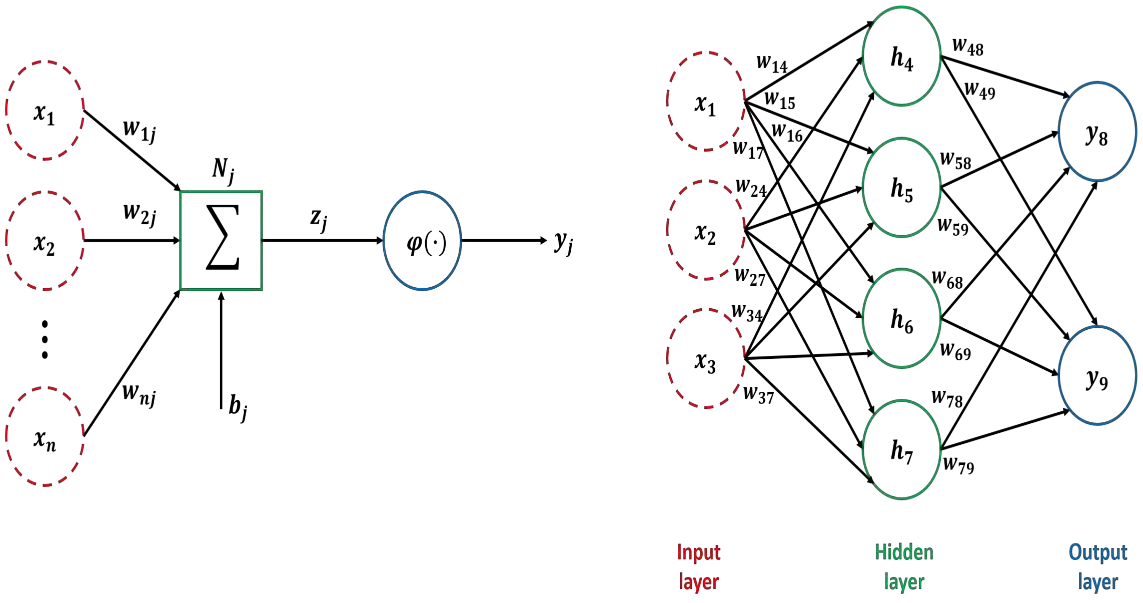


FIGURE 2.1. **Neural Network**. Left: Perceptron architecture. Right: Multi-Layer Perceptrons architecture. Inspired by (Iba and Noman 2020).

In the last decade, the progress and use of MNNs in different research areas has generated the trend in ML called **Deep Learning** (DL). The central idea of deep learning lies in exploiting many non-linear layers of information processing for useful feature extraction and transformation for supervised or unsupervised learning. A Deep Neural Network, also called DNN or only NN, can utilize hundreds even thousands of layers those collectively learn a hierarchy of representations. There is no strict division between shallow and deep NN based on the number of layers. However, any architecture with more than two or three layers can be considered as deep (Noman 2020).

2.1.2 Activation Functions

The activation functions $\varphi(\cdot)$ limits the output range of a neuron and a non-linear function is required to exhibit complex behaviour. Activation functions are scalar-to-scalar function. When a neuron passes on a nonzero output to another neuron, it is said to be *activated*.

Some of the most used activation functions will be described below.

Identity function: The most basic activation function is the identity or linear function, it means that the function outputs the unchanged signal:

$$\varphi(v) = v \tag{2.2}$$

This function does not provide non-linearity and is often used in the input layer and the output layer when the target is a real value (Aggarwal and C 2018). The graphical representation of this function is seen in Figure 2.2.

Sigmoid function: The Sigmoid activation, also called logistic function, outputs a value in the range (0,1), which is helpful in performing computations that should be interpreted as probabilities:

$$\varphi(v) = \frac{1}{1 + e^{-v}} \tag{2.3}$$

Furthermore, it is also helpful for creating loss functions derived from maximum-likelihood models (Aggarwal and C 2018). The graphical representation of this function is seen in Figure 2.3.

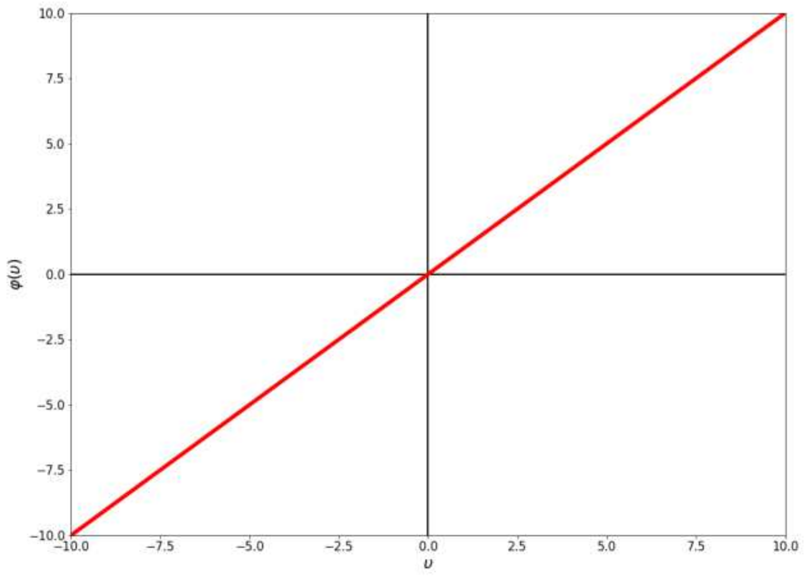


FIGURE 2.2. **Identity function.**

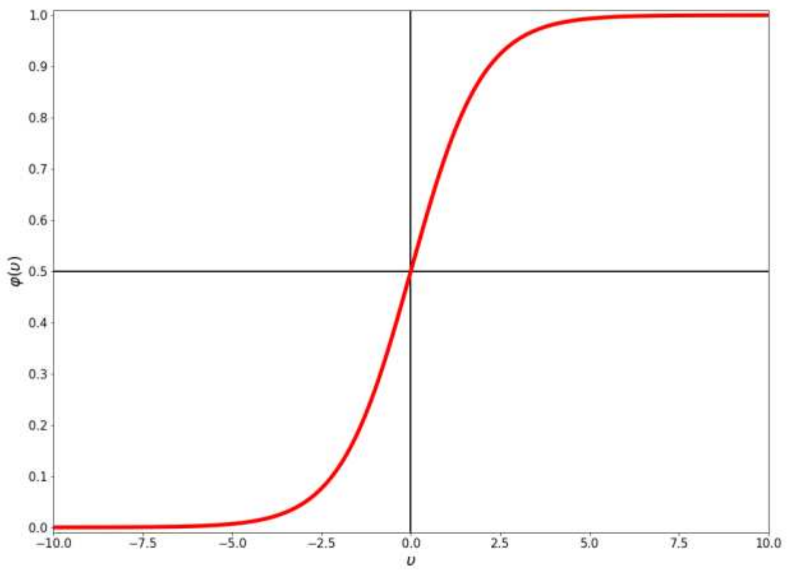


FIGURE 2.3. **Sigmoid function.**

Softmax function: Is a generalization of the sigmoid function inasmuch as it can be applied to continuous data (rather than binary classification) and can contain multiple decision boundaries. It is used to represent a probability distribution over a discrete variable with *k* possible classes (Goodfellow et al. 2016). It is defined as:

$$\varphi(\overline{v})_i = \frac{e^{v_i}}{\sum_{j=1}^{k} e^{v_j}} \quad \forall i \in \{1, ..., k\} \tag{2.4}$$

where $\overline{v}$ = $[v_1, ..., v_k]$ represents the *k* outputs of the neurons of a given layer. The *k* values obtained from softmax represent the probabilities of every *k* class to predict. The sum of this values is equal to one because they represent a multi-class probability distribution.

Tanh function: Tanh means Hyperbolic Tangent. This activation function has a shape similar to the sigmoid function except that it is vertically re-scaled in a range [-1,1]:

$$\varphi(v) = \frac{e^{2v} - 1}{e^{2v} + 1} \tag{2.5}$$

The tanh function is preferable to the sigmoid when the outputs of the computations are desired to be both positive and negative. Furthermore, its mean-centering and larger gradient with respect to the sigmoid function leads to an easier training. (Aggarwal and C 2018). The graphical representation of this function is seen in Figure 2.4.

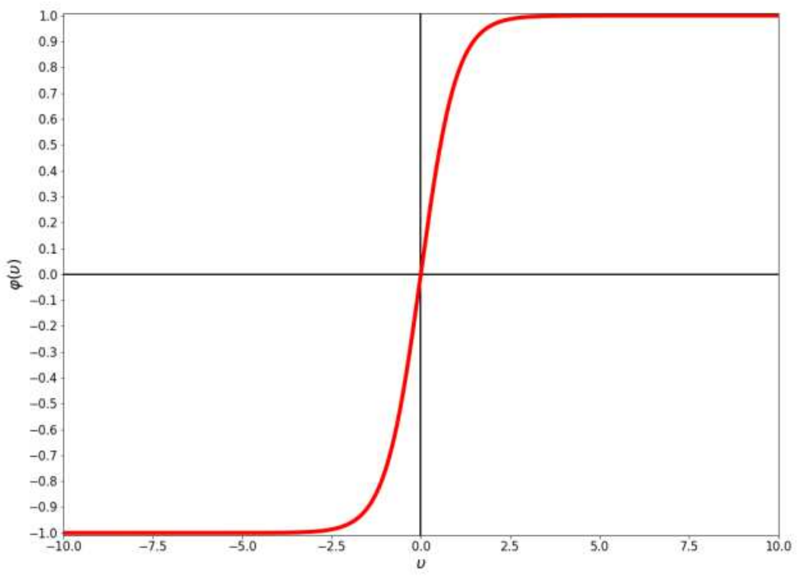


FIGURE 2.4. **Tanh function.**

Rectified Linear Unit: Also called ReLU, this activation function allows to avoid saturation problems of the sigmoid and tanh functions. This means that large values snap to 1.0 and small values snap to -1 or 0 for tanh and sigmoid respectively. Further, the functions are only really sensitive to changes around their mid-point of their input, such as 0.5 for sigmoid and 0.0 for tanh. Because ReLU is nearly linear, they preserve many of the properties that make linear models generalize well and easy to optimize with gradient-based methods (Goodfellow et al. 2016). Its formula is the following:

$$\varphi(\upsilon) = \max\{\upsilon, 0\} \tag{2.6}$$

The graphical representation of this function is seen in Figure 2.5.

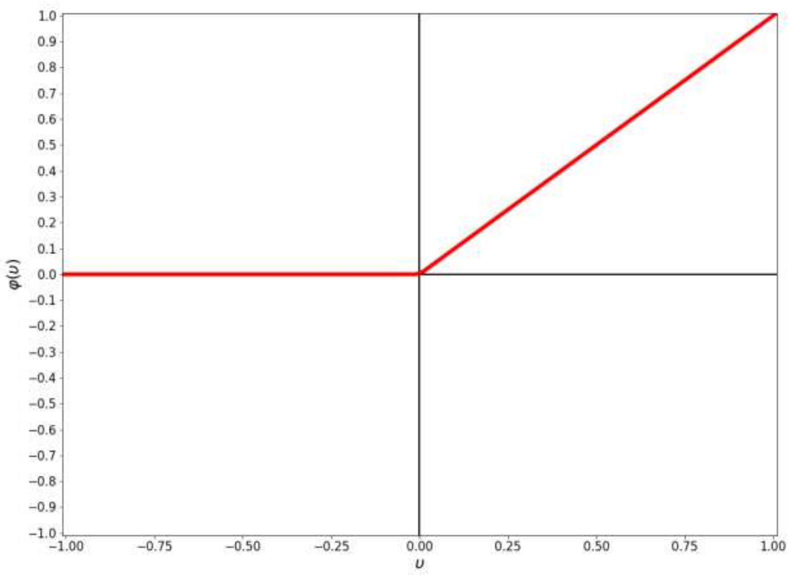


FIGURE 2.5. **ReLU function.**

Leaky ReLU: This function is ReLU function with a small slope of negative values instead of altogether zero:

$$\varphi(\upsilon) = \left\{ \begin{array}{ll} \alpha \times (\upsilon), & \text{if } \upsilon < 0 \\ \upsilon, & \text{if } \upsilon \geq 0 \end{array} \right\} \tag{2.7}$$

Where α is called *slope* which is originally *0.01*. This function attempts to minimize one's sensitivity to the dying ReLU problem. This problem is caused because ReLU is not continuously differentiable when $\upsilon = 0$. Also, ReLU sets all values smaller that zero to zero and hence neurons arriving at large negative values cannot recover from being stuck at 0. The neuron effectively *dies* and hence the problem is known as the dying ReLU problem. With the slope of Leaky ReLU the outputs are slightly descending. The thesis is that these small numbers reduce the death of ReLU activated neurons (Goodfellow et al. 2016). The graphical representation of this function is seen in Figure 2.6.

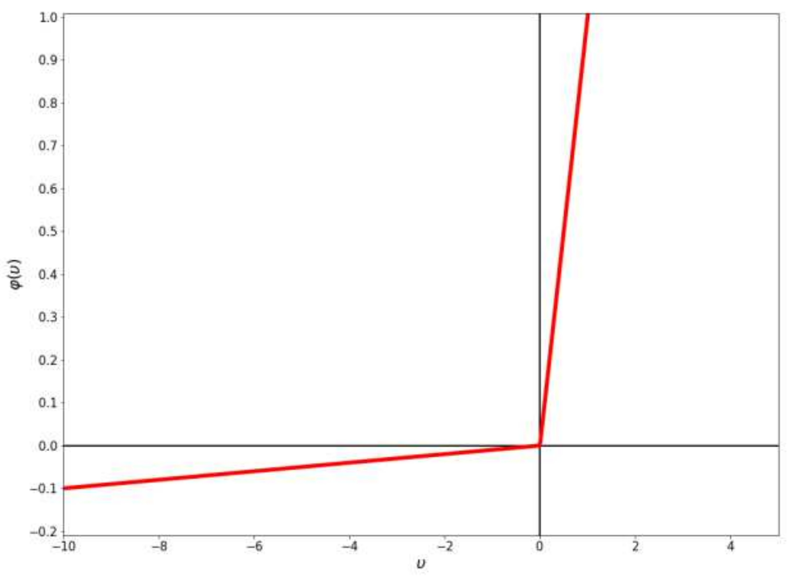


FIGURE 2.6. **Leaky ReLU function with a slope** (α) **of 0.01.**

In the Perceptron, the original activation function used was the sigmoid function because it is a binary classifier. Later, with the use of MLPs, each layer of neurons could use the same or a different activation function according to the needs of the task. The use of non-linear activations plays a fundamental role in increasing the modeling power of the network. If a network used only linear activations, it would not provide better modeling power than a single-layer linear network (Aggarwal and C 2018).

2.1.3 Loss Functions

Loss functions, also named cost functions or error functions, quantify how close a given NN is to the ideal output in its training (Patterson and Gibson 2017). They are metrics based on the observed error in the NN's predictions. The choice of the loss function is critical in defining the outputs in a way that it is sensitive to the application at hand.

In a dataset from a supervised learning problem, **N** obtained samples could be represented as the tuple **(X,Y)**. Where **X** represents feature variables and **Y** represents the observed value or class to predict. Let $\hat{Y}$ be the output value of the network, also called *prediction*. The notation $h_{W,b}(\cdot)$ denotes the NN as a function that depends of the weights (**W**) and bias (**b**) of the network. Thus:

$$h_{W,b}(X_i) = \hat{Y}_i \tag{2.8}$$

represents the transformation of the feature values of the *ith* sample to a prediction value through the NN.

The loss function is represented as **L(W, b)** emphasizing that the cost or error of a network's predicted values depends exclusively on its weights and biases.

One of the principal approaches of loss functions is for classification tasks. One of the most used loss function for classification is Negative log-likelihood loss.

Maximum Likelihood Estimation is a way to finding the best possible parameters which make the observed data most probable. In classification problems, it seeks to maximize the probability of correctly predicting the class. In binary classification it is described by the next equation:

$$P(Y_i|X_i; W, b) = (h_{W,b}(X_i))^{Y_i} \times (1 - h_{W,b}(X_i))^{1-Y_i} \tag{2.9}$$

The previous equation could be rewritten as:

$$P(Y_i|X_i; W, b) = \prod_{i=1}^{N} (\hat{Y})^{Y_i} \times (1 - \hat{Y})^{1-Y_i} \tag{2.10}$$

When working with multiplication of probabilities it is useful to use its logarithm. This has the purpose of converting the multiplication into a sum of probabilities. Also, the monotonic growth of the logarithm is a property of interest when using probabilities. Thus, minimizing the negative log-likelihood is equivalent to maximizing the probability. That is why applying the logarithm to equation (2.10) obtains the next loss function called Negative Log-Likelihood loss (Goodfellow et al. 2016):

$$L(W,b) = -\sum_{i=1}^{N} Y_i \times log(\hat{Y_i}) + (1 - Y_i) \times log(1 - \hat{Y_i}) \tag{2.11}$$

The Cross-Entropy is one of the main cost functions used for supervised learning (Goodfellow et al. 2016). It is used for multiclass classification. This is represented by the following equation:

$$L(W,b) = -\sum_{n=1}^{N}\sum_{c=1}^{C} Y_{nc} \times log(\hat{Y}_{nc}) \tag{2.12}$$

where C represent the classes of the supervised problem, Y_{nc} represent the groundtruth for the c class of the n input and $\hat{Y}_{nc}$ represent the prediction of the same input. In binary classification, where $C = 2$, the Cross-Entropy can also be represented as in Equation 2.11.

Minimizing this cost functions allows to apply the method known as Gradient Descent to train the NN using Backpropagation with the purpose of finding the optimal parameters that allow reducing the error rate of the network predictions as much as possible (Khan et al. 2018).

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs/ConvNets) are one of the most widely used NN architectures with many Computer Vision (CV) applications. Some of the applications areas include image classification and segmentation, object detection, video processing, even being used beyond CV, in tasks like natural language processing and speech recognition (Aloysius and Geetha 2017). The attractive feature of a CNN is its ability to exploit spatial or temporal correlation in data. The biological inspiration for CNN is the visual cortex in animals (Hubel and Wiesel 1959).

The CNN is a feedforward multilayered hierarchical network very similar to classic NN, being the main difference that a hidden neuron is only connected to a subset of neurons in the previous layer. This characteristic allows to learn relevant features implicitly. The deep architecture of the model results in hierarchical feature extraction (Khan et al. 2020).

CNNs are a useful class of models for image classification tasks. The CNN learns to map a given image to its corresponding category by detecting a number of abstract feature representations, ranging from simpler to more complex ones. These discriminatory features are then used within the network to make the correct prediction of an input image (Khan et al. 2018).

The topology of a CNN is divided into multiple learning stages composed of a combination of the convolutional layers, non-linear processing units, and subsampling layers (Khan et al. 2020). The figure 2.7 represent the general architecture of a CNN. The different types of layers that compose a CNN will be explored next.

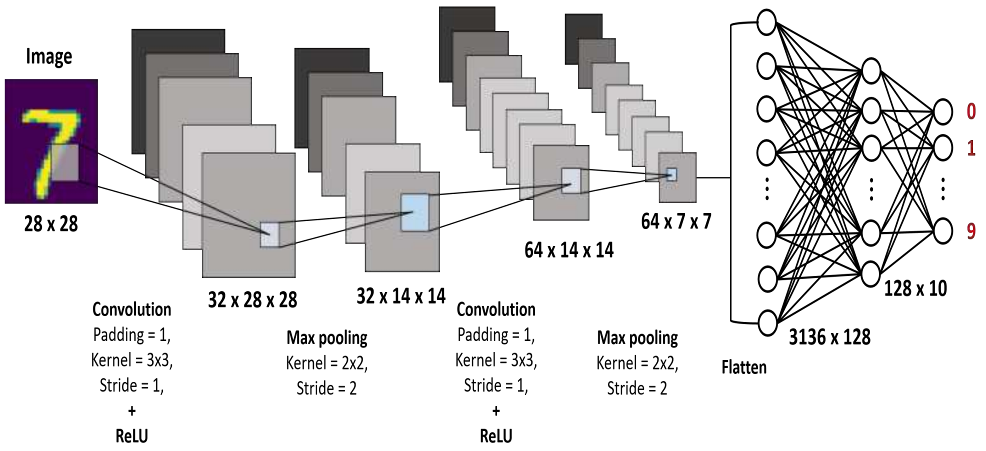


FIGURE 2.7. **Architecture of a basic CNN.** A CNN is a concatenation of different types of layers in charge of learning high-level characteristics to carry out tasks such as classification. Inspired by (Human 2020).

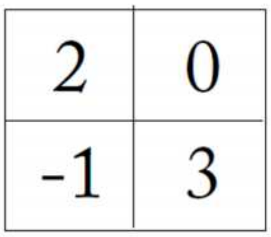
2.2.1 Convolutional Layer

The convolutional layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. It comprises a set of filters which are convolved with a given input to generate an output feature map. This layers are essentially feature-extractors (Noman 2020).

2.2.1 Filters

Each filter, also called *kernel*, in a convolutional layer is a grid of discrete numbers, which are called the weights. Similar to conventional neurons in NN, these weights are trained to learn features that allow to fulfill the required activity. The main difference from NN is that these neurons are connected only to a small region of neurons in the layer before them. The convolutional kernel works by dividing the image into small pieces, commonly know as *receptive fields*. The division of an image into small blocks helps in extracting feature motifs that hierarchically build more complex features.

The kernel structure has *height* and *width* and commonly has square-shape (*height* = *width*). For example, Figure 2.8 shows a kernel of height two and width two.



2	0
-1	3

FIGURE 2.8. **Example of a filter 2x2.**

Also, kernels have a third dimension called *depth*, not to be confused with the depth of the network. This dimension refers to the number of *channels* in each layer of the CNN, such as the number of primary color channels (e.g. red, green, and blue in RGB images) in the input image or the number of feature maps in the hidden layers (Aggarwal and C 2018). Each filter has its own weights to train and each one will generate a different *feature map*. When applying the filters to the image or to the previous feature maps, the new maps will be smaller than or equal to the previous ones with respect to *height* and *width*, but the number of new feature maps, *depth*, is not limited (Khan et al. 2018). Each filter applies the *convolution operation* over the previous feature maps.

2.2.2 Convolution Operation

The convolution operation places the filter at each possible position in the image (or hidden layer) so that the filter fully overlaps with the image, and performs a dot product between the filter and the matching spacial region of the input volume (Aggarwal and C 2018). The convolution operation is defined as:

$$h_{ijp}^{(q+1)} = (\sum_{r=1}^{F_q}\sum_{s=1}^{F_q}\sum_{k=1}^{d_q} w_{rsk}^{(p,q)} h_{i+r-1,j+s-1,k}^{(q)}) + b^{(p,q)} \tag{2.13}$$

$$\forall i \in \{1, ..., L_q - F_q + 1\} \quad \forall j \in \{1, ..., B_q - F_q + 1\} \quad \forall p \in \{1, ..., d_{q+1}\}$$

where:

- $h^{(q)}$ represents the feature map of the *qth* layer.
- F_q is the *height* and *width* of the filter. Remember that *height* = *width* is common.
- d_q is the number of *channels* of the filter and also the number of feature maps in the input volume. Recall that the *depth* of the filter is conditional on the number of feature maps of the previous layer.
- $w^{p,q}$ represents the weight value of the *pth* kernel in the *qth* layer.
- L_q and B_q refer to the height (or *length*) and width (or *breadth*) of the input volume, respectively.
- $b^{(p,q)}$ refers to the bias of the *pth* kernel in the *qth* layer.
- $L_q - F_q + 1$ and $B_q - F_q + 1$ represent the new dimensions (*length* and *breadth*) of the feature map of the *q+1* layer. The dimensions after applying the convolution are exemplified in the figure 2.9.

As an example of the convolution operation Figure 2.10 shows the convolution of a 3x3 filter over a 2D grid.

Besides of the *height*, *width* and *depth* of the filters in a convolutional layer, the **stride** and **padding** parameters are also required to define the convolutional layer.

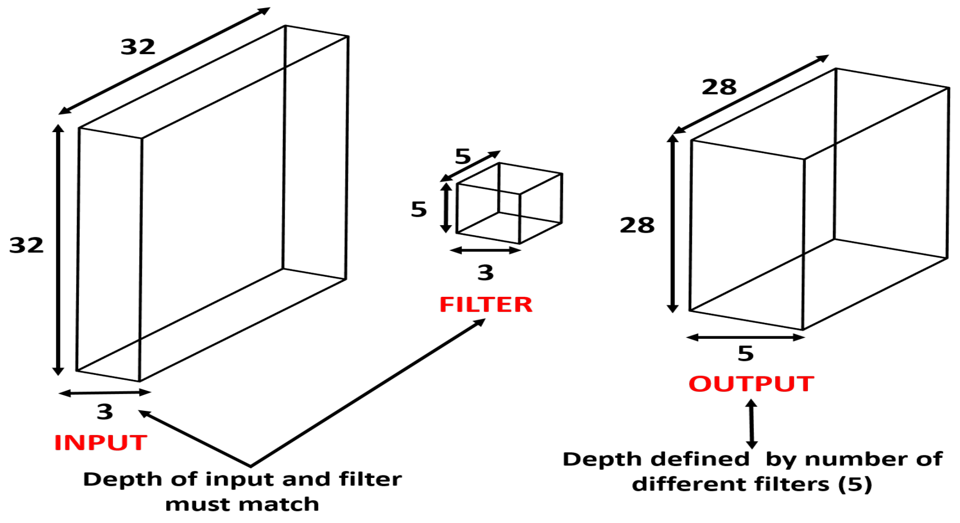

32
32
3
INPUT
5
5
3
FILTER
28
28
5
OUTPUT
Depth of input and filter must match
Depth defined by number of different filters (5)

FIGURE 2.9. **Dimensions in convolution.** The convolution between an input layer of size 32×32×3 and a filter of size 5×5×3 produces an output layer with spatial dimensions 28×28. The depth of the resulting output depends on the number of distinct filters and not on the dimensions of the input layer or filter. Inspired by (Aggarwal and C 2018).

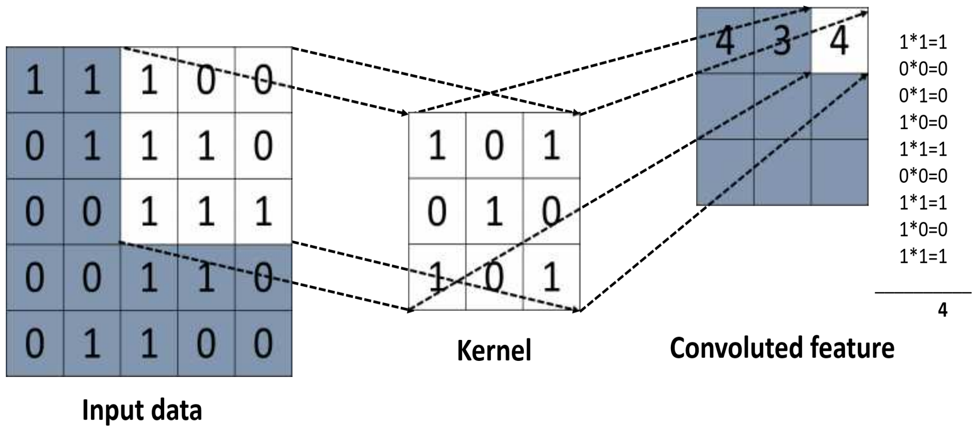


FIGURE 2.10. **Convolution operation example**. Inspired by (Patterson and Gibson 2017).

2.2.3 Stride

In the above description of convolution, in order to calculate each value of the output feature map, the filter takes a step of 1 along the horizontal or vertical position i.e. along the column or the row of the input, respectively. This step is termed as the stride of the convolution filter, which can be set to a different (other than 1) value if is required (Khan et al. 2018).

Stride configures how far to slide the filter to create the feature map. This parameter can have a slip value by the length and other by the breadth of the input volume, but commonly the same value is used for both dimensions. Without using **padding** the convolution causes a reduction in the previous dimensions that is known as the sub-sampling operation. Figure 2.11 shows an example of the convolution with stride values of one and two.

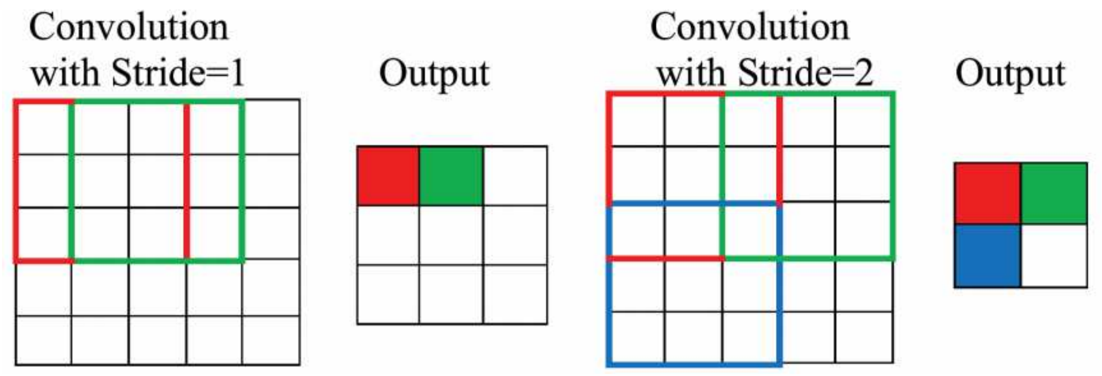


FIGURE 2.11. **Convolution with stride values of 1 and 2.**

2.2.4 Padding

One observation is that the convolution operation reduces the size of the q+*1*th layer in comparison with the size of the qth layer. This type of size reduction is not desirable in general, because it tends to lose some information along the borders of the image (or of the feature map, in the case of hidden layers). This problem can be solved by using padding (Aggarwal and C 2018). Padding controls the spatial size of output volume by adding rows and columns of "*pixels*" to the input feature map. Commonly, this new values are zeros, in which case it is called *Zero-padding*. Zero-padding is represented in Figure 2.12.

2.2.5 Output Size

The dimensions of the output volume, after the application of the convolution, can be calculated with the following formulas:

$$L_{(q+1)} = \left\lfloor \frac{L_q - F_q + stride + 2 \times padding}{stride} \right\rfloor \tag{2.14}$$

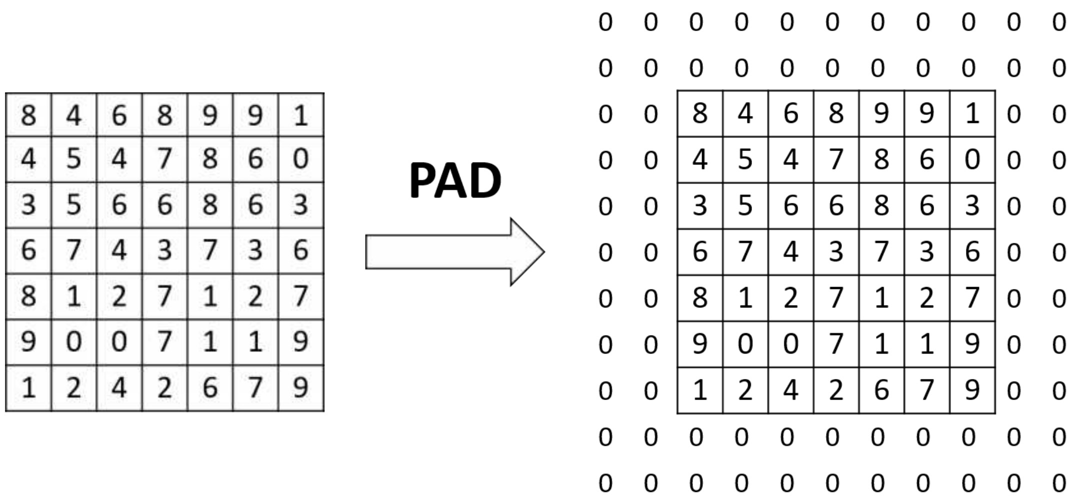


FIGURE 2.12. **Example of *zero-padding***. The padding value is 2 i.e. two rows or columns are added to each side of the feature map. Inspired by (Aggarwal and C 2018).

$$B_{(q+1)} = \left\lfloor \frac{B_q - F_q + stride + 2 \times padding}{stride} \right\rfloor \tag{2.15}$$

Recall that L_q and B_q are the *length* and *breadth* of the input volume, respectively. The symbol $\lfloor \cdot \rfloor$ represents the floor function. As an example, an image in gray scale (1 channel input) of 64x64 pixels when a filter of 3x3 is applied with a stride of 2 and padding of 1 (represents one column or row of zeros added by every edge) will have output dimensions of 32x32 on each feature map.

2.2.2 Pooling Layer

Pooling layers are commonly inserted between successive convolutional layers. The use of convolutional layers with pooling layers is done to progressively reduce the spatial size of the data representation and to help controlling overfitting. The pooling layer operates independently on every depth slice of the input (Patterson and Gibson 2017).

Pooling or down-sampling is an interesting local operation. It sums up similar information in the neighborhood of the receptive field and outputs the dominant response within this local region (Khan et al. 2020). The most common setup for a pooling layer is to apply 2×2 filters with a stride of 2. This will downsample each depth slice in the input volume by a factor of two on the *length* and *breadth* but they keep the number of *channels*, unlike convolutional filters.

The two principal pooling layers used in CNNs are *Max-pooling* and *Average-pooling*. *Max-pooling* takes the maximum value of the filter's receptive field. Meanwhile, *Average-pooling* takes the mean of the values (Goodfellow et al. 2016). The performance of these layers is shown in Figure 2.13.

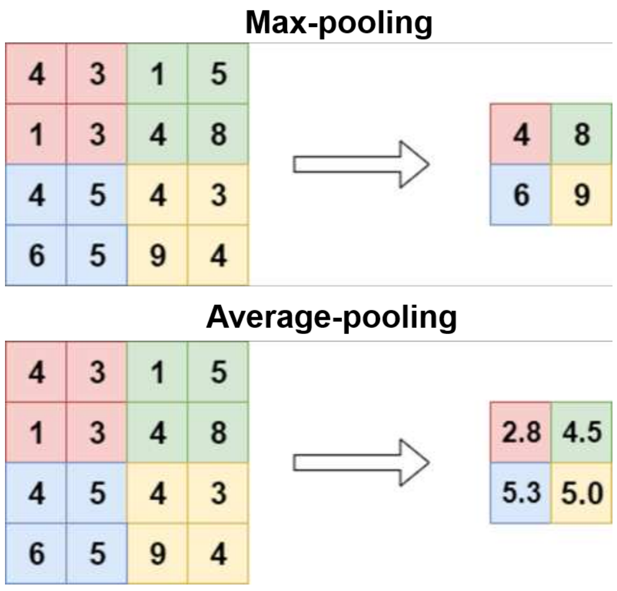


FIGURE 2.13. ***Max-pooling* and *Average-pooling*.**

2.2.3 Activation Layers

Activation layers insert non-linearity to the feature maps obtained by convolutional layers. They apply a vast collection of activation functions, such as those described in Section 2.1.2.

2.2.4 Fully Connected Layer

Fully connected layers correspond essentially to Multi-Layer Perceptrons previously described in Section 2.1.1. Each neuron in a fully connected layer is densely connected to all the units of the previous layer. This layers takes the high-level features extracted by the set of convolutional, activation and pooling layers of the CNN and generate the prediction value, $\hat{Y}$. This layers are commonly used at the end of CNN to perform tasks such as classification (Patterson and Gibson 2017).

2.2.5 Batch Normalization Layer

Batch normalization (BN) is used to address the issues related to the internal covariance shift within feature-maps. The internal covariance shift is a change in the distribution of hidden layers values, which slows down the convergence and requires careful initialization of parameters. Furthermore, it smoothens the flow of gradient and acts as a regulating factor, which thus helps to improve the generalization of the network (Khan et al. 2020). Batch normalization is represented as:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.16}$$

where:

- x_i represents the feature map of the *ith* layer.
- $\hat{x}_i$ is the normalized feature-map.
- μ_B and σ_B^2 depict mean and variance of the feature map for a mini-batch of samples, respectively.
- ϵ is a small constant to avoid division by zero for numerical stability.

Batch normalization unifies the distribution of feature-map values by setting them to zero mean and unit variance. Later, the normalized feature-map is scaled using the parameters β and γ to avoid constrains in the representation power of convolutional layers (Aggarwal and C 2018). This operation is represented as:

$$y_i = \gamma \times \hat{x}_i + \beta \tag{2.17}$$

where y_i represents the pre-activation feature-map. The Figure 2.14 shows a representation of BN.

Batch normalization is implemented as a layer and is usually used after a convolutional layer.

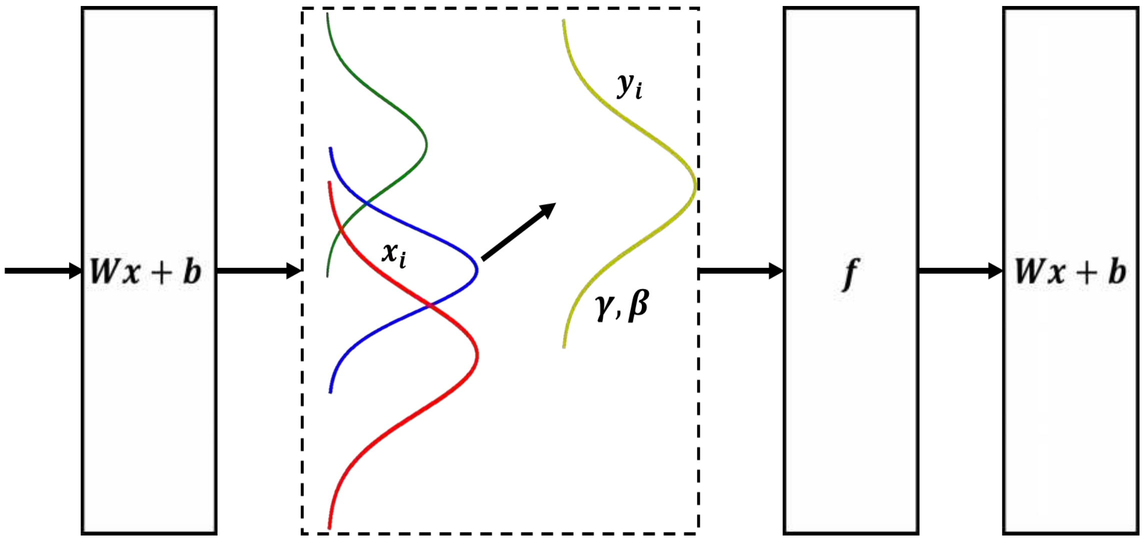


FIGURE 2.14. **Representation of Batch Normalization.** The different distributions of values (x_i) for each sample in a layer (*Wx* + *b*) are transformed into normalized and scaled signals (y_i) with BN to be introduced to an activation layer (*f*). Inspired by (Aggarwal and C 2018).

2.2.6 Transposed Convolutional Layer

Also called *fractionally strided convolutions*. This kind of layers are very similar to convolutional layer in aspects such as the use of filters, stride and padding. The main difference is that while convolutional layers create feature-maps from the pixel values of a image, transposed convolutional layers maps features to pixels when modeling images, which is the opposite of what a normal convolutional layer does (Patterson and Gibson 2017). This layer is usually carried out for upsampling i.e. to generate an output feature map which has a spatial dimension greater than that of the input feature map. This aspect of transposed convolutional layers is what enables to generate images as output from neural networks. This layer is also known as deconvolution layer, although it performs a different process than deconvolution (Khan et al. 2018).

The process that this layer applies to upsample a feature-map is the convolution operation with a filter over the input size that have borders with zero values added. This *zero-borders* have the function of making the receptive field of the filter larger and therefore obtaining a feature-map with a larger *length* and *width* than the input volume. The output size ($O_{(q+1)}$) of the *q+1th layer* where *length* = *width* ($O_{(q)}$) with a size filter F_q is calculated with the next formula:

$$O_{(q+1)} = (O_{(q)} - 1) \times stride + F_q - 2 \times padding \tag{2.18}$$

For example, an input volume with $O_{(q)}$ of 2, a kernel size of 3, stride of 1 and padding of 0 will have an output size ($O_{(q+1)}$) of 4. The Figure 2.15 shows this example.

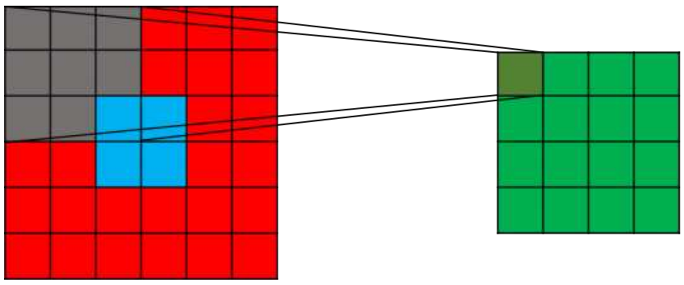


FIGURE 2.15. **Example of the transposed convolution operation.** The convolution operation is applied between a 3x3 filter (shaded grid) and a feature-map (blue grid) with 2 *zero-borders* (red grid) for each side, for upsampling and to get a new feature map larger (green grid). Inspired by (Dumoulin and Visin 2016).

The Figure 2.16 shows two ways of understanding the computation of the transposed convolution using the previous example.

The transposed convolutional layer is not used in a conventional CNN but is a fundamental component of deconvolutional networks that are used in the Deep Convolutional Generative Adversarial Networks which will be addressed later.

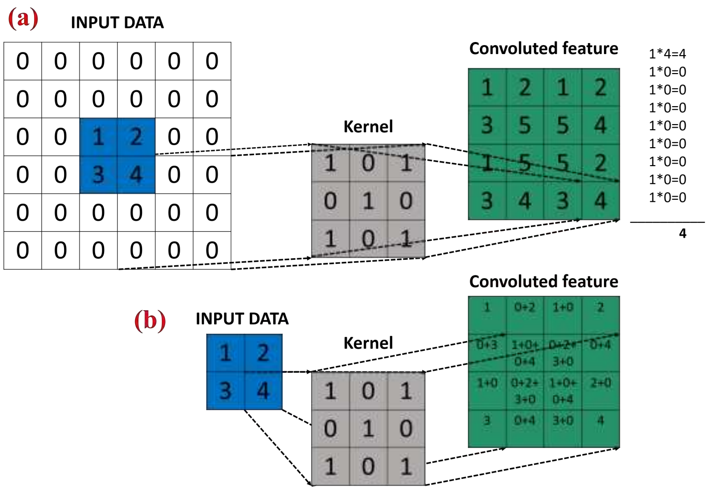


FIGURE 2.16. **Two ways to compute the transposed convolution.** (a) Perform standard convolution on input volume with *zero-borders* added. (b) Multiply each filter value by each of the input volume values and add those that overlap.

2.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are deep learning models that date back to 2014 (Goodfellow et al. 2014). These models belong to the set of generative models, a branch of unsupervised learning algorithms in charge of mapping how the data was generated. The GANs models are trained with the goal of generating synthetic data similar to some set of real data. Its operation consists of two types of neural networks, which are faced by having opposite objectives between them (hence the term *adversarial*). This confrontational training, similar to reinforcement learning, allows inherent learning by a network to generate data similar to those belonging to a real distribution.

2.3.1 GANs Structure

The two networks that compose a GAN are the following:

- **Generator** (G): This network is in charge of generating data that are highly similar to the real set with which it was trained. It takes a vector of random noise values (z) taken from a simple prior distribution (p_z) as input and process it through their inner values in the network until output data ($G(z)$) is obtained such that this new data is very similar to the same complex distribution of the original set.

- **Discriminator** (D): This model has the goal of classifying between the real samples ($x \sim p_{data}(x)$) i.e. data that belong to the original set, and the generated samples ($G(z) \sim p_g(G(z))$) i.e. those created artificially by the generator. Their input is the data, either real or fake, and their output is classification probabilities for both classes through the logistic function.

This general structure is called *Vanilla GAN* since it represents the base version from which more sophisticated architectures started but which continue to respect the two fundamental blocks of the GANs. A visual representation of this base structure is seen in Figure 2.17.

2.3.2 GANs Training

The training of a GAN is described as a zero-sum game (also called minimax) between two players with opposite objectives; these are the Generator and the Discriminator. Taking a vector of random noise sampled from prior distributions, e.g. normal or uniform, ($z \sim p_z(z)$) as input, called *latent vector*, the generator outputs samples from a more complex distribution ($G(z)$) whose goal is to be equal to the distribution of the real dataset. Meanwhile the discriminator has the task of distinguishing between the real samples($x \sim p_{data}(x)$) and the generated samples ($G(z) \sim p_g(G(z))$). In the case of real data the goal of the discriminator output ($D(\cdot)$) is to be near to 1. In the fake data scenario, the discriminator output's goal is to be close to 0 meanwhile the generator will try to make near to 1 i.e. fool the discriminator to classify his creations as real.

The training cost function of GANs, also called Minimax loss, is reflected by the following equation:

$$\min_{G} \max_{D} \mathop{\mathbb{E}}_{x \sim p_{data}} [log(D(x))] + \mathop{\mathbb{E}}_{z \sim p_z} [log(1—D(G(z)))] \tag{2.19}$$

The previous formula derives from the cross-entropy between the real and generated distributions (Goodfellow et al. 2014).

The networks are trained simultaneously, encouraging both models towards continuous improvement and adaptation. For every iteration a gradient step with backpropagation is made to reduce the cost function of each network, optimizing their internal weights. The generator optimization is treated as optimizing the Jensen–Shannon divergence, which measures how the probability distribution $G(z)$ (estimated distribution) diverges from the expected probability distribution p_{data} (the real-life distribution). The training process is repeated for a number of training iterations until the generator converges to make synthetic data indistinguishable from the real set of data i.e. $D(\cdot)$ equal to 0.5 for any sample, real or not, although this theoretical ideal is almost never fulfilled and the iterations stop when the quality of the creations is the required one, decided either by means of a measurement or the criteria of the practitioner who implements them. The representation of this training process is in the Figure 2.17.

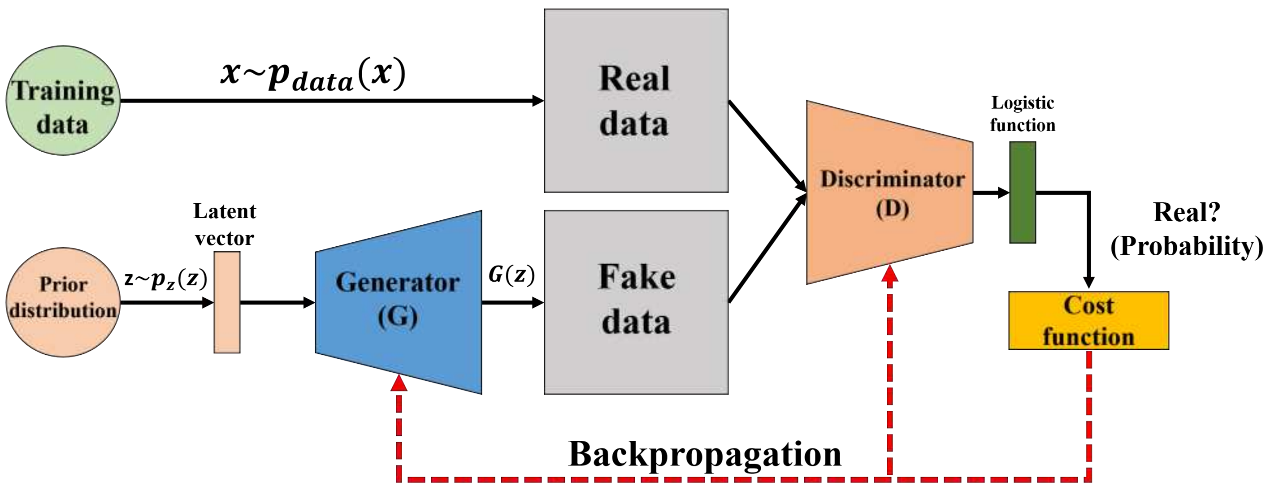


FIGURE 2.17. ***Vanilla* GAN training.**

GANs have been used in many different tasks and fields, the most prolific being the synthesis of images, their area of birth (Wang et al. 2019, Wang et al. 2020). However, the great influence of GANs as a generative model of the *state-of-the-art* has allowed their migration to other tasks such as neural style transfer, music synthesis and drug discovery, to name a few examples; the following references offer a detailed overview of the GANs application areas (including the above) as well as their development over the years (Alqahtani et al. 2019, Gui et al. 2020).

2.3.3 GANs Shortcomings

GANs training is complicated because there must be a balance between the skills of the generator and the discriminator. If there is a supremacy from one of the networks, training instability problems may arise. These issues are (Costa et al. 2020b):

- **Mode Collapse**: The situation in which the generator can only synthesize a small subset of data of the complete distribution since the training did not allow to generalize the richness of variants of the original distribution. As seen in Figure 2.18 on the left, training with the MNIST dataset, which consists of images of handwritten numbers from 0 to 9, failed to capture the diversity of the set, collapsing in the generation of only two types of digits. While on the right of the same figure the generator created an overlapping combination of digits.
- **Vanishing Gradient**: Originated when the discriminator or the generator becomes powerful enough to cause an irreversible imbalance in training and by not using adequate cost functions that allow obtaining adequate learning gradients that make possible to the opposite network to improve its performances, thus causing a stalemate. As exemplified in Figure 2.19, when the discriminator is able to easily detect between the real and synthetic samples i.e. its loss approaches zero, an stagnation in the gradients feedback to the generator is produces, thereby avoiding an improvement in the learning of the generator.

Due to the aforementioned issues it is important to have the correct hyperparameters, architectures and training procedure to obtain an useful generator.

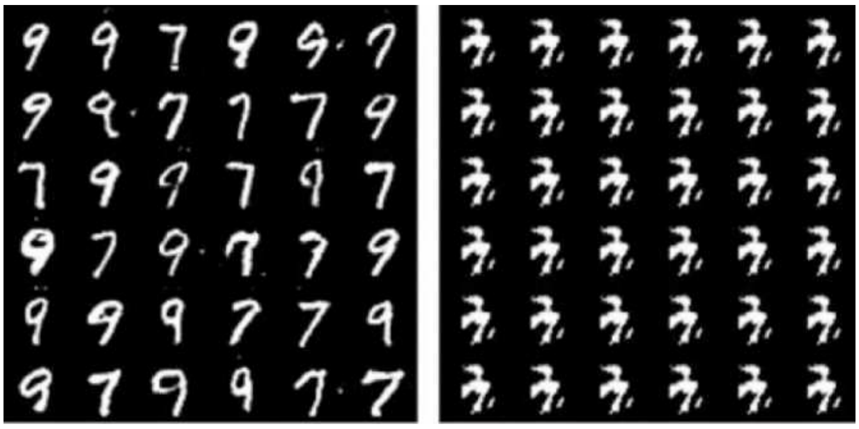


FIGURE 2.18. **Example of Mode Collapse.** Obtained from (Costa et al. 2020b).

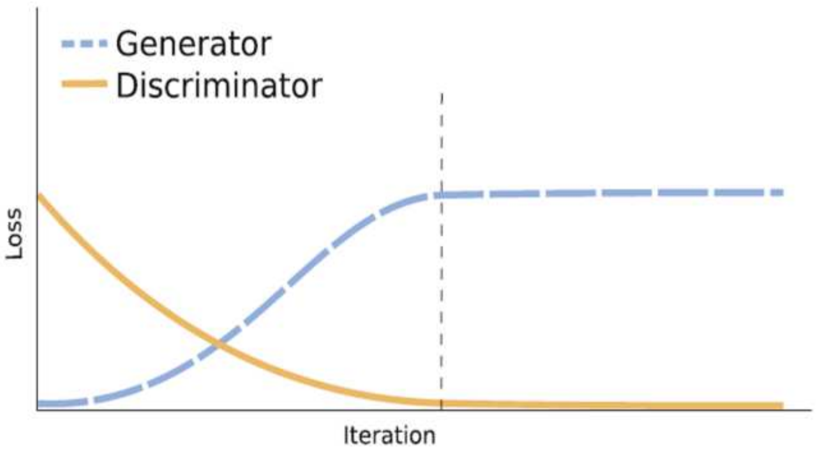


FIGURE 2.19. **Example of Vanishing Gradient.** Obtained from (Costa et al. 2020b).

2.4 Architectures and Implementations to Improve GANs

Some of the most outstanding advances in favor of improving the stability of the GANs and that are relevant in the development of this project will be explained below.

2.4.1 Deep Convolutional Generative Adversarial Network

The Deep Convolutional Generative Adversarial Networks (DCGAN) were designed in (Radford et al. 2015) and is oriented to be used in images. This class of GAN was developed focusing on improving training stability compared to the original GAN model (*Vanilla GAN*) that used Multi-Layer Perceptrons for both generator and discriminator. The training process is the same as in the original GAN model but in DCGAN, both the generator and the discriminator are inspired by the CNN architecture.

For the discriminator, a conventional CNN is used (see figure 2.7). This is because the classifying task of the discriminator is not different from the classic classifying tasks for which CNNs are commonly used. Meanwhile, for the generator, a *deconvolutional network* is required. This kind of network is similar to standar CNN with the main difference that being a generative model, it is responsible for obtaining images from high-level features provided by random noise. The deconvolutional network used deconvolutional layers (see Section 2.2.6), instead of convolutional layers like a CNN. In addition, since its purpose is to expand the feature maps (upsampling) and not reducing them (downsampling) as in CNN, the pooling layers are not used in its architecture. Figure 2.20 shows the general architecture of a DCGAN generator.

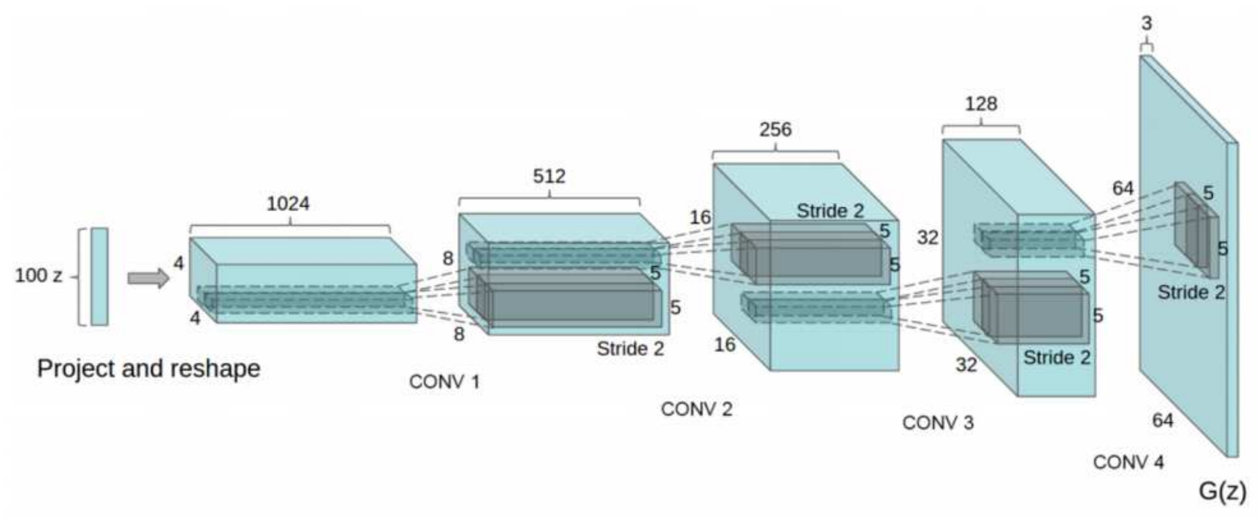


FIGURE 2.20. **General architecture of a DCGAN generator.** As can be seen when processing a random noise input with a series of deconvolutional layers, new and larger feature maps are generated, resulting in the synthesis of an RGB image (3 channels) with a resolution of 64x64 pixels. Obtained from the original DCGANs paper (Radford et al. 2015).

Another difference with respect to CNNs is in the activation layers. Commonly in CNNs, ReLU is used as activation function in every convolutional layer and sigmoid or softmax functions are used after the fully connected layer that perform classification. In a DCGAN, generator layers use ReLU as well, but in the final layer the Tanh function is used since the images used for the training are normalized in a range of [-1,1], the same that this function grants. While for the discriminator LeakyReLU activation is used in all the layers except in the last one which uses sigmoid function to obtain a binomial probability to perform the classification between fake and real images.

Since its creation, DCGAN has become a base model for the synthesis of images in multiple works of the *state-of-the-art*. Due to its superior specialized imaging capabilities and better training stability, it helps to reduce problems such as mode collapse, although it does not completely solve common GANs issues (Khan et al. 2018).

2.4.2 Conditional GANs

Conditional Generative Adversarial Networks (cGAN), created in (Mirza and Osindero 2014), is a extension of the original Vanilla GAN.

In the Vanilla GAN, if the training set contained multiple data classes, its generation by depending on the latent vector that comes from random values, generated new data of a random class from among those existing in the real set, so it was left out from the user control the generated data class.

In cGAN, this class of GAN is *conditioned* using prior information to generate distinct class of data. This is achieved by adding the label ***y***, which represents the class of the data, to the input of both the generator and the discriminator. Through this addition to the original model, it is possible to control the class of data generated, as well as to make a more punctual discrimination of a particular class. The cost function of this model is the same as that of the Vanilla GAN (see equation 2.19) with the addition of the class labels (***y***), as observed in the following equation:

$$\min_{G} \max_{D} \mathop{\mathbb{E}}_{x \sim p_{data}} [log(D(\mathbf{x}|\mathbf{y}))] + \mathop{\mathbb{E}}_{z \sim p_z} [log(1 — D(G(\mathbf{z}|\mathbf{y})))] \tag{2.20}$$

In the generator, the prior input noise $p_z(z)$ and ***y*** are combined in a joint hidden representation. While in the discriminator both types of inputs, x and $G(z)$, are also combined with their respective ***y*** label. The Figure 2.21 represents the structure of a simple cGAN.

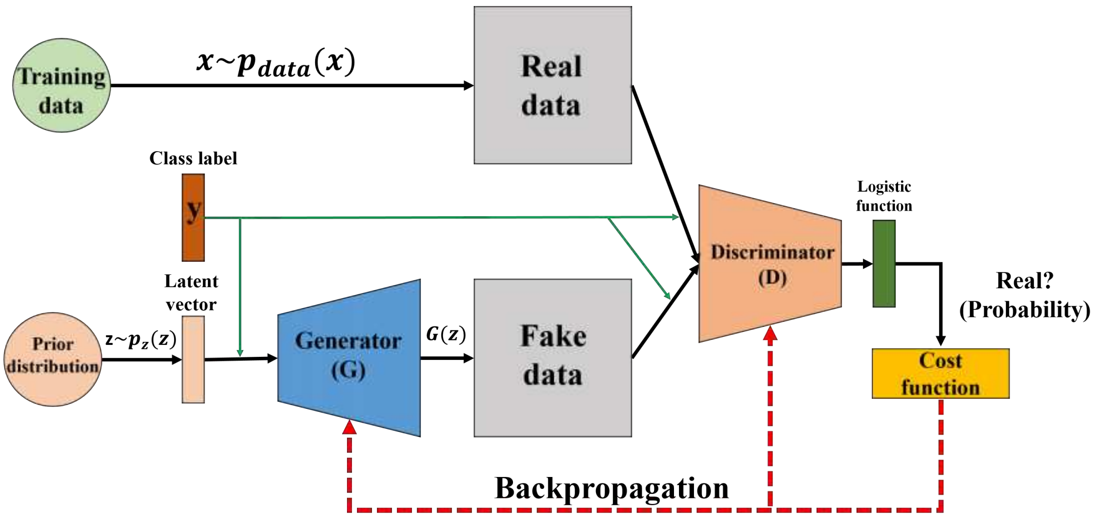


FIGURE 2.21. **General structure of a cGAN.** The green arrow represents the concatenation of the y label to the GAN's inputs.

2.4.1 cDCGANs

By joining the cGANs and the DCGANs, a model capable of controlling the class of the images generated is obtained.

The structure of the cDCGANs respects the original of the DCGANs (see Figure 2.20). The mechanisms to add the class label to the different inputs of the networks are the following:

- **Generator**: A *one-hot* vector is concatenated to the latent vector. The one-hot vector is a binary vector, that has as many positions as the number of classes. All positions contain zeros except for the position of the number that indicates the class to be encoded, which contains a one. This input is represented in the Figure 2.22.
- **Discriminator**: As many new channels, of the same length and breadth of the image, as the number of existing classes are concatenated to the input images. These new channels (matrices) are filled with zeros except for the channel that represents

the number of the class to be encoded, which is filled with ones. This input is shown in the Figure 2.23.

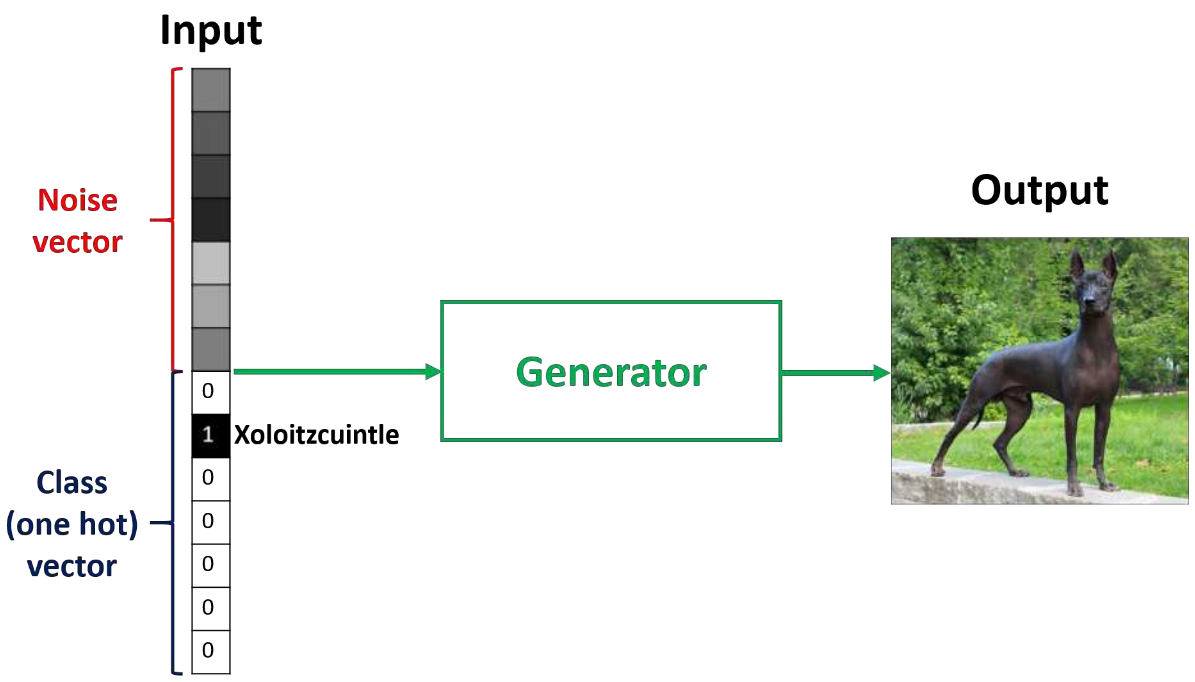


FIGURE 2.22. **Generator input in cDCGAN.** The image class (dog breed) that needs to be generated is represented by a *one-hot* vector that is concatenated to the noise vector (latent vector). Inspired by (Zhou 2020).

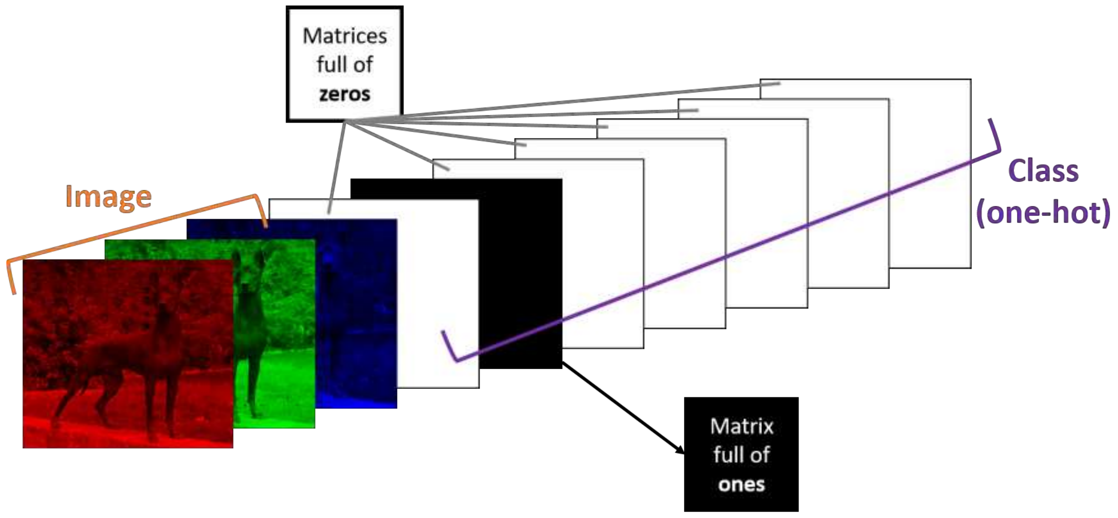


FIGURE 2.23. **Discriminator input in cDCGAN.** The image class (dog breed) is represented by a series of *one-hot* matrices that are concatenated to the input image. Inspired by (Zhou 2020).

2.4.3 Wasserstein-GAN

Wasserstein-GAN (or also called WGAN) is an extension of the Vanilla GAN, developed in (Arjovsky et al. 2017), that seeks an alternate way of training the generator model to better approximate the distribution of data observed in a given training dataset.

Instead of using a discriminator to classify or predict the probability of generated images as being real or fake, the WGAN replaces the discriminator model with a *critic* that scores the realness or fakeness of a given image.

This change is due to the fact that the cost function used in Vanilla GAN may have a vanishing gradient when saturated at the extremes of the discriminator classification i.e. when the discriminator is so good that it detects both sample cases almost perfectly. Therefore, a new cost function is used which turns the discriminator into a *critic* (as it is called in this model), which gives realism scores to the images obtained by the generator. This allows that regardless of the quality of the generator's creations, it always receives adequate feedback.

The Figure 2.24, taken from the original article, shows the differences between the gradients obtained by the discriminator of the original GAN and the critic of the WGAN when learning to differentiate two Gaussians. As can be seen, the WGAN does not have vanishing gradients as the Vanilla GAN does.

WGAN had an improvement reflected in (Gulrajani et al. 2017). This new version is called WGAN-GP. GP stands for Gradient-Penalty, a mechanism to prevent gradient explosion on the critic.

WGAN-GP is a *state-of-the-art* GAN model, as it allows for better training stability. However, these problems continue to occur. In addition, it has a slightly high computational cost that lengthens training times.

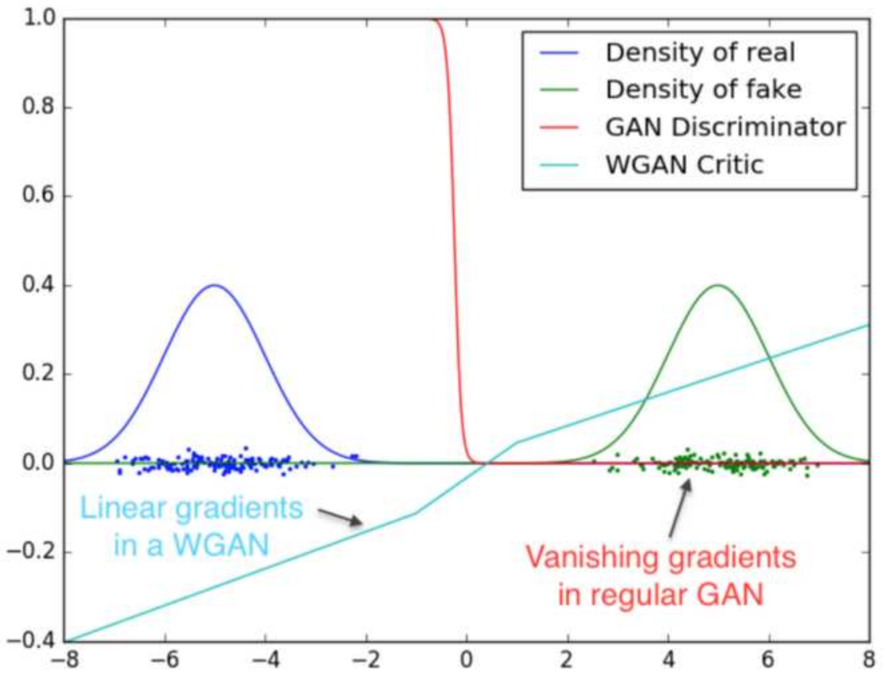


FIGURE 2.24. **WGAN and Vanilla GAN gradients.** Obtained from the original WGAN paper (Arjovsky et al. 2017).

2.4.4 Weight and Spectral Normalization

Weight normalization (Salimans and Kingma 2016) is a reparameterization of the weight vectors in a neural network which allows to stabilize the update gradients, thus avoiding gradient explosion and speeding up their training. This technique has proven its usefulness in multiple types of NN where the training can easily be destabilized due to the high number of internal parameters.

Spectral Normalization (Miyato et al. 2018) is a normalization technique used to stabilize training of the discriminator. This technique normalizes the weight for each layer (weight matrix) with its corresponding *spectral norm* also known as the matrix norm, the maximum singular value of a matrix. With spectral normalization, the weights can be normalized whenever there are updated. This creates a network that mitigates gradient explosion problems and therefore reduces instability in training. This mechanism has shown similar results to WGAN-GP with shorter training times.

Between these two techniques, spectral normalization has shown to provide greater power of representation by not restricting their area too much as Weight Normalization, as verified in the original article. However, weight normalization continues to provide competitive results when used on GANs (Xiang and Li 2017).

2.5 Fréchet Inception Distance

Fréchet Inception Distance also known as FID (Heusel et al. 2017) has stood out in recent years as a *state-of-the-art* performance metric in GANs. Although the similarity between sets of images remains an open problem in image processing, FID has been one of the latest heuristics designed to cope with this. This is a metric for evaluating the quality of generated images and specifically developed to evaluate the performance of GANs (Lucic et al. 2018).

FID use the pre-trained Inception-v3 CNN (Szegedy et al. 2016) for the feature extraction of the real (x) and synthetic (g) images. Specifically, the coding layer of the CNN (the last pooling layer prior to the output classification of images) is used to capture computer-vision-specific features of an input image, thus obtaining a feature-vector of 2048 numerical values. This feature space is interpreted as a continuous multivariate Gaussian distribution. Therefore, from the features obtained, it is calculated the Fréchet (also named Wassertein-2) distance between both distributions using their estimated mean (μ) and covariance (Σ). The lower this metric is, the more similar the two sets of images are, being zero when they are equal. The FID's formula is the following:

$$FID(x,g) = ||\mu_x—\mu_g||_2^2 + Tr(\Sigma_x + \Sigma_g—2(\Sigma_x\Sigma_g)^{\frac{1}{2}}) \tag{2.21}$$

where $||\mu_x—\mu_g||_2^2$ refers to the sum squared difference between the two mean vectors, $Tr(\cdot)$ refers to the trace linear algebra operation, i.e., the sum of the elements along the main diagonal of the square matrix and $(\Sigma_x\Sigma_g)^{\frac{1}{2}}$ is the square root of the square matrix, given as the product between the two covariance matrices.

The Figure 2.25, obtained from the FID original article, shows examples of its increase depending on the level of disturbance in the images.

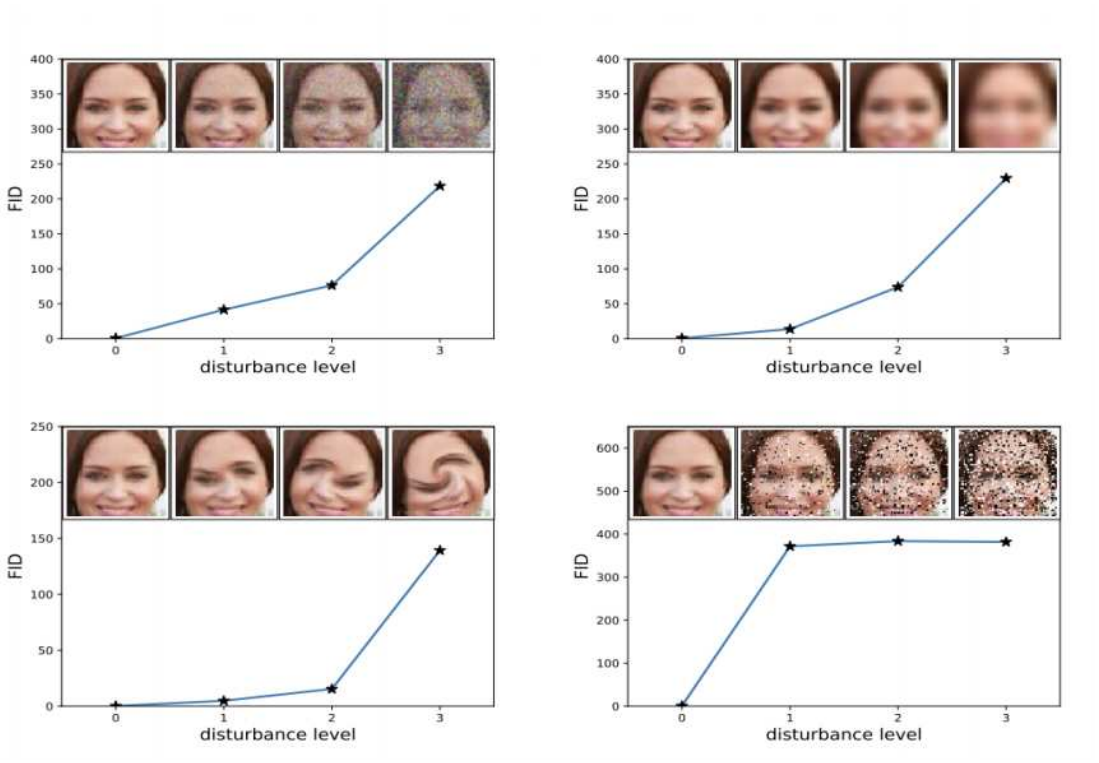


FIGURE 2.25. **FID evaluations with different disturbances.** Left to right, top to bottom: Gaussian noise, Gaussian blur, swirled faces, salt and pepper noise. Obtained from the original FID paper (Heusel et al. 2017).

2.6 Evolutionary Computing

Evolutionary Computing (EC) in essence refers to a set of search algorithms obtained from the theory of biological evolution used to tackle complex problems (Nayyar et al. 2018). EC is a set of engineering methods that imitates the mechanism of evolution in organisms and applies this to the deforming, synthesis, and selection of data structures. Their aim is to solve optimization problems (Iba 2018).

Despite the fact that there are numerous evolutionary computing algorithms, the common idea behind all of them remains the same; begin by randomly generating a set of potential solutions called *population*. Then a new population is obtained by iteratively modifying these potential solutions (*offspring*). The modification is done by iterative application of selection, crossover, and mutation operators. This process stochastically discards poor solutions and evolves fitter (better) ones. Due to the nature of this operators, it is expected that the evolved solutions will become better iteration by iteration (*generation*) (Bansal et al. 2019).

The adaptation of these solutions to their environment is measured by their fitness function (i.e. objective function), which is responsible for measuring the quality of the solution in the search space for a defined problem e.g. an optimization problem to minimize the value of a function $f(x)$, which is defined as:

$$\begin{aligned} &\underset{x}{minimize} && f(x) \\ &subject\ to && g_i(x) \leq 0, \quad i = 1, ..., m \\ & && h_j(x) = 0, \quad j = 1, ..., p \end{aligned} \tag{2.22}$$

where *x* is a solution vector of *n* variables and the function f maps the value of *x* ($\mathbb{R}^n$) to a real value ($\mathbb{R}$). The $g_i(x) \leq 0$ are called inequality constraints and $h_j(x) = 0$ are equality constraints, both must be fulfilled by the solution vector. The goal is to find x^* called the optimal vector, that minimizes f in such a way that any other solution (x) is greater than the value of this i.e. $f(x) > f(x^*)$. Therefore the purpose of EAs is to find the solution x^* or failing that, a competitive solution (or a set of them).

The representation of the solutions in the EAs has an important role, since depending on this choice, a solution can be evaluated directly through the fitness function or it will need a previous decoding step that maps it to a suitable representation to evaluate. For example, an encoding through the binary system will need to be decoded to the decimal system if the function requires it, or directly evaluated according to its sequence of bits. The choice of the encoding of the potential solutions is strongly linked to the representation of the search space, so its choice must be rigorously made in order not to cause bias in the search or restrict access to certain areas of the search space (Bansal et al. 2019).

Evolutionary computing differs from the traditional searching and optimizing in the following ways:

- **EC is population-based**: Utilizing potential solutions simultaneously to aid the searching process through the *search space*, the set of solutions among which the desired solution resides. They mostly use recombination to form a new solution by mixing the information of previous candidates.
- **EC is a metaheuristic**: Rather than using function derivatives or related knowledge, it uses direct *fitness* information that provides a heuristic estimate of solution quality.
- **EC is stochastic**: Using probabilistic, as opposed to deterministic, transition guidelines. EC gives good solutions, implying that the solutions are not necessarily optimal.

The previous characteristics allow customizable solutions for specific issues that are applicable on a variety of problems and deliver good solutions in adequate time. Evolutionary computing algorithms cater to these needs and thus answer the challenge of making automated solution methods for a large number of problems, which are increasingly mind-boggling, in the least possible time (Nayyar et al. 2018).

There are many families of algorithms that come under the umbrella of EC such as Genetic Algorithms (GAs), Genetic Programming (GP), Evolutionary Programming (EP), and Evolutionary Strategy (ES). However, evolutionary computing has broaden its scope and extended to include many areas, one of which is Swarm Intelligence.

2.6.1 Swarm Intelligence

Loosely speaking, Swarm Intelligence (SI) is part of the EC paradigm, but the interests in swarm intelligence are so overwhelming that has almost become a field of itself (Yang 2015). SI is quite a general concept that multiple agents interact and exchange information, following simple rules. Rather surprisingly, such simple systems can show complex, self-organized behaviour.

The word *swarm* refers to a collection of disorganized moving individuals or objects like insects, birds, fishes. More formally, a swarm can be considered a collection of interacting homogeneous agents or individuals. Researchers have developed many useful algorithms by modeling and simulating the foraging behavior of these individuals (Bansal et al. 2019).

The qualities of SI algorithms that researchers find attractive are their simplicity, ease of implementation and very few control parameters. Literature shows that lots of research papers reported the successful use of SI-based algorithms in a wide range of applications, like structural optimization, scheduling, bioinformatics, machine learning, data mining, medical informatics, image analysis, industrial problems, operations research, dynamical systems and even finance and business (Bansal et al. 2019, Nayyar et al. 2018).

Some merits of swarm intelligence are (Kumar et al. 2019):

- **Scability**: These algorithms are applicable for wide range of problems. SI systems are highly scalable, as they are able to explore the complete solution search space, and it is interpreted as the control mechanisms are independent on swarm size.
- **Adaptability**: These algorithms easily adapt the environmental conditions and try to converge into optimal solutions and react very quickly for varying surroundings and make use of self-organization capabilities and inherit auto-configuration. The adaptability permits them to adapt individual's behaviour to the environment on a run-time basis, with wide-ranging flexibility.
- **Cooperative robustness**: There is no central control in these algorithms and all the individuals work collectively because of its robust structure. However, the fault-tolerance skills are curiously high in SI systems, as all individuals communicate with others, therefore such systems do not have any chance of failure. The risk factor is reduced because the system works independently.
- **Individual Simplicity**: The individual agents in the swarm are very simple and not intelligent due to limited capabilities, but they collectively show intelligence.

The most representative algorithms in the field of SI are Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Firefly Algorithm (FA), among others. The present work focuses on the use of PSO for neuroevolution.

2.6.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is inspired on the flying pattern of bird flocks. Originally developed by Kennedy and Eberhart (1995); it was intended to be applied to optimization problems. In PSO the individuals, called *particles*, move in a multidimensional space (search space) and each of them represents a potential solution.

Each particle stores its own *position* vector in the search space expressed by a vector of real numbers (in the original version of PSO) and their velocity vector, which represents the rate of the change of direction and magnitude of its movement, also represented by a real vector. For a D-dimensional search space, the position vector of the *i-th* particle of the swarm at generation (iteration) *t* is represented by a D-dimensional vector $x_i^t = (x_{i1}^t, x_{i2}^t, ..., x_{iD}^t)^T$. The velocity is represented by another D-dimensional vector $v_i^t = (v_{i1}^t, v_{i2}^t, ..., v_{iD}^t)^T$. Each particle has also knowledge of its best historical position at which the individual obtained the highest fitness value, called ***pBest*** ($pBest_i^t = (pBest_{i1}^t, pBest_{i2}^t, ..., pBest_{iD}^t)^T$). All individuals also share information regarding the position with the highest fitness value which has been found historically up to generation *t*, known as ***gBest*** ($gBest^t = (gBest_1^t, gBest_2^t, ..., gBest_D^t)^T$). The position vector and the pBest and gBest vectors have a fitness value assigned relative to the quality of the solution they represent ($fitness_i^t$).

The particle swarm is not merely a collection of particles. A particle alone is powerless to solve any kind of problem; headway occurs only when the particles communicate. Each member of the swarm interacts with others using simple and limited actions. Particle swarms usually use three components to determine the searching behaviour of each individual (Nayyar et al. 2018):

- **Cognitive**: Responsible for weighing the importance an individual gives to its own knowledge of the world.

- **Social**: Weighs the importance an individual with respect to the cumulative knowledge of the swarm as a whole.
- **Inertia**: Specifies how fast individuals move and change direction over time.

With the above in mind, the presence of these components can be observed in the formula in charge of updating the velocity of the *i-th* particle:

$$v_{id}^{t+1} = w \times v_{id}^{t} + c_p \times r_p \times (pBest_{id}^{t} — x_{id}^{t}) + c_g \times r_g \times (gBest_{d}^{t} — x_{id}^{t}) \tag{2.23}$$

where *w* is a constant called *inertia weight* that adjusts how much the previous velocity affects the new velocity (**Inertia component**); c_p and r_p are a constant (user defined) and a random uniform number ([0,1)), respectively, that determine the influence of pBest (**Cognitive component**) and finally, similar to the last pair, c_g and r_g determine the influence of gBest (**Social component**).

The formula in charge of updating the position of the particles is:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{2.24}$$

The Figure 2.26 represents the behavior of a particle when updating its position, and Algorithm 1 shows the general pseudocode of the classic version of PSO.

PSO has the characteristic of allowing a fast convergence (Sahu et al. 2012) i.e. to find a solution in an optimum, generally a local one, which causes the rest of the population to turn in that direction. This could generate an early stagnation of the search, reducing the exploration power of the algorithm (and promoting exploitation). However, this ability is useful when computational resources are limited since it allows to shorten search times and then in fewer generations to find competent solutions. This allows avoiding extensive searches that, on the contrary, more popular evolutionary computing algorithms such as the Genetic Algorithm would take longer to converge towards this type of solutions and that

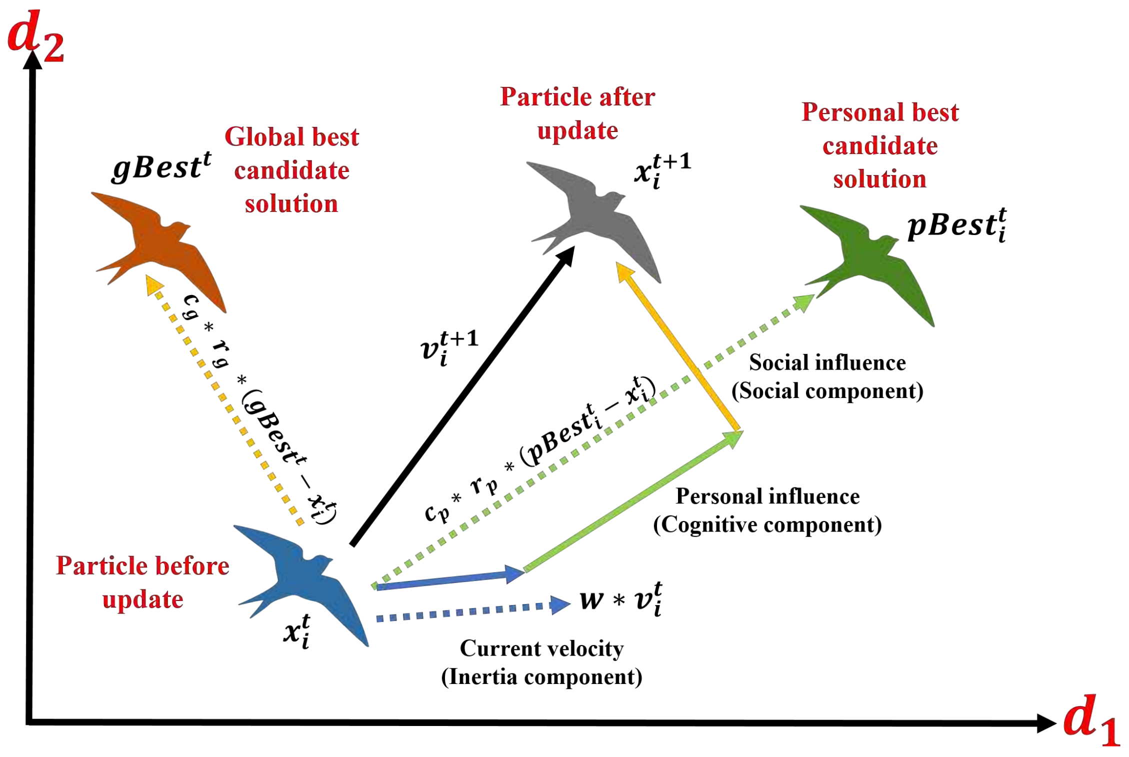

d_2
d_1
Global best candidate solution
$gBest^t$
Particle after update
x_i^{t+1}
Personal best candidate solution
$pBest_i^t$
$c_g * r_g * (gBest^t - x_i^t)$
v_i^{t+1}
$c_p * r_p * (pBest_i^t - x_i^t)$
Social influence (Social component)
Personal influence (Cognitive component)
Particle before update
x_i^t
$w * v_i^t$
Current velocity (Inertia component)

FIGURE 2.26. **Example of a PSO 2D particle update mechanism.**

PSO variants have been shown to obtain better results than these, when they are limited to a low number of calls of the fitness function, as shown in (Piotrowski et al. 2017).

2.6.3 Neuroevolution

Neuroevolution, which draws inspiration from the biological process that produced the human brain, design and train neural networks with evolutionary computing algorithms (Miikkulainen et al. 2019). Such methods enable important capabilities that are typically unavailable to classical approaches. Such capabilities for neural networks include learning their building blocks (e.g. activation functions), hyperparameters (e.g. learning rates), architectures (e.g. number of neurons per layer, number of layers) and even the rules for learning themselves (Stanley et al. 2019).

This EC application has allowed to obtain better performances in multiple areas of DL, from building more accurate, more complex or even shallower networks, with methods and sets

Algorithm 1 Classic PSO Pseudo-code

```
Input: Number of particles, N; number of dimensions, D; number of generations (iterations), T.
Output: $gBest^T$ (Best solution found)
1: Random initialization: D-dimensional swarm of N particles S ($x_1^1, ..., x_N^1$); velocity vector of each particle ($v_1^1, ..., v_N^1$).
2: Evaluate fitness of each particle ($fitness_1^1, ..., fitness_N^1$) and choice pbest (($pBest_1^1, ..., pBest_N^1$)) and gBest ($gBest^1$).
3: for t=1 to T do
4:     for i=1 to N do
5:         for d=1 to D do
6:             Update $v_{id}^t$ with (2.23)
7:             Update $x_{id}^t$ with (2.24)
8:         end for
9:         Evaluate fitness of $x_i^{t+1}$ ($fitness_i^{t+1}$)
10:        if $fitness_i^{t+1}$ better that fitness of $pBest_i^t$ then
11:            $pBest_i^{t+1} \leftarrow x_i^{t+1}$
12:        end if
13:    end for
14:    for i=1 to N do
15:        if $pBest_i^{t+1}$ better that fitness of $gBest^t$ then
16:            $gBest^{t+1} \leftarrow pBest_i^{t+1}$
17:        end if
18:    end for
19: end for
```

of training parameters not previously developed, all according to the needs of the task or the avaible computing resources (Galván and Mooney 2020, Martinez et al. 2020).

As GANs are DL models, their designing and training through these EC methods have been explored and thereby improving their performance and helping to solve the major problems that afflict them.

2.7 Biomedical Imaging and Data Augmentation

Biomedical imaging concentrates on the capture of images for both diagnostic and therapeutic purposes. Snapshots of *in vivo* physiology and physiological processes can be garnered through advanced sensors and computer technology (Lu et al. 2017).

Biomedical imaging technologies utilize either X-rays (CT scans), sound (ultrasound), magnetism (MRI), radioactive pharmaceuticals (nuclear medicine: SPECT, PET) or light (endoscopy, OCT) to assess the current condition of an organ or tissue and can monitor a patient over time for diagnostic and treatment evaluation (Suetens 2017).

These types of images supported with Deep Learning models have had innumerable successes in tasks related to the diagnosis and prediction of diseases (Litjens et al. 2017, Bakator and Radosav 2018, Mohapatra et al. 2021). However, these DL models require a vast amount of data for proper generalization and thereby raise their performances. The foregoing are conflicts with the use of biomedical images, as are scarce for four main reasons (Guibas et al. 2017):

- **Privacy:** Taking biomedical images of a patient implies confidentiality clauses since they are dealing with sensitive information that may violate their right to privacy. Therefore, the patient's approval is not always available to use these data outside of those strictly necessary for his/her diagnosis and treatment.
- **Health bias:** Those patients who are afflicted by illnesses are more likely to attend to health centers than those that are healthy, thus more information about ill patients is available.
- **Cost:** Some tests to obtain these types of images are usually expensive so not every patient can take them.
- **Risk:** Taking these images may involve exposing the patient to radiation doses that can affect them in large quantities.

That is why the use of techniques that allow to artificially increase the number of available samples has been very useful in DL. These techniques are called Data Augmentation (DA) (Abdollahi et al. 2020). Classic DA methods involve oversampling the available data by applying a series of spatial transformations to images, such as rotation, translation, clipping, and filter application. The Figure 2.27 shows a series of examples of these techniques.

DA has been proven to improve the performance of DL models (Perez and Wang 2017). However, DA tools have a limit in improving DL systems (Buda et al. 2018), so a generative

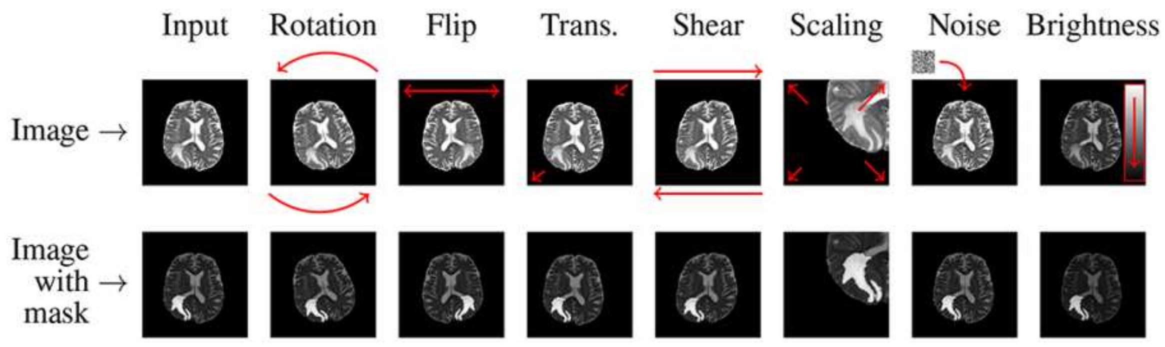


FIGURE 2.27. **Example of classic DA techniques in a brain MRI.** Obtained from (Nalepa et al. 2019).

approach such as GANs has allowed to further extend the limits of exploitation of the set of original images for being used in multiple medical applications (Shin et al. 2018, Sorin et al. 2020, Kazeminia et al. 2020).

2.8 Chest X-ray Images

Chest X-rays (CXR) are a type of biomedical images with multiple uses in the medical area, many of these applications have made use of Deep Learning. DL-based computer vision models are used to detect the attributes of these diseases and design appropriate models to support the biomedical area. Its most prominent applications are the diagnosis of multiple diseases (e.g. lung cancer or tuberculosis) or the segmentation of areas of interest (Gordienko et al. 2018, Stirenko et al. 2018, Bhandary et al. 2020). However, as previously mentioned, these DL models require a vast amount of data. Therefore, the use of Generative Adversarial Networks has made it possible to increase the available CXR image bases for training systems that support medical professionals in the diagnosis of this ailments.

The use of CXR images has increased with the sudden appearance of the Severate Acuate Respiratory Syndrome Coronavirus (SARS-Cov-2), known as COVID-19, since the end of 2019 (Bonilla-Aldana et al. 2020). This pathology has caused an exponential growth of infected people around the world, escalating to the level of a pandemic, causing the loss of countless lives. The disease generated by this new virus, belonging to the Coronavirus family (CoV), infects the lungs and causes life-threatening respiratory syndromes (Huang et al. 2020). One of the main complications caused by this disease is pneumonia with distinctive characteristics, different from pneumonia by other causes such as bacterial infection. The search for these singularities is accomplished by CXR images of patients afflicted with pneumonia and suspected of COVID-19, in which the lungs of patients are shown. An example of this kind of images is shown in Figure 2.28. However, these characteristics are hardly detectable with the naked eye by health experts (Ng et al. 2020, Huang et al. 2020). Motivated by that, in recent months projects and works have been developed focusing on designing DL models that perform the task of detecting these qualities through the processing of CXR images in order to support the diagnosis and medical decision making. This would also reduce the waiting time of the current COVID-19 detection technique called Reverse-Transcriptase Polymerase Chain Reaction (RT-PCR) (Corman et al. 2020). The main approach to detecting COVID-19 using CXR images has been the use of CNN for its classification, which has obtained quite good results (Kumar et al. 2020, Shi et al. 2020, Zhang et al. 2020, Jain et al. 2021, Ismael and Şengür 2021).

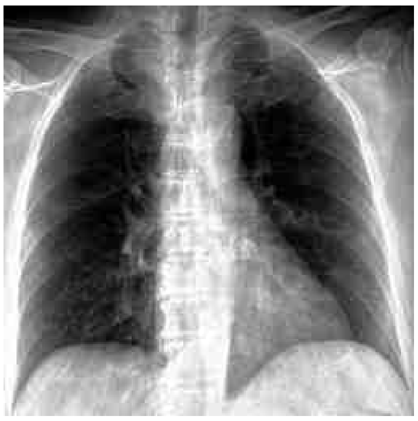


FIGURE 2.28. **Example of chest X-ray image from a patient with COVID-19 pneumonia.** Obtained from (Chowdhury et al. 2020).

However, prior to the COVID-19 scenario, CXR imaging has also been used for other types of illnesses such as pneumonia from bacterial or viral causes (Rajpurkar et al. 2017,Baltruschat et al. 2019). That is why there are datasets focused on this type of diseases, also enriched with CXR images of healthy patients in order to have the necessary data to make an adequate discrimination and contrast between cases with any of these diseases and healthy cases.

Having available public data sets of CXR images of various pathologies (including COVID-19 pneumonia) allows for an adequate case study to apply the neuroevolution of GANs in the area of biomedical images.

2.9 Chapter Summary

This chapter addressed the basic fundamentals of Deep Learning, the Convolutional Neural Networks as well as the Generative Adversarial Networks and their variants and additions in order to improve their stability. Additionally, Evolutionary Computing and Swarm Intelligence were presented as well as the branch focused on evolving Neural Networks called Neuroevolution. Finally, the contributions of GANs to Data Augmentation were mentioned as well as the case study addressed in this research work, the chest X-ray images.

CHAPTER 3

Literature Review

This chapter presents a review of the literature related to the progressive growth of GANs as well as the neuroevolution of GANs. Further, the works that use GANs for the synthesis of chest X-ray images are exposed. To the best of our knowledge this is all the work done in these areas until May 2021.

3.1 Progressive Augmentation of GANs

Since the conception of the GANs in 2014 there have been a plethora of works dedicated to improving the stability of their training and thereby avoiding related problems e.g. mode collapse and vanishing gradient. Part of that work has been dedicated to proposing better GAN architectures, such as DCGAN (see section 2.4.1); implement better cost functions that improve learning e.g. W-GAN (see section 2.4.3); or even limit the learning gradients of the NN, in order to preserve the balance between both modules of the GANs, as proposed by Spectral Normalization (see section 2.4.4).

In recent years, one approach that has led to promising results in improving GANs has been their *progressive augmentation*. This term refers to when the NN have an addition of sets of layers or increase the complexity of the task to be carried out by one of the NN as their training progresses, allowing to obtain more complex outputs than in the previous step, until reaching a desired final complexity or size of the output (e.g. resolution of the image).

Table 3.1 shows a synthesis of the works related to GANs progressive augmentation. This table condenses the name and reference of the work, the datasets used, the evaluation metrics of its results and its highlights.

As can be seen, there have been few works with this approach all of them applied to popular datasets and without any focus on biomedical images or any real applications. However, all of these have presented an improvement in the stability of the GANs. Among these works, the most prominent and the one that has been used as a starting point in the present thesis is **Pro-GAN**, a brief description of it will be provided below.

3.1.1 Pro-GAN

Progressive growing of GANs is the approach taken by Pro-GAN (also known as PGAN or PGGAN). This presents a process to progressively add layers to the GAN networks, as the training is carried out. This set of layers predefined by the authors gradually doubles the resolution of the output images until the desired final resolution is obtained. The principle on which they were based was that training GANs at low resolutions is less complex, which is why the GAN obtains good results and has a more stable training. Through increasing the complexity to learn, the GAN has a moderate learning of the more complex distributions given with greater resolution. This allows to obtain better and more stable results compared to training from the beginning to the GAN with the complexity of the final resolution.

The representation of the Pro-GAN process is shown in Figure 3.1 taken from the original article. In that image it is observed how as the training progresses, new layers are added to the generator and the discriminator, both networks as a *mirror* image with respect to the other.

Multiple works have been based on the Pro-GAN mechanism, including the synthesis of biomedical images, those works are concentrated in Table 3.2, where the reference and the type of synthesized biomedical image are included. As shown, this mechanism has many successes in the area of biomedical images, allowing to expand the number of instances available to generate and train pro-health models, such as the support of diagnosis through

Name (Authors, year)	**Architecture**	**Datasets**	**Evaluation metrics**	**Highlights**
Laplacian GAN (Denton et al. 2015)	Vanilla GAN	• STL • LSUN • CIFAR-10	Log-Likelihood	• Use a sequence of generators to scale the resolution. • Each generator has an output conditioned by the image synthesized by the generator of the previous resolution.
Pro-GAN (Karras et al. 2017)	DCGAN-based	• CelebA • LSUN • CIFAR-10	• Sliced Wasserstein Distance (SWD). • Multi-scale structural SIMilarity (MS-SSIM).	• Adding preset layers (in generator and discriminator) as the training goes on. • Training speedup compared to fixed models.
PA-GAN (Zhang and Khoreva 2019)	DCGAN-based (with spectral normalization)	• CIFAR-10 • MNIST • F-MNIST • CelebA	FID	• Progressively increase the discriminator input so that the complexity of its classification increases to avoid early performance saturation. • Architecture-agnostic.
SinGAN (Shaham et al. 2019)	Own-design	Web images	• Amazon Mechanical Turk • Single FID	• Single image learning. • The synthesized image of the previous resolution is concatenated to a random noise image and is input to the generator of the next resolution. • Applications in paint to image, editing, harmonization, Super-resolution and animation.
SP-GAN (Song et al. 2020)	DCGAN-based	• MNIST • CelebA • CIFAR-10 • CUFS	• Structural SIMilarity (SSIM). • Riesz-transform based Feature SIMilarity metric (RFSIM).	• Self-growing and pruning of GAN with preset layers. • Design and use of a new GAN's cost function. • Training speedup.

TABLE 3.1. GANs Progressive Augmentation literature review summary.

the detection of pathologies in this type of images. So far, no work has used Pro-GAN for the synthesis of CXR images, so based on its previous successes in biomedical imaging its use in this type of images is promising, which is why it has been one of the approaches used in this thesis.

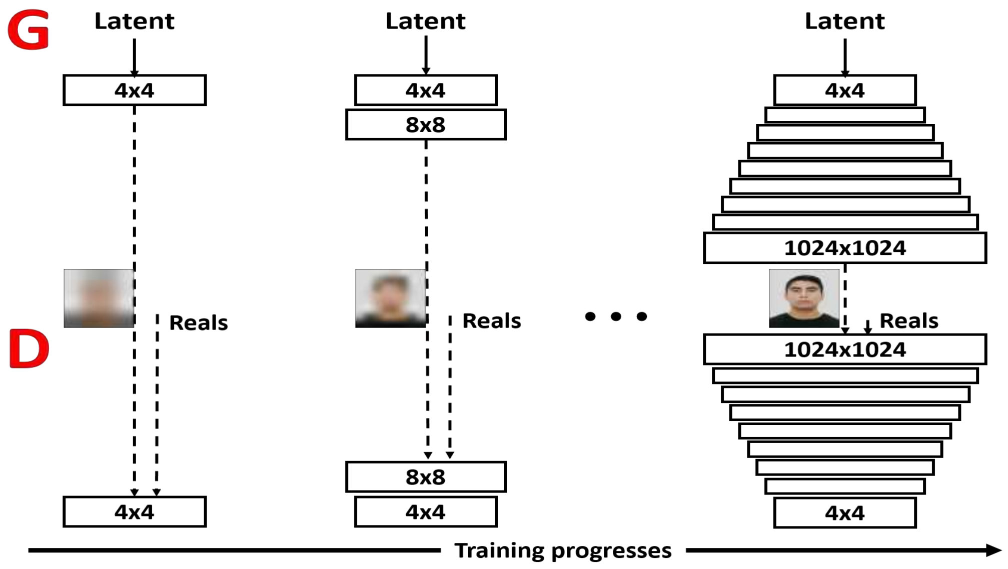


FIGURE 3.1. **Progressive Growing of GANs.** Inspired by (Karras et al. 2017)

Reference	**Synthesized biomedical image**
Beers et al. 2018	Retinal images
Korkinof et al. 2018	Mammograms
Eklund 2019	Brain volumes (voxels)
Han et al. 2019	Magnetic resonances
Teramoto et al. 2020	Lung cytological images
Abdelhalim et al. 2021	Skin lesions

TABLE 3.2. Biomedical image synthesis using Pro-GAN literature review summary.

3.2 GANs Neuroevolution

Previous works regarding the neuroevolution of GANs are summarized in Table 3.3, which includes the reference and the name given to the algorithm, the evolutionary algorithm used, the GAN's architecture (if it is evolvable, the used network encoding), variation operators, fitness function, as well as the type of offspring selection, the evaluation metrics and the training datasets.

As can be seen, most of the papers are focused on the use of Genetic Algorithms and Co-evolution. The use of Swarm Intelligence algorithms is a great absence. Also, many of these works have a DCGAN-based architecture which shows its status as a *state-of-the-art* architecture.

All works that evolve its architecture use a list-based encoding scheme, in which there are declarative blocks that contain the hyperparameters of each layer in the NN.

In the field of evaluation metrics there is no general consensus, so multiple approaches have been used. However, the Fréchet Inception Distance (FID) (see Section 2.5) has stood out in recent years as a *state-of-the-art* metric.

Something else that can be seen in the revision of the literature is that the most of the work in GANs neuroevolution has focused on the use of well-known image benchmarks (e.g. CIFAR-10, MNIST, CelebA), which contain thousands of images, yet without exploring sets of real applications, such as the use of biomedical images, which usually do not usually exceed a thousand images.

Name (Authors, year)	**EA type**	**Architecture (Encoding scheme)**	**Variation Operators**	**Fitness**	**Selection**	**Evaluation metrics**	**Datasets**
CAGAN (Ni et al. 2018)	Cultural algorithm	DCGAN-based	Mutation (discriminator weights)	Custom	Best individual	Inception score	• CIFAR-10 • STL-10
Pareto GAN (Garciarena et al. 2018)	Genetic algorithm	Evolvable (List-based)	• Crossover • Mutation	IGD (Pareto front)	Pareto dominance	IGD	Bi-objetive functions
Lipizzaner (Al-Dujaili et al. 2018)	Coevolution	DCGAN-based	Mutation (weights)	GAN loss	Spatial	Qualitative	• MNIST • CelebA
Mustangs (Toutouh et al. 2019)	Coevolution	DCGAN-based	Mutation (weights and loss)	GAN loss	Spatial	FID	• MNIST • CelebA
E-GAN (Wang, C. et al. 2019)	Evolutionary learning algorithm	DCGAN-based	Mutation (loss function)	Custom	Best individual	• Inception score • FID	• CIFAR-10 • LSUN • CelebA
COEGAN (Costa et al. 2019)	Coevolution.	Evolvable (List-based)	Mutation (architecture)	• FID • GAN loss	NEAT-based	FID	• MNIST • F-MNIST • CelebA
CA-GAN (Mehta et al. 2019)	Cultural algorithm	Evolvable (List-based)	Mutation (architecture, weights and loss)	GAN loss	Best individuals	Classification performance	Face images
DEGAN (Zheng et al. 2019)	Differential evolution	DCGAN-based	• Mutation • Crossover (images in training set)	Custom	Best individual	Average precision (image outlines)	• BSDS500 • NYUD
GAN-GA (Cho and Kim 2019)	Genetic algorithm	DCGAN-based	• Crossover • Mutation (both for generated images)	Discriminator value of the generated image	Best individuals	GAN loss	MNIST
HEO-GAN (Korde et al. 2019)	Genetic algorithm	• DCGAN-based • WGAN • Vanilla GAN	• Crossover • Mutation (weights)	FID	Best individuals	FID	• MNIST • CelebA
CO-EGAN (Shu et al. 2019)	Coevolution.	CycleGAN (Binary strings)	• Mutation • Crossover (Generator architecture)	Custom	Best individual	• FLOPs • FID • FCN	Multiples focused on style transfer
NSLC-COEGAN (Costa et al. 2020a)	Coevolution.	Evolvable (List-based)	Mutation (architecture)	NSLC-based	Best individual	FID	MNIST
COEGAN+Skill (Costa et al. 2020c)	Coevolution.	Evolvable (List-based)	Mutation (architecture)	• FID • Skill rating	NEAT-based	FID	• SVHN • F-MNIST • CelebA

TABLE 3.3. GANs neuroevolution literature review summary.

3.3 CXR Synthesis with GANs

Table 3.4 summarizes GANs implementations to synthesize CXR images. It contains the work reference, the GAN architecture used, the resolution of the synthetic images, the evaluation metrics of the results and the synthesized CXR images classes.

Once again, the hegemony of the used DCGAN-based models can be observed. Disadvantageously, all of these architectures are handcrafted and they might not be generalizable to other lung pathologies or biomedical images.

The resolutions worked are diverse, but most take a resolution equal to or less than 256x256 pixels, proving that this is sufficient for use in subsequent activities. Regarding the metrics to evaluate their results, they put aside the similarity between the sets of images to focus on improving the performance of specific tasks such as classification.

As can be seen, most of the work was carried out in 2020, due to the increase in implementations developed in order to obtain synthetic images to improve the DL models focused on the detection of COVID-19 pneumonia from CXR images. For the rest of the works, the majority was in charge of synthesizing mainly CXR of pneumonia and healthy classes, mainly due to the fact that these are the most abundant in open access sources.

Authors, year	**Architecture**	**Image size**	**Evaluation metrics**	**Synthesized CXR class**
Madani et al. 2018	DCGAN-based	128x128 pixels	Classification performance	• Healthy • Cardiomegaly
Salehinejad et al. 2018	DCGAN-based	256x256 pixels	• Radiologists • Classification performance	• Cardiomegaly • Pneumothorax • Edema • Effusion • Healthy
Middel et al. 2019	WGAN-GP	128x128 pixels	• FID • Structural SIMilarity (SSIM) • Mean Squared Error (MSE) • Specificity • Generalization ability • PCA	• Pneumonia • Healthy
Zhang et al. 2019	Own design	512x512 pixels	• Sliced Wasserstein Distance (SWD) • Multi-Scale Structural SIMilarity (MS-SSIM) • FID	• Pneumonia • Healthy
Karakanis and Leontidis. 2020	Conditional GAN	446x446 pixels	Classification performances	COVID-19
Khalifa et al. 2020	DCGAN-based	256x256 pixels	Classification performance	• Healthy • Pneumonia
Loey et al. 2020	DCGAN-based	512x512 pixels	Classification performance	• Healthy • Pneumonia bacterial • Pneumonia virus • COVID-19
Menon et al. 2020	MTTGAN	128x128 pixels	• Classification performance • Radiologists	COVID-19
Shams et al. 2020	DCGAN-based	64x64 pixels	GAN loss	• Healthy • COVID-19
Waheed et al. 2020	DCGAN-based	112x112 pixels	• Classification performance • PCA	• Healthy • COVID-19
Zulkifley et al. 2020	DCGAN-based	224x224 pixels	Classification performances	COVID-19
Karbhari et al. 2021	ACGAN	112x112 pixels	• FID • Inception score • Classification performances	• Healthy • COVID-19
Kora Venu and Ravula. 2021	DCGAN-based	128x128 pixels	• Classification performances • FID	Healthy
Sheykhivand et al. 2021	Vanilla GAN	224x224 pixels	Classification performances	COVID-19

TABLE 3.4. CXR synthesis with GANs literature review summary.

3.4 Chapter Summary

This chapter presented the background in the areas of progressive augmentation of GANs, GANs neuroevolution and synthesis of CXR images using GANs. Through these studies, the following observations were obtained:

- There is an improvement in the stability of the GANs when they are obtained through a progressive regime i.e. a gradual increase in the complexity of training.
- Until now, Swarm Intelligence has not been used in Evolutionary Computing methods for neuroevolution of GANs, a gap that misuses the potential of these algorithms to obtain faster convergences that implies savings in computational costs.
- The networks for the synthesis of CXR are entirely developed under the expertise of the researchers and based on the DCGANs, an approach that may not be generalizable to other pathologies or areas different from the original tasks.

CHAPTER 4

Proposed Algorithm

This chapter presents the algorithm proposed for the neuroevolution of GANs, named DCGAN-PSO and its multi-class version named cDCGAN-PSO.

4.1 DCGAN-PSO

The algorithm proposed here is based on three fundamental principles supported by the literature:

- *Training a DCGAN with progressive resolution growth allows for better training stability.*
- *Using PSO allows rapid convergence, which shortens search times.*
- *DCGANs being composed by CNNs can exploit previous knowledge in CNN neuro-evolution.*

Based on these ideas, DCGAN-PSO was developed. This algorithm evolves DCGAN-based architectures and train them by gradually increasing the resolution of the generated images adding layers not preset but constantly evolving using a version of Particle Swarm Optimization.

DCGAN-PSO presents the following novelties to the field of GANs neuroevolution:

- It is the first algorithm on GANs neuroevolution based on Swarm Intelligence, to the best of the author's knowledge.

- First to use biomedical imaging as a target area in GANs neuroevolution, to the best of the author's knowledge.

The different aspects that make up the DCGAN-PSO will be addressed below.

4.1.1 Base Works

The proposed algorithm was based primarily on the work carried out by **Pro-GAN** in the progressive growth of DCGANs (work previously seen in section 3.1.1) and **Junior and Yen** (2019) work to be detailed below.

psoCNN is the name given to the algorithm created by Junior and Yen which uses Particle Swarm Optimization to evolve the architecture of CNN with the purpose of finding classifiers with high performances in different benchmark datasets (e.g. MNIST) automatically. Some of the achievements obtained by his work are:

- Reduction of search times (mainly due to the rapid convergence of the PSO) with few particles and iterations, then reducing computational costs with respect to other CNN neuroevolution algorithms.
- Using an almost standard PSO algorithm with few parameters that avoid excessive complexity both in its implementation and in its operation.
- Design of operators that allow the evolution of the networks represented respecting the nature of the original PSO.
- Obtaining better results than other *state-of-the-art* neuroevolution algorithms in CNN (including another PSO).

It is for these highlights that psoCNN was taken as a basis for this proposal, making the appropriate adjustments to be able to be used with DCGANs.

4.1.2 Pseudocode of DCGAN-PSO

Algorithm 2 has the pseudocode of the mechanism implemented by DCGAN-PSO for the evolution of GANs for the synthesis of CXR images of COVID-19 pneumonia.

As seen in the pseudocode, the algorithm receives as input the datasets of CXR images in the different resolutions used, as well as the parameters that define its behavior. As output, the best trained particle (gBest) will be obtained, which will generate images with 256^2 pixels of resolution.

The algorithm begins the iteration through the various resolutions, starting from 4^2 pixels of resolution until gradually reaching 256^2 pixels, doubling it at each step (line 1). For each resolution, the swarm will be initialized with particles that generate images in the current resolution, as well as the respective pBest of each particle and the gBest (line 2). Therefore, for a defined number of generations, the particles will be decoded in their corresponding DCGANs, they will be then trained and later their fitness will be evaluated (lines 3-15). According to their fitness, the pBests and the gBest will be updated (lines 16-21). After that, the velocity and position of each particle will be updated (lines 22-24). At the end of the iterations through all the necessary resolutions, the trained gBest is returned (line 28) which is the best trained DCGAN.

The detailed explanation of each mechanism in DCGAN-PSO will be presented in the following sections.

4.1.3 Encoding-Scheme

A list-based encoding is used to represent the topology of the CNNs in the DCGAN model. Each module in the list represents a layer and the sequence of modules in the list is used to implement the generator and by using a *mirror* image of this, the discriminator is implemented. This list is the representation of the *particle* in the proposed PSO. However, a faithful mirror image is not implemented, because the nature of the tasks of both NN, the layers used and the restrictions implemented are different between the two parts of the DCGAN.

Algorithm 2 DCGAN-PSO

Input: Training CXR dataset of each resolution used, *N° generations per resolution*, *N° epochs per particle training*, *Swarm size*, C_g, *resolutions list*.

Output: Trained DCGAN of CXR images of pneumonia caused by COVID-19 in 256^2 pixels resolution (gBest).

```
1:  for resolution in resolutions list do
2:    Initialization: Swarm ← Initialize Swarm(resolution), pBests, gBest.
3:    for N° generations per resolution do
4:      for particle in swarm do
5:        if resolution = $4^2$ pixels then
6:          Network ← DCGAN particle architecture
7:        else
8:          Network ← DCGAN previous resolution pBest architecture
9:          Network weights ← Previous resolution pBest weights
10:         Network ← Add DCGAN particle architecture at the end of the Network
11:       end if
12:       for N° epochs per particle training do
13:         Train Network with CXR dataset of its respective resolution
14:       end for
15:       particle fitness ← FID(Network)
16:       if particle fitness $\leq$ particle pBest fitness then
17:         particle pBest ← particle
18:         particle pBest weights ← particle weights
19:       end if
20:     end for
21:     gBest ← pBest with lowest fitness
22:     for particle in swarm do
23:       particle velocity ← UpdateVelocity(particle)
24:       particle ← UpdateParticle(particle velocity)
25:     end for
26:   end for
27: end for
28: RETURN gBest
```

The differences in the implementation between both NN are the following:

- The transposed convolutional layers in the generator are replaced by convolutional layers in the discriminator. This is because the task of the generator is to expand the feature maps of the previous layer and the discriminator aims to obtain finer motifs that allow the classification of the images.

- While the generator could use convolutional layers to process the feature maps of the previous layer while keeping them with the same two dimensions i.e. length and width, they are not used in the discriminator. This is in order to reduce computational costs, since it could cause very deep discriminators with a very high number of weights to train; this in addition to avoid over-complexity that can lead to problems such as over-fitting or power imbalance in the GAN.

The different layers used in the generator and their counterpart in the discriminator, as well as the hyperparameters that define them are shown in Table 4.1. The stride has a fixed value in the layers since this allows them to fulfill their nature of keeping, doubling or reducing the size of the input. Padding is calculated at the time of implementation taking into account the characteristics of the filters to also ensure preserving or modifying the input size.

An example of a DCGAN's list and their respective decoded NN can be seen in Figure 4.1. In that image, section **(a)** shows a list (particle) with three blocks, where each block represents a different layer. The sequence of these blocks is faithfully implemented in the generator ((**b**)-left), while for the discriminator ((**b**)-right) the *mirror* image is implemented with the equivalent layers but in the opposite direction (i.e. first the convolutional part and then the fully connected layer). The dotted lines indicate the relationship between the module in the list and the implemented layer. Both sets of layers use the respective parameters that are marked in each block of the particle.

4.1.4 Difference Between Particles

The first step prior to updating the particles in the PSO is to measure the *difference* between two particles, P_1 and P_2, since these may have different sizes and design patterns. This difference will be represented in a difference vector $(P_1 - P_2)$. The difference only takes into consideration the type of layer of each particle. If two layers in the same position of both particles are the same type then the difference is zero, independent of their hyperparameters. This *zero-difference* indicates that the layer will remain when the particle architecture

Generator layer	Description	Discriminator layer	Description	Hyperparameters
Transposed convolutional	Increases the size (length and width) of the output of the previous layer by a factor of 2 (stride of 2).	Convolutional	Decreases the size (length and width) of previous layer output depending on filter characteristics.	• Filter size • Number of filters • Stride = 2 • Padding: Calculated according to filter
Convolutional	Keeps the size (length and width) of the previous layer output	Does not apply	Does not apply	• Filter size • Number of filters • Stride = 1 • Padding: Calculated according to filter
Fully connected	Processes and resizes the random noise input	Fully connected	Processes the high-level features (obtained by the convolutional part of the NN) to perform the classification.	Number of neurons

TABLE 4.1. Different types of layers used with their equivalents according to the GAN module. The first four columns represent the pair of equivalent layers used in the generator and discriminator, respectively, with the definition of the activity they perform. The last column shows the hyperparameters that define both layers.

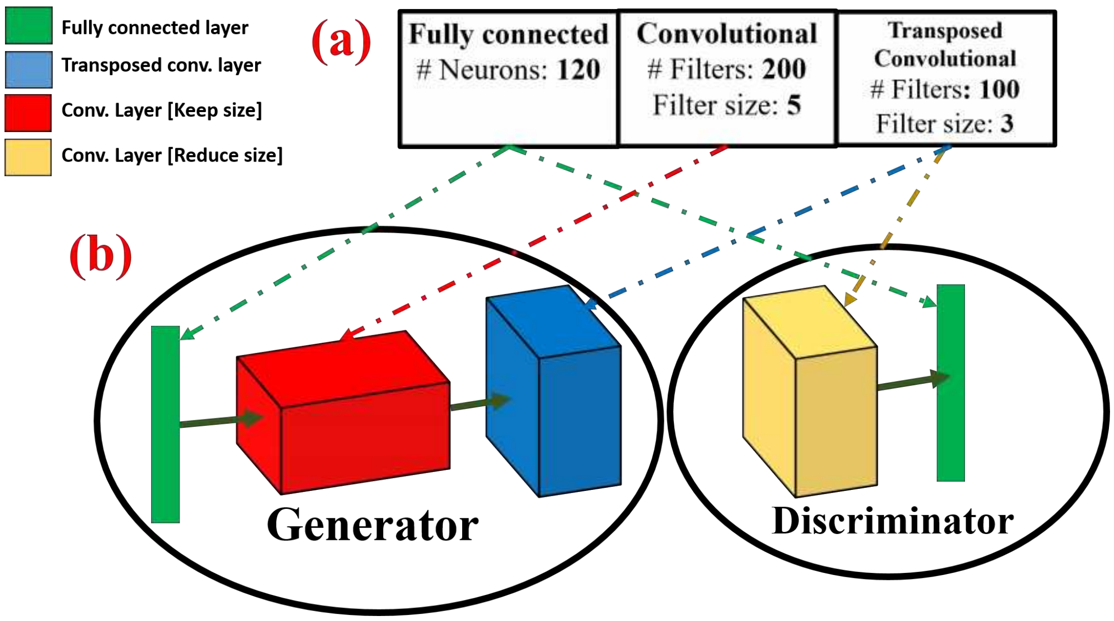


FIGURE 4.1. **Particle encoding-decoding.**

updates. If both layers are of different type, P_1 will have priority and its layer will be preserved, including its hyperparameters. If P_1 has fewer layers than P_2 then **-1** will be added at the end of the difference vector to denote that these extra layers in P_2 should be eliminated when updating the particle. In the opposite case, if P_1 has more layers than P_2, then indicators **+L** will be added that represent that layers of type *"L"* will be added in those positions when updating the particle (with randomly chosen hyperparameters). Figure 4.2 shows examples to measure the difference between particles.

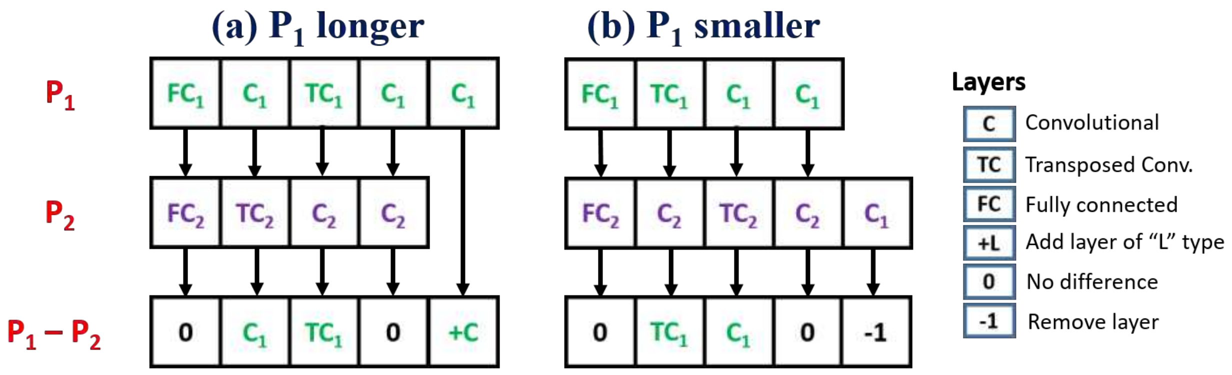


FIGURE 4.2. **Measurement of differences between particles.** (a): P_1 has more layers than P_2. (b): P_1 has fewer layers than P_2. Inspired by (Junior and Yen 2019).

4.1.5 Velocity Operator

The *velocity* computation of a particle (P), is done by comparing it with their *pBest* and *gBest*. Thus, the differences *(pBest — P)* and *(gBest — P)* are computed. With both difference vectors the velocity vector is calculated. For this operation the parameter C_g is needed, called *decision factor*, and also **r**, which is a random number from a uniform distribution in the range [0,1). For each position of the difference vectors, if $r \leq C_g$, the velocity operator will take the layer from the difference (gBest — P) and from (pBest — P), otherwise. Therefore, C_g will control the convergence of the particle towards the global best. This operation is made by *UpdateVelocity()* and the mechanism of this operator is exemplified in Figure 4.3 (a).

4.1.6 Position Operator

The operator to update the position of the particles i.e. evolving their DCGAN architecture, is *UpdateParticle()*. This operator takes the particle's *velocity* to modify the pertinent layers. Layers are added or removed from the particle architecture according to its *velocity*. An example of the action of this operator is found in Figure 4.3 (b).

When the particle is updated, there must be a record of the transposed convolutional layers in the architecture, so that there is the number allowed by the parameters according to the current resolution that is evolving. A number greater than the allowed would create images of a higher resolution than needed. When there are extra layers, the same extra number of layers is chosen randomly and replaced by a randomly initialized convolutional layer that keeps the size of the previous layer output.

Similarly, only one fully connected layer will be available at the beginning of architecture. This layer will be evolved only when the resolution is 4^2 pixels, as the DCGAN architecture only uses this layer at the beginning to process the noise input. Since the fully connected layer will always be in the first position of the particles there will always be *zero-difference* in that position. Therefore, to avoid stagnation in evolution, the number of neurons in these layers will always be randomly changed in each update.

4.1.7 Fitness Function

In the present work Fréchet Inception Distance (FID), presented and detailed previously in Section 2.5, is used as fitness function and performance metric (for final results) of the different training evolved GANs.

The FID is measured using the set of real training images and a sample of the same size of synthetic images obtained by the generator of the trained GAN to be evaluated. By means of this metric the fitness of the particle is qualified, remembering that the lower the better.

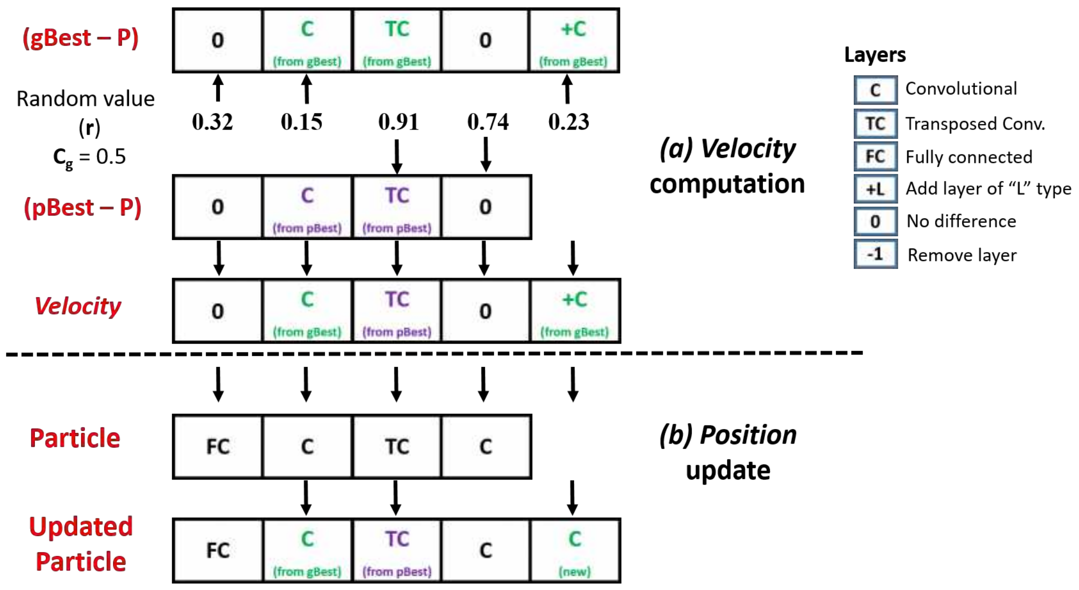


FIGURE 4.3. **Example of *velocity* and *position* operators.** (a): *Velocity* computation of a particle. (b): Particle updated using their *velocity*. Inspired by (Junior and Yen 2019).

4.1.8 Progressive Growth

The training of the various DCGANs (particles) was carried out progressively, starting with a resolution of 4^2 to 256^2 pixels in steps that doubled the resolution. The final resolution was chosen because it is one that equals or exceeds the resolution adopted by previous GANs work for CXR synthesis.

The swarm is initialized with particles that generate images with a resolution of 4^2 pixels. These particles are trained for a number of epochs by backpropagation and gradient descent with the dataset of CXR images with the same resolution. After training, the quality of the creations of these particles is evaluated by FID, which acts as a fitness function. Using the fitness values, the pBest of each particle and the gBest are selected. If an architecture is selected as pBest, the weights of the trained DCGAN are saved. The particles are updated by the pBests and gBest for a certain number of generations. At the end of the cycle of iterations, the pBests of the particles are taken as the basis for the next resolution, fixing that part of the architecture and only adding and modifying the layers ahead of this part to obtain the images of the next resolution. When the DCGANs of the particles of the new swarm are implemented, the weights for the layers belonging to the previous pBest are loaded. To avoid reducing the learning rate to smooth the training of the previous pre-trained layers with respect to the newly initialized layers, Weigth Normalization (WN) technique is used in the generator (see Section 2.4.4). WN has proven to stabilize training in GANs in addition to avoiding exploding gradients when learning rates are relatively high for certain layers. Meanwhile in the discriminator, Spectral Normalization is used to stabilize the training. The experimentation carried out to decide the training of the particles using Spectral Normalization can be seen in Appendix 1.

The previous process is repeated until obtaining the trained gBest of the final resolution. In Figure 4.4 the mechanism of progressive growth is represented. In such figure the growth of the generator is represented but the equivalent procedure is used in the discriminator. **(a)** At the beginning there are particles that generate CXR images in 4^2 pixels, from where the pBests are selected. **(b)** When the resolution is doubled, the architecture and the trained weights of particle's pBest in the previous resolution are used. pBest's layers remains fixed without evolving (but still training), while a new population is generated with particles that will be concatenated to the pbest part and which will be evolved by the PSO. **(c)** The previous process is repeated doubling the resolution until obtaining CXR images of 256^2 pixels resolution.

4.1.9 Swarm Initialization

When the swarm is initialized (*Initialize Swarm()*), the number of layers in each particle is selected at random with an uniform distribution (to avoid bias) within a defined range. This range is calculated from the ranges of allowed layers of each type in the particle. In turn, as previously mentioned, this number of allowed layers is dependent on the resolution to be obtained in the current generation. To clarify this mechanism, an example will be presented.

As the particle is a composition of the three types of layers previously seen, they have a minimum and maximum number of those that can be found in the particle in a certain part

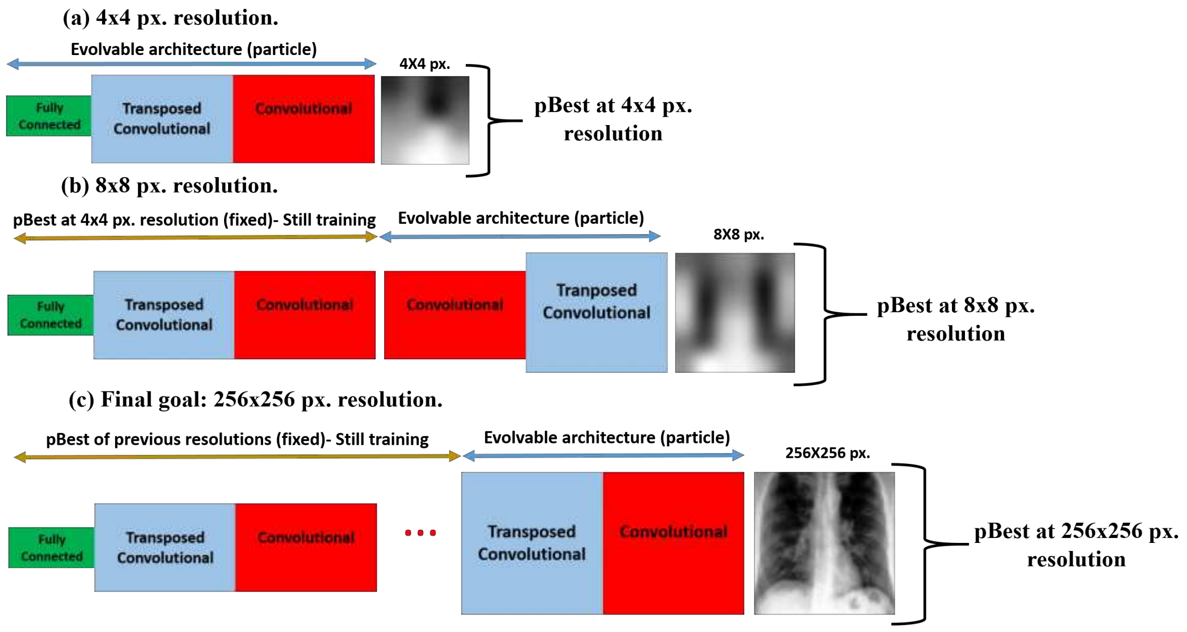

(a) 4x4 px. resolution.
Evolvable architecture (particle)
4X4 px.
Fully Connected
Transposed Convolutional
Convolutional
pBest at 4x4 px. resolution
(b) 8x8 px. resolution.
pBest at 4x4 px. resolution (fixed)- Still training
Evolvable architecture (particle)
8X8 px.
Fully Connected
Transposed Convolutional
Convolutional
Convolutional
Tranposed Convolutional
pBest at 8x8 px. resolution
(c) Final goal: 256x256 px. resolution.
pBest of previous resolutions (fixed)- Still training
Evolvable architecture (particle)
256X256 px.
Fully Connected
Transposed Convolutional
Convolutional
Transposed Convolutional
Convolutional
pBest at 256x256 px. resolution

FIGURE 4.4. **Progressive growing of DCGAN particle.**

of the evolution according to the resolution sought. Suppose that the ranges (**[minimum number, maximum number]**) for each layer are as follows:

- Fully connected: [1,1]
- Transposed convolutional: [0,1]
- Convolutional: [0,2]

With the above, the minimum and maximum size of the particles will be defined (limited) by the sum of the minimum and maximum numbers of each layer. This is calculated as follows:

- **Minimum number of layers in the particle** = minimum of fully connected + minimum of transposed convolutional + minimum of convolutional
- **Maximum number of layers in the particle** = maximum of fully connected + maximum of transposed convolutional + maximum of convolutional

These previous values are 1 and 4, respectively, for the present example. These values represent the lower and upper limits, respectively, from which a random number with uniform distribution is taken, which will represent the size of the particle.

Once with the defined size, layers of the three types are inserted randomly, with their hyper-parameters also randomly initialized, and it is also verified that they do not exceed the limits established for each type.

4.2 cDCGAN-PSO

A second version of DCGAN-PSO was developed, named **cDCGAN-PSO**.

The main difference with respect to the previous version is that in this one a Conditional DCGAN (see section 2.4.2) is used, which allows the synthesis of multiple classes of images with a single trained DCGAN, unlike the original version where they can only be synthesized images of a single class. This addition made it possible to synthesize CXR images not only of the COVID-19 pneumonia class but also of Non-COVID-19 pneumonia and Healthy classes.

All the sections related in the previous section to define the DCGAN-PSO are similar for the cDCGAN-PSO. Only one section is modified, which is realted to measure the particle fitness, a change that will be detailed below.

4.2.1 cDCGAN-PSO Fitness Function

The fitness function is also performed by calculating the FID using the images generated by the trained cDCGAN and the real set. However, since there are multiple classes, the FID of each one is calculated separately using a set of generated images belonging to that class and its respective real set. When obtaining the different FID measurements for each class, they are averaged and with such mean value the quality of the particle is evaluated.

4.2.2 Pseudocode of cDGAN-PSO

Algorithm 3 has the pseudocode of cDCGAN-PSO. As can be seen, this is the same as the first version of the algorithm (see Algorithm 2) with slight adaptations that are highlighted in red. These slight changes are related to the use of a multi-class CXR images dataset, the use of cDCGAN instead of DCGAN, and the average of the FID as fitness function.

Algorithm 3 cDCGAN-PSO

Input: Training labeled dataset of c classes of CXR images of each resolution used, *N° generations per resolution*, *N° epochs per particle training*, *Swarm size*, C_g, *resolutions list*.

Output: Trained DCGAN of CXR images of pneumonia caused by COVID-19 in 256^2 pixels resolution (gBest).

```
 1: for resolution in resolutions list do
 2:     Initialization: Swarm ← Initialize Swarm(resolution), pBests, gBest.
 3:     for N° generations per resolution do
 4:         for particle in swarm do
 5:             if resolution = $4^2$ pixels then
 6:                 Network ← cDCGAN particle architecture
 7:             else
 8:                 Network ← cDCGAN previous resolution pBest architecture
 9:                 Network weights ← Previous resolution pBest weights
10:                 Network ← Add cDCGAN particle architecture at the end of the Network
11:             end if
12:             for N° epochs per particle training do
13:                 Train Network with CXR dataset of its respective resolution
14:             end for
15:             //Measure FID for each class.
16:             particle fitness ← 0
17:             for class in CXR dataset do
18:                 particle fitness ← particle fitness + FID(Network for class)
19:             end for
20:             //Average FID.
21:             particle fitness ← particle fitness / $c$
22:             if particle fitness $\leq$ particle pBest fitness then
23:                 particle pBest ← particle
24:                 particle pBest weights ← particle weights
25:             end if
26:         end for
27:         gBest ← pBest with lowest fitness
28:         for particle in swarm do
29:             particle velocity ← UpdateVelocity(particle)
30:             particle ← UpdateParticle(particle velocity)
31:         end for
32:     end for
33: end for
34: RETURN gBest
```

4.3 Chapter Summary

In this chapter, the DCGAN-PSO and cDCGAN-PSO algorithms based on Particle Swarm Optimization were presented for the evolution of the topology of DCGANs and cDCGANs respectively, at the same time as their progressive training is carried out.

The studies that were taken as a basis, the representation of the particles used, the evolution operators, as well as the fitness function were described in detail to understand the mechanisms of the algorithms.

CHAPTER 5

Experiments and Results

This chapter will detail the obtaining of the dataset used as well as the methodology carried out to test the previously mentioned hypotheses.

The results of experimentation and their discussion are also mentioned.

5.1 Methodology

The methodology used to carry out this project is summarized in Figure 5.1. Below is the detailed information for each step.

(1) **To obtain and process datasets of CXR images**: This will be detailed in the section 5.2.

(2) **Development and implementation of DCGAN-PSO**: This is detailed in section 4.1.

(3) **To perform independent runs of the DCGAN-PSO**: Ten runs of the DCGAN-PSO were carried out using the CXR images of pneumonia caused by COVID-19.

(4) **To implement and train *state-of-the-art* models in synthesis of CXR images with GANs**: For comparison purposes, eight approaches using DCGANs reported in the literature review of CXR synthesis (see Section 3.3) were implemented and trained.

(5) **To perform experimentation using a binary classification CNN**: The binary classification of CXR images (i.e. COVID-19 and Non-COVID-19 classes) was carried out using a CNN with unbalanced and artificially balanced image sets with synthetic images.

(6) **Development and implementation of cDCGAN-PSO**: This is detailed in Section 4.2.

(7) **To perform runs of the cDCGAN-PSO**: Six runs of the cDCGAN-PSO were carried out using the CXR images of pneumonia caused by COVID-19, pneumonia Non-COVID-19, and healthy.

(8) **To perform runs of DCGAN-PSO for Pneumonia and Healthy classes to compare with cDCGAN-PSO**: Two runs of each class of the DCGAN-PSO were carried out using the CXR images of pneumonia Non-COVID-19 and healthy.

(9) **To perform experimentation using a multi-class classification CNN**: The multiclass classification of CXR images (i.e. pneumonia COVID-19, pneumonia Non-COVID-19, and healthy classes) was carried out using a CNN with unbalanced and artificially balanced image sets with synthetic images.

(10) **To perform experimentation using CViCom COVID-19**: Carry out the multi-class classification (i.e. pneumonia COVID-19, pneumonia Non-COVID-19, and healthy classes) using the UNAM CViCOM COVID-19 diagnostic system to verify the correct detection of the synthetic images of each class.

The details of activities 3-5 and 7-10 will be addressed in Section 5.4, while their results and discussion will be reflected in Section 5.5.

The schedule of activities carried out for the development of this thesis can be found in Appendix 2.

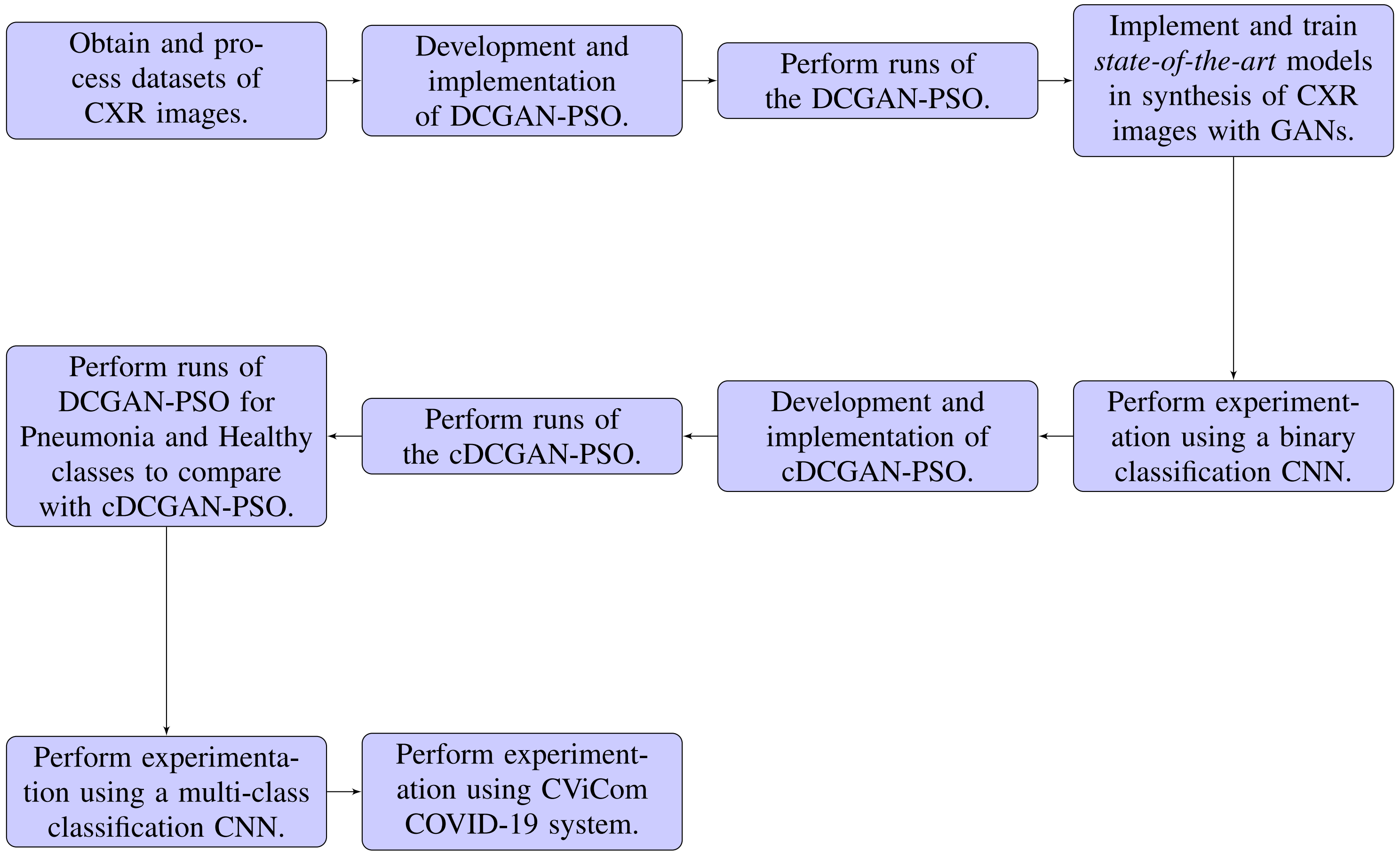

Obtain and process datasets of CXR images.
Development and implementation of DCGAN-PSO.
Perform runs of the DCGAN-PSO.
Implement and train *state-of-the-art* models in synthesis of CXR images with GANs.
Perform experimentation using a binary classification CNN.
Development and implementation of cDCGAN-PSO.
Perform runs of the cDCGAN-PSO.
Perform runs of DCGAN-PSO for Pneumonia and Healthy classes to compare with cDCGAN-PSO.
Perform experimentation using a multi-class classification CNN.
Perform experimentation using CViCom COVID-19 system.

FIGURE 5.1. **Methodology diagram.**

5.2 CXR Images Datasets

For the generation of CXR images as a case study, public image datasets available at (Chowdhury et al. 2020), (Cohen et al. 2020) and (Tabik et al. 2020) for the COVID-19 pneumonia class were used. For the pneumonia non-COVID-19 and healthy classes, the dataset from (Mooney 2018) was used. From these sets, 994 CXR images from the COVID-19 class and 918 from each of the pneumonia and healthy classes, were selected for their quality.

Subsequently, a processing was carried out which consisted of three steps:

(1) Converting images to grayscale (one channel).
(2) Resize the images to the different used resolutions (one dataset per resolution) using bicubic interpolation.
(3) Applying histogram equalization, to increase the contrast of the images.
(4) Normalize in a range of [-1,1] in order to be adequate for the training of the DCGAN or cDCGAN. In addition, and as it was mentioned before, the output of the generator is the Tanh function that provides outputs in the same range.

These steps were performed with the purpose of reducing the large variability of the image appearance, depending on the acquisition source, radiation dose as well as proprietary non-linear post-processing (Guendel et al. 2019). Examples of the images before and after processing are shown in Figure 5.2.

5.3 Technical Implementation Details

5.3.1 DCGAN Training details

The design rules for architectures based on the DCGAN original paper (Radford et al. 2015) were used:

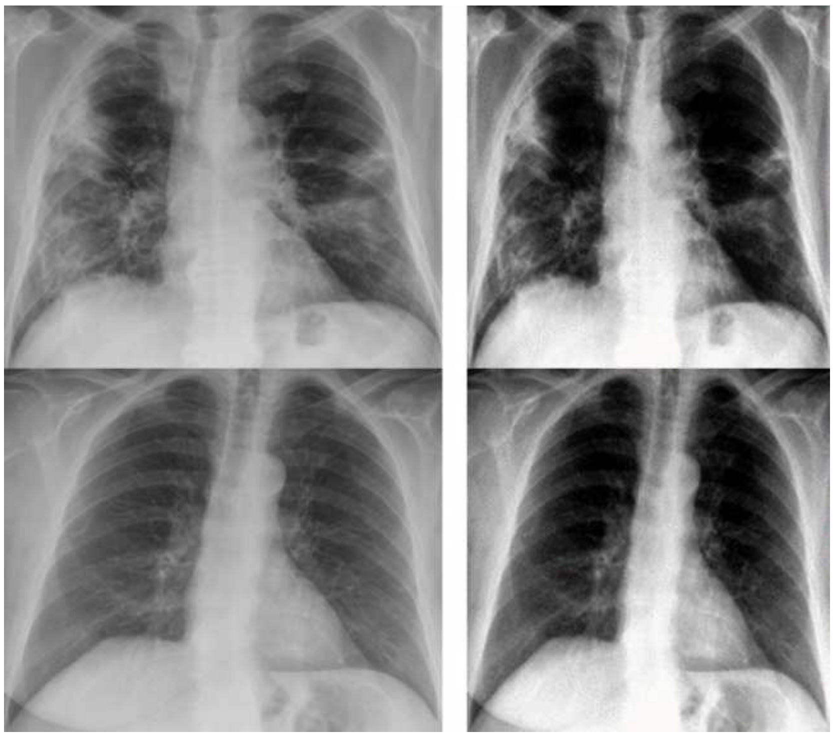


FIGURE 5.2. **Sample image processing.** The images belong to CXR of pneumonia generated by COVID-19. The left column shows original images from public image datasets while the right column shows the same images after processing.

- Batch Normalization layers were used between the layers of convolutional nature in the generator and the discriminator, except in the last one of the generator and the first of the discriminator.
- ReLU is used as activation function in generator, except in the last layer where Tanh was employed; while in discriminator, LeakyReLU was used in all layers, except in the last one, where Sigmoid was required to perform the classification.

Adam optimizer (Kingma and Ba 2014) was used in DCGAN training method for stochastic optimization. Python 3.6.9 programming language with the framework PyTorch (Paszke et al. 2017) were used for the implementation of all the NN used on the free access online platform Google Colaboratory[1] to have free access to GPUs for the NN training. These

[1]https://colab.research.google.com/

resources were limited in time and capacity and each new usage granted different computational resources, then an analysis of the run times with respect to such infrastructure was not possible.

5.3.2 Amount of Data

In the experiments, only 600 images of the created dataset were used, this with the purpose of representing the low amounts of biomedical images. However, for the FID evaluation, the total images of each dataset were used together with a batch of the same size of synthetic images.

5.3.3 Parameter Selection

The parameters used in all experiments for both versions of the neuroevolution algorithm are found in Table 5.1.

The parameters C_g and number of generations per resolution were taken from the original article of psoCNN (Junior and Yen 2019), while the swarm size was reduced (originally it is 20) to 15 to reduce costs and computational times. The rest of the parameters were selected through a brief experimentation, depending on the limited computational capacity available.

5.4 Experimentation

To verify compliance with the hypotheses developed at the beginning of the thesis (see Section 1.5) the following experiments were carried out.

Parameter	**Value**
Particle Swarm Optimization	
Swarm size	15
N° generations per resolution	10
C_g	0.5
Resolutions list	[4^2 px., 8^2 px., 16^2 px., 32^2 px., 64^2 px., 128^2 px., 256^2 px., 320^2 px.]
Ranges of particle parameters	
N° Fully connected layers	[1,1] (4^2 px.) ; [0,0] (otherwise)
N° Convolutional layers	[0,2]
N° Transposed convolutional layers	[0,0] (4^2 px.) ; [1,1] (otherwise)
Filter size	[2,5]
N° filters	4^2:64^2 px. = [1,256]; 128^2 px. = [1,64]; 256^2:320^2 px. = [1,32]
N° neurons	[1,300]
DCGAN & cDCGAN training	
CXR images classes	COVID-19 (DCGAN-PSO); COVID-19, Pneumonia, and Healthy (cDCGAN-PSO)
N° epochs per particle training	200
N° images used by class	600
Batch size	4^2:128^2 px. = 16; 256^2 px. = 14
Optimizer	Adam
Learning rate (**G** and **D**)	2×10^{-4}
β_1, β_2 (optimizer)	0.5 , 0.999
LeakyReLU slope (**D**'s activations)	0.2
Weight's initializer (**G** and **D**)	$\mathcal{N}(0, 0.02)$
Noise distribution (p_z): $\mathbb{R}^{100}$	$\mathcal{N}(0, 1)$
CNN training	
Batch size	32
N° epochs	100
N° images per class used in the validation set	300 of each real class
Optimizer	Adam
Learning rate	2×10^{-4}
β_1, β_2 (optimizer)	0.5 , 0.999
Weight's initializer	$\mathcal{N}(0, 0.02)$
Cost function	Cross-Entropy loss

TABLE 5.1. Parameter values of DCGAN-PSO, cDCGAN-PSO, and CNN used in the experiments.

5.4.1 Algorithm Executions

5.4.1 DCGAN-PSO

The execution of DCGAN-PSO, using the CXR images belonging to the COVID-19 class, was performed 10 times, being this number is due to computational limitations. Each run lasted approximately one week and with each change of resolution the time was doubled with respect to the previous resolution.

In addition, two runs of each class of the DCGAN-PSO were carried out using the CXR images of pneumonia and healthy classes. The results of these runs were only used for FID comparison with the cDCGAN-PSO, the second version of the neuroevolution algorithm.

5.4.2 cDCGAN-PSO

The execution of cDCGAN-PSO, using the CXR images belonging to the COVID-19, pneumonia and healthy classes, was performed six times, being this lower number with respect to the previous experiment due to computational limitations. Each run lasted approximately one week and half and with each change of resolution the time was doubled with respect to the previous resolution. Two runs were scaled up to a resolution of 320^2 pixels resolution because this is the base resolution to experiment in the CViCom COVID-19 system. The parameters used for the rest of the runs were respected and those necessary to scale up to 320^2 pixels are found in Table 5.1.

5.4.2 Fitness Evolution

The fitness values of the $gBest$ were monitored during the different generations and evolution stages of the runs.

In addition, the FID of the $gBest$ in cDCGAN-PSO runs was monitored for each of the CXR images classes used in the runs.

5.4.3 Qualitative Evaluation

A sample of the COVID-19 class images synthesized by the DCGAN generator evolved by DCGAN-PSO is visually compared with a sample from the real set.

Samples of images of each class synthesized by the cDCGAN generator evolved by cDCGAN-PSO are visually compared with samples from the real sets.

The synthetic images were obtained from the generator with the FID value closest to the median of the FID evaluation of each algorithm version.

5.4.4 FID Evaluation

The FID measurement was performed with the complete sets of real images and a batch of the same size synthesized by the generator of the evolved GANs obtained at the end of the runs by the two algorithm versions.

5.4.5 COVID-19 CXR Synthesis Comparison

Eight of the *state-of-the-art* CXR synthesis DCGANs seen in the literature review (see section 3.3) were implemented for comparison purposes. These models were trained with the same parameters established in their source articles and using the same set of CXR images for COVID-19. Each GAN was trained five independent times.

The reason for only using the models for the synthesis of the COVID-19 class was due to the limited computational time available which was used to carry out the other types of experimentation.

In addition, the random creation of DCGANs was carried out (with the same parameters for network depth and layers that were used in both versions of the algorithm) for images wit resolution of 256^2 pixels.

The FID evaluation of the synthetic images obtained by the *state-of-the-art* models and the random DCGANs was carried out. These values were compared with those obtained by the FID evaluation of the GANs evolved by both versions of the algorithm for the COVID-19 class.

The 95%-confidence Wilcoxon rank-sum test was used to verify that the results come from the same distributions (null hypothesis). If the significance values of this test (p-values) are less than 0.05, the alternative hypothesis is accepted i.e. the samples come from different distributions.

5.4.6 Classification Performance

A CNN was implemented to perform binary and multiclass classification for images obtained using the DCGAN-PSO and cDCGAN-PSO, respectively.

The CNN was designed based on VGG16 models (Simonyan and Zisserman 2014), which tend to perform well for image classification. Furthermore, CNNs designed based on this structure were used in previous CXR images classification work (Waheed et al. 2020, Panwar et al. 2020, Rajaraman et al. 2018, Civit-Masot et al. 2020). Through a brief experimentation, the topology of the network was established, this is shown in figure 5.3. The same network was used for binary and multiclass classification concatenating at the end different layers according to the scenario, which are the following:

- **Binary**: One fully connected layer with one neuron and the sigmoid activation function to obtain a binomial probability.
- **Multiclass**: One fully connected layer with three neurons and the SoftMax activation function to obtain a multinomial probability.

The binary classification using the synthetic CXR images obtained by the DCGAN-PSO was performed for the COVID-19 and Non-COVID-19 classes (composed of pneumonia and healthy classes). Five independent CNN trainings were conducted for each of the following scenarios:

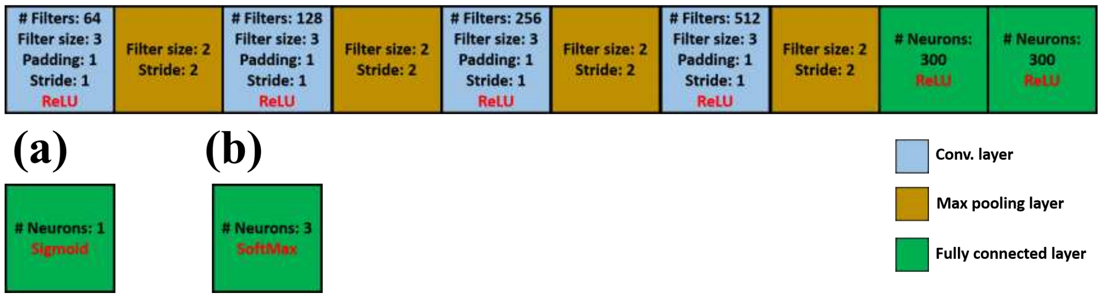


FIGURE 5.3. **CNN architecture for classification.** (a) Final layer used for binary classification. (b) Final layer used for multiclass classification. The red text on the layers indicates the activation function used.

(1) **Unbalanced dataset**:
- 600 COVID-19 real images.
- 1200 Non-COVID-19 real images (600 from pneumonia and 600 from healthy).

(2) **Balanced dataset**:
- 600 COVID-19 real images + 600 COVID-19 synthetic images.
- 1200 Non-COVID-19 real images (600 from pneumonia and 600 from healthy).

The multiclass classification using the synthetic CXR images obtained by the cDCGAN-PSO was performed for the COVID-19, pneumonia and healthy classes. Five independent CNN trainings were conducted for each of the following scenarios:

(1) **Unbalanced COVID-19 dataset**:
- 300 COVID-19 real images.
- 600 pneumonia real images.
- 600 healthy real images.

(2) **Balanced COVID-19 dataset**:
- 300 COVID-19 real images + 300 COVID-19 synthetic images.
- 600 pneumonia real images.
- 600 healthy real images.

(3) **Unbalanced pneumonia dataset**:
- 600 COVID-19 real images.
- 300 pneumonia real images.

- 600 healthy real images.

(4) **Balanced COVID-19 dataset**:

- 600 COVID-19 real images.
- 300 pneumonia real images + 300 pneumonia synthetic images.
- 600 healthy real images.

(5) **Unbalanced healthy dataset**:

- 600 COVID-19 real images.
- 600 pneumonia real images.
- 300 healthy real images.

(6) **Balanced healthy dataset**:

- 600 COVID-19 real images.
- 600 pneumonia real images.
- 300 healthy real images + 300 healthy synthetic images.

(7) **Balanced dataset**:

- 600 COVID-19 real images.
- 600 pneumonia real images.
- 600 healthy real images.

(8) **Extended balanced dataset**:

- 600 COVID-19 real images + 600 COVID-19 synthetic images.
- 600 pneumonia real images + 600 pneumonia synthetic images.
- 600 healthy real images + 600 healthy synthetic images.

The synthetic images were obtained from the generator with the FID value closest to the median of the FID evaluation results of each version of the neuroevolution algorithm.

The classification performance was measured in terms of accuracy which is the fraction of predictions that the model made correctly. This is calculated using the following formula:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{5.1}$$

The parameters used in the CNN training are found in their corresponding section in Table 5.1.

The 95%-confidence Wilcoxon rank-sum test was used to verify that the results come from the same of different distributions.

It is worth mentioning that the results obtained by this classification experimentation do not seek to obtain *state-of-the-art* results for the classification of CXR images, since this is beyond the scope of this work. Such experiment only has the purpose of verifying the improvement in performance obtained through the use of synthetic images of evolved GANs. Later uses of the algorithms could have this application as an objective to improve.

5.4.7 CViCom COVID-19 Classification Performance

The CViCom-COVID-19 system[2], which was designed to detect COVID-19, pneumonia and healthy cases from CXR images, was used to classify the synthetic images obtained from both versions of the algorithm. The synthetic images were obtained from the generators with the FID value closest to the median of the FID evaluation results of each class and version.

Images with 256^2 pixels of resolution were used for the DCGAN-PSO, since this was the only resolution handled in their experimentation batch. While for cDCGAN-PSO the images from one of the two runs with a resolution of 320^2 pixels were used. In addition, real images of each class in 320^2 pixel resolution were used as control and comparison.

The batch size for each set of images was 30.

The 95%-confidence Wilcoxon rank-sum test was used to verify that the results come from the same of different distributions.

[2]https://www.imagensalud.unam.mx/

5.4.8 Two-Dimensional Visualizations

Two-dimensional visual comparison of the real sets and a batch of the same size of images synthesized by the evolved GANs of both versions of the neuroevolution algorithm was performed only for the purpose of visualizing and condensing in a simpler way the results obtained. The feature vectors of both batches of images were extracted using the pre-trained Inception-v3 network (the same one used for the FID calculation) thus obtaining vectors with a length of 2048 numerical values. This with the purpose of comparing the high-level characteristics extracted from both types of images. The synthetic images were obtained from the generators with the FID value closest to the median of the FID evaluation. The methods used for dimension reduction are as follows:

- **t-SNE**: Like the strategy shown in (Costa et al. 2020a) and (Sheykhivand et al. 2021), the t-Distributed Stochastic Neighbor Embedding (t-SNE) (Maaten and Hinton 2008) method was used, which is an unsupervised nonlinear technique used primarily for data exploration and high-dimensional data visualization. This method models each high-dimensional object by a two-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability.
- **PCA**: Principal Component Analysis is a method which reduces the dimension of feature space such that new variables are independent. PCA retains large pair distances in order to optimize variance (Wold et al. 1987). The two first principal components were used, obtained from the feature vector batches. The values of these principal components were normalized in the range [0,1]. This visualization technique has previously been used in the synthesis of CXR images with GANs for the comparison of the results with the real sets (Middel et al. 2019, Waheed et al. 2020).

5.5 Results and Discussion

5.5.1 Fitness Evolution

The FID value of the $gBest$ through the evolution of the resolutions of the images of the ten runs with the COVID-19 class and the two runs of each class pneumonia and healthy using the DCGAN-PSO are shown in Fig. 5.4.

The evolution of $gBest$ FID of the six runs of cDCGAN-PSO for all three classes of CXR images is shown in Fig. 5.5. Moreover, the individual $gBest$ FID of each class through the runs is shown in Fig. 5.6.

Additionally, the runs with the result closest to the median of the FID values obtained at the end of the executions for the COVID-19 class with DCGAN-PSO and the three classes with cDCGAN-PSO are shown in Fig. 5.7 as convergence plots.

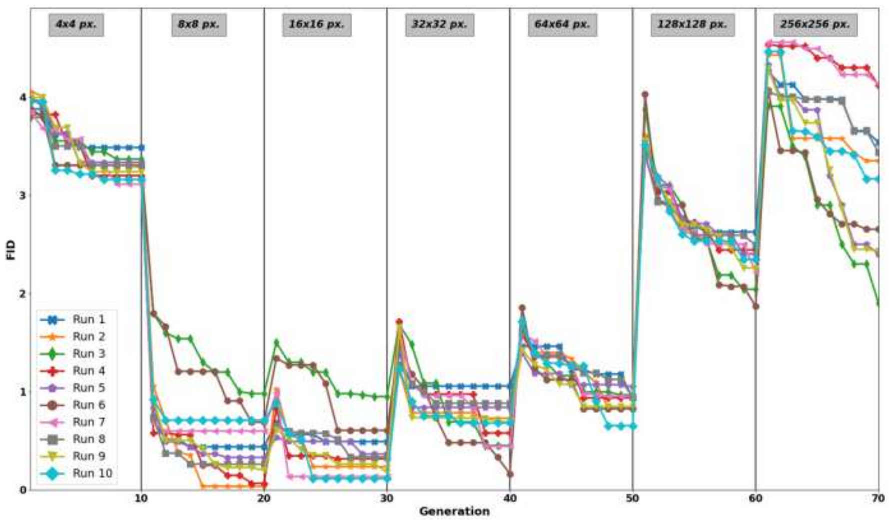
The best DCGAN and cDCGAN architectures found and trained by the respective versions of the algorithm, as well as those obtained in the executions closest to the median of the FID evaluation are reflected in Fig. 5.8.

As expected from PSO, a rapid convergence is achieved in both version of the neuroevolution algorithm. This can be confirmed by observing that the performance does not improve substantially after approximately the fifth generation of each resolution. This behavior is more clearly noted in the runs belonging to the median. As theorized in the development of the algorithm, this characteristic, despite having a high probability of being trapped in a local best, allows a reduction in search times, something of great importance due to the extensive need for resources to train and evaluate the evolved architectures.

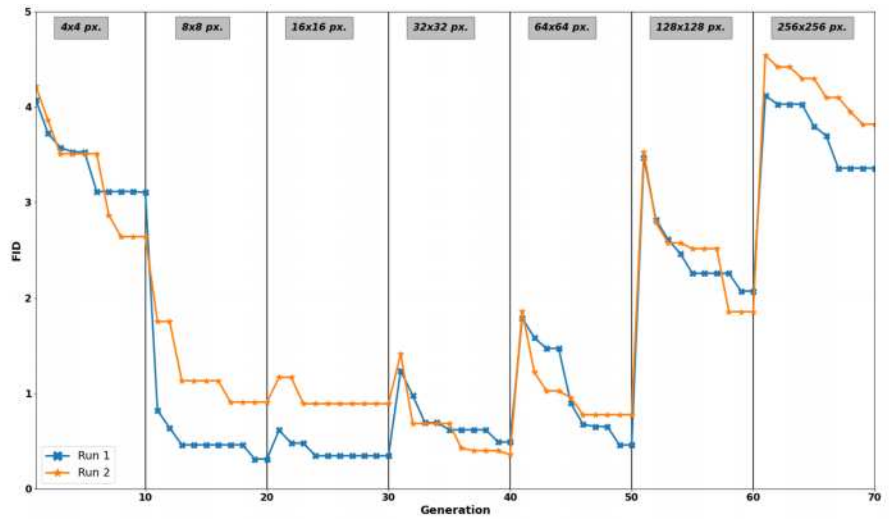
The upward behavior in each change of resolution shown by FID has an origin. FID is not a deterministic measure because it uses a CNN to extract high-level features of the images and later make a statistical comparison between them. These high-level features may vary depending on multiple factors, such as the resolution of the image, greater detail which allows it to have finer motifs and increases the complexity to be learned by the GANs. Therefore, the FID values will be slightly higher since the direct similarity with the real set will be increasingly difficult. However, regardless of the resolution, the lower the FID value is, the more similar the images will be. In such way, the achievement of reducing FID value in each part of the run relative to a single resolution proves the existence of a successful search for architectures and good training.

The up-down behavior shown in the $gBest$ FID for each class in cDGAN-PSO is because the fitness function is the average FID for the three classes used, then in some cases not all classes can improve. In summary, sometimes the fitness of one or some classes will worsen while the rest will improve. However, this is part of a trade-off among the three classes to improve the overall quality of the creations.

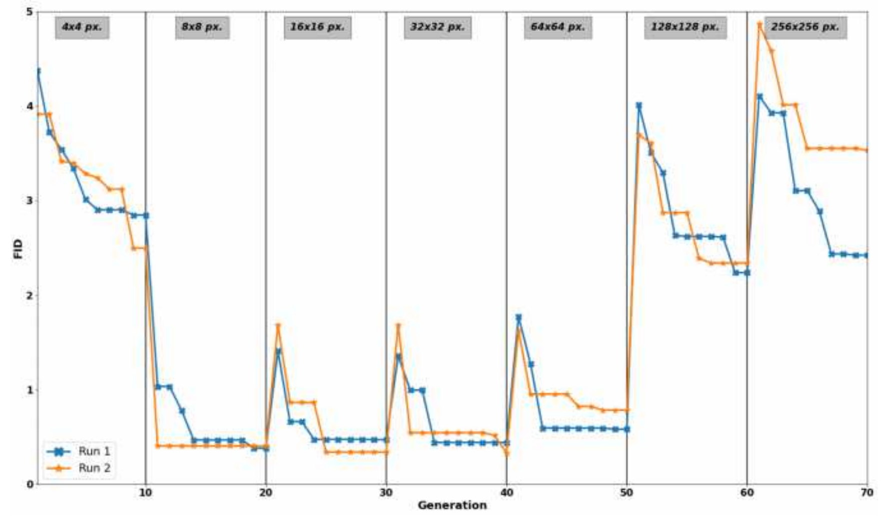
One detail to mention is that the architectures evolved by the different versions of the algorithm have a similar pattern, first use the transposed convolutional layer and then convolutional layers. This strategy could mean using the first type of layer to scale the resolution of the previous input (resolution) and later with the convolutional layers to polish the details. It can also be observed that the architectures referring to cDCGAN-PSO are more extensive than those of DCGAN-PSO. This behavior may be due to the fact that when handling a greater number of classes, more complex GANs are needed to learn to perform the differentiating details between the CXR images.



(a) COVID-19 class (10 runs).



(b) Healthy class (2 runs).



(c) Pneumonia class (2 runs).

FIGURE 5.4. **DCGAN-PSO *gBest* FID evolution.**

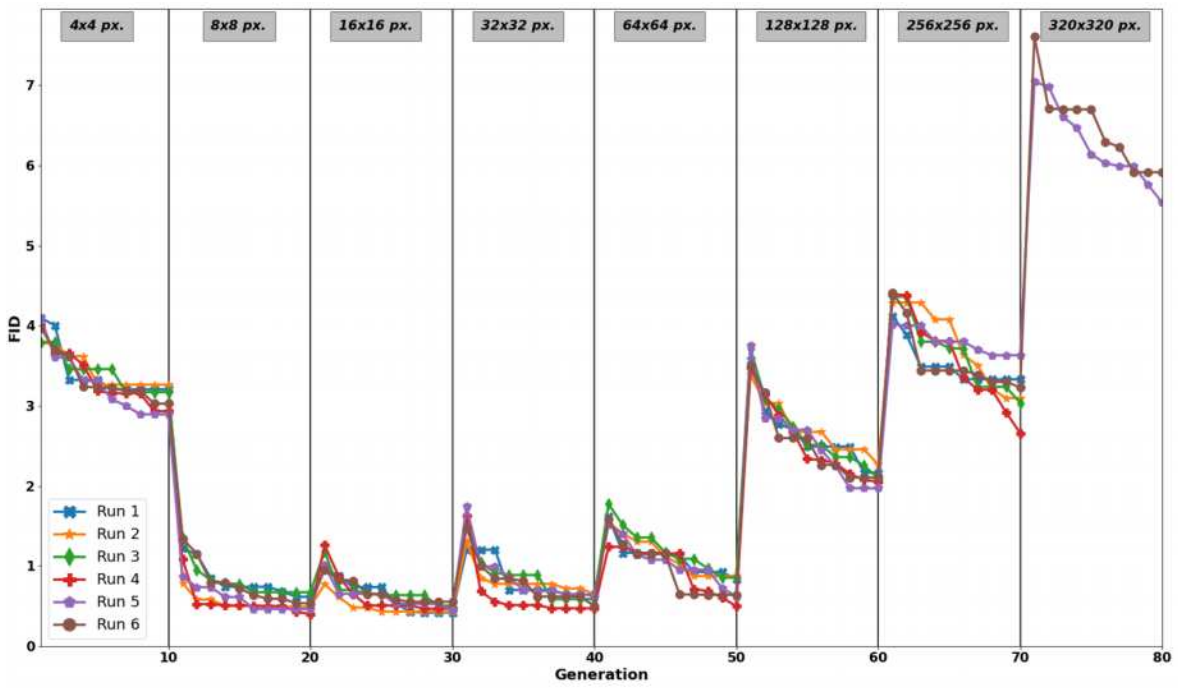
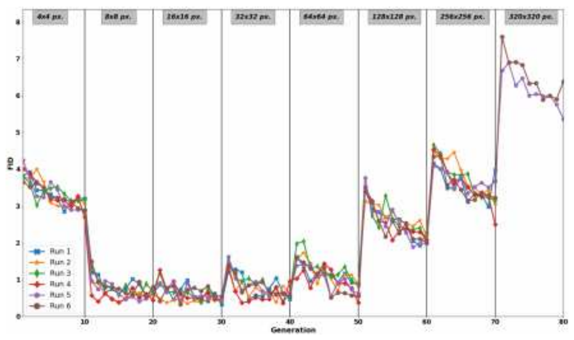
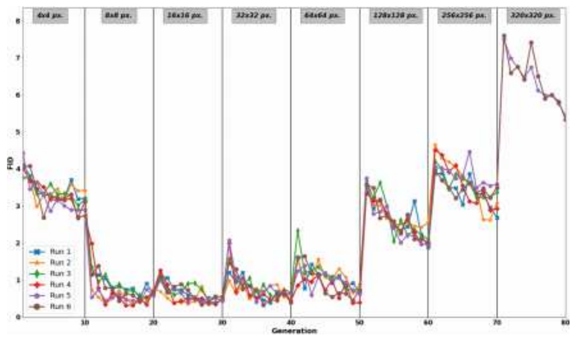


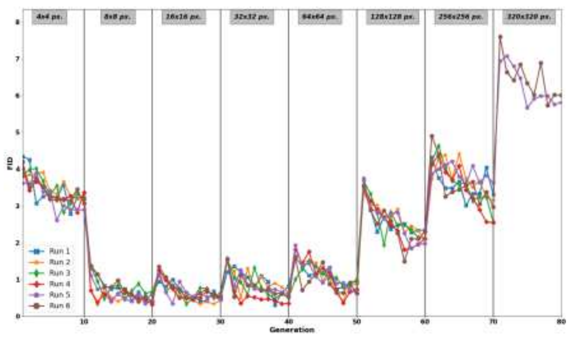
FIGURE 5.5. **cDCGAN-PSO *gBest* FID evolution.** COVID-19, healthy, and pneumonia classes (6 runs).



(a) COVID-19 class.

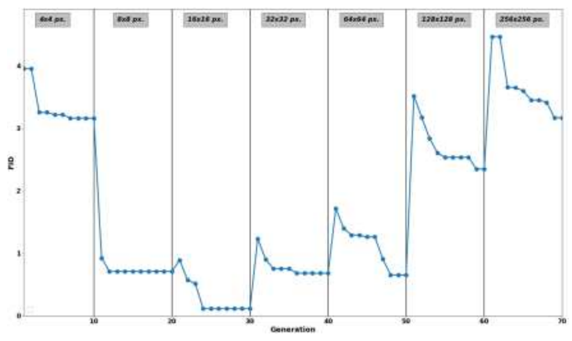


(b) Healthy class.

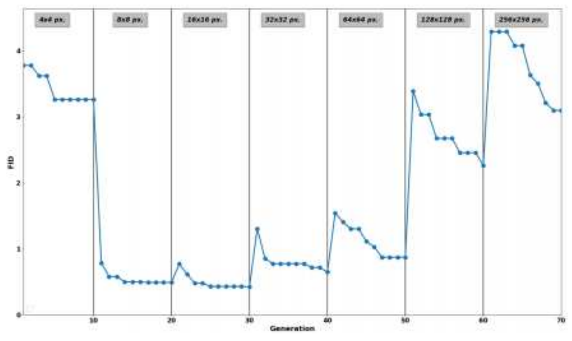


(c) Pneumonia class.

FIGURE 5.6. **cDCGAN-PSO *gBest* FID evolution of each class (6 runs).**



(a) DCGAN-PSO.



(b) cDCGAN-PSO.

FIGURE 5.7. **Convergence plots over 70 generations of the median runs of DCGAN-PSO and cDCGAN-PSO.**

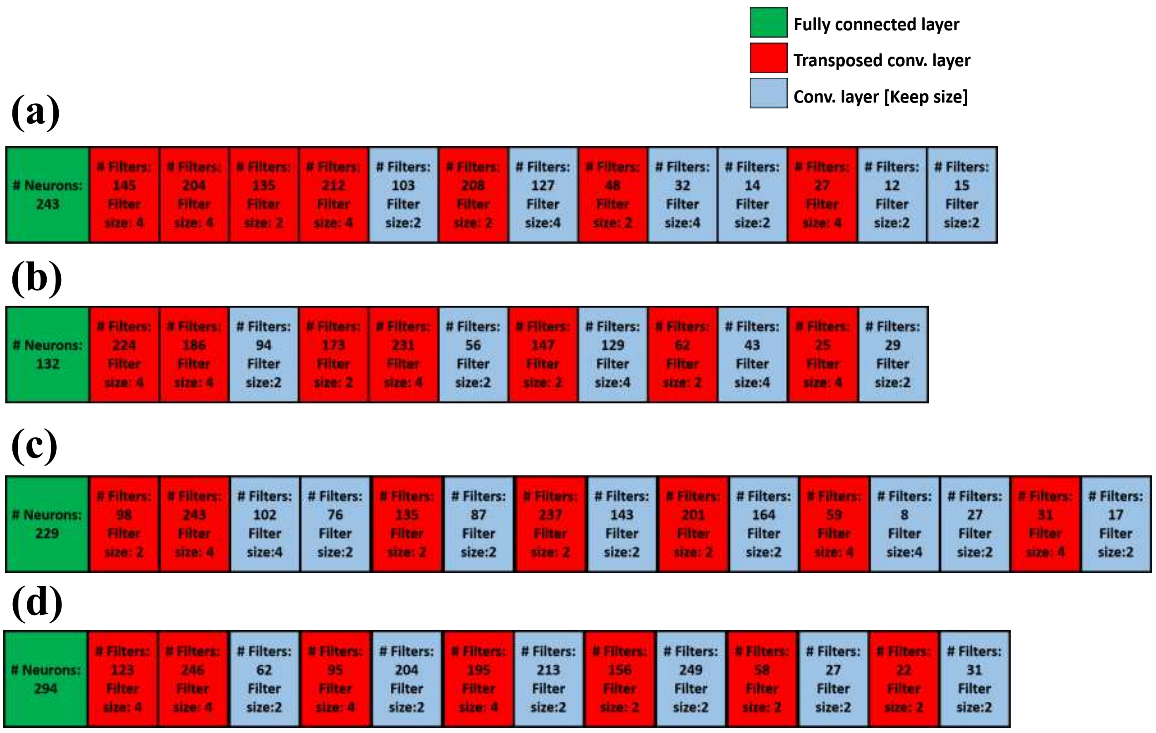

Fully connected layer
Transposed conv. layer
Conv. layer [Keep size]
(a)
# Neurons: 243
# Filters: 145 Filter size: 4
# Filters: 204 Filter size: 4
# Filters: 135 Filter size: 2
# Filters: 212 Filter size: 4
# Filters: 103 Filter size:2
# Filters: 208 Filter size: 2
# Filters: 127 Filter size:4
# Filters: 48 Filter size: 2
# Filters: 32 Filter size:4
# Filters: 14 Filter size:2
# Filters: 27 Filter size: 4
# Filters: 12 Filter size:2
# Filters: 15 Filter size:2
(b)
# Neurons: 132
# Filters: 224 Filter size: 4
# Filters: 186 Filter size: 4
# Filters: 94 Filter size:2
# Filters: 173 Filter size: 2
# Filters: 231 Filter size: 4
# Filters: 56 Filter size:2
# Filters: 147 Filter size: 2
# Filters: 129 Filter size:4
# Filters: 62 Filter size: 2
# Filters: 43 Filter size:4
# Filters: 25 Filter size: 4
# Filters: 29 Filter size:2
(c)
# Neurons: 229
# Filters: 98 Filter size: 2
# Filters: 243 Filter size: 4
# Filters: 102 Filter size:4
# Filters: 76 Filter size:2
# Filters: 135 Filter size: 2
# Filters: 87 Filter size:2
# Filters: 237 Filter size: 2
# Filters: 143 Filter size:2
# Filters: 201 Filter size: 2
# Filters: 164 Filter size:2
# Filters: 59 Filter size: 4
# Filters: 8 Filter size:4
# Filters: 27 Filter size:2
# Filters: 31 Filter size: 4
# Filters: 17 Filter size:2
(d)
# Neurons: 294
# Filters: 123 Filter size: 4
# Filters: 246 Filter size: 4
# Filters: 62 Filter size:2
# Filters: 95 Filter size: 4
# Filters: 204 Filter size:2
# Filters: 195 Filter size: 4
# Filters: 213 Filter size:2
# Filters: 156 Filter size: 2
# Filters: 249 Filter size:2
# Filters: 58 Filter size: 2
# Filters: 27 Filter size:2
# Filters: 22 Filter size: 2
# Filters: 31 Filter size:2

FIGURE 5.8. **Best found GAN architectures.** (a) The run with the best FID evluation for DCGAN-PSO. (b) The run with the closest FID to the FID evaluation median for DCGAN-PSO. (c) The run with the best FID evluation for cDCGAN-PSO. (d) The run with the closest FID to the FID evaluation median for cDCGAN-PSO.

5.5.2 Qualitative Evaluation

Figure 5.9 shows samples of the real image dataset of COVID-19 class and examples of those obtained from the evolved architecture with DCGAN-PSO.

Samples of the real images of the three CXR classes and synthesized from the evolved architecture with cDCGAN-PSO are shown in Fig. 5.10.

The synthetic images of the COVID-19 class for the DCGAN-PSO, qualitatively speaking, display a similar visual quality (morphologically) with respect to the images belonging to the real set, with only small errors (blank areas). The sampled images captures the global structural elements such as the lungs and the ribs. A diversity such as that seen in the original set can also be observed, giving indications of no mode collapse in the generators what can be derived from stable training. Similar results were obtained from the remaining runs for the COVID-19 class and those carried out for the pneumonia and healthy classes.

Regarding the images obtained by the cDCGAN-PSO, the visual quality of the results is slightly better than those obtained by DCGAN-PSO, noting an increase in the quality of the fine details of the CXR images, e.g. more defined ribs. The reason for this is because the size of the training set was tripled by adding two new classes of CXR images. This allowed the evolved cDCGANs to use this extra amount of data to learn more finely the morphological details common to the three types of images. Similar results were obtained from the remaining runs.

Collecting the expert opinion of radiologists about synthetic images would be useful to highlight the quality of the results obtained or, failing that, to detect the main errors that may be useful to improve the collection and processing of image sets. A higher resolution would be recommended, since at low resolutions it is difficult for experts to detect fine details that sift through the different types of CXR images, as is the case of diagnosing COVID-19 (Ng et al. 2020, Huang et al. 2020). This is because the images commonly used tend to have more than 1000^2 pixels (Suetens 2017) and this has been previously mentioned in works of CXR image synthesis with GANs that resort to obtaining the opinion of these medical experts (Salehinejad et al. 2018, Menon et al. 2020).

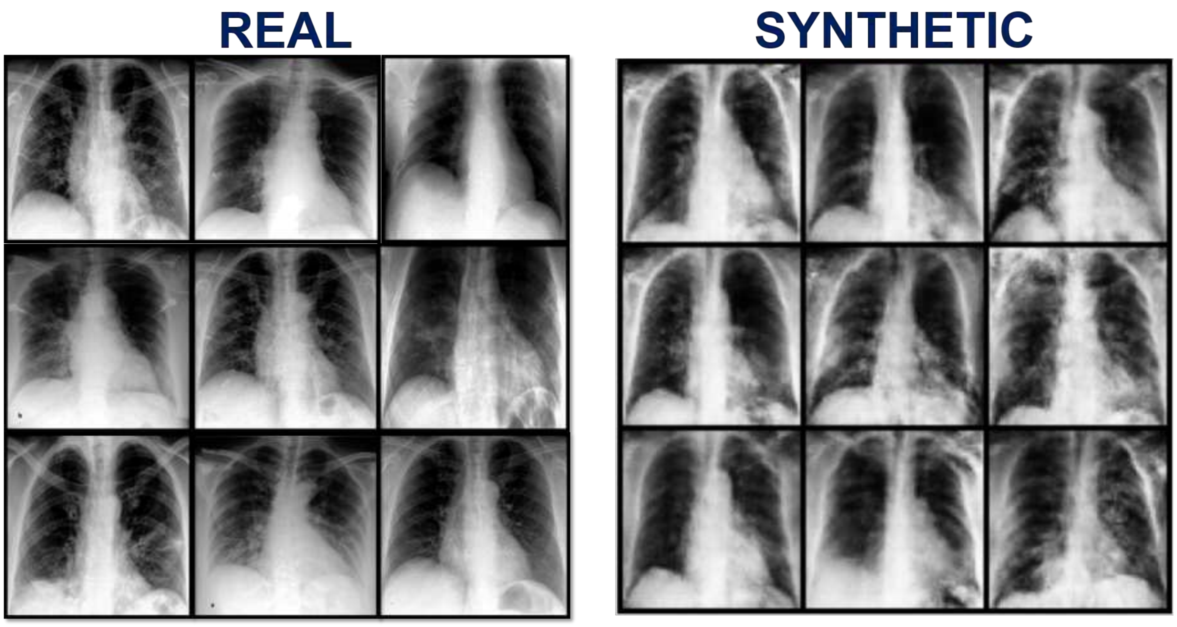


FIGURE 5.9. **Sample of synthesized COVID-19 CXR images from DCGAN-PSO in 256x256 pixels.**

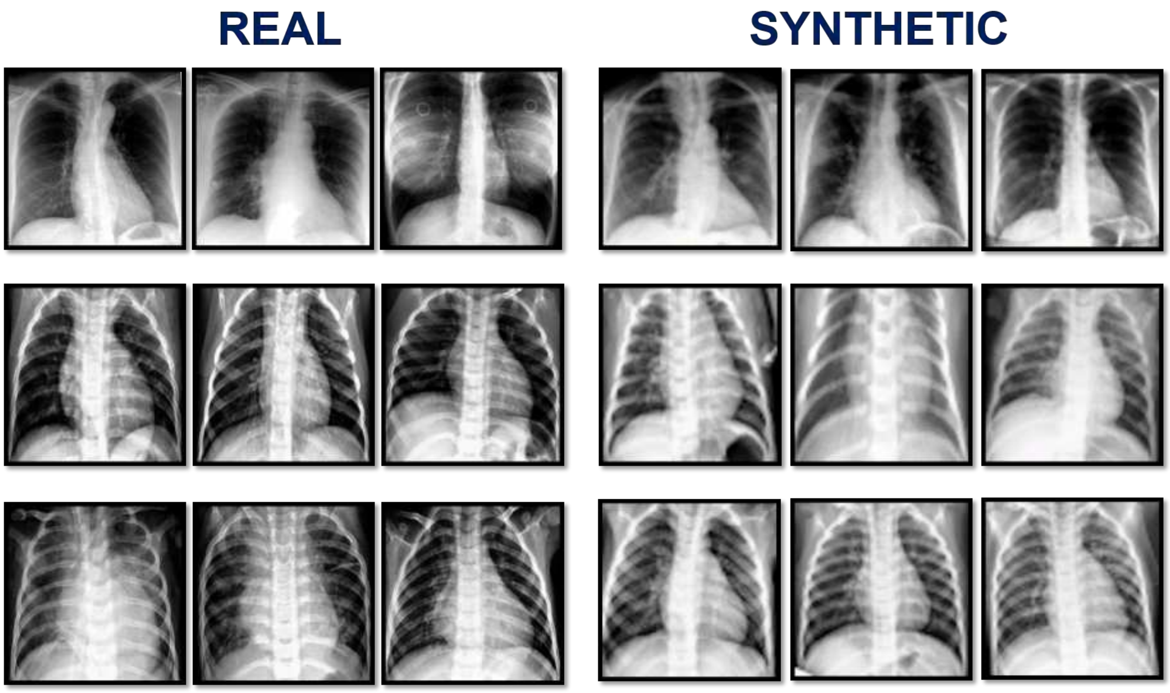


FIGURE 5.10. **Sample of synthesized CXR images from cDCGAN-PSO in 256x256 pixels.** Rows from top to bottom: COVID-19, pneumonia, and healthy classes.

5.5.3 FID Evaluation Comparison

Using the FID values of each CXR image class of each algorithm version, a performance comparison was made between them.

Table 5.2 contains the class of CXR images synthesized to be compared, the averages of the FID evaluation for each version of the algorithm in the class of its corresponding row, the p-value of the comparison between values obtained by both versions and the result of the Wilcoxon rank-sum test between both evaluations.

As can be seen in the aforementioned table, the average results of the final FID quality obtained by each class is slightly better in the cDCGAN-PSO. The reason for this may be because, as discussed in the previous section, the greater amount of training data allowed for better results than simply using a single class of CXR images as performed in the DCGAN-PSO.

Despite the previous observation, the results obtained by the Wilcoxon rank-sum test suggest that the performances of both algorithms are equal. In this case, the advantage is obtained by the second version, the cDCGAN-PSO, due to the fact that by handling multiple classes and not having a significant extension in the waiting times (half a week more). This would be a more viable option that using the first version individually for each class, which implies a substantial saving in the necessary computation time.

However, these results could be not very robust due to the size of the samples. Therefore, as future work a more exhaustive comparison between both versions could be carried out with a greater number of executions.

Class	DCGAN-PSO Average FID	cDCGAN-PSO Average FID	p-value	Wilcoxon rank-sum test
COVID-19	3.052 ± 0.773	2.988 ± 0.631	0.8282	(=)
Pneumonia	3.2506 ± 0.666	3.111 ± 0.709	0.7388	(=)
Healthy	3.5916 ± 0.3242	3.345 ± 0.991	1	(=)

TABLE 5.2. Results and comparison of FID evaluation values of both version of the algorithm. (=) means that the two sets of data compared have the same performance based on the Wilcoxon rank-sum test.

5.5.4 COVID-19 CXR Synthesis Comparison

Table 5.3 presents the FID evaluation of the synthetic images obtained from DCGAN-PSO and cDCGAN-PSO generators and compares them with the *state-of-the-art* GANs in CXR synthesis, as well as with the random creation of DCGANs. This table contains the implemented GAN model as well as its average FID in the first two columns. Columns 3 and 4 show the p-value obtained from comparing the results of each model with the DCGAN-PSO and the result of the Wilcoxon rank-sum referring to comparing the DCGAN-PSO with the approach in the corresponding row. Finally, columns 5 and 6 contain the same type of information as the previous two columns but for the cDCGAN-PSO.

As can be seen in the aforementioned table, the results obtained through the evolution of GAN architectures with progressive training improved the results of previous works in the synthesis of CXR images using handcrafted GANs, showing that the networks obtained by the proposed algorithm have a better average (lower) in the FID evaluation values than the rest of the works, supported by the Wilcoxon rank-sum test.

The behavior of the runs shown above indicating the progressive improvement of the evolved networks with respect to their fitness function together with the best results obtained respect to the random DCGANs, allow to establish that a random search is not carried out but rather an intelligent search is performed allowing the obtaining of good trained GAN networks using both versions of the algorithm.

The promising results are an indication of the potential for improvement that both versions of the neuroevolution algorithm can offer to obtain better trained GANs and then with better quality in their results.

Model	**Average FID**	**p-value (DCGAN-PSO)**	**Wilcoxon rank-sum test (DCGAN-PSO)**	**p-value (cDCGAN-PSO)**	**Wilcoxon rank-sum test (cDCGAN-PSO)**
Madani et al. 2018	5.372 ± 1.030	0.0021	(+)	0.0061	(+)
Salehinejad et al. 2018	5.308 ± 0.842	0.0048	(+)	0.0061	(+)
Khalifa et al. 2020	4.254 ± 0.165	0.00705	(+)	0.0061	(+)
Loey et al. 2020	5.846 ± 0.855	0.00219	(+)	0.0061	(+)
Shams et al. 2020	5.938 ± 0.601	0.00219	(+)	0.0061	(+)
Waheed et al. 2020	4.296 ± 0.533	0.01	(+)	0.0061	(+)
Zulkifley et al. 2020	4.492 ± 0.633	0.0101	(+)	0.0061	(+)
Kora Venu and Ravula 2021	5.576 ± 0.482	0.00219	(+)	0.0061	(+)
Random DCGANs	9.851 ± 2.028	$3.178e^{-5}$	(+)	$4.5e^{-4}$	(+)
DCGAN-PSO	3.052 ± 0.773	——-	——-	0.8282	(=)
cDCGAN-PSO	2.988 ± 0.631	0.8282	(=)	——-	——-

TABLE 5.3. Results and comparison of FID evaluation values. (=) means that the algorithm version in the column has the same performance that the compared approach in the corresponding row. (+) means that the algorithm version in the column outperformed the compared approach in the corresponding row. In red the best values are remarked.

5.5.5 Classification Performance

Table 5.4 shows the results obtained from the experimentation in binary classification using the GAN evolved by the DCGAN-PSO for the COVID-19 class. The results of the experimentation in multiclass classification using the GAN evolved by cDCGAN-PSO for the three classes of CXR images are found in Table 5.5. Both tables include the dataset used, the number of images by each class, the average accuracy obtained, the p-value obtained from the comparison between the unbalanced and balanced versions of the datasets and the result of the Wilcoxon rank-sum test between both versions of the datasets.

As can be seen, the binary and multiclass classifications benefited from the artificial increase of the datasets through synthetic images by both versions of the algorithm. This is supported by the Wilcoxon rank-sum test. These results remark the value of the synthetic images of the evolved GAN with progressive growth through both versions of the algorithm used for an external application such as classification, one of the main uses of biomedical images. The usage of the proposed algorithms in other types of biomedical images, or even in applications outside this area, migth have a high potential to support other types of Deep Learning-based tasks for which the amount of data is highly relevant and having an increase can be very supportive.

Dataset	**Classes**	**N° images**		**Average acurracy**	**p-value**	**Wilcoxon rank-sum test**
		Real	**Synthetic**			
Unbalanced	COVID-19	600	—	72.3 ± 4.3	0.0282	(-)
	Non-COVID-19	1200	—			
Balanced	COVID-19	600	600	80.4 ± 3.12		(+)
	Non-COVID-19	1200	—			

TABLE 5.4. Results of binary classification with synthesized images from DCGAN-PSO. In red the best value obtained. (-) means that the dataset used is worse than the other dataset compared, (+) means the opposite.

Dataset	Classes	N° images		Average acurracy	p-value	Wilcoxon rank-sum test
		Real	Synthetic			
Unbalanced COVID-19	COVID-19	300	—	75.43 ± 3.78	0.0472	(-)
	Pneumonia	600	—			
	Healthy	600	—			
Balanced COVID-19	COVID-19	300	300	81.31 ± 3.04		(+)
	Pneumonia	600	—			
	Healthy	600	—			
Unbalanced Pneumonia	COVID-19	600	—	72.78 ± 2.87	0.0282	(-)
	Pneumonia	300	—			
	Healthy	600	—			
Balanced Pneumonia	COVID-19	600	—	78.91 ± 3.4		(+)
	Pneumonia	300	300			
	Healthy	600	—			
Unbalanced Healthy	COVID-19	600	—	74.37 ± 1.97	0.009	(-)
	Pneumonia	600	—			
	Healthy	300	—			
Balanced Healthy	COVID-19	600	—	82.96 ± 2.39		(+)
	Pneumonia	600	—			
	Healthy	300	300			
Balanced	COVID-19	600	—	84.871 ± 3.21	0.0282	(-)
	Pneumonia	600	—			
	Healthy	600	—			
Extended balanced	COVID-19	600	600	88.361 ± 2.34		(+)
	Pneumonia	600	600			
	Healthy	600	600			

TABLE 5.5. Multiclass classification results with synthesized images from cDCGAN-PSO. In red the best values obtained. (-) means that the dataset used is worse than the other dataset compared, (+) means the opposite.

5.5.6 CViCom COVID-19 Classification Performance

The results obtained from the classification of CXR images using the CViCom COVID-19 system are condensed in Table 5.6. That table compares the classification results obtained by the sets of real images of each class and those obtained for the same class of synthetic images of both algorithms. The average predictions, the p-values of the comparison between the real set and the respective synthetic set and the result of the Wilcoxon rank-sum test between both sets are shown. Furthermore, the results obtained for the same classes with both algorithms are compared.

As can be seen, the results obtained between the real and synthetic sets obtained by cDCGAN-PSO are the same for the COVID-19 and pneumonia classes, and this is validated by the Wilcoxon rank-sum test. These results suggest that the quality of the synthetic images is close enough to the real ones that even the system gives quite similar prediction values.

In contrast, the synthetic results obtained by the DCGAN-PSO are different and lower than those obtained with the real images, this also happens with the healthy class of the cDCGAN-PSO. The reason for this with respect to DCGAN-PSO may be that the resolution of the images is less than the ideal minimum required by the system to carry out a correct prediction. Despite this, the average results do not show an extensive gap between those obtained by the real sets or by cDCGAN-PSO, so the quality of the images is not so different.

Regarding the synthetic images of the healthy class in cDCGAN-PSO, and also applies to those of DCGAN-PSO, the lower prediction certainty may be due to the fact that, as previously mentioned, the synthetic images have small white areas which denotes small defects in their synthesis, so that these artifacts could hinder the prediction, perhaps confusing themselves with the opacities that usually characterize pneumonia seen through CXR images.

Regarding the comparisons between the same classes for both algorithms, as can be seen, the predictions obtained for the cDCGAN-PSO are better than for the DCGAN-PSO. This again may be due to the differences between the resolutions of both versions. However, these differences are not too large.

It is necessary to mention that the healthy class in all its versions was classified in the same way by the classifier system. However, the COVID-19 and pneumonia classes were classified as COVID-19. The reason for this may be mainly due to the low resolution of the synthetic images and a lack of fine details that allow sifting between the two classes, since as it will be recalled, the CXR images of COVID-19 represent pneumonia due to this disease, for which the pneumonia class can be confused with the COVID-19 class.

Dataset 1	Average prediction	Dataset 2	Average prediction	p-value	Wilcoxon rank-sum test
COVID-19 real	0.9988 ± 0.005	**COVID-19 DCGAN-PSO**	0.9388 ± 0.082	$5.78e^{-5}$	(-)
		COVID-19 cDCGAN-PSO	0.9885 ± 0.028	0.1691	(=)
Pneumonia real	0.9549 ± 0.101	**Pneumonia DCGAN-PSO**	0.8913 ± 0.092	$7.19e^{-5}$	(-)
		Pneumonia cDCGAN-PSO	0.9413 ± 0.112	0.2115	(=)
Healthy real	0.9998 ± 0.0001	**Healthy DCGAN-PSO**	0.8598 ± 0.124	$2.87e^{-11}$	(-)
		Healthy cDCGAN-PSO	0.9129 ± 0.132	$2.15e^{-10}$	(-)
COVID-19 DCGAN-PSO	0.9388 ± 0.082	**COVID-19 cDCGAN-PSO**	0.9885 ± 0.028	0.00485	(+)
Pneumonia DCGAN-PSO	0.8913 ± 0.092	**Pneumonia cDCGAN-PSO**	0.9413 ± 0.112	$1.67e^{-3}$	(+)
Healthy DCGAN-PSO	0.8598 ± 0.124	**Healthy cDCGAN-PSO**	0.9129 ± 0.132	0.00181	(+)

TABLE 5.6. Classification results using CViCom COVID-19 system. (-) means that Dataset 2 used is worse than Dataset 1, (+) means the opposite. (=) means the both datasets have the same performance.

5.5.7 Two-Dimensional Visualizations

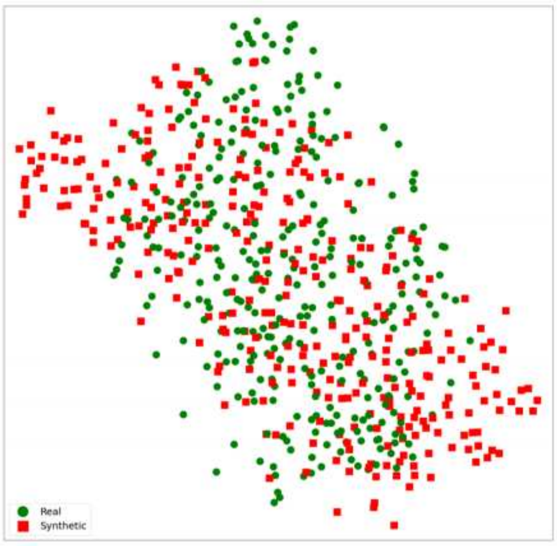
The two-dimensional representation by t-SNE of the feature vector sets of real CXR images and synthesized by the evolved GAN using DCGAN-PSO are found in Figure 5.11. While those relating to the cDCGAN-PSO are found in Figure 5.12.

The representation in two dimensions using the first two principal components of PCA of the feature vectors from the real images and those synthesized with the GANs evolved with DCGAN-PSO and cDCGAN-PSO are shown in Figure 5.13 and Figure 5.14, respectively.

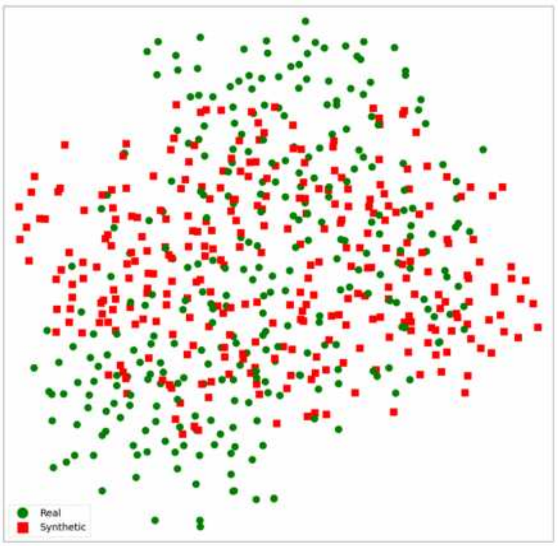
As seen in both visual representations for both algorithms, the area covered by real images of all classes is highly similar to that of synthetic images of the same class. Although there are cases in which a part of the synthetic images overflows the area of the real images (e.g.

COVID-19 class in Fig. 5.11), the highest density of synthetic images is positioned in the area of the real ones. In none of the visualizations can be observed a set of cases completely isolated from the real set, which would suggest that there is no similarity between both sets of images.

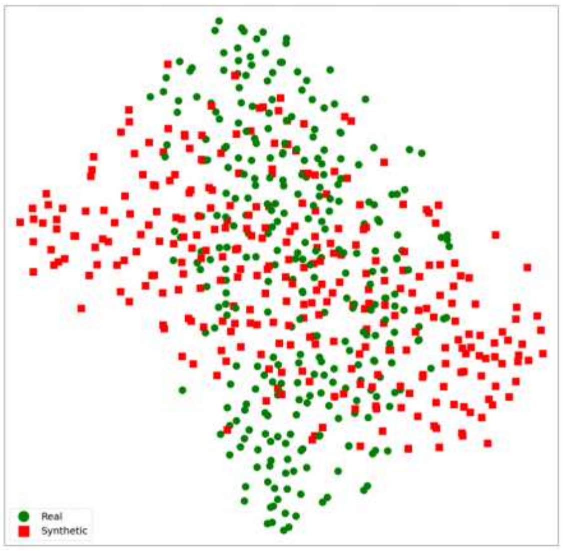
The above mentioned represents that the visual diversity of the synthetic images is highly similar to that of the real images. This may represent the absence of mode collapse, which is an indicator of a more stable training of GANs using the designed algorithms.



(a) COVID-19 class.

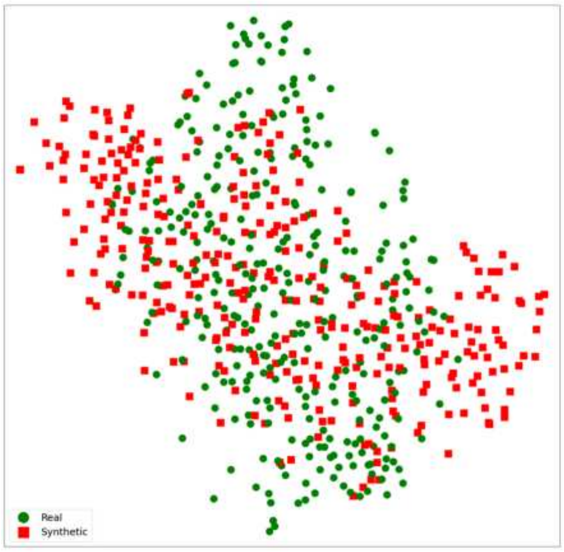


(b) Healthy class.

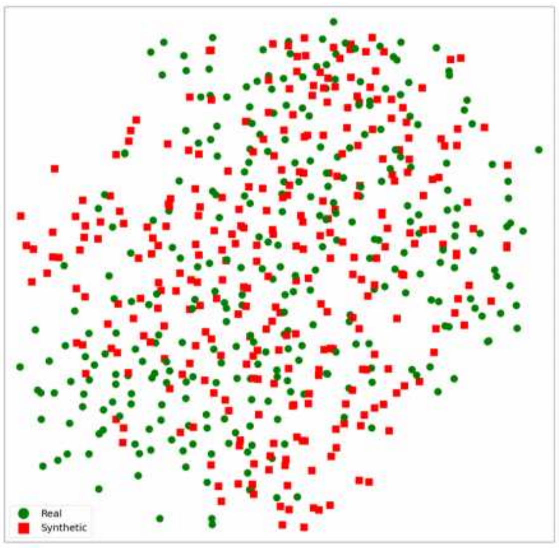


(c) Pneumonia class.

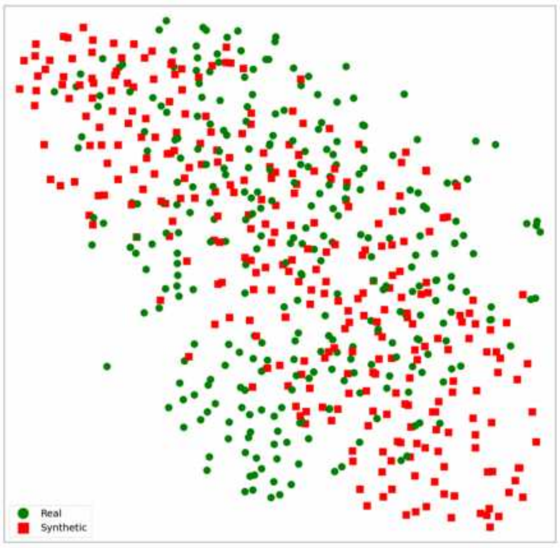
FIGURE 5.11. **t-SNE comparison of real and synthetic CXR images from DCGAN-PSO.** Green for real images and red for synthetic ones.



(a) COVID-19 class.

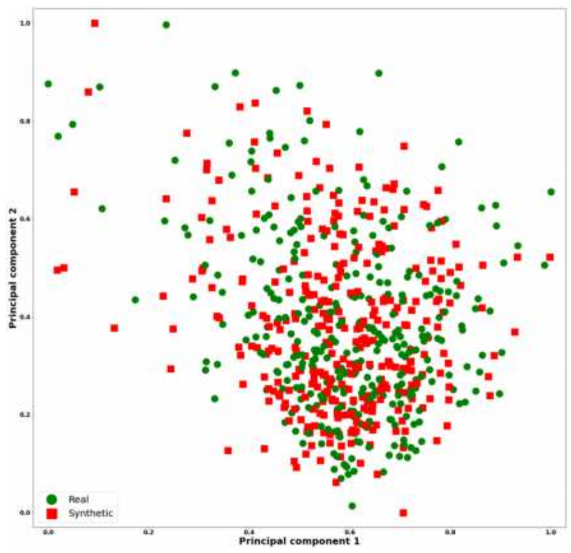


(b) Healthy class.

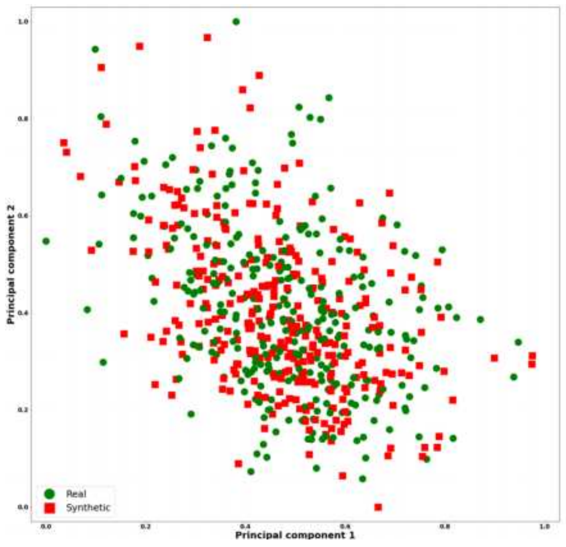


(c) Pneumonia class.

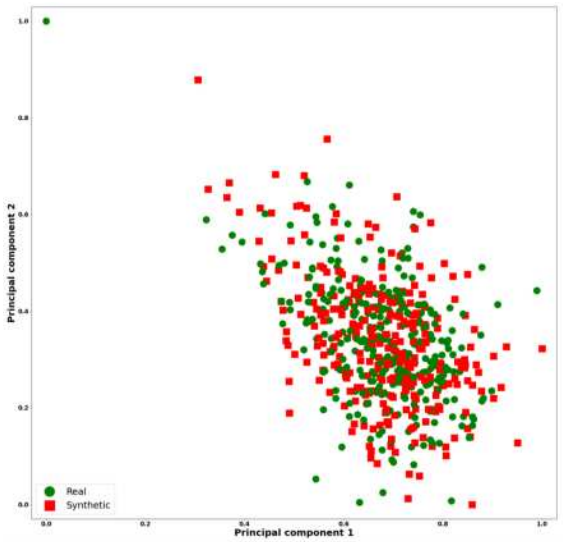
FIGURE 5.12. **t-SNE comparison of real and synthetic CXR images from cDCGAN-PSO.** Green for real images and red for synthetic ones.



(a) COVID-19 class.

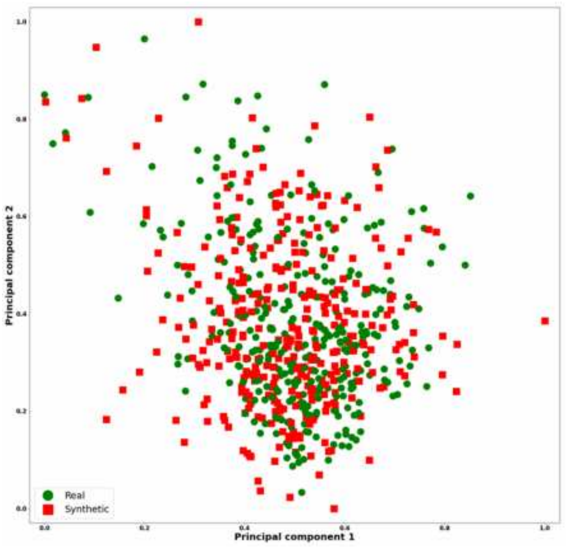


(b) Healthy class.

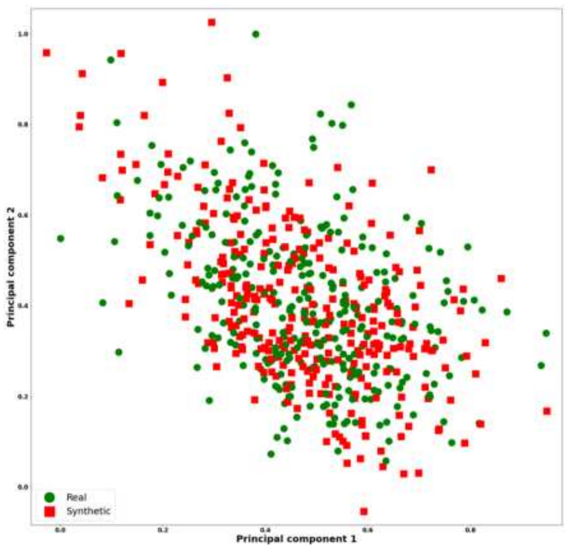


(c) Pneumonia class.

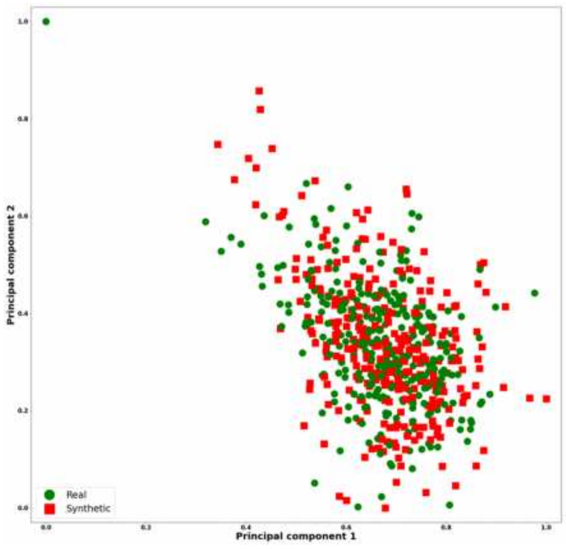
FIGURE 5.13. **PCA comparison of real and synthetic CXR images from DCGAN-PSO.** Green for real images and red for synthetic ones.



(a) COVID-19 class.



(b) Healthy class.



(c) Pneumonia class.

FIGURE 5.14. **PCA comparison of real and synthetic CXR images from cDCGAN-PSO.** Green for real images and red for synthetic ones.

5.6 Hypothesis Validation

The purpose of the experimentation already presented is to test and validate the hypotheses designed for the present algorithms of neuroevolution of GANs, which are stated again:

- **Hypothesis 1**: Generate chest X-ray images with high similarity to the set of real images (measured by FID), preserving the quality and diversity of that, what would be an indicator of the avoidance or reduction of training instability problems.
- **Hypothesis 2**: Obtaining better FID results than handcrafted GANs for the synthesis of CXR images of pneumonia by COVID-19.
- **Hypothesis 3**: Improve the classification, in terms of accuracy, of CXR images in CNN, with respect to unbalanced sets, using artificially balanced sets using synthetic images obtained from evolved GANs.

The observations and related tests to check each hypothesis are as follows:

- **Hypothesis 1**:
  - The low FID values obtained through the FID evaluation using the complete real set (see Section 5.5.4), because the low values of this metric is an indicator of similarity in quality and diversity with the real set of images.
  - The quality and visual diversity of synthetic images (see Section 5.5.2). No mode collapse indicator, a problem related to training instability.
  - Similar classification results between real and synthetic classes using the CViCom COVID-19 system (see Section 5.5.6). This is an indicator of similarity between the synthetic and real images.
  - Similar areas between real and synthetic images in two-dimensional visualizations (see Section 5.5.7), which shows a similar diversity and quality between real and synthetic images.
- **Hypothesis 2**:
  - Better FID values than *state-of-the-art* handcrafted architectures for COVID-19 CXR images synthesis using GANs (see Section 5.5.4).
- **Hypothesis 3**:
  - Better performances in binary and multiclass classification with CNN using synthetic images obtained from evolved GANs (see Section 5.5.5).

The above allows to verify and validate the hypotheses established for the development of this thesis.

5.7 Chapter Summary

In this chapter the methodology, the experimentation carried out as well as the results and respective discussion are covered.

The results obtained allow to establish the evidence to validate the hypotheses developed in this thesis. Both versions of the GANs neuroevolution algorithm presented similar performances in the multiple tests carried out, highlighting the good level of diversity and quality obtained, highly similar to those of the real sets. However, the cDCGAN-PSO presented advantages related to the quality of its creations and its savings in computational time.

The adequate progressive design and evolutionary training of the GANs allowed to obtain better performances than handcrafted GANs of the *state-of-the-art* for CXR synthesis, which indicate the support that neuroevolution brings to the GANs.

The good performance in this case study makes it possible to establish the potential of the algorithm to be used in other types of biomedical images or even outside of this area to support increasing the amounts of data available for tasks based on Deep Learning, as verified by its use in CNN classification.

Part of the experimentation of this chapter related to the DCGAN-PSO was used for the publication of a conference paper in the Congress on Evolutionary Computation (CEC) 2021[3]. This paper is attached in Appendix 3.

Part of the experimentation of this chapter related to the cDCGAN-PSO was used for the publication of a conference paper in the International Symposium on Medical Information Processing and Analysis (SIPAIM) 2021[4]. This paper is attached in Appendix 4.

[3]https://cec2021.mini.pw.edu.pl/

[4]https://sipaim.org/

CHAPTER 6

Conclusion and Future Work

In this work DCGAN-PSO and cDCGAN-PSO algorithms were presented. They are, to the best of the author's knowledge, the first Swarm Intelligence algorithms for Generative Adversarial Networks (GANs) neuroevolution based on Particle Swarm Optimization (PSO) that performs the search and training of DCGAN architectures through their progressive growth.

The present work also focused on the use of GANs evolution in the automatic design and training of architectures for biomedical images generation (previously unexplored area by GANs neuroevolution), specifically chest X-ray images.

The obtained results showed that the visual quality and diversity of the synthetic CXR images from the evolution of GANs were better than the synthesized results by handcrafted architectures from previous works, measured through Fréchet Inception Distance (FID).

The low FID values obtained from the synthetic images generated by the evolved GANs indicate the similarity of diversity and quality with respect to the real images, which is an indication of the reduction in common problems of GANs training instability, such as mode collapse (lack of diversity in synthetic images) or gradient fading (poor quality of results). This similarity can be observed more clearly by observing batches of synthetic images, which manage to mimic the general morphological characteristics of the real CXR images.

These results make it possible to verify that the use of progressive growth allows greater stability in training, as mentioned in its original article (Karras et al. 2017), and in conjunction with the intelligent search of the GAN topology using PSO, high quality results are obtained.

In addition, through the use of synthetic images in the improvement of the CNN classification of CXR images and through an external classification system, the similarity that exists between both types of images in various groups of CXR images is reinforced. Two-dimensional visualizations using t-SNE and PCA allow to accentuate this conclusion.

In addition, the premature convergence behavior by the developed PSO could allow to reduce search times, necessary in cases of limited computational budget, since finding a network with a good performance in just a few generations avoids an extensive search that other evolutionary computation algorithms with slower convergence might need. This characteristic can be an attractive feature in various applications to find a GAN that meets the specific needs of the task with good quality results in reasonable time.

For future work the proposed algorithm can be expanded through the following approaches:

- Reducing the size of the networks (reducing the range of parameters) and limiting the use to only transposed convolutional layers in the generator, since many DCGAN-based architectures only use these layers. This is in order to reduce computational costs, since a run of the versions lasts at least a week with limited computational budget.
- Obtaining higher resolution synthetic CXR images that contain greater and finer details with two purposes: 1. To get a better performance in tasks that use synthetic images e.g. CNN classification. 2. To consult radiologist experts to carry out a detailed study of the synthetic images.
- Modifying the PSO operators, in such a way that they not only take into account the evolution of the layers as a whole but also evolve their inner parameters.
- Expand the final resolution of synthetic images, to obtain a better quality that can better support Deep Learning-based systems.
- Use the algorithms in other types of biomedical images and other subsequent applications, such as segmentation.
- Using other types of GANs architectures compatible with the encoding used based on DCGANs.

Bibliography

Abdelhalim, Ibrahim Saad Aly, Mamdouh Farouk Mohamed and Yousef Bassyouni Mahdy (2021). 'Data augmentation for skin lesion using self-attention based progressive generative adversarial network'. In: *Expert Systems with Applications* 165, p. 113922. DOI: 10.1016/j.eswa.2020.113922.

Abdollahi, Behnaz, Naofumi Tomita and Saeed Hassanpour (2020). 'Data Augmentation in Training Deep Learning Models for Medical Image Analysis'. In: *Deep Learners and Deep Learner Descriptors for Medical Applications*. Springer, pp. 167–180. DOI: 10.1007/978-3-030-42750-4_6.

Aggarwal and Charu C (2018). *Neural networks and deep learning*. Springer. DOI: 10.1007/978-3-319-94463-0.

Aloysius, Neena and M Geetha (2017). 'A review on deep convolutional neural networks'. In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, pp. 0588–0592. DOI: 10.1109/ICCSP.2017.8286426.

Alqahtani, Hamed, Manolya Kavakli-Thorne and Gulshan Kumar (2019). 'Applications of generative adversarial networks (gans): An updated review'. In: *Archives of Computational Methods in Engineering*, pp. 1–28. DOI: 10.1007/s11831-019-09388-y.

Arjovsky, Martin, Soumith Chintala and Léon Bottou (Jan. 2017). 'Wasserstein GAN'. In: *arXiv e-prints*. arXiv: 1701.07875 [stat.ML].

Bakator, Mihalj and Dragica Radosav (2018). 'Deep learning and medical diagnosis: A review of literature'. In: *Multimodal Technologies and Interaction* 2.3, p. 47. DOI: 10.3390/mti2030047.

Baltruschat, Ivo M et al. (2019). 'Comparison of deep learning approaches for multi-label chest X-ray classification'. In: *Scientific reports* 9.1, pp. 1–10. DOI: 10.1038/s41598-019-42294-8.

Bansal, Jagdish Chand, Pramod Kumar Singh and Nikhil R Pal (2019). *Evolutionary and swarm intelligence algorithms*. Springer. DOI: 10.1007/978-3-319-91341-4.

Beers, Andrew et al. (May 2018). 'High-resolution medical image synthesis using progressively grown generative adversarial networks'. In: *arXiv e-prints*. arXiv: 1805.03144 [cs.CV].

Bhandary, Abhir et al. (2020). 'Deep-learning framework to detect lung abnormality–A study with chest X-Ray and lung CT scan images'. In: *Pattern Recognition Letters* 129, pp. 271–278. DOI: 10.1016/j.patrec.2019.11.013.

Bonilla-Aldana, D Katterine, Kuldeep Dhama and Alfonso J Rodriguez-Morales (2020). 'Revisiting the one health approach in the context of COVID-19: a look into the ecology of this emerging disease'. In: *Adv Anim Vet Sci* 8.3, pp. 234–237. DOI: 10.17582/journal.aavs/2020/8.3.234.237.

Buda, Mateusz, Atsuto Maki and Maciej A Mazurowski (2018). 'A systematic study of the class imbalance problem in convolutional neural networks'. In: *Neural Networks* 106, pp. 249–259. DOI: 10.1016/j.neunet.2018.07.011.

Cho, Hwi-Yeon and Yong-Hyuk Kim (2019). 'Stabilized Training of Generative Adversarial Networks by a Genetic Algorithm'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, pp. 51–52. DOI: 10.1145/3319619.3326774.

Chowdhury, M. E. H. et al. (2020). 'Can AI Help in Screening Viral and COVID-19 Pneumonia?' In: *IEEE Access* 8, pp. 132665–132676. DOI: 10.1109/ACCESS.2020.3010287.

Civit-Masot, Javier et al. (2020). 'Deep learning system for COVID-19 diagnosis aid using X-ray pulmonary images'. In: *Applied Sciences* 10.13, p. 4640. DOI: 10.3390/app10134640.

Cohen, Joseph Paul et al. (June 2020). 'COVID-19 Image Data Collection: Prospective Predictions Are the Future'. In: *arXiv e-prints*. arXiv: 2006.11988 [q-bio.QM]. URL: https://github.com/ieee8023/covid-chestxray-dataset.

Corman, Victor M et al. (2020). 'Detection of 2019 novel coronavirus (2019-nCoV) by real-time RT-PCR'. In: *Eurosurveillance* 25.3, p. 2000045. DOI: 10.2807/1560-7917.ES.2020.25.3.2000045.

Costa, Victor et al. (2019). 'COEGAN: Evaluating the Coevolution Effect in Generative Adversarial Networks'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, pp. 374–382. DOI: 10.1145/3321707.3321746.

— (2020a). 'Exploring the evolution of GANs through quality diversity'. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 297–305. DOI: 10.1145/3377930.3389824.

— (2020b). 'Neuroevolution of Generative Adversarial Networks'. In: *Deep Neural Evolution*. Springer, pp. 293–322. DOI: 10.1007/978-981-15-3685-4_11.

— (2020c). 'Using Skill Rating as Fitness on the Evolution of GANs'. In: *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, pp. 562–577. DOI: 10.1007/978-3-030-43722-0_36.

Denton, Emily et al. (2015). 'Deep Generative Image Models Using a Laplacian Pyramid of Adversarial Networks'. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'15. Montreal, Canada: MIT Press, pp. 1486–1494. DOI: 10.5555/2969239.2969405.

Al-Dujaili, Abdullah, Tom Schmiedlechner and Una-May O'Reilly (July 2018). 'Towards Distributed Coevolutionary GANs'. In: *arXiv e-prints*. arXiv: 1807.08194 [cs.NE].

Dumoulin, Vincent and Francesco Visin (Mar. 2016). 'A guide to convolution arithmetic for deep learning'. In: *ArXiv e-prints*. eprint: 1603.07285.

Eiben, Agoston E and James E Smith (2015). *Introduction to evolutionary computing*. Springer. DOI: 10.1007/978-3-662-44874-8.

Eklund, Anders (Dec. 2019). 'Feeding the zombies: Synthesizing brain volumes using a 3D progressive growing GAN'. In: *arXiv e-prints*. arXiv: 1912.05357 [eess.IV].

Galván, Edgar and Peter Mooney (June 2020). 'Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges'. In: *arXiv e-prints*. arXiv: 2006.05415 [cs.NE].

Garciarena, Unai, Roberto Santana and Alexander Mendiburu (2018). 'Evolved GANs for Generating Pareto Set Approximations'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, pp. 434–441. ISBN: 9781450356183. DOI: 10.1145/3205455.3205550.

Goodfellow, Ian, Yoshua Bengio and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Goodfellow, Ian J. et al. (2014). 'Generative Adversarial Networks'. In: *arXiv e-prints*. arXiv: 1406.2661 [stat.ML].

Gordienko, Yu et al. (2018). 'Deep learning with lung segmentation and bone shadow exclusion techniques for chest X-ray analysis of lung cancer'. In: *International Conference on Computer Science, Engineering and Education Applications*. Springer, pp. 638–647. DOI: 10.1007/978-3-319-91008-6_63.

Guendel, Sebastian et al. (May 2019). 'Multi-task Learning for Chest X-ray Abnormality Classification on Noisy Labels'. In: *arXiv e-prints*. arXiv: 1905.06362 [cs.CV].

Gui, Jie et al. (Jan. 2020). 'A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications'. In: *arXiv e-prints*. arXiv: 2001.06937 [cs.LG].

Guibas, John T., Tejpal S. Virdi and Peter S. Li (Sept. 2017). 'Synthetic Medical Images from Dual Generative Adversarial Networks'. In: *arXiv e-prints*. arXiv: 1709.01872 [cs.CV].

Gulrajani, Ishaan et al. (2017). 'Improved training of wasserstein gans'. In: *Advances in neural information processing systems*, pp. 5767–5777. DOI: 10.5555/3295222.3295327.

Han, Changhee et al. (2019). 'Learning more with less: Conditional PGGAN-based data augmentation for brain metastases detection using highly-rough annotation on MR images'. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 119–127. DOI: 10.1145/3357384.3357890.

Heusel, Martin et al. (2017). 'Gans trained by a two time-scale update rule converge to a local nash equilibrium'. In: *Advances in neural information processing systems*, pp. 6626–6637. DOI: 10.5555/3295222.3295408.

Huang, Chaolin et al. (2020). 'Clinical features of patients infected with 2019 novel coronavirus in Wuhan, China'. In: *The lancet* 395.10223, pp. 497–506. DOI: 10.1016/S0140-6736(20)30183-5.

Huang, He, Philip S. Yu and Changhu Wang (Mar. 2018). 'An Introduction to Image Synthesis with Generative Adversarial Nets'. In: *arXiv e-prints*. arXiv: 1803.04469 [cs.CV].

Hubel, David H and Torsten N Wiesel (1959). ‘Receptive fields of single neurones in the cat’s striate cortex’. In: *The Journal of physiology* 148.3, pp. 574–591. DOI: 10.1113/jphysiol.1959.sp006308.

Human, Becoming (2020). *Building a Convolutional Neural Network (CNN) Model for Image classification.* URL: https://becominghuman.ai/building-a-convolutional-neural-network-cnn-model-for-image-classification-116f77a7a236 (visited on 02/06/2021).

Iba, Hitoshi (2018). *Evolutionary approach to machine learning and deep neural networks*. Springer. DOI: 10.1007/978-981-13-0200-8.

Iba, Hitoshi and Nasimul Noman (2020). *Deep Neural Evolution*. Springer. DOI: 10.1007/978-981-15-3685-4.

Ismael, Aras M and Abdulkadir Şengür (2021). ‘Deep learning approaches for COVID-19 detection based on chest X-ray images’. In: *Expert Systems with Applications* 164, p. 114054. DOI: 10.1016/j.eswa.2020.114054.

Jain, Rachna et al. (2021). ‘Deep learning based detection and analysis of COVID-19 on chest X-ray images’. In: *Applied Intelligence* 51.3, pp. 1690–1700. DOI: 10.1007/s10489-020-01902-1.

Junior, Francisco Erivaldo Fernandes and Gary G Yen (2019). ‘Particle swarm optimization of deep neural networks architectures for image classification’. In: *Swarm and Evolutionary Computation* 49, pp. 62–74. DOI: 10.1016/j.swevo.2019.05.010.

Karakanis, Stefanos and Georgios Leontidis (2020). ‘Lightweight deep learning models for detecting COVID-19 from chest X-ray images’. In: *Computers in Biology and Medicine* 130, p. 104181. DOI: 10.1016/j.compbiomed.2020.104181.

Karbhari, Yash et al. (2021). ‘Generation of Synthetic Chest X-ray Images and Detection of COVID-19: A Deep Learning Based Approach’. In: *Diagnostics* 11.5, p. 895. DOI: 10.3390/diagnostics11050895.

Karras, Tero et al. (Oct. 2017). ‘Progressive Growing of GANs for Improved Quality, Stability, and Variation’. In: *arXiv e-prints*. arXiv: 1710.10196 [cs.NE].

Kazeminia, Salome et al. (2020). ‘GANs for medical image analysis’. In: *Artificial Intelligence in Medicine* 109, p. 101938. DOI: 10.1016/j.artmed.2020.101938.

Kennedy, James and Russell Eberhart (1995). ‘Particle swarm optimization’. In: *Proceedings of ICNN’95-International Conference on Neural Networks*. Vol. 4. IEEE, pp. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

Khalifa, Nour Eldeen M. et al. (Apr. 2020). ‘Detection of Coronavirus (COVID-19) Associated Pneumonia based on Generative Adversarial Networks and a Fine-Tuned Deep Transfer Learning Model using Chest X-ray Dataset’. In: *arXiv e-prints*. arXiv: 2004.01184 [eess.IV].

Khan, Asifullah et al. (2020). ‘A survey of the recent architectures of deep convolutional neural networks’. In: *Artificial Intelligence Review* 53, pp. 5455–5516. DOI: 10.1007/s10462-020-09825-6.

Khan, Salman et al. (2018). ‘A guide to convolutional neural networks for computer vision’. In: *Synthesis Lectures on Computer Vision*. DOI: 10.2200/S00822ED1V01Y201712COV015.

Kingma, Diederik P. and Jimmy Ba (Dec. 2014). ‘Adam: A Method for Stochastic Optimization’. In: *arXiv e-prints*. arXiv: 1412.6980 [cs.LG].

Kora Venu, Sagar and Sridhar Ravula (2021). ‘Evaluation of Deep Convolutional Generative Adversarial Networks for Data Augmentation of Chest X-ray Images’. In: *Future Internet* 13.1, p. 8. DOI: 10.3390/fi13010008.

Korde, Charudatta G et al. (2019). ‘Training of Generative Adversarial Networks with Hybrid Evolutionary Optimization Technique’. In: *2019 IEEE 16th India Council International Conference (INDICON)*. IEEE, pp. 1–4. DOI: 10.1109/INDICON47234.2019.9030352.

Korkinof, Dimitrios et al. (July 2018). ‘High-Resolution Mammogram Synthesis using Progressive Generative Adversarial Networks’. In: *arXiv e-prints*. arXiv: 1807.03401 [cs.CV].

Kumar, Aishwarya, Puneet Kumar Gupta and Ankita Srivastava (2020). ‘A review of modern technologies for tackling COVID-19 pandemic’. In: *Diabetes & Metabolic Syndrome: Clinical Research & Reviews* 14.4, pp. 569–573. DOI: 10.1016/j.dsx.2020.05.008.

Kumar, Sandeep, Anand Nayyar and Anand Paul (2019). *Swarm Intelligence and Evolutionary Algorithms in Healthcare and Drug Development*. CRC Press. ISBN: 9780367257576.

Litjens, Geert et al. (2017). ‘A survey on deep learning in medical image analysis’. In: *Medical image analysis* 42, pp. 60–88. DOI: 10.1016/j.media.2017.07.005.

Loey, Mohamed, Florentin Smarandache and Nour Eldeen M Khalifa (2020). ‘Within the Lack of Chest COVID-19 X-ray Dataset: A Novel Detection Model Based on GAN and Deep Transfer Learning’. In: *Symmetry* 12.4, p. 651. DOI: 10.3390/sym12040651.

Lu, Le et al. (2017). ‘Deep learning and convolutional neural networks for medical image computing’. In: *Advances in Computer Vision and Pattern Recognition*. DOI: 10.1007/978-3-319-42999-1.

Lucic, Mario et al. (2018). ‘Are gans created equal? a large-scale study’. In: *Advances in neural information processing systems*, pp. 700–709. DOI: 10.5555/3326943.3327008.

Maaten, Laurens van der and Geoffrey Hinton (2008). ‘Visualizing data using t-SNE’. In: *Journal of machine learning research* 9.Nov, pp. 2579–2605. URL: https://www.jmlr.org/papers/v9/vandermaaten08a.html.

Madani, Ali et al. (2018). ‘Semi-supervised learning with generative adversarial networks for chest x-ray classification with ability of data domain adaptation’. In: *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. IEEE, pp. 1038–1042. DOI: 10.1109/ISBI.2018.8363749..

Mao, Xudong et al. (2017). ‘Least squares generative adversarial networks’. In: *Proceedings of the IEEE international conference on computer vision*, pp. 2794–2802. DOI: 10.1109/ICCV.2017.304..

Martinez, Aritz D et al. (2020). ‘Lights and shadows in Evolutionary Deep Learning: Taxonomy, critical methodological analysis, cases of study, learned lessons, recommendations and challenges’. In: *Information Fusion* 67, pp. 161–194. DOI: 10.1016/j.inffus.2020.10.014.

McCulloch, Warren S and Walter Pitts (1943). ‘A logical calculus of the ideas immanent in nervous activity’. In: *Bulletin of mathematical biophysics* 5.4, pp. 115–133. DOI: 10.1007/BF02478259.

Mehta, Kaitav et al. (2019). ‘Data Augmentation using CA Evolved GANs’. In: *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, pp. 1087–1092. DOI: 10.1109/ISCC47284.2019.8969638.

Menon, Sumeet et al. (2020). ‘Generating Realistic COVID-19 x-rays with a Mean Teacher + Transfer Learning GAN’. In: *2020 IEEE International Conference on Big Data (Big Data)*, pp. 1216–1225. DOI: 10.1109/BigData50022.2020.9377878.

Middel, Luise, Christoph Palm and Marius Erdt (2019). ‘Synthesis of medical images using gans’. In: *Uncertainty for Safe Utilization of Machine Learning in Medical Imaging and Clinical Image-Based Procedures*. Springer, pp. 125–134. DOI: 10.1007/978-3-030-32689-0_13.

Miikkulainen, Risto et al. (2019). ‘Evolving deep neural networks’. In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, pp. 293–312. DOI: 10.1016/B978-0-12-815480-9.00015-3.

Mirza, Mehdi and Simon Osindero (Nov. 2014). ‘Conditional Generative Adversarial Nets’. In: *arXiv e-prints*. arXiv: 1411.1784 [cs.LG].

Miyato, Takeru et al. (Feb. 2018). ‘Spectral Normalization for Generative Adversarial Networks’. In: *arXiv e-prints*. arXiv: 1802.05957 [cs.LG].

Mohapatra, Subhashree, Tripti Swarnkar and Jayashankar Das (2021). ‘Deep convolutional neural network in medical image processing’. In: *Handbook of Deep Learning in Biomedical Engineering*. Elsevier, pp. 25–60. DOI: 10.1016/B978-0-12-823014-5.00006-5.

Mooney, Paul (2018). ‘Chest x-ray images (pneumonia)’. In: *kaggle, Marzo*. URL: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia.

Nalepa, Jakub, Michal Marcinkiewicz and Michal Kawulok (2019). ‘Data augmentation for brain-tumor segmentation: A review’. In: *Frontiers in Computational Neuroscience* 13.83. DOI: 10.3389/fncom.2019.00083.

Nayyar, Anand, Dac-Nhuong Le and Nhu Gia Nguyen (2018). *Advances in swarm intelligence for optimizing problems in computer science*. CRC Press. ISBN: 9781138482517.

Neyshabur, Behnam, Srinadh Bhojanapalli and Ayan Chakrabarti (May 2017). ‘Stabilizing GAN Training with Multiple Random Projections’. In: *arXiv e-prints*. arXiv: 1705.07831 [cs.LG].

Ng, Ming-Yen et al. (2020). ‘Imaging profile of the COVID-19 infection: radiologic findings and literature review’. In: *Radiology: Cardiothoracic Imaging* 2.1. DOI: 10.1148/ryct.2020200034.

Ni, Yao et al. (July 2018). 'CAGAN: Consistent Adversarial Training Enhanced GANs'. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, pp. 2588–2594. DOI: 10.24963/ijcai.2018/359.

Noman, Nasimul (2020). 'A Shallow Introduction to Deep Neural Networks'. In: *Deep Neural Evolution*. Springer, pp. 35–63. DOI: 10.1007/978-981-15-3685-4_2.

Pan, Zhaoqing et al. (2019). 'Recent progress on generative adversarial networks (GANs): A survey'. In: *IEEE Access* 7, pp. 36322–36333. DOI: 10.1109/ACCESS.2019.2905015.

Panwar, Harsh et al. (2020). 'Application of deep learning for fast detection of COVID-19 in X-Rays using nCOVnet'. In: *Chaos, Solitons & Fractals* 138, p. 109944. DOI: 10.1016/j.chaos.2020.109944.

Paszke, Adam et al. (2017). 'Automatic differentiation in pytorch'. In: URL: https://openreview.net/pdf?id=BJJsrmfCZ.

Patterson, J. and A. Gibson (2017). *Deep Learning: A Practitioner's Approach*. O'Reilly. ISBN: 9781491914250. URL: https://books.google.com.mx/books?id=BdPrrQEACAAJ.

Perez, Luis and Jason Wang (Dec. 2017). 'The Effectiveness of Data Augmentation in Image Classification using Deep Learning'. In: *arXiv e-prints*. arXiv: 1712.04621 [cs.CV].

Piotrowski, Adam P et al. (2017). 'Swarm intelligence and evolutionary algorithms: Performance versus speed'. In: *Information Sciences* 384, pp. 34–85. DOI: 10.1016/j.ins.2016.12.028.

Radford, Alec, Luke Metz and Soumith Chintala (Nov. 2015). 'Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks'. In: *arXiv e-prints*. arXiv: 1511.06434 [cs.LG].

Rajaraman, Sivaramakrishnan et al. (2018). 'Visualization and interpretation of convolutional neural network predictions in detecting pneumonia in pediatric chest radiographs'. In: *Applied Sciences* 8.10, p. 1715. DOI: 10.3390/app8101715.

Rajpurkar, Pranav et al. (Nov. 2017). 'CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning'. In: *arXiv e-prints*. arXiv: 1711.05225 [cs.CV].

Rosenblatt, Frank (1958). 'The perceptron: a probabilistic model for information storage and organization in the brain.' In: *Psychological review* 65.6, pp. 386–408. DOI: 10.1037/h0042519.

Sahu, Amaresh, Sushanta Kumar Panigrahi and Sabyasachi Pattnaik (2012). 'Fast convergence particle swarm optimization for functions optimization'. In: *Procedia Technology* 4, pp. 319–324. DOI: 10.1016/j.protcy.2012.05.048.

Salehinejad, Hojjat et al. (2018). 'Generalization of deep neural networks for chest pathology classification in x-rays using generative adversarial networks'. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 990–994. DOI: 10.1109/ICASSP.2018.8461430..

Salimans, Tim and Diederik P. Kingma (2016). 'Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., pp. 901–909. ISBN: 9781510838819. DOI: 10.5555/3157096.3157197.

Shaham, Tamar Rott, Tali Dekel and Tomer Michaeli (2019). 'Singan: Learning a generative model from a single natural image'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4570–4580. DOI: 10.1109/ICCV.2019.00467.

Shams, MY et al. (2020). 'Why Are Generative Adversarial Networks Vital for Deep Neural Networks? A Case Study on COVID-19 Chest X-Ray Images'. In: *Big Data Analytics and Artificial Intelligence Against COVID-19: Innovation Vision and Approach*. Springer, pp. 147–162. DOI: 10.1007/978-3-030-55258-9_9.

Sheykhivand, Sobhan et al. (2021). 'Developing an efficient deep neural network for automatic detection of COVID-19 using chest X-ray images'. In: *Alexandria Engineering Journal* 60.3, pp. 2885–2903. DOI: 10.1016/j.aej.2021.01.011.

Shi, Feng et al. (2020). 'Review of artificial intelligence techniques in imaging data acquisition, segmentation and diagnosis for covid-19'. In: *IEEE Reviews in Biomedical Engineering*. DOI: 10.1109/RBME.2020.2987975.

Shin, Hoo-Chang et al. (2018). 'Medical image synthesis for data augmentation and anonymization using generative adversarial networks'. In: *International workshop on simulation* *and synthesis in medical imaging*. Springer, pp. 1–11. DOI: 10.1007/978-3-030-00536-8_1.

Shorten, Connor and Taghi M Khoshgoftaar (2019). 'A survey on image data augmentation for deep learning'. In: *Journal of Big Data* 6.1, p. 60. DOI: 10.1186/s40537-019-0197-0.

Shu, Han et al. (2019). 'Co-Evolutionary Compression for Unpaired Image Translation'. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3234–3243. DOI: 10.1109/ICCV.2019.00333.

Shukla, Nagesh et al. (2020). 'Half a century of computer methods and programs in biomedicine: A bibliometric analysis from 1970 to 2017'. In: *Computer methods and programs in biomedicine* 183, p. 105075. DOI: 10.1016/j.cmpb.2019.105075.

Simonyan, Karen and Andrew Zisserman (Sept. 2014). 'Very Deep Convolutional Networks for Large-Scale Image Recognition'. In: *arXiv e-prints*. arXiv: 1409.1556 [cs.CV].

Song, Xiaoning et al. (2020). 'SP-GAN: Self-growing and pruning generative adversarial networks'. In: *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2020.3005574..

Sorin, Vera et al. (2020). 'Creating Artificial Images for Radiology Applications Using Generative Adversarial Networks (GANs)-A Systematic Review'. In: *Academic Radiology*. DOI: 10.1016/j.acra.2019.12.024.

Stanley, Kenneth O et al. (2019). 'Designing neural networks through neuroevolution'. In: *Nature Machine Intelligence* 1.1, pp. 24–35. DOI: 10.1038/s42256-018-0006-z.

Stirenko, Sergii et al. (2018). 'Chest X-Ray Analysis of Tuberculosis by Deep Learning with Segmentation and Augmentation'. In: *2018 IEEE 38th International Conference on Electronics and Nanotechnology (ELNANO)*, pp. 422–428. DOI: 10.1109/ELNANO.2018.8477564.

Suetens, Paul (2017). *Fundamentals of medical imaging*. Cambridge university press. DOI: 10.1017/CBO9780511596803.

Szegedy, Christian et al. (2016). 'Rethinking the inception architecture for computer vision'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308.

Tabik, S. et al. (2020). 'COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images'. In: *IEEE Journal of Biomedical and Health Informatics* 24.12, pp. 3595–3605. DOI: 10.1109/JBHI.2020.3037127.

Teramoto, Atsushi et al. (2020). 'Deep learning approach to classification of lung cytological images: Two-step training using actual and synthesized images by progressive growing of generative adversarial networks'. In: *PloS one* 15.3, e0229951. DOI: 10.1371/journal.pone.0229951.

Toutouh, Jamal, Erik Hemberg and Una-May O'Reilly (2019). 'Spatial Evolutionary Generative Adversarial Networks'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, pp. 472–480. DOI: 10.1145/3321707.3321860.

Waheed, Abdul et al. (2020). 'Covidgan: Data augmentation using auxiliary classifier gan for improved covid-19 detection'. In: *IEEE Access* 8, pp. 91916–91923. DOI: 10.1109/ACCESS.2020.2994762..

Wang, L. et al. (2020). 'A State-of-the-Art Review on Image Synthesis With Generative Adversarial Networks'. In: *IEEE Access* 8, pp. 63514–63537. DOI: 10.1109/ACCESS.2020.2982224.

Wang, Zhengwei, Qi She and Tomas E. Ward (June 2019). 'Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy'. In: *arXiv e-prints*. arXiv: 1906.01529 [cs.LG].

Wang, C. et al. (2019). 'Evolutionary Generative Adversarial Networks'. In: *IEEE Transactions on Evolutionary Computation* 23.6, pp. 921–934. DOI: 10.1109/TEVC.2019.2895748.

Wold, Svante, Kim Esbensen and Paul Geladi (1987). 'Principal component analysis'. In: *Chemometrics and intelligent laboratory systems* 2.1-3, pp. 37–52. DOI: 10.1016/0169-7439(87)80084-9.

Xiang, Sitao and Hao Li (Apr. 2017). 'On the Effects of Batch and Weight Normalization in Generative Adversarial Networks'. In: *arXiv e-prints*. arXiv: 1704.03971 [stat.ML].

Yang, Xin-She (2015). *Recent advances in swarm intelligence and evolutionary computation*. Springer. DOI: 10.1007/978-3-319-13826-8.

Yi, Xin, Ekta Walia and Paul Babyn (2019). ‘Generative adversarial network in medical imaging: A review’. In: *Medical image analysis* 58, p. 101552. DOI: 10.1016/j.media.2019.101552.

Zhang, Dan and Anna Khoreva (Jan. 2019). ‘Progressive Augmentation of GANs’. In: *arXiv e-prints*. arXiv: 1901.10422 [cs.CV].

Zhang, Jianpeng et al. (2020). ‘Viral Pneumonia Screening on Chest X-rays Using Confidence-Aware Anomaly Detection’. In: *IEEE transactions on medical imaging*. DOI: 10.1109/TMI.2020.3040950.

Zhang, Tianyang et al. (2019). ‘SkrGAN: Sketching-rendering unconditional generative adversarial networks for medical image synthesis’. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, pp. 777–785. DOI: 10.1007/978-3-030-32251-9_85.

Zheng, W. et al. (2019). ‘Differential-Evolution-Based Generative Adversarial Networks for Edge Detection’. In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 2999–3008. DOI: 10.1109/ICCVW.2019.00362.

Zhou, Sharon (2020). *Build Basic Generative Adversarial Networks (GANs). Conditional Generation: Intuition.* URL: https://www.coursera.org/lecture/build-basic-generative-adversarial-networks-gans/conditional-generation-inputs-2OPrG (visited on 23/06/2021).

Zulkifley, Mohd Asyraf, Siti Raihanah Abdani and Nuraisyah Hani Zulkifley (2020). ‘COVID-19 Screening Using a Lightweight Convolutional Neural Network with Generative Adversarial Network Data Augmentation’. In: *Symmetry* 12.9, p. 1530. DOI: 10.3390/sym12091530.

Appendices

1 Vanilla DCGAN vs. WGAN vs. Spectral Normalization Experimentation

The WGAN-GP (see section 2.4.3) was implemented as a training mechanism for DCGANs to evolve due to its stability properties in training and good results obtained as a *state-of-the-art* model. For these tests, only 64^2 pixels resolution images were synthesized and no progressive growth was performed. These results were compared with training using Spectral Normalization (see section 2.4.4) for the DCGAN discriminator (SN-DCGAN) and with the original version of DCGAN (Vanilla DCGAN) (see section 2.4.1).

The results are compared in three aspects, quality of the synthetic images, variance of the final results obtained between different runs (the FID value was used in these two) and training speed (minutes).

The observations obtained in these aspects in the three implementations are the following:

- **Quality and Variance**: The results in the table .1 show the mean and standard deviation of the FID value obtained from training two different architectures with the three variants for 200 epochs each with 5 runs. As can be seen, WGAN-GP obtains the best FID values (lower is better) and also has a lower variance. SN-DCGAN is positioned as second best, although the average is not as good as WGAN-GP, it is quite low and with a not so high variance. Lastly, the worst approach on both architectures turned out to be Vanilla DCGAN with large FID values and quite high variance compared to its competitors. These quality results are exemplified with figure .1, where a batch of synthetic images is shown for the same architecture trained with the three variants.
- **Training speed**: The results of this field are found in the table .2, where the same architecture was trained with the three variants during 1000 epochs in 5 runs and its time was recorded. As can be seen, the shortest time was obtained by SN-DCGAN, requiring approximately only one third and one seventh of the time required by

Vanilla DCGAN and WGAN-GP respectively. This advantage has a great weight since it has limited computational resources.

With the previous results, the conclusion was reached that SN-DCGAN offers a trade-off between the quality and stability capabilities (little variance) of WGAN-GP and the speed of Vanilla DCGAN, and although it does not obtain a very good final quality, it is more stable compared to Vanilla DCGAN, which allows to obtain robustness in the fitness measurements carried out within DCGAN-PSO. That is why SN-DCGAN is the method used to train the particles of the proposed algorithm.

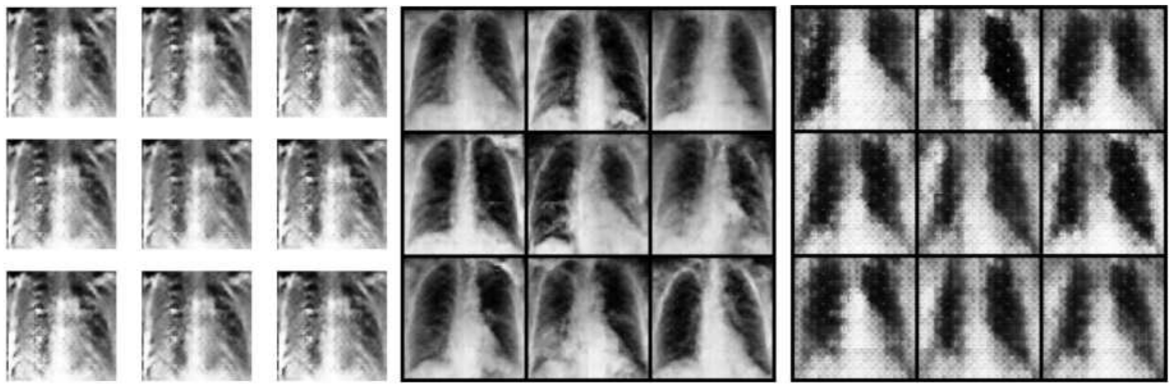


FIGURE .1. **Synthetic images obtained from the same DCGAN architecture trained with the three variants with 1000 epochs.** From left to right: Vanilla DCGAN, WGAN-GP and SN-DCGAN.

Architecure	**Vanilla-DCGAN**	**WGAN-GP**	**SN-DCGAN**
1	19.66 ± 17.59	1.42 ± 0.11	3.9755 ± 0.73
2	22.28 ± 36.55	1.91 ± 0.13	7.49 ± 1.12

TABLE .1. FID results ($mean \pm std.dev.$) of the three DCGAN variants. Two different architectures were trained for 200 epochs and 5 times with each variant.

Model	Training time per 1000 epochs (min.)
Vanilla-DCGAN	25.15 ± 2.30
WGAN-GP	61.4 ± 5.32
SN-DCGAN	9.21 ± 0.45

TABLE .2. Time results in minutes for 3 runs of each variant for 1000 epochs ($mean \pm std.dev.$).

2 Thesis Activities Schedule

Year	**2020**				**2021**						
Activity	September	October	November	December	January	February	March	April	May	June	July
Literature review	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	
Obtaining CXR images and processing	**X**	**X**	**X**				**X**				
EA development and implementation	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**			
First batch of experiments (DCGAN-PSO)	**X**	**X**	**X**	**X**	**X**	**X**					
Research stay at IIMAS-UNAM							**X**	**X**			
Second batch of experiments (cDCGAN-PSO)							**X**	**X**	**X**	**X**	
Thesis redaction	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	
Thesis submission and defense										**X**	**X**
Design and submission of article for CEC 2021 (DCGAN-PSO)			**X**	**X**	**X**	**X**					
Presentation of accepted article in CEC 2021 (DCGAN-PSO)											**X**
Design and submission of article for SIPAIM 2021 (cDCGAN-PSO)										**X**	**X**

TABLE .3. Schedule of activities.

3 Congress on Evolutionary Computing 2021 Paper

J.-A. Rodríguez-de-la-Cruz, H.-G. Acosta-Mesa and E. Mezura-Montes, "Evolution of Generative Adversarial Networks Using PSO for Synthesis of COVID-19 Chest X-ray Images," 2021 IEEE Congress on Evolutionary Computation (CEC), 2021, pp. 2226-2233, doi: 10.1109/CEC45853.2021.9504743.

Evolution of Generative Adversarial Networks Using PSO for Synthesis of COVID-19 Chest X-ray Images

Juan-Antonio Rodríguez-de-la-Cruz
Artificial Intelligence Research Institute
University of Veracruz
Xalapa, Veracruz, México
juanantonio_2604@hotmail.es

Héctor-Gabriel Acosta-Mesa
Artificial Intelligence Research Institute
University of Veracruz
Xalapa, Veracruz, México
heacosta@uv.mx

Efrén Mezura-Montes
Senior Member, IEEE
Artificial Intelligence Research Institute
University of Veracruz
Xalapa, Veracruz, México
emezura@uv.mx

Abstract—**The use of biomedical images for the training of various Deep Learning (DL) systems oriented to health has reported a competitive performance. However, DL needs a large number of images for a correct generalization and, particularly in the case of biomedical images, these can be scarce. Generative Adversarial Networks (GANs) as Data Augmenting tools have reaped significant results to improve performance in tasks that involve the use of this kind of image. However, the architectural design of these generative models in the biomedical image area has been usually relegated to the expertise of researchers. Moreover, GANs are affected by training instability that may lead to poor quality results. This paper presents a neuroevolution algorithm based on Particle Swarm Optimization for the design and training of GANs for the generation of biomedical Chest X-Ray (CXR) images of pneumonia caused by COVID-19. The proposed approach allows having a swarm of GANs topologies, where each one of them grows progressively while being trained at the same time. The fitness value is based on the Frechet Inception Distance (FID). The proposed algorithm is able to obtain better FID results compared to handcrafted GANs for the synthesis of CXR images.**

Index Terms—**GANs neuroevolution, Particle Swarm Optimization, Biomedical Imaging Synthesis, COVID-19.**

I. Introduction

Computer-aided diagnosis (CAD) with biomedical images has made use of Deep Learning (DL) models in recent years. This approach has proven to improve the performance in disease detection and areas of interest segmentation [1]. One of the main obstacles in DL-based models in CAD is the size restrictions of the available image datasets. Biomedical images are scarce because of four main reasons: **(1)** patient privacy; **(2)** cost of the tests; **(3)** health bias, i.e. more available data of sick patients; and **(4)** risk, as some tests expose the patient to high doses of radiation [2].

For the aforementioned, the use of Data Augmentation (DA) techniques that allow artificially increasing the number of available samples, has been very useful in DL. However, as those DA classic tools usually focus on oversampling the original images by altering them, the expected performance improvement in the learning task may be limited.

Therefore, the efforts of the last decade have turned around developing more and better techniques to support model training with small datasets and unbalanced classes. In recent years, the use of Generative Adversarial Networks (GANs) for the synthesis of images that expand the size of available datasets has significantly improved the capabilities provided by previous efforts in DA for CAD [3]. However, the usage of GANs comes with training instability issues that have plagued these generative models since their creation. Furthermore, the design of a topology for a specific task depends most of the time on the iterative design process based on the researcher's expertise. Different works have been developed with the support of Evolutionary Computation (EC) to design GANs [4], i.e., neuroevolution. The research efforts have been particularly centered in the use of Genetic Algorithms (GAs) and Coevolution. In contrast, to the best of the authors' knowledge, Swarm Intelligence (SI) algorithms are yet to be explored.

This paper presents the so-called Deep Convolutional Generative Adversarial Networks with Particle Swarm Optimization (**DCGAN-PSO**), an algorithm to design and train DCGANs specialized in image synthesis. PSO has showed a faster convergence in different domains than that of Genetic Algorithms [5] (one of the main approaches for GANs neuroevolution). Such property is attractive in the context of neuroevolution, as it may reduce the usually costly search process. Therefore, this work aims to analyze the capabilities of PSO to get competitive DCGANs with a limited computational budget.

The remaining of this paper is organized as follows. In Section II brief backgrounds about GANs, PSO and COVID-19 CXR are presented. The related work about GANs neuroevolution and CXR synthesis is shown in Section III. The proposed algorithm is described in detail in Section IV. Section V recounts the experiments carried out and their results, while Section VI discusses them. For last, Section VII concludes this paper and proposes directions for future work.

II. Background

A. Generative Adversarial Networks

GANs were proposed in 2014 [6] as generative models and consist of two Neural Networks (NNs), known as Generator (**G**) and Discriminator (**D**).

The Generator's goal is to synthesize data that are very similar (in quality and diversity) to the original training dataset distribution (p_{data}). Such process takes a random values vector from a simple prior distribution ($z \sim p(z)$), e.g., normal or uniform, as input, and then passes it through the NN's inner layers until output data ($G(z) \sim p_g(G(z))$) is obtained. Meanwhile, the Discriminator is in charge of classifying (discriminating) between two classes, the real set ($x \sim p_{data}(x)$) and the generated samples ($G(z) \sim p_g G(z)$). The input of the Discriminator is real or fake data, while its outputs are classification probabilities for both classes ($D(\cdot)$), e.g., values close to 0 for fake data and values close to 1 for real data.

The GANs training is described as a zero-sum game between two players, generator and discriminator, with opposite objectives. When the input is a real sample, the **D**'s goal is to make $D(x)$ near to 1. In the fake data scenario, the goal of **D** is to get $D(G(z))$ close to 0, while **G** will try to make near to 1 i.e. causing the discriminator to confuse the samples with the real set. The GANs cost function is stated as in Eq. (1):

$$\min_G \max_D \mathop{\mathbb{E}}_{x \sim p_{data}} [log(D(x))] + \mathop{\mathbb{E}}_{z \sim p_z} [log(1—D(G(z)))] \quad (1)$$

Both NNs are trained simultaneously. In every iteration, a gradient step is made to reduce the cost function of each network. The training process is repeated for a number of iterations until the generator learns to make synthetic data similar to the real distribution. The training process is represented in Fig. 1(b).

Since their inception, GANs have presented training instability problems [4]. This issue is caused by the fact that a balance between both NNs abilities must be maintained so as to avoid a disproportion that generates an irrecoverable learning gap. This instability creates common problems such as mode collapse and vanishing gradient. In the mode collapse problem, the generator captures only a small portion of the dataset distribution provided as input to the discriminator. This is not desirable since it is expected that a generative model reproduces the whole distribution of the data to achieve variability on the output samples. Meanwhile, the vanishing gradient occurs when either, the discriminator or the generator, becomes powerful enough to harm the balance required on the training, e.g. the discriminator becomes too strong that almost perfectly classifies both types of data, then the cost function is too small, the gradient does not flow with enough power to the generator, and the GANs progress stagnates.

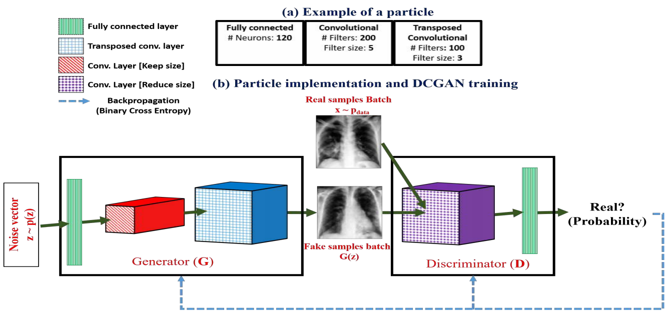

Fully connected layer
Transposed conv. layer
Conv. Layer [Keep size]
Conv. Layer [Reduce size]
Backpropagation (Binary Cross Entropy)
(a) Example of a particle
Fully connected # Neurons: 120
Convolutional # Filters: 200 Filter size: 5
Transposed Convolutional # Filters: 100 Filter size: 3
(b) Particle implementation and DCGAN training
Real samples Batch x ~ p_data
Noise vector z ~ p(z)
Generator (G)
Fake samples batch G(z)
Discriminator (D)
Real? (Probability)

Fig. 1. DCGAN representation. (a): Encoding: the parameters of each layer are explicitly written in the slots of the list. (b): Decoded particle and DCGAN training process.

The efforts to overcome the above mentioned problems include the creation of Deep Convolutional Generative Adversarial Networks [7]. This is a type of GAN, in which both NNs are Convolutional Neural Networks (CNN). This model is focused on the synthesis of images. The layers used in this architecture are transposed convolutional, convolutional and fully connected. Unlike the traditional CNN, pooling is not used. These networks have significantly improved the training stability, but they do not completely avoid it.

There are other approaches, using DCGAN as a basis, as the progressive growth presented in [8]. In that work the size of the generated images is gradually scaled by adding preset layers as the training goes on. With this, it is argued that learning by steps of increasingly complex resolutions stabilizes the GANs training, obtaining better results compared to learning the final resolution from the beginning. Another work focused on improving GANs training stability is related to the use of *Spectral Normalization* [9]. This technique normalizes the weight matrices in the discriminator by their corresponding spectral norm, which helps to control the Lipschitz constant of the discriminator. Lipschitz continuity is important to ensure the boundedness of the optimal discriminator.

B. Particle Swarm Optimization

Particle Swarm Optimization is a Swarm Intelligence algorithm, originally proposed in 1995 [10], that simulates the flying pattern of a bird flock. In PSO, every solution is called *particle* and the set of all the solutions is named *swarm*. Each particle has knowledge of its best historical configuration called $pBest$ and the best current solution in the swarm known as $gBest$. In each iteration, a particle updates its *velocity* so that its new *position* will be influenced to be closer to $pBest$ and $gBest$. The *velocity* and *position* of the *i-th* particle in its *j-th* dimension at iteration *t+1* is updated with Eq. (2) and Eq. (3), respectively,

$$\begin{aligned} v_{i,j}(t+1) = & w \times v_{i,j}(t) + c_p \times r_p \times (pBest_{i,j} — Pos_{i,j}(t)) \\ & + c_g \times r_g \times (gBest_j — Pos_{i,j}(t)) \end{aligned} \quad (2)$$

$$Pos_{i,j}(t+1) = Pos_{i,j}(t) + v_{i,j}(t+1) \quad (3)$$

where *v* is the particle's velocity; *Pos* is the particle's position; *w* is a constant called *inertia weight* that adjusts the influence of the previous velocity; c_p and c_g are constants that determine the influence of $pBest$ and $gBest$, respectively and therefore the exploration and exploitation capacity of the algorithm; finally r_p and r_g are random numbers within [0,1) that provide the search a stochastic behavior.

C. *Chest X-Ray images of pneumonia generated by COVID-19*

The sudden appearance in late 2019 of the Severe Acute Respiratory Syndrome Coronavirus (SARS-Cov-2), known as COVID-19, has caused an exponential growth of infected people around the world, escalating to the level of a pandemic and causing the loss of countless lives. One of the main complications caused by this disease is pneumonia with distinctive characteristics [11]. The search for these singularities is accomplished by Chest X-Ray (CXR) images of patients afflicted with pneumonia. However, these characteristics are hardly detectable with the naked eye by health experts. Therefore it is relevant to create DL models to help in the detection while reducing the diagnostic waiting times [12]. Nevertheless, DL models require a vast amount of data for proper learning. Hence, the use of GANs can help to increase the available image datasets. Therefore, this is the case of study in this paper.

III. Related Work

A. *GANs Neuroevolution*

Previous works regarding the neuroevolution of GANs are summarized in Table I, which includes the reference and the name given to the algorithm, the year of publication, the type of evolutionary algorithm used, the GAN's architecture (if it is evolvable, the encoding scheme used), the variation operators used, the fitness metric used as well as the type of offspring selection, the performance measures and the datasets used.

As it can be seen, most of the papers are focused on the use of GAs and Coevolution. The use of SI algorithms is a notable absence. Moreover, many of those works have a DCGAN-based architecture. The proposals that evolve its architecture use a list-based encoding scheme, in which there are declaratively blocks that contain the hyperparameters of each layer in the NNs.

Regarding evaluation metrics, there is no general consensus as different options have been adopted. However, the Frechet Inception Distance (FID) [13] has stood out in recent years as a *state-of-the-art* metric, specifically developed to evaluate the performance of GANs. FID uses the Inception-v3 CNN for the feature extraction of the real (x) and synthetic (g) images. Based on that, the Frechet distance between both multivariate Gaussian feature distributions using their estimated mean (μ) and covariance (Σ) is calculated. A lower value of this metric indicates a higher similarity between the two sets of images, being zero when they are equal. The FID's formula is showed in Eq. (4).

$$FID(x,g) = \|\mu_x — \mu_g\|_2^2 + Tr(\Sigma_x + \Sigma_g — 2(\Sigma_x \Sigma_g)^{\frac{1}{2}}) \quad (4)$$

Most of the GAN neuroevolution works have focused on the use of well-known image benchmarks (e.g. CIFAR-10, MNIST, CelebA) yet without exploring sets of real applications such as the use of biomedical images.

B. *GANs to synthesize CXR*

Previous works related to the use of GANs in the synthesis of chest X-ray images are condensed in Table II. It specifies the reference, the year of publication, the GAN's architecture, the resolution of the synthesized images in pixels (px.), the metrics used for performance evaluation and if they were used in the detection of pneumonia due to COVID-19.

Most of the studies were developed in 2020, with the goal of increasing the available sets of CXR for COVID-19 pneumonia for subsequent implementations. The other works focused on obtaining CXR from lungs afflicted with various pathologies. Furthermore, most architectures are DCGAN-based, proving the usefulness of this model in the synthesis of these biomedical images. On the other hand, all these architectures are handcrafted and they might not be easy to adapt to other lung pathologies or biomedical images.

The sizes of the synthetic images are different, but most of them have a resolution equal to or less than 256^2 px., indicating that such value is suitable to use in subsequent activities. Regarding the metrics to evaluate their results, they put aside the similarity between the sets of images to focus on improving the performance of specific tasks such as classification.

IV. Proposed Algorithm

The proposed algorithm, was inspired by three main foundations:

- *Training a DCGAN with progressive resolution growth favors training stability.*
- *Using PSO leads to get competitive results with a limited search time.*
- *DCGANs being composed by CNNs can exploit previous knowledge in CNN neuroevolution.*

Based on those ideas, DCGAN-PSO was developed. This algorithm evolves DCGAN-based architectures using a PSO,

TABLE I
GANs NEUROEVOLUTION WORK.

Reference and Name	**Year**	**EA type**	**Architecture (Encoding scheme)**	**Variation Operators**	**Fitness**	**Selection**	**Evaluation metrics**	**Datasets**
[14] **CA-GAN**	2018	Cultural algorithm	DCGAN-based	Mutation (discriminator weights)	Custom	Best individual	Inception score	CIFAR-10 STL-10
[15] **Pareto GAN**	2018	Genetic Alg.	Evolvable (List-based)	Crossover and Mutation	IGD (Pareto front)	Pareto dominance	IGD	Bi-objetive functions
[16] **Lipizzaner**	2018	Coevolution	DCGAN-based	Mutation (weights)	GAN loss	Spatial	Qualitative	MNIST CelebA
[17] **E-GAN**	2019	Genetic Alg.	DCGAN-based	Mutation (loss function)	Custom	Best individual	Inception and FID score	CIFAR-10 LSUN CelebA
[18] **Mustangs**	2019	Coevolution	DCGAN-based	Mutation (weights and loss)	GAN loss	Spatial	FID score	MNIST CelebA
[19] **CO-EGAN**	2019	Coevolution.	Evolvable (List-based)	Mutation (architecture)	FID and GAN loss	NEAT-based	FID score	MNIST F-MNIST CelebA
[20] **CA-GAN**	2019	Cultural algorithm	Evolvable (List-based)	Mutation (architecture, weights and loss)	GAN loss	Best individuals	Classification performance	Face images
[21] **GAN-GA**	2019	Genetic algorithm	DCGAN-based	Crossover and Mutation (both for generated images)	Discriminator score	Best individuals	GAN loss	MNIST
[22] **HEO-GAN**	2019	Genetic algorithm	DCGAN-based, WGAN and Vanilla GAN	Crossover and Mutation (weights)	FID	Best individuals	FID	MNIST CelebA

TABLE II
CXR SYNTHESIS WITH GANS

Reference	Year	Architecture	Image size (pixels)	Evaluation metrics	COVID-19
[23]	2018	DCGAN-based	256x256	Radiologists Classification performance	No
[24]	2018	DCGAN-based	128x128	Classification performance	No
[25]	2019	Own design	512x512	SWD MS-SSIMS FID	No
[26]	2020	DCGAN-based	256x256	Classification performance	Yes
[27]	2020	DCGAN-based	112x112	Radiologists Classification performance PCA	Yes
[28]	2020	DCGAN-based	512x512	Classification performance	Yes
[29]	2020	DCGAN-based	64x64	GAN loss	Yes
[30]	2020	Conditional GAN	446x446	Classification performances	Yes
[31]	2020	DCGAN-based	224x224	Classification performances	Yes
[32]	2021	DCGAN-based	128x128	Classification performances FID	No

taking advantage of its inherent qualities of faster convergence with a low number of evaluations compared to other EC algorithms [5] which allows a saving in the computational budget. Evolved architectures are trained by gradually increasing the resolution of the generated images. The main difference with respect to the approach in [8] (where the layers are preset by the authors) is that the inserted layers to scale the size of the generated images are obtained through the PSO search, i.e., evaluating the quality of the DCGANs using FID as fitness function and saving the trained weights of the best networks obtained, thus having the opportunity to improve the type of topology that would be obtained by manual methods.

The method to measure the difference between particles and the operators to update the *velocity* and *position*, which will be detailed later, are taken from [33], a *state-of-the-art* CNN neuroevolution algorithm, and modified to use them in the current approach to evolve DCGANs.

Algorithm 1 presents the pseudocode of DCGAN-PSO. The mechanism begins the iteration through the various resolutions, starting from 4^2 px. until gradually reaching 256^2 px., doubling it at each step **(line 1)**. For each resolution, the swarm will be initialized with particles that generate images in such resolution, as well as their corresponding *pBest* values and also the *gBest* is identified **(line 2)**. After that, for a predefined number of generations, the DCGAN represented in each particle will be trained with backpropagation and then its fitness will be evaluated **(lines 3-15)**. According to the fitness value already computed, the *pBest* of each particle and also the *gBest* of the whole swarm will be updated **(lines 16-21)**. Afterwards, the velocity and position of each particle will be updated **(lines 22-24)**. At the end of the iterations through all the resolutions, the trained *gBest* is returned **(line 28)**. The detailed explanation of each process in DCGAN-PSO is included in the following subsections.

A. Encoding-scheme

A list-based encoding scheme is used to represent the topology of the CNNs in the DCGAN model (*particle*). This encoding was selected because it is the one used by the operators taken and adapted from [33]. Each module in the list represents a layer and the sequence of layers in the list is used to implement both, the Generator and the Discriminator using a *mirror* image, i.e. the Generator uses the sequence from the list while the Discriminator is made up of the reverse sequence. The different layers used in the generator and their counterpart in the discriminator, as well as the hyperparameters that define them, are the following:

- **Transposed convolutional** and **Convolutional**. The first layer increases the size of the output of the previous layer by a factor of 2 in *G*; the second layer decreases the size of previous layer output in *D*: **Filter size** and **Number of filters**.

Algorithm 1 DCGAN-PSO

```
Input: Training CXR dataset of each resolution used, N° generations per resolution, N° epochs per particle training, Swarm size, $C_g$, resolutions list.
Output: Trained DCGAN of CXR images of pneumonia caused by COVID-19 in $256^2$ pixels resolution (gBest).
1: for resolution in resolutions list do
2:     Initialization: Swarm ← Initialize Swarm (resolution), pBests, gBest.
3:     for N° generations per resolution do
4:         for particle in swarm do
5:             if resolution = $4^2$ pixels then
6:                 Network ← DCGAN particle architecture
7:             else
8:                 Network ← DCGAN previous resolution pBest architecture
9:                 Network weights ← Previous resolution pBest weights
10:                Network ← Add DCGAN particle architecture at the end of the Network
11:            end if
12:            for N° epochs per particle training do
13:                Train Network with CXR dataset of its corresponding resolution
14:            end for
15:            particle fitness ← FID (Network)
16:            if particle fitness < particle pBest fitness then
17:                particle pBest ← particle
18:                particle pBest weights ← particle weights
19:            end if
20:        end for
21:        gBest ← pBest with lowest fitness
22:        for particle in swarm do
23:            particle velocity ← UpdateVelocity (particle)
24:            particle ← UpdateParticle (particle velocity)
25:        end for
26:    end for
27: end for
28: RETURN gBest
```

- **Convolutional**. This layer keeps the size of the previous layer output in G; does not apply to D in order to reduce computational costs and to avoid an over-complexity that could lead to overfitting: **Filter size** and **Number of filters**.
- **Fully connected layer**. This layer processes and resizes the random noise input in G and processes the high-level features (obtained by the convolutional part of the CNN) to perform the classification in D: **Number of neurons**.

The number of layers of each particle is limited by the minimum and maximum number of layers allowed by each type of layer, these ranges are found in Table III. A graphical example of a particle and its respective decoded DCGAN can be seen in Fig. 1.

When the swarm is initialized (*Initialize Swarm()*), the number of layers in each particle is selected at random with uniform distribution (to avoid bias) as well as the hyperparameters that are associated to each layer. The ranges of the parameter values are resolution-dependent (see Table III). These parameter values were selected through a brief experimental design considering the high computational cost of each run and the infrastructure available to carry out the experiments.

B. Difference between particles

The first step prior to updating the particles is to measure the *difference* between two particles, P_1 and P_2, since these may have different sizes and design patterns. This difference is represented in a difference vector $(P_1 - P_2)$. The difference only takes into consideration the type of layer of each particle. If two layers in the same position of both particles are the same type then the difference is zero, regardless their hyperparameters. This *zero-difference* indicates that the layer will remain when the particle architecture is updated. If both layers are of different type, P_1 will have priority and its layer will be preserved (including its hyperparameters). If P_1 have fewer layers that P_2 then **-1** will be added at the end of the difference vector to denote that these extra layers in P_2 should be eliminated when updating the particle. In the opposite case, if P_1 has more layers than P_2 then indicators **+L** will be added that represent that layers of type L will be added in those positions when updating the particle (with randomly chosen hyperparameters). Fig. 2 shows examples to measure the difference between particles.

TABLE III
DCGAN-PSO PARAMETER VALUES USED IN THE EXPERIMENTS.

Parameter	Value
Particle Swarm Optimization	
Swarm size	15
N° generations per resolution	10
C_g	0.5
Resolutions list	[4^2 px., 8^2 px., 16^2 px., 32^2 px., 64^2 px., 128^2 px., 256^2 px.]
Ranges of particle parameters	
N° Fully connected layers	[1,1] (4^2 px.) ; [0,0] (otherwise)
N° Convolutional layers	[0,2]
N° Transposed convolutional layers	[0,0] (4^2 px.) ; [1,1] (otherwise)
Filter size	[2,5]
N° filters	4^2:64^2 px. = [1,256]; 128^2 px. = [1,64]; 256^2 px. = [1,32]
N° neurons	[1,300]
DCGAN training	
N° epochs per particle training	200
Batch size	4^2:128^2 px. = 16; 256^2 px. = 14
Optimizer	Adam [39]
Learning rate (**G** and **D**)	2×10^{-4}
β_1, β_2 (optimizer)	0.5 , 0.999
LeakyReLU slope (**D**'s activations)	0.2
Weight's initializer (**G** and **D**)	$\mathcal{N}(0, 0.02)$
Noise distribution (p_z): $\mathbb{R}^{100x1}$	$\mathcal{N}(0, 1)$

C. Velocity and Position Operators

The *velocity* update of particle (P), is computed by comparing it with its *pBest* and the *gBest*. Thus, the differences *(pBest — P)* and *(gBest — P)* are required. With both difference vectors the velocity vector is calculated. For this operation the parameter C_g, called *decision factor*, is needed, while **r** is a random number from a uniform distribution within [0,1). For each position of the difference vectors, if $r \leq C_g$, the velocity operator will take the layer from the difference (*gBest* — P), and from (*pBest* — P) otherwise. Therefore, C_g will control the convergence of the particle towards the global best. This operation is represented as *UpdateVelocity()*.

The operator to update the position of the particles i.e. modifying their DCGAN architecture, is *UpdateParticle()*. This operator takes the particle´s *velocity* to modify the pertinent layers. Layers are added or removed from the particle architecture according to its *velocity*. The Velocity and Position operators are depicted in Fig. 3.

When the particle is updated, there must be a record of the transposed convolutional layers, when there are extra layers, they are replaced by convolutional layers that keeps the size. Also, only one fully connected layer will be available at the beginning of the architecture. This layer will be evolved only when the resolution is 4^2 px., as the DCGAN architecture only uses this layer at the beginning to process the noise input in G and perform the classification in D. Since the fully connected layer will always be in the first position of the particles there will always be *zero-difference* in that position. Therefore, to avoid stagnation in evolution, the number of neurons in these layers will always be randomly changed in each update.

D. Progressive Particle Growth

The training of the various DCGAN was carried out progressively, starting with a resolution of 4^2 to 256^2 px. in steps that doubled the resolution. The final resolution was chosen because it is one that equals or exceeds the resolution adopted by previous GANs for CXR (Table II).

The swarm is initialized with particles that generate images with a resolution of 4^2 px. These particles are trained with backpropagation for a number of epochs with the dataset of

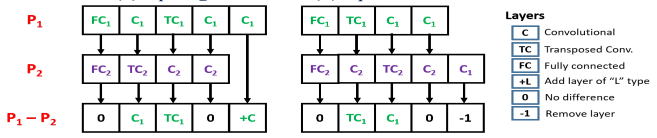


Fig. 2. Differences measurement between particles inspired by [33]. (a): P_1 has more layers that P_2. (b): P_1 has fewer layers that P_2.

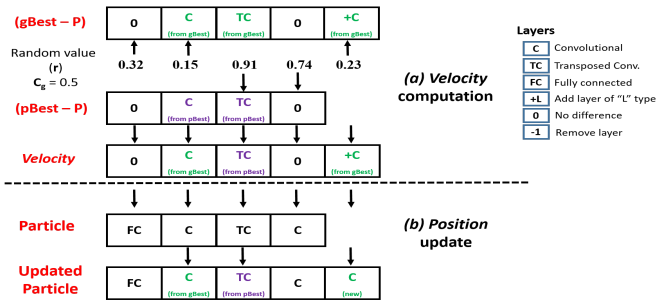


Fig. 3. Example of *velocity* and *position* operators inspired by [33]. (a): *Velocity* update. (b): Particle position updated using its *velocity*.

CXR images with the same resolution. After training, their quality is evaluated by FID, which acts as a fitness function. Using such fitness values, the *pBest* of each particle and also the *gBest* of the whole swarm are updated. If an architecture is selected as *pBest*, the weights of the trained DCGAN are saved. The particle updates continue for a certain number of generations. At the end, the *pBest* of each particle is taken as the basis for the next resolution, fixing that part of the architecture, but still training it, and only adding and modifying the layers ahead of this part to obtain the images of the next resolution. When the DCGAN of a particle in the swarm with the increased resolution is implemented, the weights for the layers belonging to the previous *pBest* are loaded. To avoid reducing the learning rate to smooth the training of the previous pre-trained layers with respect to the newly initialized layers, Weigth Normalization (WN) technique is used in the generator. WN has proven to stabilize training in GANs in addition to avoid exploding gradients when learning rates are relatively high for certain layers [34]. Meanwhile, in the discriminator, Spectral Normalization is used to stabilize the training. Fig. 4 represents the progressive growth approach. The whole DCGAN-PSO process repeats until the final resolution is reached.

V. Experimental results

A. CXR Images Datasets

For the generation of CXR images of COVID-19 pneumonia as a case study, public image datasets available at [35], [36] and [37] were used. From these sets, 994 CXR images were selected for their quality, although only 600 were used for training. Three pre-processing steps were considered: **(1)**: grayscale conversion (one channel); **(2)**: image resizing to the different used resolutions (one dataset per resolution); **(3)**: histogram equalization to increase contrast.

B. Technical implementation details

The design rules regarding the Batch Normalization and Activation layers for DCGANs defined in [7] were used when DCGANs were implemented. Details about the optimizer and initialization of the networks can be found in Table III.

Python 3.6.9 programming language with the PyTorch framework [38] were used for the implementation on the free access online platform Google Colaboratory[1] to have access to GPUs for the NNs training.

C. CXR Synthesis Works Comparison

For comparison purposes, the eight approaches using DC-GANs reported in the literature review of CXR synthesis (Table II) were implemented and trained five independent times, with the same dataset used for DCGAN-PSO. Such approaches were coded based on the specifications provided by their authors in their corresponding references.

D. Performance Evaluation

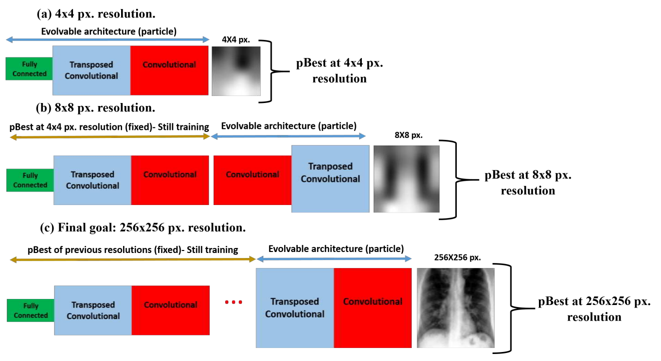
1) Qualitative evaluation: Fig. 5 shows samples of the real image dataset and examples of those obtained by DCGAN-PSO.

2) Fitness evolution: The FID values of the best solution obtained by DCGAN-PSO in ten independent runs (this number was chosen due to the significant computational cost of each run i.e. a run takes a week with the available resources) along the different image resolutions are shown in Fig. 6.

3) FID evaluation: The FID was evaluated using the complete CXR image dataset (994 images) and a sample of the same size taken from the images synthesized by the trained generators of the DCGAN-PSO runs and by the compared approaches. The FID statistical results obtained are summarized in Table IV. The results of the randomized design and training of fifteen DCGANs (with the same parameters for network depth and layers used by DCGAN-PSO) for 256^2 px. images were also included. The 95%-confidence Wilcoxon rank-sum test was used to validate the results.

VI. Discussion

The results in Table IV indicate that DCGAN-PSO was able to outfperform the nine compared approaches based on the



(a) 4x4 px. resolution.
Evolvable architecture (particle)
4X4 px.
Fully Connected
Transposed Convolutional
Convolutional
pBest at 4x4 px. resolution
(b) 8x8 px. resolution.
pBest at 4x4 px. resolution (fixed)- Still training
Evolvable architecture (particle)
8X8 px.
Tranposed Convolutional
pBest at 8x8 px. resolution
(c) Final goal: 256x256 px. resolution.
pBest of previous resolutions (fixed)- Still training
256X256 px.
pBest at 256x256 px. resolution

Fig. 4. Progressive growth of a DCGAN represented in a particle. Here the growth of the generator is represented but the equivalent procedure is used in the discriminator. (a): At the beginning particles generate CXR images in 4^2 px. and *pBests* are selected. (b): When the resolution increases, the architecture and the trained weights of the particle's *pBest* in the previous resolution are used. *pBest*'s layers remains fixed (not evolved), but still training, while a new population is generated and evolved with particles that will be concatenated to the already known *pBest* part. (c): The process stops when 256^2 px. resolution is reached.

[1]https://colab.research.google.com/

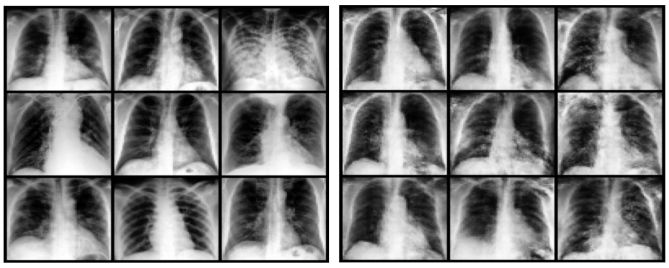


Fig. 5. 256^2 px. image samples. Left: Real CXR. Right: DCGAN-PSO generated CXR.

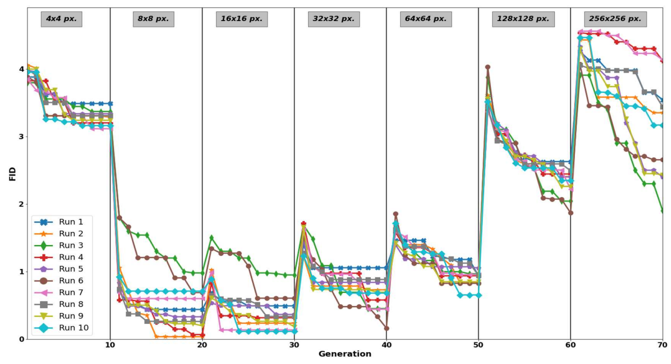


Fig. 6. *gBest* FID value obtained by DCGAN-PSO along generations and resolutions (10 runs).

TABLE IV
FID VALUES OBTAINED BY DCGAN-PSO AND THE COMPARED APPROACHES. (+) MEANS THAT DCGAN-PSO OUTPERFORMED THE COMPARED APPROACH IN THE CORRESPONDING ROW BASED ON THE 95%-CONFIDENCE WILCOXON RANK-SUM TEST

Model	**Average & St. Deviation**	**p-value**	**Wilcoxon rank-sum test**
Salehinejad, et al. 2018 [23]	5.308 ± 0.842	0.0048	(+)
Madani, et al. 2018 [24]	5.372 ± 1.030	0.0021	(+)
Khalifa, et al. 2020 [26]	4.254 ± 0.165	0.00705	(+)
Waheed, et al. 2020 [27]	4.296 ± 0.533	0.01	(+)
Loey, et al. 2020 [28]	5.846 ± 0.855	0.00219	(+)
Shams, et al. 2020 [29]	5.938 ± 0.601	0.00219	(+)
Zulkifley, et al. 2020 [31]	4.492 ± 0.633	0.0101	(+)
Kora Venu and Ravula 2021 [32]	5.576 ± 0.482	0.00219	(+)
Random DCGANs	9.851 ± 2.028	$3.178e^{-5}$	(+)
DCGAN-PSO	$\mathbf{3.052 \pm 0.773}$	——-	——-

FID value obtained and such performance was validated by the 95%-confidence Wilcoxon rank-sum test.

Qualitatively speaking, the images generated by DCGAN-PSO shown in Fig. 5 display a similar visual quality (morphological) with respect to the images belonging to the real set. Moreover, the visual diversity obtained can be an indicator of the absence of mode collapse. Just minor errors (blank areas) are observed in the generated images.

An interesting convergence behavior was observed in Fig. 6, where, regardless the resolution, DCGAN-PSO was able to improve the fitness value in just a few generations. It is important to remark that the fitness decreasing observed between resolution increments is due to the fact that FID is not a deterministic measure. FID uses a CNN to extract high-level features of the images and later makes a statistical comparison among them. Those high-level features may vary depending on multiple factors, such as the resolution of the image, the greater detail which allows them to have finer motifs and increases the complexity to be learned by DCGAN. At the end of the graph (256^2 px. resolution), the values obtained are those reported in Table IV which showed to be better (lower FID values suggest a reduction in training instability problems) than those of the compared approaches.

As part of a preliminary experimentation, a CNN was implemented to classify between two classes of CXR, COVID-19 and Non-COVID-19 (composed of healthy patients and pneumonia not caused by COVID). Five training processes were carried out with 600 images of COVID-19 class and 1200 images of Non-COVID-19 class. In addition, five training processes were carried out by adding 600 synthetic images to the COVID-19 class to balance the dataset. The synthetic images were obtained from the generator with the FID value in the median of the results (2.85). This experimentation shows an improvement in the average classification accuracy of the COVID-19 class when the dataset is balanced, i.e. 72.3 ± 4.3 with unbalanced data and 80.4 ± 3.12 with a balanced dataset. Further experimentation is required, which is one of the future goals in the development of this research.

The overall above presented promising results suggest that the search promoted by PSO has the potential to improve DCGANs designs when applied to biomedical image synthesis.

VII. Conclusions and Future Work

In this work the DCGAN-PSO algorithm was presented. The novelty of this Swarm Intelligence algorithm for Generative Adversarial Networks (GANs) neuroevolution based on PSO is (1) the usage of a progressive growth approach to perform the search and training of the DCGAN architectures and (2) the application domain, i.e., biomedical images generation (previously unexplored area, to the best of the authors' knowledge, by GANs designed by neuroevolution), specifically Chest X-Ray images of pneumonia caused by COVID-19. The obtained results showed that the quality of the synthetic CXR images from the evolution of DCGANs were better than those synthesized results provided by handcrafted architectures from previous works, measured through the Frechet Inception Distance. Furthermore, the expected fast convergence to competitive results by PSO was confirmed in this particular type of search space.

For future work, the proposed algorithm can be expanded through modifying the PSO operators, in such a way that they not only take into account the evolution of the layers as a whole but also evolve their inner hyperparameters. Finally, the experiments on classification will be extended.

Acknowledgment

The first author, with CVU number 1001447, acknowledges support from the Mexican National Council of Science and Technology (CONACyT) through a scholarship to pursue graduate studies at the University of Veracruz.

References

[1] Lu, L., Zheng, Y., Carneiro, G., and Yang, L., "Deep learning and convolutional neural networks for medical image computing," in Advances in Computer Vision and Pattern Recognition, vol. 10, 2017, pp. 978-3.

[2] Suetens, P., "Fundamentals of medical imaging," 2nd ed., Cambridge university press, 2017.

[3] Yi, Xin; Walia, Ekta, and Babyn, Paul, "Generative adversarial network in medical imaging: A review," in Medical image analysis, vol. 58, 2019, p. 101552.

[4] Costa, V., Lourenço, N., Correia, J., and Machado, P., "Neuroevolution of generative adversarial networks," in Deep Neural Evolution, Springer, Singapore, 2020, pp. 293-322.

[5] Piotrowski, A. P., Napiorkowski, M. J., Napiorkowski, J. J., and Rowinski, P. M., "Swarm intelligence and evolutionary algorithms: Performance versus speed," in Information Sciences, vol. 384, 2017, pp. 34-85.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative Adversarial Nets," in Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.

[7] Radford, Alec, Luke Metz, and Soumith Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2015.

[8] Karras, T., Aila, T., Laine, S., and Lehtinen, J., "Progressive growing of GANs for improved quality, stability, and variation," in 6th International Conference on Learning Representations, ICLR 2018, Vancouver, Conference Track Proceedings, 2018.

[9] Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y., "Spectral normalization for generative adversarial networks," arXiv preprint arXiv:1802.05957, 2018.

[10] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proceedings of IEEE International Conference on Neural Networks (Perth, Australia) (IEEE Service Center, Piscataway, NJ), 1995 , pp. 1942–1948.

[11] Huang, C. et al., "Clinical features of patients infected with 2019 novel coronavirus in Wuhan, China," The Lancet, vol. 395(10223), 2020, pp. 497–506.

[12] Shoeibi, A., Khodatars, M., Alizadehsani, R., Ghassemi, N., Jafari, M., Moridian, P., and Srinivasan, D, "Automated detection and forecasting of covid-19 using deep learning techniques: A review", arXiv preprint arXiv:2007.10785, 2020.

[13] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S., "Gans trained by a two time-scale update rule converge to a local Nash equilibrium," in Advances in neural information processing systems, 2017, pp. 6626-6637.

[14] Ni, Y., Song, D., Zhang, X., Wu, H. and Liao, L., "Cagan: consistent adversarial training enhanced gans," in IJCAI, 2018, pp. 2588–2594.

[15] Garciarena, U., Santana, R. and Mendiburu, A., "Evolved gans for generating pareto set approximations", in Proceedings of the Genetic and Evolutionary Computation Conference, 2018, pp. 434–441.

[16] Al-Dujaili, A., Schmiedlechner, T., and O'Reilly, U. M., "Towards distributed coevolutionary gans," arXiv preprint arXiv:1807.08194, 2018.

[17] Wang, C., Xu, C., Yao, X. and Tao, D., "Evolutionary generative adversarial networks," in IEEE Transactions on Evolutionary Computation vol.23(6), 2019, pp.921–934.

[18] Toutouh, J., Hemberg, E., and O'Reilly, U.M., "Spatial evolutionary generative adversarial networks," in Proceedings of the Genetic and Evolutionary Computation Conference, 2019, pp. 472–480.

[19] Costa, V., Lourenco, N., Correia, J., and Machado, P., "Coegan: evaluating the coevolution effect in generative adversarial networks," in Proceedings of the Genetic and Evolutionary Computation Conference, 2019, pp. 374–382.

[20] Mehta, K., Kobti, Z., Pfaff, K. and Fox, S., "Data augmentation using ca evolved gans," in 2019 IEEE Symposium on Computers and Communications (ISCC), 2019, pp. 1087–1092.

[21] Cho, Hwi-Yeon, and Yong-Hyuk Kim, "Stabilized Training of Generative Adversarial Networks by a Genetic Algorithm," in Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 51–52.

[22] Korde, Charudatta G et al., "Training of Generative Adversarial Networks with Hybrid Evolutionary Optimization Technique," in 2019 IEEE 16th India Council International Conference (INDICON), IEEE, 2019, pp. 1–4.

[23] Salehinejad, H., Valaee, S., Dowdell, T., Colak, E., and Barfett, J., "Generalization of deep neural networks for chest pathology classification in x-rays using generative adversarial networks," in 2018 IEEE International Conference on Acoustics, Speechand Signal Processing (ICASSP), 2018, pp. 990–994.

[24] Madani, A., Moradi, M., Karargyris, A., and Syeda-Mahmood, T., "Semi-supervised learning with generative adversarial networks for chest x-ray classification with ability of data domain adaptation," in 2018 IEEE 15th International Symposiumon Biomedical Imaging (ISBI 2018), 2018, pp. 1038–1042.

[25] Zhang, Tianyang et al., "SkrGAN: Sketching-rendering unconditional generative adversarial networks for medical image synthesis," International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2019, pp. 777–785.

[26] Khalifa, N.E.M., Taha, M.H.N., Hassanien, A.E., and Elghamrawy, S., "Detection of coronavirus (covid-19) associated pneumonia based on generative adversarial networks and a fine-tuned deep transfer learning model using chest x-ray dataset," arXiv preprint arXiv:2004.01184, 2020.

[27] Waheed, A., Goyal, M., Gupta, D., Khanna, A., Al-Turjman, F., and Pinheiro, P.R., "Covidgan: Data augmentation using auxiliary classifier gan for improved covid-19 detection," IEEE Access, vol.8, 2020, pp. 91916–91923.

[28] Loey, M., Smarandache, F., and Khalifa, N.E.M., "Within the lack of chest covid-19 x-ray dataset: a novel detection model based on gan and deep transfer learning," Symmetry, vol.12(4), 2020, p. 651.

[29] Shams, MY et al., "Why Are Generative Adversarial Networks Vital for Deep Neural Networks? A Case Study on COVID-19 Chest X-Ray Images," Big Data Analytics and Artificial Intelligence Against COVID-19: Innovation Vision and Approach, Springer, 2020, pp. 147–162.

[30] Karakanis, Stefanos, and Georgios Leontidis, "Lightweight deep learning models for detecting COVID-19 from chest X-ray images," in Computers in Biology and Medicine, vol.130, 2020, p. 104181.

[31] Zulkifley, M. A., Abdani, S. R., and Zulkifley, N. H., "COVID-19 Screening Using a Lightweight Convolutional Neural Network with Generative Adversarial Network Data Augmentation," Symmetry, vol. 12(9), 2020, p. 1530.

[32] Kora Venu, Sagar and Sridhar Ravula, "Evaluation of Deep Convolutional Generative Adversarial Networks for Data Augmentation of Chest X-ray Images," in Future Internet, vol. 13(1), 2021, p. 8.

[33] Junior, F. E. F., and Yen, G. G., "Particle swarm optimization of deep neural networks architectures for image classification," in Swarm and Evolutionary Computation, vol. 49, 2019, pp. 62-74.

[34] Xiang, S., and Li, H., "On the effects of batch and weight normalization in generative adversarial networks," arXiv preprint arXiv:1704.03971, 2017.

[35] Chowdhury, M. E. et al., "Can AI help in screening viral and COVID-19 pneumonia?," arXiv preprint arXiv:2003.13145, 2020.

[36] Cohen, J.P., Morrison, P., Dao, L., Roth, K., Duong, T. Q., and Ghassemi, M., "Covid-19 image data collection: prospective predictions are the future," arXiv preprint arXiv 2006.11988, https://github.com/ieee8023/covid-chestxray-dataset7, 2020.

[37] Tabik, S., et al., "COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images," in IEEE Journal of Biomedical and Health Informatics, vol.24(12), 2020, pp. 3595–3605.

[38] Paszke, A. et al, "Automatic differentiation in PyTorch," 2017.

[39] Kingma, D. P., and Ba, J., "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

4 17th International Symposium on Medical Information Processing and Analysis (SIPAIM) Paper

Evolution of conditional-GANs for the synthesis of chest X-ray images.

Juan-Antonio Rodríguez-de-la-Cruz[a], Héctor-Gabriel Acosta-Mesa[a], Efrén Mezura-Montes[a], Fernando Arámbula Cosío[b], Borís Escalante-Ramírez[c], and Jimena Olveres Montiel[c]

[a]Instituto de Investigaciones en Inteligencia Artificial, Universidad de Veracruz, Campus Sur, Paseo 112, Col. Nueva Xalapa, C.P. 91097 Xalapa, Veracruz, México
[b]Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Circuito Escolar S/N, Ciudad Universitaria, Cd. Mx., C.P. 04510, México
[c]Facultad de Ingeniería, Universidad Nacional Autónoma de México, Universidad 3000, Ciudad Universitaria, Coyoacán, Cd. Mx., C.P. 04510, México

ABSTRACT

Deep learning (DL) is now widely used to perform tasks involving the analysis of biomedical imaging. However, the small amounts available of annotated examples of these types of images make it difficult to use DL-based systems, since large amounts of data are required for adequate generalization and performance. For this reason, in recent years, Generative Adversarial Networks (GANs) have been used to obtain synthetic images that artificially increase the amount available. Despite this, the usual training instability in GANs, in addition to their empirical design, does not always allow for high-quality results. Through the neuroevolution of GANs it has been possible to reduce these problems, but many of these works use benchmark datasets with thousands of images, a scenario that does not reflect the real conditions of cases in which it is necessary to increase the data due to the limited amount available. In this work is presented cDCGAN-PSO, an algorithm for the neuroevolution of conditional-GANs (cGAN) that adapts the concepts of a previously reported neuroevolutionary algorithm called DCGAN-PSO, which was focused on the design and training of DCGANs through the use of Particle Swarm Optimization, a Swarm Intelligence algorithm that uses a set of potential solutions to approximate a highly competitive solution. The evolved cGANs allows the synthesis of three classes of chest X-ray images and they were trained with only 600 images of each class. The synthetic images obtained of each class show good similarity with real chest X-ray images.

Keywords: GANs, Neuroevolution, Biomedical Imaging Synthesis, Chest X-Ray Images, conditional-GAN, DCGAN-PSO

1. INTRODUCTION

Systems based on Deep Learning (DL) used in multiple tasks that involve the use of biomedical images have made it possible to obtain the best performances in the *state of the art*.[1] Activities such as classification, segmentation of areas of interest, noise elimination and registration, to name a few, have benefited from the use of these complex systems, which in turn support the biomedical area. However, the use of DL-based models implies the need for high amounts of data for the purpose of adequate training and generalization, since the very complexity of these is what can cause problems, e.g. overfitting, if there is a very limited amount of data. This problem

Further author information: (Send correspondence to J-A.R-C.)
J-A.R-C.: E-mail: juanantonio_2604@hotmail.es
H-G.A-M.: E-mail: heacosta@uv.mx
E.M-M.: E-mail: emezura@uv.mx
F.A.C.: E-mail: fernando.arambula@iimas.unam.mx
B.E-R.: E-mail: boris@unam.mx
J.O.M: E-mail: jolveres@cecav.unam.mx

is present in biomedical images, where their limited quantity is due to factors such as data privacy, the risk of some tests, e.g. exposure to radiation, or even the prohibitive cost of some tests.[2]

The use in recent years of Generative Adversarial Networks (GANs), for the synthesis of images with a high similarity to the real ones, has allowed to increase the amounts available to train biomedical systems.[3] However, GANs usually have a training instability that ends up affecting the quality of the synthetic images obtained. Therefore, multiple neuroevolution systems, an area of Evolutionary Computation (EC) focused on improving and automating the design and/or training mechanisms of Neural Networks (NNs), focused on GANs have been developed, which has allowed progress to overcome or reduce classic GANs problems.[4] Most of these algorithms have been tested using popular benchmark datasets with thousands of images, none of them being tested in real applications or scenarios with little amount of data available, being DCGAN-PSO, to the best of the authors' knowledge, the first algorithm to be used not only with a low amount of data but also in the area of biomedical images, a field that can be highly benefited with the increase of data through GANs. However, by using DCGAN-PSO, only GANs that can synthesize a single class of images are obtained. In this paper, that proposal is extended and adapted to use conditional-GANs, a variant that allows the synthesis of multiple classes of images conditioned by the user, which allows fewer algorithm executions to be carried out to obtain different classes of synthetic images. Like the original version of the algorithm, Chest X-Ray (CXR) images were used as a case study, handling three different classes: pneumonia due to COVID-19, non-COVID-19 pneumonia and healthy.

To validate the results obtained using this new version, the quality was evaluated using FID and compared against the original version in the three classes of CXR images.

The rest of the paper is organized as follows: Section 2 contains the introduction to the key concepts related to GANs and their neuroevolution and CXR images, as well as related works. Section 3 contains the presentation and novelty of our proposal. Section 4 defines the experimentation performed to evaluate our version, as well as its results, while Section 5 contains the corresponding discussion. Section 6 presents the conclusions obtained and the proposals for future work.

2. BACKGROUND AND RELATED WORK

GANs[5] are Deep Learning models that belong to the set of generative models, a branch of unsupervised learning algorithms in charge of mapping how the data was generated. The training of a GAN is described as a zero-sum game (also called minimax) between two players with opposite objectives; these are two Neural Networks; the Generator and the Discriminator. Taking a vector of random noise sampled from prior distributions, e.g. normal or uniform, $(z \sim p_z(z))$ as input, called *latent vector*, the generator outputs samples from a more complex distribution $(G(z))$ whose goal is to be equal to the distribution of the real dataset. Meanwhile the discriminator has the task of distinguishing between the real samples$(x \sim p_{data}(x))$ and the generated samples $(G(z) \sim p_g(G(z)))$. In the case of real data, the goal of the discriminator output $(D(\cdot))$ is to be near to 1. In the fake data scenario, the discriminator output's goal is to be close to 0 meanwhile the generator will try to make near to 1, i.e., fool the discriminator to classify his creations as real.

The training of both networks is carried out by means of independent backpropagation optimizations. The generator is optimized using the discriminator's predictions about its creations, and once this step is completed, the discriminator is trained using the synthetic data obtained by the generator. This optimization is carried out using the training cost function of GANs, called Minimax loss, which is reflected by the following equation:

$$\min_G \max_D \mathop{\mathbb{E}}_{x \sim p_{data}} [log(D(x))] + \mathop{\mathbb{E}}_{z \sim p_z} [log(1 — D(G(z)))] \tag{1}$$

The networks are trained through multiple cycles of the two previously mentioned training steps, encouraging both models towards continuous improvement and adaptation. For every iteration a gradient step with backpropagation is made to reduce the cost function of each network, optimizing their internal weights.

Conditional-GAN (cGAN)[6] is a variant of the classic GAN, which uses class labels to condition the class of the generated image from among all the classes available in the training set and thus obtain greater control over the synthesized images, contrary to the original GAN, where the synthesis of different classes from the same training set is completely random. The cost function of the cGAN is the same as the original GAN with the addition of the class labels (y):

$$\min_G \max_D \mathop{\mathbb{E}}_{x \sim p_{data}} [log(D(x|y))] + \mathop{\mathbb{E}}_{z \sim p_z} [log(1 — D(G(z|y)))] \qquad (2)$$

In Fig. 1, the representation of the general structure of cGAN as well as its training process is shown.

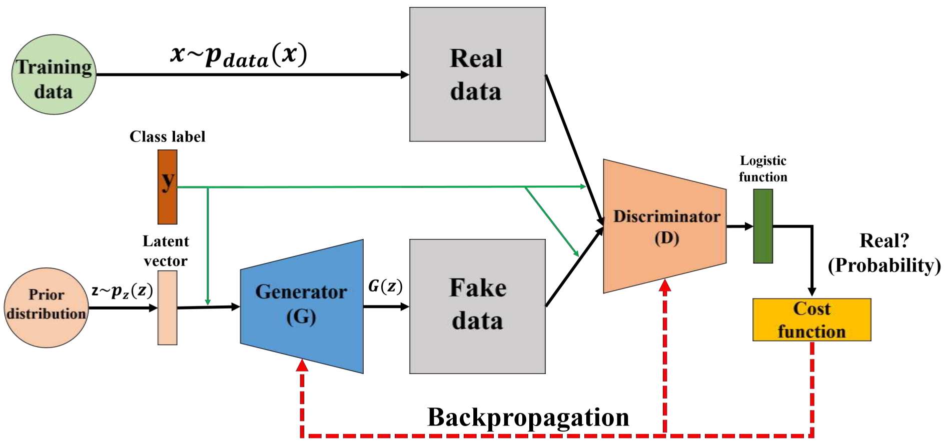


Figure 1. **General structure of a cGAN.** The green arrow represents the concatenation of the y label to the GAN's inputs.

Multiple GANs have been developed focused on the synthesis of various biomedical images, with Chest X-Ray (CXR) images being highlighted in the last year due to the need to increase the available quantities that represent COVID-19 pneumonia[7–10] to train systems that support in the diagnosis of this disease. However, other works have mainly used CXR images of the pneumonia (bacterial and viral) and healthy classes.[11–15] Nonetheless, all these works use empirically hand-designed networks, which are usually not generalizable to other applications or types of images. In addition, GANs commonly present training instability.[4]

GANs training is complicated because there must be a balance between the skills of the generator and the discriminator. If there is a supremacy from one of the networks, training instability problems may cause the following problems: **(1)**Mode collapse: The situation in which the generator can only synthesize a small subset of data of the complete distribution since the training did not allow to generalize the richness of variants of the original distribution; **(2)** Vanishing gradient: Originated when the discriminator or the generator becomes powerful enough to cause an irreversible imbalance in training that does not make possible to the opposite network to improve its performances, thus causing a stalemate, resulting in poor visual quality synthetic results.

The use of neuroevolution to carry out the design and training of GANs has been addressed in multiple works recently.[4] Evolutionary Computation takes inspiration on the mechanism found in nature to evolve a population of potential solutions on the production of better outcomes for a given problem, being Neuroevolution the branch of EC in charge of the evolution of neural networks.[16] Among the works in GANs neuroevolution, DCGAN-PSO[17] stands out, which, unlike the previous works, uses a set of biomedical images as a real case of application. This algorithm uses a low number of images (compared to benchmark datasets with thousands of images used previously) for the evolution of DCGANs,[18] a variant of GAN focused on image synthesis. DCGAN-PSO can be extended and improved through the use of conditional-DCGANs to handle the creation of multiple classes of images through the use of a single evolved DCGAN, without the need to perform as many runs as classes are needed, as is necessary in DCGAN-PSO.

3. OUR APPROACH

Our approach to using conditional-DCGANs as an architecture to evolve in a neuroevolution algorithm is presented in this section. For this, we use the previously introduced method called DCGAN-PSO, adapting it to our proposal.

3.1 DCGAN-PSO

DCGAN-PSO[17] presents the use of a neuroevolution algorithm for the search for architectures and training of DCGANs, a variant of the GANs in which both networks that compose it are Convolutional Neural Networks (CNN), which allows obtaining networks capable of generating a single class of CXR images with a high similarity in quality and diversity with respect to the real set, an indicator of a more stable training.

DCGAN-PSO is based on Pro-GAN,[19] an approach carried out in the progressive growth of GANs, increasing the resolution of the images generated through the addition of layers, pre-established by the researchers, at specific moments of the GANs training. In addition, the variation operators of the CNN neuroevolution algorithm called psoCNN[20] are used.

The representation of the evolved DCGANs is by means of lists, where each slot represents a layer and the sequence of these the network architecture. Each of these architectures represents *particles* of a *swarm*, which when interacting can generate complex search behaviors.

For the progressive growth of the networks, a *swarm* of *particles* is initialized (random architectures) where each particle represents a DCGAN that can synthesize images at 4x4 pixels resolution. Each particle is then modified having the influence of the best architecture found so far by that *particle* (*pBest*) and the best obtained by the entire *swarm* (*gBest*). Through these better architectures, variation operators update the particles in order to find better networks. Each potential network is trained and evaluated by means of the fitness function, if the network results to improve the performance of the particle's *pBest*, its architecture and its weights are saved. At the end of a number of cycles (*generations*) determined for a resolution, the best architectures of each particle are fixed and new layers are added from these that allow doubling the resolution of the images generated. The new added layers are evolved by the same number of cycles as the previous resolution. These steps are repeated, doubling the resolution of the images until a final resolution of 256x256 pixels is obtained. Fig. 2 shows the general scheme of progressive growth carried out by this algorithm.

The fitness function used is the Frechet Inception Distance (FID).[21] FID uses the pre-trained Inception-v3 CNN[22] for the feature extraction of the real (x) and synthetic (g) images, thus obtaining a feature-vector of 2048 numerical values. This feature space is interpreted as a continuous multivariate Gaussian distribution.

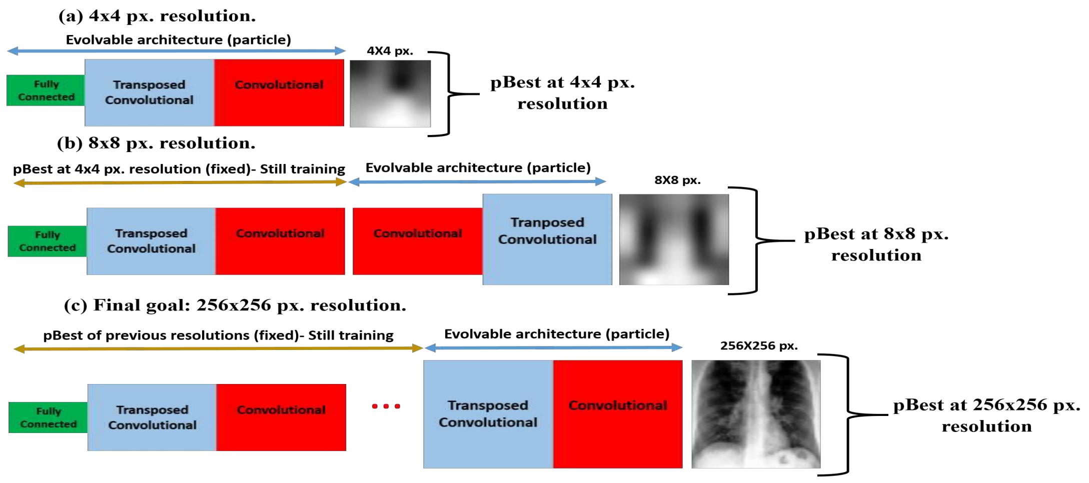

(a) 4x4 px. resolution.
Evolvable architecture (particle)
4X4 px.
Fully Connected
Transposed Convolutional
Convolutional
pBest at 4x4 px. resolution
(b) 8x8 px. resolution.
pBest at 4x4 px. resolution (fixed)- Still training
Evolvable architecture (particle)
8X8 px.
Fully Connected
Transposed Convolutional
Convolutional
Convolutional
Tranposed Convolutional
pBest at 8x8 px. resolution
(c) Final goal: 256x256 px. resolution.
pBest of previous resolutions (fixed)- Still training
Evolvable architecture (particle)
256X256 px.
Fully Connected
Transposed Convolutional
Convolutional
Transposed Convolutional
Convolutional
pBest at 256x256 px. resolution

Figure 2. Progressive growth of a DCGAN represented in a particle. Here the growth of the generator is represented but the equivalent procedure is used in the discriminator. (a): At the beginning particles generate CXR images in 4^2 px. and *pBests* are selected. (b): When the resolution increases, the architecture and the trained weights of the particle's *pBest* in the previous resolution are used. *pBest*'s layers remains fixed (not evolved), but still training, while a new population is generated and evolved with particles that will be concatenated to the already known *pBest* part. (c): The process stops when 256^2 px. resolution is reached. Obtained from the original paper.[17]

Therefore, from the features obtained, the Fréchet distance between both distributions is calculated using their estimated mean (μ) and covariance (Σ) by means of the following formula:

$$FID(x, g) = \|\mu_x — \mu_g\|_2^2 + Tr(\Sigma_x + \Sigma_g — 2(\Sigma_x\Sigma_g)^{\frac{1}{2}}) \quad (3)$$

The lower this metric is, the more similar the two sets of images are, being zero when they are equal.

The results obtained by this algorithm show an improvement in the quality of the synthetic images compared to handcrafted DCGANs for the CXR images of pneumonia due to COVID-19. In addition to verifying that the intelligent search carried out by the algorithm allows to improve the trained networks as the cycles advance, until having an early convergence. In the original paper, a brief experimentation was carried out improving the performance of a CNN for the binary classification of CXR images using the synthetic images of the evolved DCGANs, thus testing the usefulness of these synthetic biomedical images obtained through the neuroevolution of GANs.

3.2 cDCGAN-PSO

Our proposal focuses on the use of conditional-DCGANs instead of DCGANs for the neuroevolutionary algorithm, which allow generating different classes of images, controlled by the user, with only one evolved architecture. This new variant would allow the evolution of GANs architectures that generate multiple classes of biomedical images at the same time, thus reducing the number of necessary executions compared to obtain multiple classes with the original version, which only handles one class at a time per evolved network.

The changes necessary to adapt DCGAN-PSO to our proposal are the following:

- The architecture decoded from the particles is a cDCGAN instead of a DCGAN. Since the difference between architectures only concerns concatenating the class labels in the inputs of the generator and the discriminator to condition their creation and criticism, respectively, the representation of the architectures does not need any addition, since the change is made only in the implementation of the networks for their training. Therefore, those architectures that are generated from the evolved particles are also usable for the cDCGANs.
- The fitness function adopted is FID (as in DCGAN-PSO), but due to the handling of multiple classes it is required to measure the FID for each class. Therefore, we resort to using the FID averaged for each of the classes that synthesize the evolved GANs.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

The parameters used in all experiments for both versions of the neuroevolution algorithm are the same used in the reference of the original version. These are detailed in Tab. 1.

4.2 CXR Images Datasets

The sets of CXR images of pneumonia and COVID-19 used for the cDCGAN-PSO experimentation were collected from the set available in Ref. 24, which has images of pneumonia (bacterial and viral) in addition to healthy cases. 918 images of each class were selected for their quality and processed in the same way as in the former version of the algorithm: **(1)**: grayscale conversion; **(2)**: image resizing to the different used resolutions (one dataset per resolution); **(3)**: histogram equalization to increase contrast. The COVID-19 image set is the same as the one used in the original version article. Only 600 images of each class were used in the GANs evolution.

4.3 Technical Details

Details about the optimizer and initialization of GANs used can be found in Tab. 1. Python 3.6.9 programming language with the PyTorch framework[25] were used for the implementation on the free access online platform Google Colaboratory[*] to have access to GPUs for the NNs training.

[*]https://colab.research.google.com/

Table 1. Experimental parameters of DCGAN-PSO and cDCGAN-PSO used.

Parameter	Value	Parameter	Value
Particle Swarm Optimization			
Swarm size	15	N° generations per resolution	10
C_g	0.5	Resolutions list	[4^2 px., 8^2 px., 16^2 px., 32^2 px., 64^2 px., 128^2 px., 256^2 px.]
Ranges of particle parameters			
N° Convolutional layers	[0,2]	N° Fully connected layers	[1,1] (4^2 px.) ; [0,0] (otherwise)
Filter size	[2,5]	N° Transposed convolutional layers	[0,0] (4^2 px.) ; [1,1] (otherwise)
N° neurons	[1,300]	N° filters	4^2:64^2 px. = [1,256]; 128^2 px. = [1,64]; 256^2 px. = [1,32]
DCGAN & cDCGAN training			
CXR images classes	COVID-19, Pneumonia, and Healthy	N° epochs per particle training	200
N° images used by class	600	Batch size	4^2:128^2 px. = 16; 256^2 px. = 14
Optimizer	Adam[23]	Learning rate (**G** and **D**)	2×10^{-4}
β_1,β_2 (optimizer)	0.5 , 0.999	LeakyReLU slope (**D**'s activations)	0.2
Weight's initializer (**G** and **D**)	$\mathcal{N}(0, 0.02)$	Noise distribution (p_z): $\mathbb{R}^{100}$	$\mathcal{N}(0, 1)$

4.4 Fitness Evolution

The execution of cDCGAN-PSO, using the CXR images belonging to the COVID-19, pneumonia and healthy classes, was performed six times, this low number is due to computational limitations. Each run lasted approximately one week and half and with each change of resolution the time was doubled with respect to the previous resolution. However, these times could be shortened if better computer systems are available for training, the main restriction of this research.

The fitness values of the *gBest* solution were monitored during different generations and evolution stages of the cDCGAN-PSO runs. The FID optimization is shown in Fig. 3.

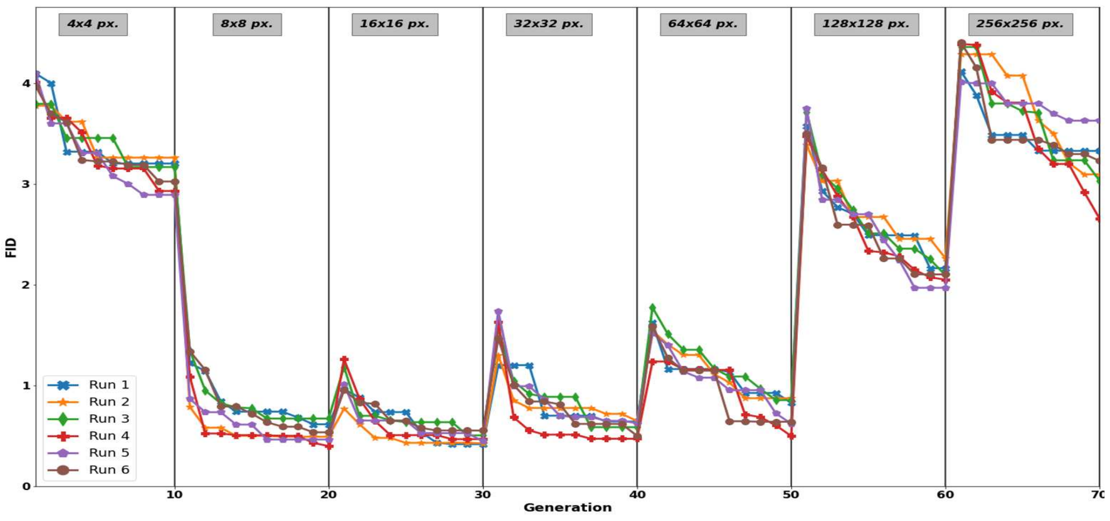


Figure 3. **cDCGAN-PSO *gBest* FID evolution.** COVID-19, healthy, and pneumonia classes (6 runs).

4.5 Qualitive Evaluation

Samples of the real images belonging to each class used as well as synthetics can be seen in Fig. 4.

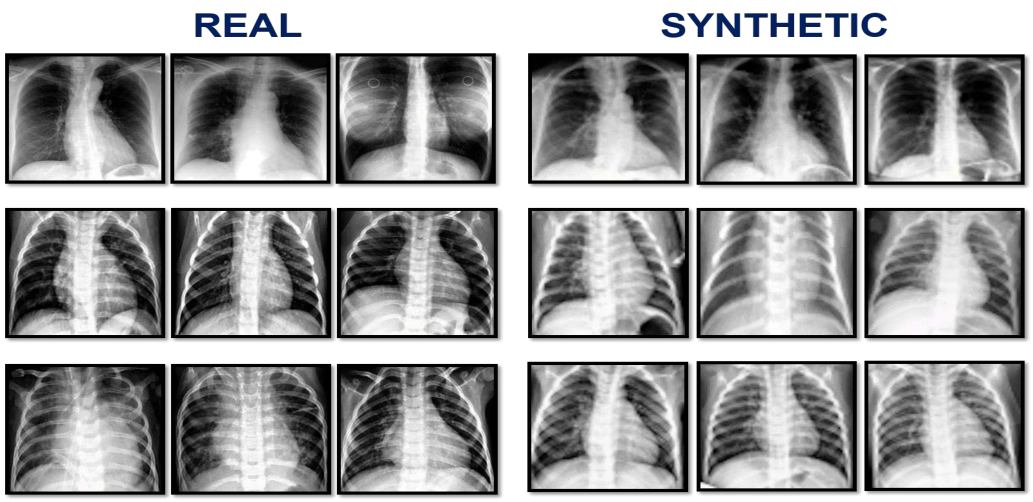


Figure 4. Sample of synthesized CXR images from cDCGAN-PSO in 256x256 pixels. Rows from top to bottom: COVID-19, pneumonia, and healthy classes.

4.6 FID Evaluation Comparison

The values of the FID evaluation obtained for the COVID-19 class of the original version were compared with the FID evaluation of the evolved networks of our proposal. In addition, two executions of the first version of the neuroevolution algorithm were performed per class pneumonia and healthy, and their FID was evaluated with the complete real set (918 for each class) and an equal size batch of synthetic images, the same was done for the networks obtained by our version, in order to compare the performance of each version of the algorithm for each of the CXR classes used in the present work. The low number of executions of the first version is due to computational restrictions. The results of these evaluations are shown in Tab. 2. The 95%-confidence Wilcoxon rank-sum (WRS) test was used to validate the statistical significance of the results.

Table 2. Results and comparison of FID evaluation values of both version of the algorithm. (=) means that the two sets of data compared have the same performance.

Class	Average FID (DCGAN-PSO)	Average FID (cDCGAN-PSO)	p-value	WRS test
COVID-19	3.052 ± 0.773	2.988 ± 0.631	0.8282	(=)
Pneumonia	3.2506 ± 0.666	3.111 ± 0.709	0.7388	(=)
Healthy	3.5916 ± 0.3242	3.345 ± 0.991	1	(=)

5. DISCUSSION

From Fig. 3 it can be observed that the fast convergence shown by the original version of the algorithm (for more details go to the article of the original version[17]) is also obtained by this new version. The behavior with sudden increases or decreases in each resolution change is as expected since the FID is a metric highly dependent on the level of detail shown by the different resolutions, then it has different ranges of values in each resolution. However, the descending behavior after each resolution change shows that the evolved networks are achieving better performances thanks to the intelligent search carried out by the neuroevolution algorithm.

Regarding the visual qualities of the synthetic images, shown in Fig.'4, these managed to mimic the general morphological features of the real images, e.g. shape of the lungs and ribs, with only small errors (blank areas). A diversity such as that seen in the original set can also be observed. Giving indications of not existing mode collapse in the cDCGANs evolved what can be derived from more stable training. The visual quality of the results is slightly better than those obtained by DCGAN-PSO, noting an increase in the quality of the fine details of the CXR images, e.g. the more defined ribs. The reason for this is because the size of the training set was tripled by adding two new classes of CXR images. This allowed the evolved cDCGANs to use this extra amount of data to learn more finely the morphological details common to the three types of images. Hence, a visual improvement over the use of a single class was obtained when compared with the former version of the algorithm.

As can be seen in Tab. 2 , the average results of the final quality of FID obtained by each class is slightly better in the cDCGAN-PSO. The reason for this may be because, as discussed previously, the greater amount of training data allowed for better results than simply using a single class of CXR images as performed in the DCGAN-PSO.

Despite the previous observation, the results obtained by the Wilcoxon rank-sum test indicate that the performances of both algorithms are equal (i.e., there are not significant statistical differences). In this case, the advantage is obtained by the new version, the cDCGAN-PSO, because it can handle multiple classes in a single execution, then decreasing the total processing times by avoiding the sequential need the former version requires. Furthermore, considering the low number of executions due to technical restrictions, further experiments with more single runs need to be carried out so as to analyze the robustness of the new proposed approach.

Finally, as shown in the Ref. 26, the use of the FID as a metric for evaluating synthetic images obtained through GANs has proven to be a heuristic that allows for improvements in the quality of the results, as well as giving a good approximation to which are the best methods and which are the least effective, as shown, by allowing to improve the synthetic images in both versions of the neuroevolutionary algorithm and its use in multiple handcrafted CXR GANs[10–13] to assess the quality of the results. However, FID it is not always robust when is used in biomedical imaging because the CNN used a non-medical dataset for their training, so it couldn't detect very fine features to aid comparison. An alternative would be to use objective task-based evaluations using synthetic images, e.g., classification as it has been used in previous CXR GANs[9,10,13–15] that use images in a resolution equal to or less than that used in the present work (256^2 px.). This is a future line of work.

6. CONCLUSIONS AND FUTURE WORK

In this work we present cDCGAN-PSO, a new version of DCGAN-PSO, a neuroevolutionary algorithm of GANs for the synthesis of biomedical images. Our version uses conditional-DCGANs which have the ability to synthesize multiple classes from CXR images, unlike the original version which can only handle one class at a time. The results obtained show that our version obtains similar performances compared to the original version of the algorithm, however, being able to synthesize multiple classes at the same time without a large increase in search times, i.e. about a week for the first version and a week and a half for our version, provides an attractive quality for use in scenarios with the need to balance or increase multiple classes with synthetic images for use in DL-based biomedical applications.

The experimentation carried out also verified the good quality of the synthetic images verified by their low FID values, statistically equal to those of the original version of the algorithm.

As future work, the proposed version can be used for the synthesis of images in higher resolution that allow to improve various tasks with CXR images of the biomedical area and that can be analyzed in detail by radiologist experts as a means of further corroborating their fidelity to real images.

ACKNOWLEDGMENTS

The first author, with CVU number 1001447, acknowledges support from the Mexican National Council of Science and Technology (CONACyT) through a scholarship to pursue graduate studies at the University of Veracruz. All UNAM authors gratefuly acknowledge the financial support of DGAPA, UNAM, under grants PAPIIT TA101121 and IV100420.

REFERENCES

[1] Mohapatra, S., Swarnkar, T., and Das, J., "Deep convolutional neural network in medical image processing," in [*Handbook of Deep Learning in Biomedical Engineering*], 25–60, Elsevier (2021).

[2] Guibas, J. T., Virdi, T. S., and Li, P. S., "Synthetic Medical Images from Dual Generative Adversarial Networks," *arXiv e-prints* (Sept. 2017).

[3] Kazeminia, S., Baur, C., Kuijper, A., van Ginneken, B., Navab, N., Albarqouni, S., and Mukhopadhyay, A., "Gans for medical image analysis," *Artificial Intelligence in Medicine* **109**, 101938 (2018).

[4] Costa, V., Lourenço, N., Correia, J., and Machado, P., "Neuroevolution of generative adversarial networks," in [*Deep Neural Evolution*], 293–322, Springer (2020).

[5] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., "Generative Adversarial Networks," *arXiv e-prints* (2014).

[6] Mirza, M. and Osindero, S., "Conditional Generative Adversarial Nets," *arXiv e-prints* (Nov. 2014).

[7] Loey, M., Smarandache, F., and M Khalifa, N. E., "Within the lack of chest covid-19 x-ray dataset: A novel detection model based on gan and deep transfer learning," *Symmetry* **12**(4), 651 (2020).

[8] Shams, M., Elzeki, O., Abd Elfattah, M., Medhat, T., and Hassanien, A. E., "Why are generative adversarial networks vital for deep neural networks? a case study on covid-19 chest x-ray images," in [*Big Data Analytics and Artificial Intelligence Against COVID-19: Innovation Vision and Approach*], 147–162, Springer (2020).

[9] Waheed, A., Goyal, M., Gupta, D., Khanna, A., Al-Turjman, F., and Pinheiro, P. R., "Covidgan: Data augmentation using auxiliary classifier gan for improved covid-19 detection," *IEEE Access* **8**, 91916–91923 (2020).

[10] Karbhari, Y., Basu, A., Geem, Z.-W., Han, G.-T., and Sarkar, R., "Generation of synthetic chest x-ray images and detection of covid-19: A deep learning based approach," *Diagnostics* **11**(5), 895 (2021).

[11] Zhang, T., Fu, H., Zhao, Y., Cheng, J., Guo, M., Gu, Z., Yang, B., Xiao, Y., Gao, S., and Liu, J., "Skrgan: Sketching-rendering unconditional generative adversarial networks for medical image synthesis," in [*International Conference on Medical Image Computing and Computer-Assisted Intervention*], 777–785, Springer (2019).

[12] Middel, L., Palm, C., and Erdt, M., "Synthesis of medical images using gans," in [*Uncertainty for Safe Utilization of Machine Learning in Medical Imaging and Clinical Image-Based Procedures*], 125–134, Springer (2019).

[13] Kora Venu, S. and Ravula, S., "Evaluation of deep convolutional generative adversarial networks for data augmentation of chest x-ray images," *Future Internet* **13**(1), 8 (2021).

[14] Salehinejad, H., Valaee, S., Dowdell, T., Colak, E., and Barfett, J., "Generalization of deep neural networks for chest pathology classification in x-rays using generative adversarial networks," in [*2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*], 990–994, IEEE (2018).

[15] Khalifa, N. E. M., Taha, M. H. N., Hassanien, A. E., and Elghamrawy, S., "Detection of Coronavirus (COVID-19) Associated Pneumonia based on Generative Adversarial Networks and a Fine-Tuned Deep Transfer Learning Model using Chest X-ray Dataset," *arXiv e-prints* (Apr. 2020).

[16] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al., "Evolving deep neural networks," in [*Artificial Intelligence in the Age of Neural Networks and Brain Computing*], 293–312, Elsevier (2019).

[17] Rodríguez-de-la Cruz, J.-A., Acosta-Mesa, H.-G., and Mezura-Montes, E., "Evolution of generative adversarial networks using pso for synthesis of covid-19 chest x-ray images," in [*2021 IEEE Congress on Evolutionary Computation (CEC)*], 2226–2233 (2021).

[18] Radford, A., Metz, L., and Chintala, S., "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv e-prints* (Nov. 2015).

[19] Karras, T., Aila, T., Laine, S., and Lehtinen, J., "Progressive Growing of GANs for Improved Quality, Stability, and Variation," *arXiv e-prints* (Oct. 2017).

[20] Junior, F. E. F. and Yen, G. G., "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm and Evolutionary Computation* **49**, 62–74 (2019).

[21] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S., “Gans trained by a two time-scale update rule converge to a local nash equilibrium,” in [*Advances in neural information processing systems*], 6626–6637 (2017).

[22] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z., “Rethinking the inception architecture for computer vision,” in [*Proceedings of the IEEE conference on computer vision and pattern recognition*], 2818–2826 (2016).

[23] Kingma, D. P. and Ba, J., “Adam: A Method for Stochastic Optimization,” *arXiv e-prints* (Dec. 2014).

[24] Mooney, P., “Chest x-ray images (pneumonia),” *kaggle, Marzo* (2018).

[25] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A., “Automatic differentiation in pytorch,” (2017).

[26] Skandarani, Y., Jodoin, P.-M., and Lalande, A., “GANs for Medical Image Synthesis: An Empirical Study,” *arXiv e-prints* (May 2021).