Coevolutionary Generative Adversarial Networks for Medical Image Augmentation at Scale

Diana Flores
ALFA, MIT CSAIL
floresd@alumn.mit.edu

Erik Hemberg
ALFA, MIT CSAIL
hembergerik@csail.mit.edu

Jamal Toutouh
ALFA, MIT CSAI
toutouh@mit.edu

Una-May O'Reily
ALFA, MIT CSAI
unamay@csail.mit.edu

ABSTRACT

Medical image processing can lack images for diagnosis. Generative Adversarial Networks (GANs) provide a method to train generative models for data augmentation. Synthesized images can be used to improve the robustness of computer-aided diagnosis systems. However, GANs are difficult to train due to unstable training dynamics that may arise during the learning process, e.g., mode collapse and vanishing gradients. This paper focuses on Lipizzaner, a GAN training framework that combines spatial coevolution with gradient-based learning, which has been used to mitigate GAN training pathologies. Lipizzaner improves performance by taking advantage of its distributed nature and running at scale. Thus, the Lipizzaner algorithm and implementation robustness can be scaled to high-performance computing (HPC) systems to provide more accurate generative models. We address medical imaging data augmentation to create chest X-Ray images by using Lipizzaner on the HPC infrastructure provided by Oak Ridge National Labs' Summit Supercomputer. The experimental analysis shows improved performance by increasing the scale of the Lipizzaner GAN training. We also demonstrate that distributed coevolutionary learning improves performance even when using suboptimal neural network architectures due to hardware constraints.

CCS CONCEPTS

• **Computing methodologies → Unsupervised learning**; **Neural networks**; *Distributed algorithms.*

KEYWORDS

Generative adversarial networks, coevolution, high performance computing, medical imaging

ACM Reference Format:
Diana Flores, Erik Hemberg, Jamal Toutouh, and Una-May O'Reily. 2022. Coevolutionary Generative Adversarial Networks for Medical Image Augmentation at Scale. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Dual artificial neural networks (ANN) adversarial structures have emerged as effective means of deep learning in which the training of a deep ANN for a specific task is assisted by another deep ANN training for an adversarially coupled task. They have found traction in certified robustness [18], generative modeling [20], and reinforcement learning [37].

This paper focuses on generative adversarial networks (GANs) [20], which train generative models through an adversarial process. GANs provide a method for learning an estimate of the training data distribution to produce new information units (samples) that approximate the original data set. Thus, GANs combine two ANN: a generator and a discriminator, which optimize their network weights to address a minmax optimization problem by applying an adversarial paradigm. The training objective of the discriminator is to distinguish real samples in the training data set from samples synthesized by the generator. The generator aims to deceive the discriminator with the samples it produces from a latent input space and a non-linear function. After a successfully converged training, the generator serves as a generative model.

GANs have been successfully applied to many problems, e.g., generating images and video [34]. One important application area for GANs is medical assistance [24], where they provide new insights to the interpretation of medical information stored in different media, such as X-ray images. However, despite their competitive results, GANs are notoriously hard to train. GANs learning process frequently shows a variety of unstable training dynamics or pathologies, such as collapse, discriminator collapse, and vanishing gradients [4, 17, 52].

Co-evolutionary algorithms (coEA) can help address GAN training pathologies. These methods optimize the minmax objective of GAN training by evolving two populations: a population of generators and a population of discriminators [12, 14, 16, 21]. The main idea is to apply the same solutions that coEAs provide to mitigate similar pathologies as those observed in GAN training. In coEAs, degenerate behaviors, such as focusing, relativism, and loss of gradient, are attributed to a lack of diversity [38]. Thus, spatially distributed populations (cellular algorithms) have shown to be effective in mitigating and resolving these types of problems.

This paper focuses on `Lipizzaner` [21], a GAN training framework that implements GAN training by applying a spatially distributed competitive coEA. In each cell of a spatial grid, an individual of each population is located, i.e., a pair generator-discriminator. During the evolutionary process, `Lipizzaner` fosters an arms race between the two populations. Thus, in each cell at each training epoch in parallel, the discriminator is evaluated against all the generators collected from the cell and its adjacent neighbors, the same with the generator. Then they apply a selection/replacement procedure to update the sub-population with the best networks (generator/discriminator).

Previous studies have shown that `Lipizzaner` provides robustness that arises from the diversity driven by the use of populations and multiple competitions [47]. The generative models trained by `Lipizzaner` improves their quality when larger grids are used, but the scale of the hardware platform used to run the algorithm limits their size. Thus, using a high-performance computing (HPC) platform may lead `Lipizzaner` to achieve better results.

The emergence of accurate image processing and analysis methods based on machine learning (ML) has improved computer-aided diagnosis (CAD) systems [11, 26]. Most of the learning models applied to CAD require a vast number of medical images for training, which are not easy to acquire [24, 41]. So image data augmentation is a possible solution to increase the number of images used to train the CAD models. Different authors have studied the use of GANs to generate diagnostic images [5, 28, 53]. We explore data augmentation of medical images with co-evolutionary GANs, specifically to generate chest X-Ray images of COVID-19 patients. This can help improve the CAD of novel emerging medical conditions.

Mainly, this research is devoted to discussing the following research questions: **RQ1:** *What is the effect on the accuracy of the generative model trained with a cellular algorithm when the spatial grid scales?* **RQ2:** *Can large-scale overcome a sub-optimal selection in the network architecture in the event of hardware limitations?* **RQ3:** *How can spatially distributed GAN training benefit from large-scale hardware platforms?* **RQ4:** *Is the combination of both distributed GAN training and HPC useful to address medical imaging generation?*

The paper is organized as follows. Section 2 presents the background of this research. Section 3 introduces the parallel distributed coEA GAN training analyzed to address medical image data augmentation on HPC environments. Section 4 describes the implementation work performed to adapt `Lipizzaner` to make the most of an HPC platform. The experimental setup is in Section 5 and results in Section 6. Finally, conclusions are drawn and future work is outlined in Section 7.

2 BACKGROUND

This paper focuses on using GANs to generate chest X-Ray images of COVID-19 patients. The following subsections discuss relevant studies about *(a)* GANs in CAD *(b)* data augmentation applied to the generation of chest X-Ray images *(c)* coEA GAN training methods.

2.1 Generative Adversarial Networks in CAD

GANs are machine learning methods applied to learn the specific distribution of a given training data set to generate new synthesized samples that follow the same estimated distribution. GANs consist of (at least) two ANNs: a generator and a discriminator, which optimize their parameters by solving a minmax optimization problem by applying adversarial learning [20]. During the learning process, the discriminator is trained to distinguish between the real samples from the training data set from the artificial/synthesized samples produced by the generator. Simultaneously, the generator learns how to transform inputs from a random latent space into synthesized samples to deceive the discriminator.

Generally, the generator and the discriminator optimize the minmax GAN objective by performing simultaneous gradient-based updates to their parameters, which rarely converges to an equilibrium suffering from degenerate behaviors or training pathologies [27].

Learning models have been applied to develop CAD systems. Given diagnostic procedure results, such as medical images, these learning models provide new insights to help the interpretation of these results by the physicians. These models rely on large data sets, and the lack of data limits their accuracy. In the particular case of X-Ray images data sets, it is not always easy to access a fair number of them to train a model [5, 28, 53]. In this scenario, (image) data augmentation techniques can help overcome the lack of images to train these intelligent models for diagnostic assistance. Thus, GAN image data augmentation approaches provide a generative model to synthesize new images that follow the same distribution as the real images. Figure 1 schemes this proposal.

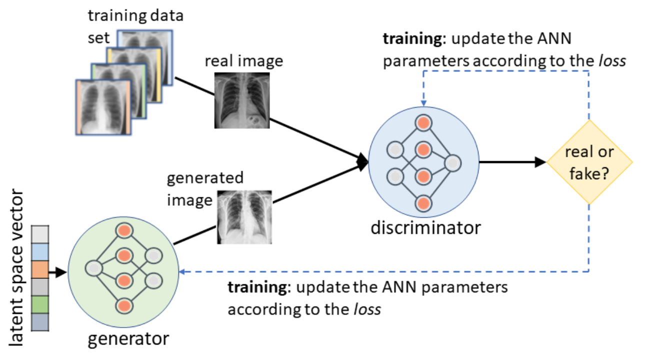

training data set
real image
training: update the ANN parameters according to the *loss*
real or fake?
discriminator
generated image
latent space vector
generator
training: update the ANN parameters according to the *loss*

Figure 1: Scheme on GANs chest X-Ray images generation.

2.2 Chest X-Ray image data augmentation

Chest X-Ray images are used to train classification models to develop CAD systems to detect health problems such as pneumonia. The accuracy of these classifiers has increased by applying chest X-Ray image data augmentation methods. Thus, GAN data augmentation was applied to train AlexNet, SqueezeNet, GoogLeNet, and ResNet-18 models with only 10% of real data and 90% of synthesized data. [23]. The same problem (i.e., detecting pneumonia) was studied by training a deep convolutional ANN (CNN) classifier with synthesized and real chest X-Ray images [6]. The generator and discriminator architectures grow to deal with higher resolution images during the training process. The main results showed that using GANs as a data augmentation technique improves the robustness of the classifier models, making them less prone to overfitting.

Focusing on the automatic detection of COVID-19 based on chest X-Ray images [22, 42], it is still a challenging task mainly because of the limited number of publicly available COVID-19 chest X-Ray image data sets. Thus, GANs are applied to create new synthesized images. Most of these studies proposed to address a multinomial classification problem considering the following labels: normal, COVID-19, and another kind of pneumonia (e.g., virus or bacterial pneumonia). The same approach proposed by Khalifa et al. [23] was applied to address this classification problem. The synthesized images were able to contribute to improving classification accuracy. Augmenting chest X-Ray images for training a CNN classifier using GANs was also studied [53]. The experimental analysis showed that the detection accuracy improved from 85% to 95%. A semi-supervised CycleGAN was applied to augment the training data set [5]. The proposed classifier achieved 94% of accuracy. A preliminary research analyzed a simplified parallel version of `Lipizzaner` to synthesize COVID-19 chest X-Ray images, which was able to produce new X-Ray images with lungs affected with COVID-19 [46].

2.3 Robust GAN training using coEAs

Avoiding or mitigating GAN training pathological behaviors is still an open question. A promising research line proposes training simultaneously multiple generators and/or discriminators, which has been shown to improve training robustness [9, 30, 32, 32, 45].

Evolutionary computation (EC) has been used to address deep learning problems, e.g., optimizing the ANN parameters through neuroevolution and evolving ANN architectures [8, 29, 44, 55]. In turn, EC has been specifically applied to GANs training. Evolutionary GAN (E-GAN) evolves a population of generators that are mutated by randomly switching the loss function [54]. Evolved GAN applies a genetic algorithm (GA) to evolve the architectures of the generators and discriminators [19]. A cooperative coEA has been proposed to conduct adversarial multi-objective optimization [12]. The minmax optimization problem is decomposed into two sub-problems (generation and discrimination). Each problem is solved by separated populations of generators and discriminators that evolve by their own evolutionary algorithm (EA).

GAN training can be seen as a two-player game solved using gradient-based updates to optimize a minmax objective simultaneously. Comp-coEA-based GAN training methods have also been proposed to co-evolve two populations of ANNs (one of generators and one of discriminators) against each other towards convergence. Co-evolutionary GAN (COEGAN) combines a Comp-coEA and and neuroevolution [14, 15]. A Comp-coEA models the training of two populations of generators and discriminators that are trained using an *all vs. all* approach. The genes of the individuals represent their architecture, which is modified by applying a mutation operator.

Theoretical studies and empirical results showed that the spatially distributed Comp-coEA GAN training mitigates non-convergence pathologies [1, 21, 51]. This paper focuses on `Lipizzaner`, which locates the individuals of the generator and discriminator populations on each cell (i.e., each cell contains a generator-discriminator pair). Overlapping Von Neumann neighborhoods determine the communication among the cells to propagate the models through the grid. Each generator is evaluated against all the discriminators of its neighborhood. The same happens with each discriminator. An SGD-based mutation is applied to the best individuals (generator and discriminator) by training them against a randomly chosen adversary in their neighborhood. These mechanisms evaluation and mutation intentionally foster diversity to address GAN training pathologies. `Lipizzaner` is a high-level GAN training framework. Therefore, any GAN alternative (e.g., Cycle-GAN) can be trained using `Lipizzaner` [21].

Mustangs, a version of `Lipizzaner` based on E-GANs, randomly selects a loss function to train each cell for each generation to increase diversity [50]. The spatial distribution of the cells allows data decomposition to train GANs in each cell with different subsets of data, which fosters diversity across the grid [48]. `Lipizzaner` returns a generative model that consists of an ensemble of generators defined by the best sub-population of generators. Evolutionary strategies are used to learn the ensemble [49]. `Lipizzaner` and its variations have shown competitive results on standard benchmarks.

Distributed GAN training represents a high-dimensional optimization problem with high computational requirements. Parallel/distributed implementation of algorithms allows improving their scalability. Focusing on EC solutions applied to ML, there are several examples of such parallel implementations, for example, a parallel version of GA to train deep CNNs on hundreds of CPU cores [43] and HPC environments [56]; a parallel version of natural evolution strategies to address a collection of reinforcement learning benchmark problems [39]; and EC-Star, a distributed genetic programming framework that runs upon commercial volunteer resources [33]. Likewise, `Lipizzaner` maximizes the scalability by running the sub-population training on independent computational resources CPUs or GPUs [21, 36]. However, there is no previous research on large-scale `Lipizzaner` experimentation on HPC.

The analysis of related works allows concluding that no previous proposals have explored the application of large-scale distributed GAN training, i.e., large-scale `Lipizzaner`. Furthermore, this research proposes addressing the relevant problem of data augmentation for medical images, specifically COVID-19 chest X-Ray images. We do not intend to create a new GANs method for image generation but to demonstrate how the scalability of `Lipizzaner` allows improving the results of any GAN trained with this framework.

3 LIPIZZANER PARALLEL DISTRIBUTED COEA GAN TRAINING

`Lipizzaner` GAN training algorithm is summarized in this section. The `Lipizzaner` algorithm is fully described in [21].

3.1 General GAN training

GANs train a generator $\mathbf{g}_g$ and a discriminator $\mathbf{d}_d$ in an adversarial setup. Here, $\mathbf{g}_g$ and $\mathbf{d}_d$ are models parametrized by g and d, where $g \in \mathcal{G}$ and $d \in \mathcal{D}$ with $\mathcal{G}, \mathcal{D} \subseteq \mathbb{R}^p$ represent the respective parameters space of both models.

Let G_* be the unknown target distribution to which we would like to fit our generative model [3]. The generator $\mathbf{g}_g$ receives a vector from a random latent space $z \sim P_z(z)$ and creates a sample from data space $x = \mathbf{g}_g(z)$ (in our case a chest X-Ray image). The discriminator $\mathbf{d}_d$ assigns a probability $p = \mathbf{d}_d(x) \in [0, 1]$ that represents the likelihood that the x belongs to the real training data set, i.e., G_* by applying a *measuring function* $\phi : [0, 1] \to \mathbb{R}$. The $P_z(z)$ is a prior distribution on z. The goal of GAN training is to find d and g parameters to optimize the objective function $\mathcal{L}(g, d)$.

$$\min_{g \in \mathcal{G}} \max_{d \in \mathcal{D}} \mathcal{L}(g, d), \text{ where} \tag{1}$$

$$\mathcal{L}(g, d) = \mathbb{E}_{x \sim P_{data}(x)}[\phi(\mathbf{d}_d(x))] + \mathbb{E}_{x \sim \mathbf{g}_g(z)}[\phi(1 - \mathbf{d}_d(x))]$$

The optimization problem is addressed by a gradient-based learning process in which $\mathbf{g}_g$ approximates the latent data distribution. At the same time, $\mathbf{d}_d$ learns a binary classifier that is the best possible discriminator between real and fake data (see Figure 1).

3.2 Distributed coEA GAN training

Lipizzaner evolves a population of generators $\mathbf{G} = \{g_1, \ldots, g_Z\}$ and a population of discriminators $\mathbf{D} = \{d_1, \ldots, d_Z\}$ by competition between each other. Individuals of each population are located in each cell of a toroidal grid. Thus, in each cell of the grid, there is a pair generator-discriminator named *center*. The neighborhood concept is applied to define the sub-population of generators (G) and discriminators (D) that participate in the training phase. Lipizzaner uses Von Neumann neighborhoods, which includes the cell itself and the ones in the adjacent cells to the North, South, East, and West (see Figure 2).

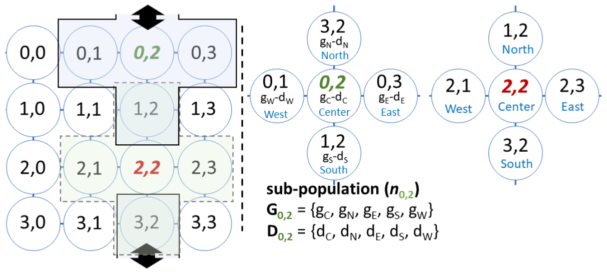


Figure 2: Lipizzaner neighborhoods and sub-population.

The coEA training is performed by a parallel cellular model for EAs [2]. Algorithm 1 presents the key steps of this method, which begins the parallel execution of the training on each cell by initializing their own learning hyper-parameters (Line 2). Then, the training process consists of a loop repeated T (generations or epochs) times with three main phases: *migration*, in which the cells gather the ANNs, i.e., individual neighbors, to build the sub-population (n); *train and evolve*, in which each cell updates the *center* by applying the coEA GANs training described in Algorithm 2; and *learning rate evolution* by using a Gaussian mutation operator. After that, each cell creates a generative model by learning an ensemble of generators using evolutionary strategies to compute the mixture weights ω (Line 8). Finally, the master process evaluates the final score of all computed generative models to select the best one.

Each cell, in parallel, applies the Comp-coEA shown in Algorithm 2 for each generation. After the evaluation of the sub-populations, The method starts by evaluating both sub-populations applying an *all-vs-all* strategy. The fitness $\mathcal{L}_{g,d}$ of a given individual is evaluated according to a loss function, e.g., *binary-cross-entropy* loss using a randomly chosen batch of data Br. Then, it selects the *best* pair of individuals, a generator and a discriminator named g_b and d_b, respectively (Lines 1 to 3). The offspring is created by training g_b and d_b against randomly chosen adversaries from the sub-populations (i.e., applying gradient-based mutations) for each batch of data (Lines 4 to 8). Then, the new individuals (i.e., mutated g_b and d_b) are added to the sub-populations. Thus, the coEA GAN training epoch ends by a a replacement procedure that removes the *weakest* individuals and updates the *center* selecting the individuals with the best fitness (Lines 11 to 13).

Algorithm 1 Lipizzaner key steps
Input: T: Total generations, E: Grid cells, s: Neighborhood size, θ_D: Training dataset, θ_{COEV}: Parameters for CoevGANsTraining, θ_{EA}: Parameters for MixtureEA, θ_{ES}: Parameters for UpdateLR
Return: n: neighborhood, ω: mixture weights

```
1:  parfor $k \in E$                                   ▷ Asynchronous parallel execution of all cells in grid
2:      $n, \omega \leftarrow$ initializeCells($k, \theta_D$)            ▷ Initialization of cells
3:      for generation do $\in [0, \ldots, T]$                 ▷ Iterate over generations
4:          $n \leftarrow$ copyNeighbours($k$)                 ▷ Collect neighbor cells
5:          $n \leftarrow$ CoevGANsTraining ($n, \theta_D, \theta_{COEV}, lr$)  ▷ Coevolve GANs
6:          $n_\delta \leftarrow$ mutateLR($n_\delta, \theta_{ES}$)           ▷ Update learning rate
7:      end for
8:      $\omega \leftarrow$ MixtureEA($\omega, n, \theta_{EA}$)            ▷ Build optimal ensemble
9:  end parfor
10: $score_k \leftarrow$ FinalScore($\omega, n$)     ▷ Compute the ensemble final score
11: $score_k \leftarrow$ CollectScore($n$)          ▷ Collect the ensemble final score
12: return $(n, \omega)^*$                          ▷ Cell with best generator mixture
```

Algorithm 2 coEA GAN training
Input: n: Cell neighborhood subpopulation, θ_D : Training dataset, τ : Tournament size, β : Mutation probability
Return: n : Cell neighborhood subpopulation trained

```
1:  $Br \leftarrow$ getRandomBatch($\theta_D$)  ▷ Random batch to evaluate GAN pairs
2:  $\mathcal{L}_{G,D} \leftarrow$ evaluate($D, G, Br$) ▷ Evaluate sub-population, i.e., GAN pairs
3:  $g_b, d_b \leftarrow$ select($n, \tau$)                 ▷ Tournament selection
4:  for $B \in \theta_D$ do                        ▷ Loop over the batches in $\theta_D$
5:      $d \leftarrow$ getRandomOpponent($\mathbf{d}$)          ▷ Get random discriminator
6:      $g_b \leftarrow$ updateNN($g_b, d, B$)              ▷ Update $g_b$ with gradient
7:      $g \leftarrow$ getRandomOpponent($\mathbf{g}$)   ▷ Get uniform random generator
8:      $d_b \leftarrow$ updateNN($d_b, g, B$)             ▷ Update $d_b$ with gradient
9:  end for
10: $\mathbf{g}, \mathbf{d} \leftarrow$ updatePopulations(G, D, $g_b, d_b$)          ▷ Add $g_b$ and $d_b$
11: $\mathcal{L}_{G,D} \leftarrow$ evaluate($G, D, Br$) ▷ Evaluate sub-population, i.e., GAN pairs
12: $n \leftarrow$ replace($n, \mathbf{g}, \mathbf{d}$)          ▷ Replace the networks with worst loss
13: $n \leftarrow$ setCenter($n$)        ▷ Best gen. and disc. are placed in the center
14: return $n$
```

Focusing on the parallelism, Lipizzaner performs an asynchronous parallel execution of all cells in the grid (see Algorithm 1). Thus, a master process starts the running by *(1)* performing the data distribution, *(2)* assigning the populations to the grid cells, and *(3)* creating the communication channels according to the defined neighborhood. Figure 3 outlines this process by detailing it for a given cell. Along the evolutionary process, communication between the processes is performed to exchange relevant information (such as ANNs parameters that define the individuals).

4 IMPLMENTATION DETAILS

This section describes the main development work performed to adapt Lipizzaner to HPC environments to be able to define larger grid sizes than the ones already investigated[1].

Better use of HPC resources. One roadblock when running Lipizzaner with large grid sizes was that the original code required a square grid [40]. The master process code was modified to find the largest rectangle that can be made with the successfully

[1]Source code - https://github.****

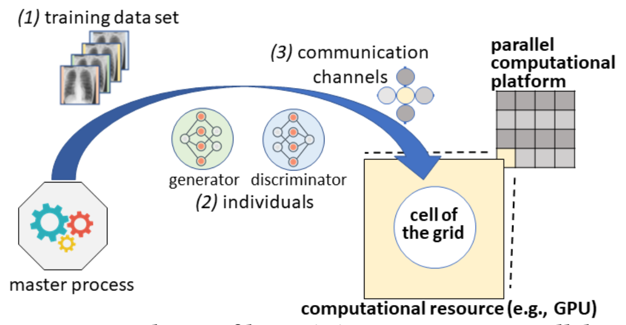

(1) training data set
(3) communication channels
parallel computational platform
generator
discriminator
(2) individuals
master process
cell of the grid
computational resource (e.g., GPU)

Figure 3: Scheme of how Lipizzaner runs in parallel.

connected clients to set the grid up. Besides, the overhead of computing the final score on the master was mitigated by moving the final score computation into the clients to run it in parallel.

With large grid sizes, training iterations could become out of sync, leaving some clients behind the rest. This can lead to the master process waiting too long to return the results. Thus, the Lipizzaner master process was allowed to end the run as soon as a proportion of the client nodes had successfully finished and a specific waiting time had expired. Previous studies have shown that allowing the Lipizzaner training to finish its execution before all clients finish does not degrade its performance [31].

Handling failures. Handling errors is critical when running Lipizzaner on large grids because the probability is higher for any client to fail. As most of these errors depend on the HPC system, we cannot avoid them but instead make Lipizzaner resilient by developing a checkpointing procedure. The main idea is to make the clients save their running status (i.e., a checkpoint) and send it to the master process to store or update it. A checkpoint consists of the parameters of the ANNs (generator and discriminator) in the center of the cell and the values of the hyperparameters that drive Lipizzaner. These checkpoints allow a master process to re-start any GAN training that abruptly halted from the last stored status as long as the necessary computational resources (client nodes) are available as long as you have the necessary computational resources. The frequency of this checkpointing operation is a configurable parameter of Lipizzaner.

Another requirement for the resiliency of Lipizzaner was to allow the client nodes to continue the training even if there is a failure. Thus, the cells in the dead client’s neighborhood simply do not try to communicate with it or update their sub-populations by copying its center ANNs. However, it is essential to set a maximum number of nodes that could fail during a run. If the number is too high, the training can be misdirected and ends up without converging to an accurate generative model.

HPC Resource Constraints. HPC can have different resource constraints compared to e.g desktop and cloud computing. Summit has various wall-time restrictions for each experiment run. To mitigate this, it was necessary to improve checkpointing in Lipizzaner. Checkpointing is the process of saving intermittent data if an experiment is killed or paused and needs to be resumed later.

When saving checkpoints for large grid sizes, the strictest limitation is memory. Storing information about a significant number (over 100) of network models should consider this limitation. As a result, the goal for saving checkpoints was to ensure that no redundant or repeating information gets stored.

We update the code to checkpoint data to only store information about the central models in the cell (i.e., the best individuals of the sub-population) instead of the original method that saved information about all models in the neighborhood. Upon re-starting training, each cell contacts its neighborhood and pulls information from them instead. The frequency of checkpointing is a configurable setting in Lipizzaner.

Further, it is difficult to guarantee the exact number of clients that will successfully connect on Summit. Lipizzaner client connection has a small probability of failure that impacts experiments of large grid sizes. As a result, guaranteeing that successive jobs will successfully connect the exact same number of clients is not always possible, which can cause inconsistency issues when an experiment requires multiple rounds of checkpointing to complete. To mitigate this, we implement configurable grid sizes. We add the fields *max_clients* and *min_clients* to the configuration files that dictate training parameters and update the Lipizzaner master code.

5 EXPERIMENTAL SETUP

The experimental analysis was carried out on two data sets: MNIST, which consists of low dimensional (28×28 pixels) hand-written digits images from 0 to 9, and COVID-19 positive chest X-Ray images provided by Cohen et al. [13] (Figure 4) shows two samples). The MNIST data set was used on additional preliminary experiments. As GPU memory constraints prohibited the use of CNNs to deal with 128×128 image generation for COVID-19 experiments, these image generation analyses considered two different lower image resolutions: 28×28 and 64×64 pixels. At the time of the experiments, the COVID-19 data set contained 190 images. A SMOTE [10] preprocess was applied to increase the number of COVID-19 images in the training data set.

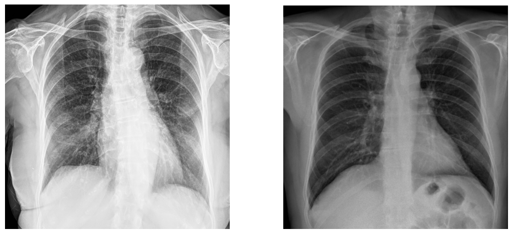


Figure 4: Chest X-Ray images data samples [13].

Experiments considered different types of ANN architectures: basic multi-layer perceptrons (MLP) and CNNs. In general, MLPs based GANs achieve less competitive results than the CNNs ones. Thus, the idea of evaluating these two types of ANN architectures was to show that scaling Lipizzaner can overcome the limitation of using non-optimal ANN architectures. Three ANN architectures were evaluated: four-layer MLPs (named 4LP), CNNs for grayscale images of 28×28 pixels (named CNN28), and CNNs for grayscale images of 64×64 pixels (named CNN64). Table 1 summarizes the main parameters of the defined ANN architectures by showing the number of neurons in the input size, the number of neurons in the output size, and the number of trainable parameters. The number of trainable parameters is representative of the ANN complexity.

Table 1: ANN architectures main parameters.

ANN arch.		Input size	Output size	Train. params.
4LP	generator	64	784	1 041 432
	discriminator	784	1	798 673
CNN28	generator	100	784	1 964 896
	discriminator	784	1	1 156 544
CNN64	generator	100	4 096	3 575 617
	discriminator	4 096	1	2 764 481

The metrics considered in the evaluation were the Frechet inception distance (FID) and the inception score (IS), which is commonly used for evaluating images generated by GANs [7], for MNIST and COVID-19 experiments, respectively. They aim at objectively assessing the quality of generated images via two relevant properties that are evaluated simultaneously: likeliness to a specific object to be generated and diversity. FID score compares (i.e., evaluates the distance between) the distribution of generated images with the distribution of real images that were used to train the generator. Therefore, a lower FID score indicates a better generative model; a perfect FID score would be close to zero. IS is within the range (1.0,M), being M the highest IS for the considered dataset. Besides, the computational clock time of each run was also evaluated.

The scalability of `Lipizzaner` was evaluated by ruining the parallel training performed in each grid cell in a different number of HPC clients depending on the grid size. Thus, `Lipizzaner` was executed on 3×3, 6×6, 12×12, and 18×18 grid sizes by using 9, 36, 144, and 324 clients, respectively. The main settings used for the experiments are summarized in Table 2.

Table 2: Main GAN training parameters.

Parameter	MNIST	COVID-19
	coEA main parameters	
Generations (epochs)	200	200
Population size per cell	1	1
Tournament size	2	2
	ANN training main parameters	
Loss function	binary-cross entropy	binary-cross entropy
Activation function	$tanh$	$tanh$
Batch size	100	70
	Learning Rate Mutation	
Optimizer	Adam	Adam
Initial learning rate	0.0002	0.0002
Mutation rate	0.0001	0.0001
Mutation probability	0.5	0.5

In order to perform the experiments, we have modified `Lipizzaner` framework [21, 40] using Python and Pytorch library [35]. The experiments were performed on the Summit Supercomputer housed at Oak Ridge National Labs (ORNL) [25]. This HPC environment provided us with a number of nodes with two IBM POWER9^{TM} processors with 1600GB of non-volatile memory and six GPUs NVIDIA Volta V100s with 32GB GPU memory.

6 RESULTS AND DISCUSSION

This section presents the results and the analyses of the studied GAN training problems. We demonstrate how spatially distributed GAN training can benefit from large scale HPC, how it overcomes the limitations of neural network architecture due to hardware constraints, and how it can generate medical images.

First, the preliminary experiments on MNIST are evaluated in terms of FID score, generated images, and computational cost time. Then, the experimental analysis on the COVID-19 data are discussed in terms of IS, synthesized images, and computational effort. Due to the limited access to the Summit Supercomputer HPC platform, different independent runs were performed for each grid size and ANN architecture complexity.

6.1 MNIST experimental results

Table 3 summarizes the experimental results by showing the number of independent runs performed for each grid size and the mean, standard deviation (stdev), minimum (min), and maximum (max) FID scores. Besides, the table presents the average run time.

The results achieved by the 3×3 grid experiments are close to the ones obtained in previous studies by using the same grid size [21]. The results indicated that the best generative models were trained by using the largest grid size, i.e., 18×18. The trend shown in Table 3 indicates that the larger the grid size, the better the results, i.e., lower FID scores. This improvement does not lead to an increase in computational time cost. The table shows that all grid sizes require a similar average run time (around 81 minutes).

Table 3: MNIST 4LP FID score (lower is better) results and average run time (in minutes). Best value in *italic.*

Grid size	Indep. runs	Mean	Stdev	Min	Max	Run time
3×3	30	41.23	2.36	36.87	47.64	82
6×6	30	24.63	2.35	20.04	29.16	*80*
12×12	15	19.89	2.54	15.41	25.91	81
18×18	8	*17.20*	*1.61*	*14.72*	*19.65*	*80*

Table 4 presents the p-values from computing the tie-corrected Rank-sum between each pair of grid size experimental results (FID score distributions) to assess the significance of the difference between them. Results in tables 3 and 4 illustrate how the difference between the results provided by different gird sizes decreases as the grid size increases. Note that all the Rank-sum p-values are lower than 0.01 but the p-value between 12×12 and 18×18 results is higher than 0.01. This suggests how well `Lipizzaner` can perform with the 4LP architecture and that a GAN based on this 4LP architecture cannot improve its results much more.

Table 4: MNIST 4LP Rank-sum p-values.

	6×6	12×12	18×18
3×3	$3.02e10^{-11}$	$6.46e10^{-8}$	$1.88e10^{-5}$
6×6		$1.05e10^{-5}$	$1.88e10^{-5}$
12×12			$1.83e10^{-2}$

Figure 5 shows a random selection of generated images for each grid size. Each grid size is able to produce a generative model that synthesizes what visually appears to be well-balanced and accurate-looking digits. It can be observed that the least clear digits are those produced by the generator trained with the 3×3 grid.

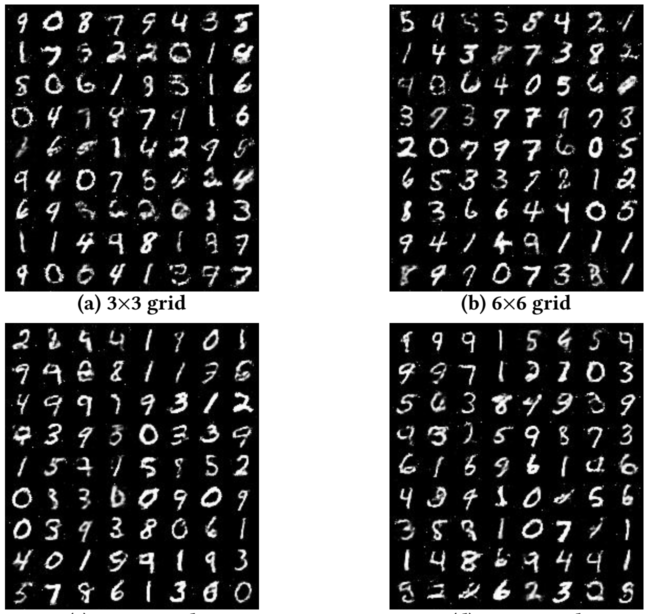

(a) 3×3 grid (b) 6×6 grid (c) 12×12 grid (d) 18×18 grid

Figure 5: Generated MNIST images with 4LP.

6.2 Chest X-Ray images experimental results

Table 5 summarizes the experimental by reporting the mean, standard deviation (stdev), minimum (min), and maximum (max) IS scores, and the average run time for each ANN architecture type and grid size. Unlike the FID score, better generative models synthesize images that provide higher IS.

Table 5: Covid-19 IS (higher is better) results and average run time (in minutes). Best value in *italic*.

Arch.	Grid size	Mean	Stdev	Min	Max	Run time
4LP	3×3	1.26	0.046	1.18	1.32	138
	6×6	1.29	0.035	1.24	1.34	164
	12×12	1.21	0.016	1.19	1.24	166
CNN28	3×3	1.31	0.021	1.28	1.34	197
	6×6	1.46	0.019	1.44	1.50	203
	12×12	1.49	0.013	1.47	1.51	208
CNN64	3×3	1.37	0.011	1.36	1.39	215
	6×6	1.53	0.016	1.50	1.55	219
	12×12	*1.61*	*0.011*	*1.59*	*1.62*	226

Considering the different ANN architectures (and same grid sizes), as the ANN complexity (in terms of the number of trainable parameters) increases, the accuracy of the generated samples improves. Thus, the lowest IS are shown by the samples generated by using 4LP and the highest scores by the ones synthesized by CNN64.

Focusing on the grid sizes, in general, larger grids provide more accurate generative models (higher IS), which implies that scale can mitigate the effects of training a poor network. However, when evaluating 4LP results in Table 5, there is a decrease in performance at the 12×12 grid experiments. This may be due to the increase in diversity (when using populations of 144 individuals) is preventing the algorithm from converging in the desired equilibrium when training generators and discriminators based on 4LP architecture.

Table 6 presents the p-values from computing the tie-corrected Rank-sum between each pair of grid size for each ANN architecture type to assess the statistical significance among the results. The results reported in the table show statistically significant differences when the grid size increases. However, the decrease in performance at the 4LP 12×12 grid experiments provokes that there is no significant difference between its results and the ones achieved by the 4LP 3×3 (0.01>p-value).

With the support of the FID scores in Table 3 and the IS in Table 5, we can answer the **RQ1:** *What is the effect on the accuracy of the generative model trained with a cellular algorithm when the spatial grid scales?.* **Answer:** The generative models trained with `Lipizzaner`, i.e., a coEA cellular algorithm, improve their accuracy as the spatial grid scales.

Table 6: Covid-19 Rank-sum p-values for different grid sizes and same architecture.

Arch.		6×6	12×12
4LP	3×3	$5.07e10^{-3}$	$9.16e10^{-2}$
	6×6		$4.92e10^{-3}$
CNN28	3×3	$4.99e10^{-3}$	$4.99e10^{-3}$
	6×6		$6.36e10^{-2}$
CNN64	3×3	$4.85e10^{-3}$	$4.77e10^{-3}$
	6×6		$4.85e10^{-3}$

Comparing only the GANs based on CNNs, when the CNN28 was trained using the 6×6 and 12×12 grids was able to provide more accurate generative models (higher IS) than the GAN based on CNN64 trained on the 3×3 grid. With the support of this observation, we can answer the **RQ2:** *Can large-scale overcome a sub-optimal selection in the network architecture in the event of hardware limitations?.* **Answer:** Yes; scaling was enough to overcome the selection of sub-optimal ANNs, allowing CNN28 to perform better than a more complex ANN architecture (i.e., CNN64).

The improvement on the obtained generative model when increasing the grid size does not lead to a significant increase in computational time costs (see Table 5). This is mainly due to the asynchronous parallelism applied by `Lipizzaner`. The delays that may occur in a cell do not generate delays in the cells of its neighborhood since the communication between them is asynchronous.

The highest run time increments occur when increasing the grid sizes when training 4LP based GANs (from 138 to 166 minutes required by 3×3 and 12×12, respectively, which implies a 20% of time increase). Evaluating the computational time for the same grid sizes when training CNNs, the increment is of 6% and 5% for CNN28 and CNN64, respectively.

According to the computational time cost results, we see that when the distributed coEA Lipizzaner GAN training relies on large-scale HPC platforms to scale the grid sizes, there is a non-significant increase in the run time. This allows scaling the distributed GAN training without significant additional computational time requirements.

Figure 6 shows a random selection of generated images for each ANN architecture and grid size. The quality of the generated images is in line with the results in Table 5. It can be seen that the 4LP architecture could not properly generate the details of the chest X-Ray images. In addition, it is shown how the 4LP results worsen when increasing the grid size from the 6×6 grid to the 12×12 grid.

Analyzing the images generated by both CNNs, i.e., the CNN28 and CNN64 architectures (figures from 6d to 6i), it can bee seen the improvement as the grid sizes increase. The most accurate chest X-Ray images were generated by the CNN64 trained in a 12×12 grid.

According to the results reported in this section, we can answer **RQ3:** *Is the combination of both distributed GAN training and HPC useful to address the generation of medical imaging? Answer:* Yes, the results in terms of IS and the generated images in Figure 6 showed that coupling GAN training and HPC is effective to train generative models to synthesize medical images.

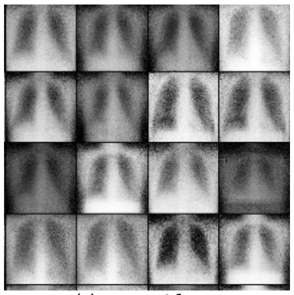
Finally, according to the experimental analysis performed in this study, the answer to the **RQ4:** *How can spatially distributed GAN training benefit from large-scale hardware platforms?* is the following one. **Answer:** The related literature showed that increasing diversity leads to robust GAN training[21, 47, 50]. Thus, large-scale hardware platforms, such as Summit Supercomputer, allow applying the spatially coEA GAN training on large grid sizes, which is robust and achieves more accurate generative models.

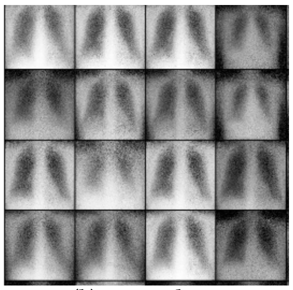
7 CONCLUSION AND FUTURE WORK

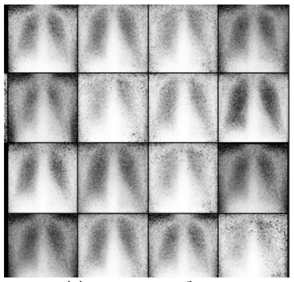
The empirical analysis of running the spatially distributed coEA GAN training applied by Lipizzaner on an HPC environment showed that it scales when computational resources are available. The preliminary results on the MNIST data set illustrated that, in general, significant improvements in FID scores (and image quality) are achieved when grid sizes grow (from 3×3 to 18×18 grids).

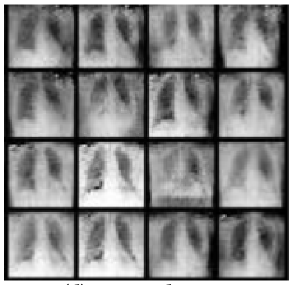
Focusing on the medical image generation (i.e., chest X-Ray images), the experimental analysis confirmed that the results improve with the scale of the grid. Except in the case of 4LP architecture trained on the 12×12 grid that showed worst IS than the same architecture trained the 6×6 grid. Thus, using the same CNN network architecture to train the generative models, the synthesized chest X-Ray images are more accurate (higher IS) as the grid size increases. When comparing the results achieved by both CNNs, it can be observed that scale may make up for reduced network complexity. Thus, CNN28 provided more accurate generative models than CNN64 when increasing the scale. The improvements observed when scaling Lipizzaner are achieved while not incurring significant additional computational effort requirements, i.e., longer execution times.

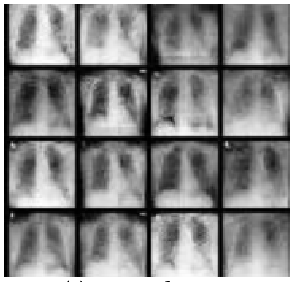
The main lines for future work are extending the experimental evaluation by increasing the grid sizes to what the actual peak

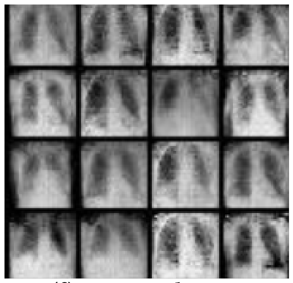

(a) 3×3 grid 4LP

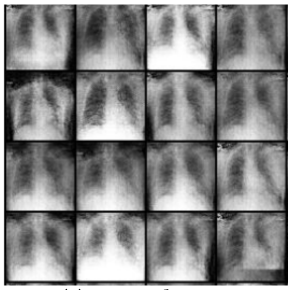

(b) 6×6 grid 4LP

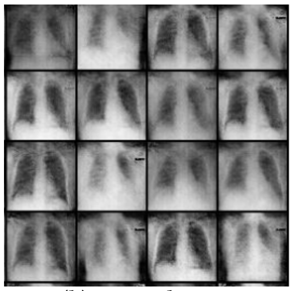

(c) 12×12 grid 4LP

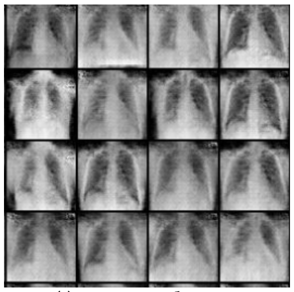

(d) 3×3 grid CNN28


(e) 6×6 grid CNN28


(f) 12×12 grid CNN28


(g) 3×3 grid CNN64


(h) 6×6 grid CNN64


(i) 12×12 grid CNN64

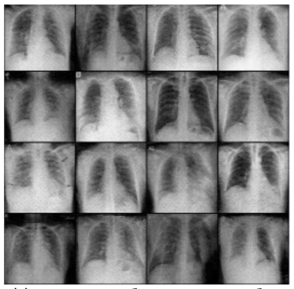

(j) Training data set samples

Figure 6: Generated and real COVID-19 samples.

of performance could be and studying data sets with medical images with larger dimensions. We will analyze the application of Lipizzaner to train other GAN solutions that have been already used to synthesize medical images, such as Cycle GAN. We will assess the robustness when using different types of neighborhoods on large grid sizes. Finally, we will analyze the evolution of the network weights through the generations to better understand the dynamics of this type of GAN training.

ACKNOWLEDGEMENTS

The work had support from Oak Ridge National Labs for project "Towards A Robust and Scalable Adversarial Learning Framework" (CSC387), and from MIT CSAIL Systems That Learn. It was partially funded by the Univ. de Málaga under grant UMA18-FEDERJA003; under grant PID 2020-116727RB-I00 funded by MCIN/AEI/ 10.13039/501100011033; and TAILOR ICT-48 Network (No 952215) funded by EU Horizon 2020 research and innovation programme. University of Málaga, Andalucía Tech.

REFERENCES

[1] Abdullah Al-Dujaili, Tom Schmiedlechner, Erik Hemberg, and Una-May O'Reilly. Towards distributed coevolutionary GANs. In *AAAI Fall Symposium*, 2018.
[2] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2012.
[3] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (GANs). *arXiv preprint arXiv:1703.00573*, 2017.
[4] Wissam J Baddar, Geonmo Gu, Sangmin Lee, and Yong Man Ro. Dynamics transfer GAN: Generating video by transferring arbitrary temporal dynamics from a source video to a single target image. *arXiv preprint arXiv:1712.03534*, 2017.
[5] Ghazal Bargshady, Xujuan Zhou, Prabal Datta Barua, Raj Gururajan, Yuefeng Li, and U. Rajendra Acharya. Application of cyclegan and transfer learning techniques for automated detection of covid-19 using x-ray images. *Pattern Recognition Letters*, 153:67–74, 2022.
[6] Vedant Bhagat and Swapnil Bhaumik. Data augmentation using generative adversarial networks for pneumonia classification in chest xrays. In 5^{th} *International Conference on Image Information Processing*, 2019.
[7] Ali Borji. Pros and cons of gan evaluation measures: New developments. *Computer Vision and Image Understanding*, 215:103329, 2022.
[8] Andrés Camero, Jamal Toutouh, and Enrique Alba. Random error sampling-based recurrent neural network architecture optimization. *Engineering Applications of Artificial Intelligence*, 96:103946, 2020.
[9] Tatjana Chavdarova and François Fleuret. Sgan: An alternative training of generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9407–9415, 2018.
[10] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique, 2002.
[11] Nabil Karim Chebbah, Mohamed Ouslim, and Sorore Benabid. New computer aided diagnostic system using deep neural network and svm to detect breast cancer in thermography. *Quantitative InfraRed Thermography Journal*, 0(0):1–16, 2022.
[12] Shiming Chen, Wenjie Wang, Beihao Xia, Xinge You, Qinmu Peng, Zehong Cao, and Weiping Ding. Cde-gan: Cooperative dual evolution-based generative adversarial network. *IEEE Transactions on Evolutionary Computation*, 25(5):986–1000, 2021.
[13] Joseph Cohen, Paul Morrison, and Lan Dao. COVID-19 Image Data Collection, 2020. Preprint arXiv:2003.11597v1.
[14] Victor Costa, Nuno Lourenço, João Correia, and Penousal Machado. Coegan: Evaluating the coevolution effect in generative adversarial networks. In *Proceedings of the genetic and evolutionary computation conference*, pages 374–382, 2019.
[15] Victor Costa, Nuno Lourenço, João Correia, and Penousal Machado. Neuroevolution of generative adversarial networks. In *Deep Neural Evolution*, pages 293–322. Springer, 2020.
[16] Victor Costa, Nuno Lourenço, and Penousal Machado. Coevolution of generative adversarial networks. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 473–487. Springer, 2019.
[17] Chris Donahue, Julian McAuley, and Miller Puckette. Synthesizing audio with generative adversarial networks. *arXiv preprint arXiv:1802.04208*, 2018.
[18] Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018.
[19] Unai Garciarena, Roberto Santana, and Alexander Mendiburu. Evolved gans for generating pareto set approximations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 434–441, 2018.
[20] 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*, pages 2672–2680, 2014.
[21] Erik Hemberg, Jamal Toutouh, Abdullah Al-Dujaili, Tom Schmiedlechner, and Una-May O'Reilly. Spatial coevolution for generative adversarial network training. *ACM Trans. Evol. Learn. Optim.*, 1(2), jul 2021.
[22] Aras M Ismael and Abdulkadir Şengür. Deep learning approaches for covid-19 detection based on chest x-ray images. *Expert Systems with Applications*, 164:114054, 2021.
[23] N. Khalifa, Mohamed Taha, Aboul Hassanien, and Sally Elghamrawy. 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, 2020. arXiv preprint 2004.01184, accessed 06/2020.
[24] Vassili Kovalev and Siarhei Kazlouski. Examining the capability of GANs to replace real biomedical images in classification models training. In *Communications in Computer and Information Science*, pages 98–107. Springer, 2019.
[25] Oak Ridge National Laboratory. SUMMIT oak ridge national laboratory's 200 petaflop supercomputer. https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/l, 2019. Accessed: 2022-12-30.
[26] Howard Lee and Yi-Ping Phoebe Chen. Image based computer aided diagnosis system for cancer detection. *Expert Systems with Applications*, 42(12):5356–5365, 2015.
[27] Jerry Li, Aleksander Madry, John Peebles, and Ludwig Schmidt. On the limitations of first-order approximation in GAN dynamics. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3005–3013. PMLR, 10–15 Jul 2018.
[28] Mohamed Loey, Florentin Smarandache, and NourEldeen Khalifa. 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.
[29] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Chapter 15 - evolving deep neural networks. In Robert Kozma, Cesare Alippi, Yoonsuck Choe, and Francesco Carlo Morabito, editors, *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Academic Press, 2019.
[30] Gonçalo Mordido, Haojin Yang, and Christoph Meinel. Dropout-gan: Learning from a dynamic ensemble of discriminators. *arXiv preprint arXiv:1807.11346*, 2018.
[31] Urmi Mustafi. Investigating system resilience in distributed evolutionary GAN training. Master's thesis, Massachusetts Institute of Technology, 2021.
[32] Behnam Neyshabur, Srinadh Bhojanapalli, and Ayan Chakrabarti. Stabilizing gan training with multiple random projections. *arXiv preprint arXiv:1705.07831*, 2017.
[33] Una-May O'Reilly, Mark Wagy, and Babak Hodjat. EC-Star: A massive-scale, hub and spoke, distributed genetic programming system. In *Genetic programming theory and practice X*, pages 73–85. Springer, 2013.
[34] Zhaoqing Pan, Weijie Yu, Xiaokai Yi, Asifullah Khan, Feng Yuan, and Yuhui Zheng. Recent progress on generative adversarial networks (GANs): A survey. *IEEE Access*, 7:36322–36333, 2019.
[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
[36] Emiliano Pérez, Sergio Nesmachnow, Jamal Toutouh, Erik Hemberg, and Una-May O'Reily. Parallel/distributed implementation of cellular training for generative adversarial neural networks. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 512–518. IEEE, 2020.
[37] David Pfau and Oriol Vinyals. Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*, 2016.
[38] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. Coevolutionary principles. In *Handbook of natural computing*, pages 987–1033. Springer, 2012.
[39] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864*, 2017.

[40] Tom Schmiedlechner, Ignavier Ng Zhi Yong, Abdullah Al-Dujaili, Erik Hemberg, and Una-May O'Reilly. Lipizzaner: A system that scales robust generative adversarial network training. In *NIPS 2018 Workshop on Systems for ML*, 2018.

[41] Jarrel Seah, Jennifer Tang, Andy Kitchen, Frank Gaillard, and Andrew Dixon. Chest radiographs in congestive heart failure: Visualizing neural network learning. *Radiology*, 290(2):514–522, 2019.

[42] Connor Shorten, Taghi M Khoshgoftaar, and Borko Furht. Deep learning applications for covid-19. *Journal of big Data*, 8(1):1–54, 2021.

[43] Kenneth O. Stanley and Jeff Clune. Welcoming the era of deep neuroevolution - uber engineering blog. https://eng.uber.com/deep-neuroevolution/, December 2017.

[44] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[45] Ilya O Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl-Johann Simon-Gabriel, and Bernhard Schölkopf. Adagan: Boosting generative models. In *Advances in Neural Information Processing Systems*, pages 5430–5439, 2017.

[46] Jamal Toutouh, Mathias Esteban, and Sergio Nesmachnow. Parallel/distributed generative adversarial neural networks for data augmentation of covid-19 training images. In Sergio Nesmachnow, Harold Castro, and Andrei Tchernykh, editors, *High Performance Computing*, pages 162–177, Cham, 2021. Springer International Publishing.

[47] Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly. Analyzing the components of distributed coevolutionary gan training. In Thomas Bäck, Mike Preuss, André Deutz, Hao Wang, Carola Doerr, Michael Emmerich, and Heike Trautmann, editors, *Parallel Problem Solving from Nature – PPSN XVI*, pages 552–566, Cham, 2020. Springer International Publishing.

[48] Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly. Data dieting in gan training. In Hitoshi Iba and Nasimul Noman, editors, *Deep Neural Evolution: Deep Learning with Evolutionary Computation*, pages 379–400. Springer Singapore, Singapore, 2020.

[49] Jamal Toutouh, Erik Hemberg, and Una-May O'Reily. Re-purposing heterogeneous generative ensembles with evolutionary computation. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 425–434, 2020.

[50] Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly. Spatial evolutionary generative adversarial networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 472–480, New York, NY, USA, 2019. Association for Computing Machinery.

[51] Jamal Toutouh and Una-May O'Reilly. Signal propagation in a gradient-based and evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 377–385, 2021.

[52] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. *arXiv preprint arXiv:1707.04993*, 2017.

[53] Abdul Waheed, Muskan Goyal, Deepak Gupta, Ashish Khanna, Fadi Al-Turjman, and Placido Rogerio Pinheiro. CovidGAN: Data augmentation using auxiliary classifier GAN for improved covid-19 detection. *IEEE Access*, 8:91916–91923, 2020.

[54] Chaoyue Wang, Chang Xu, Xin Yao, and Dacheng Tao. Evolutionary generative adversarial networks. *IEEE Transactions on Evolutionary Computation*, 23(6):921–934, 2019.

[55] Pak-Kan Wong, Man-Leung Wong, and Kwong-Sak Leung. Probabilistic grammar-based deep neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 87–88, 2019.

[56] Steven R Young, Derek C Rose, Travis Johnston, William T Heller, Thomas P Karnowski, Thomas E Potok, Robert M Patton, Gabriel Perdue, and Jonathan Miller. Evolving deep networks using hpc. In *Proceedings of the Machine Learning on HPC Environments*, page 7. ACM, 2017.