1290

UNIVERSIDADE DE
COIMBRA

Victor Franco Costa

Coevolutionary Generative Adversarial Networks

Doctoral thesis submitted in partial fulfillment of the Doctoral Program in Information Science and Technology supervised by Professor Nuno António Marques Lourenço, co-supervised by Professor João Nuno Gonçalves Costa Cavaleiro Correia and Professor Fernando Jorge Penousal Martins Machado, and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

November 2022

COEVOLUTIONARY GENERATIVE ADVERSARIAL NETWORKS

VICTOR FRANCO COSTA
vfc@dei.uc.pt

A thesis submitted to the
University of Coimbra
in partial fulfillment of the requirements for the

Doctoral Program
in Information Science and Technology

1 2 9 0

UNIVERSIDADE D
COIMBRA

Department of Informatics Engineering
Faculty of Sciences and Technology
University of Coimbra

Coimbra, November 2022

This work was partially funded by the FCT - Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit - UIDB/00326/2020 or project code UIDP/00326/2020, and by the project grant DSAIPA/DS/0022/2018 (GADgET).

Coevolutionary Generative Adversarial Networks

THIS THESIS WAS PREPARED UNDER THE SUPERVISION OF:
Nuno António Marques Lourenço
João Nuno Gonçalves Costa Cavaleiro Correia
Fernando Jorge Penousal Martins Machado

ABSTRACT

Generative Adversarial Networks (GANs) became a hot topic, presenting impressive results in the field of generative Machine Learning (ML). The GAN model consists of two neural networks that are trained simultaneously as adversaries. Research has been conducted to improve the GAN model, but there are still open challenges, such as the training stability and the necessity to manually design the architectures.

Evolutionary Algorithms (EAs) are a family of algorithms inspired by biological evolution, simulating the evolutionary mechanism found in nature. These algorithms are designed to use selective pressure to discover solutions for problems. In coevolution, more than one species are evolved simultaneously in a competitive or cooperative environment. The application of EAs in the evolution of neural networks is called neuroevolution, which can be used to automatically design the network architectures.

Our main goal is to propose a method that integrates neuroevolution and coevolution to enhance the coordination of the GAN training process, effectively applying the principles of Evolutionary Computation (EC) and ML. We hypothesize that using EAs to drive the training of GAN can improve the stability and provide the discovery of efficient network architectures. Therefore, new models are proposed in the context of this thesis. Our first contribution is a novel method called Coevolutionary Generative Adversarial Networks (COEGAN). This model combines neuroevolution and coevolution to guide the training of GANs and to discover efficient architectures. COEGAN is evaluated in different scenarios, using datasets commonly adopted to assess the performance of GANs and compared with other GAN proposals. We design a set of experiments to test our hypothesis by exploring the capacity of the model to improve the training stability and the automatic discovery of efficient network architectures. The results show that our models can overcome common issues in GANs and also outperform other non-evolutionary approaches.

COEGAN was also improved with new mechanisms not only related to EAs but also concerning advances developed for GANs. Experimental results suggest that these proposals leverage the outcome quality, approximating the results with state-of-the-art proposals. An evaluation method is also designed for simultaneous visualization and quantification of the results. When applied to COEGAN and its variations, we demonstrate the evolutionary aspects of the model and the ability to avoid issues such as mode collapse. The results of this research are important to provide a robust solution to train GANs, with stable training and automatic discovery of network architectures, showing the potential of the application of EAs in GANs. With a stable solution, a broader range of applications can be explored without the cost of manual intervention in the design process of GANs.

RESUMO

Redes Generativas Adversárias (RGA) tornaram-se um tema relevante, apresentando resultados impressionantes no campo dos modelos generativos de aprendizado de máquina (AM). O modelo RGA consiste em duas redes neuronais que são treinadas simultaneamente como adversárias. Pesquisas foram realizadas para melhorar as RGA, mas ainda existem desafios em aberto, como a estabilidade do treinamento e a necessidade de projetar manualmente as arquiteturas.

Algoritmos Evolucionários (AE) são uma família de algoritmos inspirados na evolução biológica, simulando o mecanismo evolutivo encontrado na natureza. Esses algoritmos são projetados para usar pressão seletiva para descobrir soluções para problemas. Na coevolução, mais de uma espécie evolui simultaneamente em um ambiente competitivo ou cooperativo. A aplicação de AE na evolução das redes neuronais é chamada de neuroevolução, pondendo ser usada para projetar automaticamente as arquiteturas das redes.

Nosso principal objetivo é propor um método que combine neuroevolução e coevolução para aprimorar a coordenação do processo de treinamento das RGA, aplicando efetivamente os princípios da Computação Evolutiva (CE) e da Aprendizagem de Máquina (AM). Nossa hipótese é que o uso de AE para conduzir o treinamento das RGA pode melhorar a estabilidade e proporcionar a descoberta de arquiteturas de rede eficientes. Deste modo, novos modelos são propostos no contexto desta tese. Nossa primeira contribuição é um novo método chamado COEGAN (Redes Generativas Adversárias Coevolucionárias). Este modelo combina neuroevolução e coevolução para coordenar o treinamento das RGA e descobrir arquiteturas eficientes. COEGAN foi avaliado em diferentes cenários, usando conjuntos de dados comumente adotados para avaliar o desempenho das RGA e comparando com outras propostas. Experimentos foram projetados para testar a hipótese deste trabalho de modo a explorar a capacidade do modelo de melhorar a estabilidade do treinamento e a descoberta automática de arquiteturas de rede eficientes. Os resultados mostram que nossos modelos podem evitar problemas comuns em RGA e também superar outras abordagens não evolutivas.

O COEGAN também foi aprimorado com novos mecanismos relacionados tanto ao AE quanto aos avanços desenvolvidos para RGA. Os resultados experimentais sugerem que essas melhorias alavancam a qualidade dos resultados, aproximando-os do estado da arte. Um método de avaliação foi proposto para promover a visualização e quantificação simultânea dos resultados. Quando aplicado ao COEGAN e suas variações, demonstramos os aspectos evolutivos do modelo e a capacidade de evitar problemas. Os resultados desta tese são importantes para fornecer uma solução robusta para treinar RGA, com treinamento estável e descoberta automática de arquiteturas de rede, mostrando o potencial da aplicação de AE em RGA. Com uma solução estável, uma gama mais ampla de aplicações pode ser explorada sem o custo de intervenção manual no processo de criação das RGA.

ACKNOWLEDGEMENTS / AGRADECIMENTOS[1]

Começo por expressar a minha profunda gratidão aos meus orientadores, professores Nuno Lourenço, João Correia e Penousal Machado. A orientação e o incentivo deles foram elementos indispensáveis para a realização da pesquisa e elaboração deste documento.

Aos meus familiares, especialmente aos meus pais, agradeço por acreditar em minha capacidade e pelo apoio ao ingressar nessa jornada.

Expresso um agradecimento especial à minha companheira Talita Leão, cujo apoio e incentivo foram fundamentais durante todo o meu percurso. A sua presença e apoio constante representaram uma força imensa durante este longo processo.

Por fim, expresso minha gratidão aos meus antigos professores e colegas, cuja influência no passado foi determinante para a minha formação. A todos que, de alguma forma, contribuíram para a concretização deste trabalho, o meu mais sincero obrigado.

[1] For personal reasons, the acknowledges are written in Portuguese. Apologies to the non-Portuguese speakers.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

AdaIN Adaptive Instance Normalization

AC-GAN Auxiliary Classifier Generative Adversarial Networks

ALI Adversarially Learned Inference

ANN	Artificial Neural Network
BEGAN	Boundary Equilibrium Generative Adversarial Networks
BiGAN	Bidirectional Generative Adversarial Networks
CAN	Creative Adversarial Networks
CatGAN	Category-aware GAN
CDE-GAN	Cooperative Dual Evolution based Generative Adversarial Network
CE-GAN	Evolutionary Generative Adversarial Networks with Crossover
CoDeepNEAT	Coevolution DeepNEAT
DeepNEAT	Deep NeuroEvolution of Augmenting Topologies
COEGAN	Coevolutionary Generative Adversarial Networks
DENSER	Deep Evolutionary Network Structured Representation
DCGAN	Deep Convolutional Generative Adversarial Networks
EA	Evolutionary Algorithm
EC	Evolutionary Computation
E-GAN	Evolutionary Generative Adversarial Networks
ELU	Exponential Linear Unit
EMOCGAN	Evolutionary Multiobjective Cyclic GAN
FID	Fréchet Inception Distance
GA	Genetic Algorithm
GAN	Generative Adversarial Networks
GMAN	Generative Multi-Adversarial Network
IGD	Inverted Generational Distance
InfoGAN	Information Maximizing Generative Adversarial Networks
LAPCA	Layered Pareto Coevolution Archive
LSGAN	Least Squares Generative Adversarial Networks
MAD-GAN	Multi-Agent Diverse GAN
ML	Machine Learning
MLP	Multi-Layer Perceptron
MOEA	Multi-Objective Evolutionary Algorithm
MMD	Maximum Mean Discrepancy
Mustangs	MUtation SpaTial gANs
NEAT	NeuroEvolution of Augmenting Topologies

NSLC Novelty Search with Local Competition

NSGA Nondominated Sorting Genetic Algorithm

NSGA-II Nondominated Sorting Genetic Algorithm II

PSO Particle Swarm Optimization

QD Quality Diversity

t-SNE t-Distributed Stochastic Neighbour Embedding

ReLU Rectified Linear Unit

RMSE Root Mean Squared Error

SAGAN Self-Attention Generative Adversarial Networks

SANE Symbiotic, Adaptive Neuro-Evolution

SN-GANs Spectrally Normalized Generative Adversarial Networks

SVHN Street View House Number

WGAN Wasserstein Generative Adversarial Networks

WGAN-GP Wasserstein Generative Adversarial Networks with Gradient Penalty

RGAN Relativistic Generative Adversarial Network

RSGAN Relativistic Standard Generative Adversarial Network

RaSGAN Relativistic Average Standard Generative Adversarial Network

1 INTRODUCTION

Generative models have the goal of learning an input distribution to build a model capable of producing samples based on the high-level representation of the data (Salakhutdinov and Hinton, 2009; Tu, 2007). When applied to the image domain, generative models are often trained using an image dataset for synthesizing new images with the same characteristics as the input data. Learning these models is a challenging task because of the dimensionality and complexity of the data (Tu, 2007). Generative Adversarial Networks (GAN) (Goodfellow et al., 2014) gained relevance in recent years, presenting a growing interest of the community on the improvement of generative models. GAN proposes an adversarial model used to produce samples based on an input distribution. In this context, the adversarial model are represented by the use of two components, a discriminator and a generator, trained together in a competitive process. When successful, the simultaneous training of the discriminator and the generator in a unified algorithm leads to the construction of strong components that are able to discriminate samples and synthesize realistic data. The generative component in GANs gained significant attention for representing a powerful generative model capable of producing innovative samples based on an input distribution. However, GANs can also be used to produce strong classifiers by making use of the discriminator built after successful training.

The discriminator and the generator in GANs are Artificial Neural Networks (ANNs) (Goodfellow et al., 2014). A real data distribution, usually in the form of a dataset or a probabilistic distribution, is used as input for training the discriminator. However, the generator is trained using another distribution, usually a normal or uniform distribution, that represents the latent space. Thus, the generator learns to capture the real data distribution indirectly, i.e., it will not directly receive data from the input dataset. The discriminator acts as the judge of the generator, analyzing the synthetic samples created based on the learned distribution to identify their probability to belong to the input data. In this process, the discriminator learns to distinguish between fake and real samples drawn from the input data distribution. These components are trained simultaneously as adversaries, resulting in a zero-sum game between them.

GANs have been used successfully to produce samples in a variety of domains. However, the most impressive results have been in the image domain, representing significant advances when compared to other methods by the production of realistic samples (Arjovsky, Chintala, and Bottou, 2017; Karras et al., 2018; Zhang et al., 2018a).

Despite all the success, the training of GANs is challenging, and the presence of problems such as the vanishing gradient and the mode collapse is common (Brock, Donahue, and Simonyan, 2019; Fedus et al., 2018). Moreover, it is difficult to find a way to make the training process stable, and the necessity of manual design for efficient models are relevant issues affecting the progress of GANs and a broader adoption on other applications (Wang, She, and Ward, 2019).

The vanishing gradient problem occurs when the discriminator becomes too strong and can easily distinguish between fake and real samples. In this case, the loss function will not be representative enough to contribute to the training process, preventing the gradients to flow through the generator, and the training progress stagnates. Concerning mode collapse, this problem occurs when the generator captures only a fraction of the dataset distribution provided as input to the discriminator. This is not desirable since we want to reproduce the diversity of the data distribution. Although there are strategies to minimize the effect of these problems, they remain mostly unsolved (Gulrajani et al., 2017; Salimans et al., 2016). The majority of the proposed solutions to these problems rely on the modification of the loss function used to guide the models (Arjovsky, Chintala, and Bottou, 2017; Berthelot, Schumm, and Metz, 2017; Mao et al., 2017; Zhang et al., 2018a), and/or the usage of layers in the neural networks that provide some stability (Miyato et al., 2018; Radford, Metz, and Chintala, 2015).

Another issue, not related only to GANs but also to neural networks in general, is concerned with their design. Usually, the topology and hyperparameters are chosen empirically, based on expert knowledge, which requires researchers and practitioners to put a large effort and time in repetitive tasks (e.g., fine-tuning) (Brock, Donahue, and Simonyan, 2019). In GANs, the architecture design is crucial due to the adversarial characteristics of the components. The generator and the discriminator have to be in equilibrium in order to achieve convergence on the training process. If one component becomes more powerful than the other, the GAN will suffer from training stability issues.

To tackle the problem of automatically designing ANNs, several approaches have been proposed in the literature (Assunção et al., 2019; Miikkulainen et al., 2017; Moriarty and Miikkulainen, 1997; Stanley and Miikkulainen, 2002; Yao, 1999). Neuroevolution is a field of research in artificial intelligence that is concerned with the application of Evolutionary Algorithms (EAs) to automatically design and optimize neural networks (Yao, 1999). EAs are a family of algorithms that are inspired by the evolutionary mechanism found in nature (Sims, 1994a). Algorithms in this family use a population of potential solutions that are evolved through generations to produce an optimized outcome for a given problem. An individual from the population encodes the solution through an internal representation called genotype. The algorithm applies a transformation function for transforming a genotype into the concrete representation of the solution called phenotype. Then, individuals can be evaluated and selected for reproduction, composing the next generations of potentially better solutions. Neuroevolution applies these concepts to evolve neural networks, where genotypes represent neural networks that must be optimized for a given objective.

In neuroevolution, both the network architecture (e.g., topology, hyperparameters, and optimization method) and the parameters (e.g., weights) can be evolved (Yao, 1999). NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002) is a well-known neuroevolution method that evolves the weights and topologies of neural networks. Recently, DeepNEAT (Miikkulainen et al., 2017) was proposed in order to expand NEAT to larger search spaces, such as in deep neural networks.

Taking into account that GANs use ANNs as models for the generator and the discriminator, one can use neuroevolution to design these models. However, the way GANs work, where the generator and the discriminator are competing with each other in a zero-sum game, raises some challenges, namely regarding on how to promote this competition in the context of EAs. Coevolution is the simultaneous evolution of at least two distinct species (Hillis, 1990; Rawal, Rajagopalan, and Miikkulainen, 2010). In competitive coevolution, individuals of these species are competing together, and their fitness function directly represents this competition. Thus, we can assess the applicability of a competitive coevolution environment in an EA to train GANs.

1.1 RESEARCH QUESTIONS

In spite of the success and impressive results, GANs are hard to design and train. The balance between the generator and the discriminator is essential for training stability and convergence. If the balance between the two is disrupted, the end results will be compromised, often leading to poor results for the application domain. Despite the recent efforts, the stability issues remain mostly unsolved. Besides, the choice for the neural network architecture to be used for the discriminator and the generator is not a straightforward decision, being usually decided manually through experimentation. There is also no consensus about the best architecture to use in GANs, as this choice depends on the context of the application.

The hypothesis behind this work is that the training process of GANs can be improved by the use of EAs, in the form of neuroevolution and coevolution. From this hypothesis, the following research questions arise and are the subject of the investigation in the scope of this thesis:

I How to define useful fitness functions for the discriminator and the generator of a GAN that induce evolutionary pressure to produce efficient models?

II Can neuroevolution be applied to GANs to create useful neural network architectures?

III Can the use of coevolution improve the training stability of GANs?

IV Does the proposed evolutionary algorithm solve common problems in GANs, such as the mode collapse and the vanishing gradient?

To handle Question I, we propose and evaluate different functions to be used as fitness on the discriminator and generator inside an EA. These fitness functions should provide evolutionary pressure to achieve the goals defined by the GAN model. Question II refers to the assessment of the neuroevolution strategies applied in the context of GANs. The architectures of the neural networks discovered by the proposed method should be analyzed to identify their characteristics and performance in the experiments. To answer Question III, we analyze the training convergence of the proposed algorithm through experimentation. This analysis should identify the occurrence of common problems in GANs, also answering Question IV. We also propose an evaluation method to provide a coherent visualization and quantification of the issues related to Question IV.

1.2 CONTRIBUTIONS

During the research process, the following contributions were made in the context of this thesis:

- Literature Review: A review of the state of the art on the applications of EAs in GANs.

- Coevolutionary Generative Adversarial Networks (COEGAN): We propose a method that integrates neuroevolution and coevolution to enhance the coordination of the GAN training process, effectively applying the principles of Evolutionary Computation (EC) and Machine Learning (ML). An experimental study was conducted to assess the contributions of this model compared to other GANs using datasets such as MNIST, Fashion-MNIST, SVHN, and CIFAR-10. We show that this model improves the training stability of GANs and provides automatic discovery of efficient network architectures. As the training of a GAN is recognized as an unstable process, the results of this research is important to provide a robust solution to train GANs, with stable training and automatic discovery of network architectures. By having a stable model for training GANs, it becomes possible to explore a wider range of applications without the need for manual intervention in the design process, thereby reducing associated costs.

- Improvements in Coevolutionary Generative Adversarial Networks (COEGAN): As we identified possible improvements in the original model, we propose to incorporate new mechanisms of GANs and EAs to make the model more robust. Thus, we explore the use of other fitness functions and different approaches for the Evolutionary Algorithm (EA). Furthermore, we incorporate into the model more robust mechanisms for GANs.

- Evaluation Method for Generative Models: We propose a new method to visualize and quantify the performance of generative models. This method was applied in the context of this thesis to study the performance of COEGAN.

Concerning the scientific dissemination of the work, the previous contributions resulted in a series of publications in prestigious international conferences and a book chapter. Following we list these publications and the corresponding chapter of this thesis that originated the work.

The work conducted in Chapter 2 produced the following book chapter:

- Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2020b). "Neuroevolution of Generative Adversarial Networks." In: Deep Neural Evolution. Springer, pp. 293–322.

The work conducted in Chapter 3 resulted in the following publications:

- Costa, Victor, Nuno Lourenço, and Penousal Machado (2019). "Coevolution of Generative Adversarial Networks." In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar). Springer, pp. 473–487.

- Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2019). "COEGAN: Evaluating the coevolution effect in generative adversarial networks." In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, pp. 374–382.

Furthermore, the advances in the model proposed in Chapter 4 and Chapter 5 produced the following publications:

- Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (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.

- Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2020a). "Exploring the evolution of GANs through quality diversity." In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 297–305.

- Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2021). "Improved Evolution of Generative Adversarial Networks." In: Proceedings of the Genetic and Evolutionary Computation Conference Companion.

Finally, the evaluation model proposed in Chapter 6 resulted in the following publications:

- Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2021). "Demonstrating the Evolution of GANs Through t-SNE." In: Applications of Evolutionary Computation. Ed. by Pedro A. Castillo and Juan Luis Jiménez Laredo. Cham: Springer, pp. 618–633. isbn: 978-3-030-72699-7.

1.3 DOCUMENT STRUCTURE

The remainder of this document is organized as follows. Chapter 2 introduces the background concepts related to this work, focusing on GANs, EAs, neuroevolution and coevolution, presenting state-of-the-art approaches that apply EAs to GANs. In Chapter 3 we detail our proposal to apply EAs in the context of GANs and present our first experimental results. Chapter 4 explore different evolutionary mechanisms used to improve the original proposal and achieve better results, giving directions for further improvements. Chapter 5 presents a new version of the model that incorporates recent advances regarding GANs. Chapter 6 presents our proposal to inspect the performance of generative models. Finally, Chapter 7 presents the conclusions of this thesis and research directions for future work.

2 BACKGROUND AND RELATED WORK

This chapter presents the background concepts and the recent advances on the topics related to this thesis. Section 2.1 introduces the Generative Adversarial Networks (GAN) model and its variations that are relevant to this thesis. We also present and discuss the main challenges that occur when training GANs which support the hypothesis proposed in this thesis. Section 2.2 presents the concepts related to Evolutionary Algorithms (EAs), focusing on neuroevolution and coevolution. Finally, Section 2.3 describes the current proposals that use EA in the context of GANs.

2.1 GENERATIVE ADVERSARIAL NETWORKS

Generative Adversarial Networks (GAN), proposed by Goodfellow et al. (2014), is an adversarial model that became relevant mostly due to the performance achieved in generative tasks for the image domain, representing significant improvements over other generative methods.

The vanilla model of a GAN combines two Artificial Neural Networks (ANNs): a discriminator D and a generator G. The goal of the discriminator D is to distinguish between real and fake examples, given a real data distribution, usually in the form of a dataset. The generator G outputs fake samples, attempting to capture the data distribution used in the training of D. For this, the generator receives data from an input distribution (e.g., a uniform or a normal distribution), representing the latent space of the generative model. These components are trained simultaneously as adversaries, creating strong generative and discriminative components. Even though the generator never looks directly at the data distribution given to the discriminator, in a successfully trained GAN, the created samples approximate the characteristics of the input dataset. Figure 1 depicts the interaction between these components when using a digits input dataset, such as the MNIST dataset (LeCun, Cortes, and Burges, 1998). In this thesis, we focused on the application of GANs to the image domain. Therefore, the real and fake samples mentioned above are representations of images from the dataset and from the generator, respectively.

The discriminator and generator are trained with backpropagation through a gradient descent method (LeCun, Bengio, and Hinton, 2015). Therefore, different loss functions are used in the GAN components. The loss function of the discriminator is defined as follows:

$$J^{(D)}(D, G) = -\mathbb{E}_{x \sim p_{data}}[\log D(x)] - \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]. \tag{1}$$

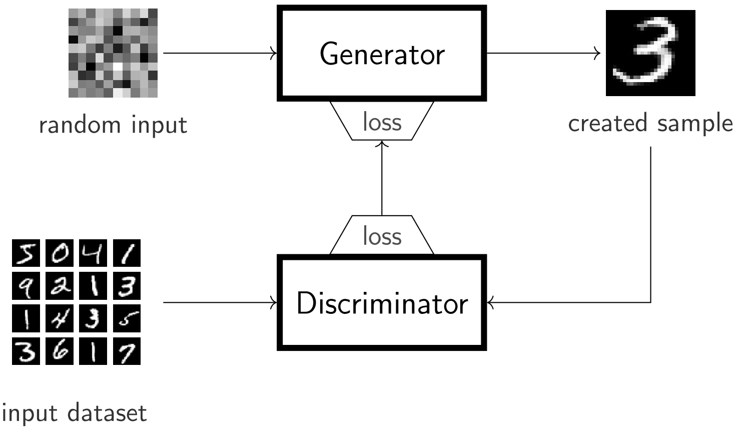


Figure 1: High-level interaction between the components of a GAN during the training process.

In Eq. 1, p_{data} represents the dataset used as input to the discriminator, z is the latent space, p_z is the latent distribution, G is the generator, and D is the discriminator. In this equation, the discriminator is used to estimate the probabilities of samples from the input dataset being real and synthetic samples to be fake. The difference between these probability distributions gives the loss used to update the parameters of the neural network.

For the generator, the loss function is defined by:

$$J^{(G)}(G) = \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]. \tag{2}$$

In Eq. 2, z, p_z, G and D represent the latent space, the latent distribution, the generator, and the discriminator, respectively. In this equation, the discriminator is used to assess the capabilities of the generator to produce samples that approximates the input data. Thus, the loss function aims to minimize the probability of synthetic samples to be classified as fake. However, gradients can be saturated in this loss function (Eq. (2)) early in training when the generator is still striving to learn the input distribution. In this case, it is easy for the discriminator to reject synthetic samples with high probability. Therefore, to tackle this issue, a non-saturating version of the loss function is defined by:

$$J^{(G)}(G) = -\mathbb{E}_{z \sim p_z}[\log(D(G(z)))]. \tag{3}$$

This version of the loss function invert the logic used in Eq. 2, aiming to maximize the probability of synthetic samples to be classified as real.

The training procedure of a GAN is presented in Algorithm 1. In lines 2-6, the discriminator is trained for k steps, using a set of m samples usually referred to as batch.

Algorithm 1 Generative Adversarial Networks (GAN) training algorithm.

```
1: for $n$ iterations do
2:     for $k$ steps do
3:         $fake\_samples \leftarrow sample(m, P_z(g))$
4:         $real\_samples \leftarrow sample(m, P_{data}(x))$
5:         $train\_discriminator(D, G, fake\_samples, real\_samples)$
6:     end for
7:     $fake\_samples \leftarrow sample(m, P_z(g))$
8:     $train\_generator(G, D, fake\_samples)$
9: end for
```

The generator is trained in lines 7-8. The losses described by Equations 1 and 3 are applied in lines 5 and 8, respectively. The optimization method used for training can be any stochastic gradient descent method, such as the Adam optimizer (Kingma and Ba, 2015) or RMSprop (Tieleman and Hinton, 2012). In this case, the errors calculated by the loss functions Eq. (1) and Eq. (3) flow through the neural networks of the discriminator and the generator, and the trainable parameters (i.e., weights and biases) are adjusted according to the optimization method.

The interactions between the generator and the discriminator is paramount to the training stability. However, the balance between them is frequently disrupted, making the training of GANs a challenging task. Often, to achieve good outcomes in training, a trial-and-error approach is used. Therefore, researchers developed a set of techniques and recommendations to increase the probability to achieve convergence. Salimans et al. (2016) proposes the use of strategies such as label smoothing and minibatch discrimination to obtain better results in training. However, these strategies only minimize the effect of the problems, and finding the right settings is challenging and time consuming.

2.1.1 *Issues*

Many researchers proposed variations over the original GAN model attempting to solve the most common problems affecting the training of GANs, namely the vanishing gradient and the mode collapse problems. These problems recurrently affects the outcome of GANs, representing a significant challenge to obtain efficient models for different applications (Creswell et al., 2018). Although several approaches have been proposed to improve the GAN model, these problems still affect the training (Arjovsky, Chintala, and Bottou, 2017; Gulrajani et al., 2017; Salimans et al., 2016).

2.1.1.1 *Mode Collapse*

The mode collapse problem occurs when the generator captures only a small portion of the distribution provided as input to the discriminator. This diminished representation is not desirable since the aim of the generator is to reproduce the whole distribution of the data and with the possibility of adding novelty to the output samples.

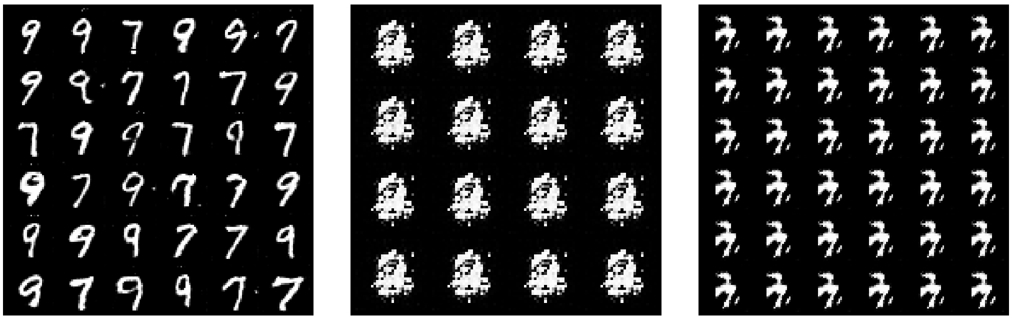


Figure 2: Mode collapse occurring in the MNIST dataset.

Figure 2 shows an example of samples created by a generator suffering from mode collapse after a failed training of a GAN using the MNIST dataset. We can see in Figure 2(a) that only the digits 9 and 7 are represented, which correspond to a small portion of the numbers presented in the dataset. Figures 2(b) and 2(c) show another issue regarding the mode collapse, where the digits cannot be identified correctly since the generator creates only a superposed combination of digits. Thus, these issues demonstrated in 2 characterizes the mode collapse problem. In this case, the generator learns to represent only a small fraction of the input distribution. The loss functions used in the original GAN model do not take into account the diversity of samples produced by generators. This is only addressed indirectly by the capacity of the discriminator to learn the behavior of the generator. Therefore, there are cases that the discriminator does not achieve the desired capabilities in the training process, benefiting the generator to exploit specific sections of the input distribution to deceive the discriminator. In these cases, the trained parameters of the generator will be focused on the generation of a specific subset of the data.

2.1.1.2 *Vanishing Gradient*

The vanishing gradient problem occurs when the discriminator becomes so powerful that it cannot be deceived by the generator anymore when distinguishing between fake and real samples. Hence, the loss function will have a small value, making the gradients so small that they are not contributing to update the trainable parameters of the generator during the gradient descent algorithm, and the GAN training stagnates.

The equilibrium between the discriminator and generator is essential to the training, and when the vanishing gradient problem happens this equilibrium is violated in an irreversible way.

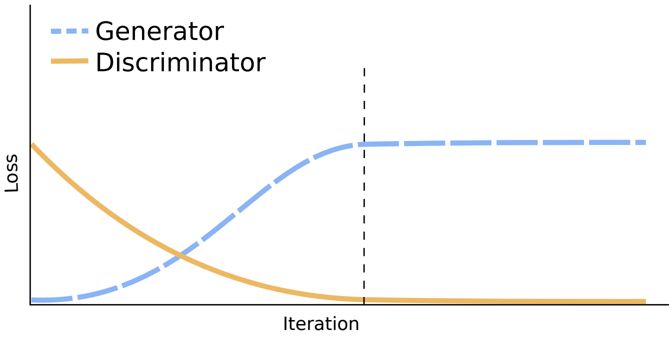


Figure 3: Losses of the generator and discriminator of a training with the vanishing gradient issue.

Figure 3 depicts an example of the training of a GAN that suffered from the vanishing gradient problem. We can see in this figure the progression of losses of the generator and discriminator through iterations. Note that when the discriminator loss becomes zero (marked by the dashed vertical line), the generator stops to improve and stagnates until the end of the training. As such, the quality of samples created by the generator will not improve anymore. As argued by Fedus et al. (2018), it is important to note that the divergence between the generator and discriminator does not need to always decrease. Even when the loss increases, the training can reach good solutions in the end. Therefore, regarding the vanishing gradient, the problem only occurs when the loss approximates zero. Otherwise, the GAN model is able to tolerate iterations where the losses deviate from the objective and produce a worse value (when compared to the previous step).

2.1.2 *Evaluation Metrics*

To evaluate the performance of GANs, samples created by the generator should be analyzed following a standard procedure, preferably by using an automatic measurement of the quality of the samples to compare the results obtained from different variations of GANs or other generative solutions. However, there is not a consensus yet in the community about the best metric to perform this analysis and it remains an open problem. In Xu et al. (2018), an empirical evaluation is presented on metrics that can be applied to GANs, such as the Kernel Maximum Mean Discrepancy (MMD), Inception Score, Fréchet Inception Distance (FID), Wasserstein distance, and the 1-Nearest Neighbor classifier. Nevertheless, we highlight in this section two of the most common metrics used by researchers: the Inception Score and the FID score.

2.1.2.1 *Inception Score*

The Inception Score, defined by Salimans et al. (2016), is an automatic metric to evaluate synthetic image samples created based on an input dataset. This method uses the Inception Network (Szegedy et al., 2015, 2016) to get the conditional label distribution of the images created by a generative algorithm. The network is previously trained using the ImageNet dataset (Russakovsky et al., 2015). Therefore, the Inception Score is defined as:

$$IS(x, y) = exp(\mathbb{E}_x KL(p(y|x)||p(y))), \tag{4}$$

where x is the input data, y is the label of the data, $p(y)$ is the label distribution, $p(y|x)$ is the conditional label distribution, and KL is the Kullback–Leibler (Goodfellow et al., 2014) divergence between the distributions $p(y|x)$ and $p(y)$. Salimans et al. (2016) recommend evaluating the metric on a large number of samples, such as 50000, to provide enough diversity to the score.

Barratt and Sharma (2018) found some issues with the Inception Score, namely its sensitivity to the weights of the Inception Network used in the calculation. Additionally, the network used in the Inception Score, which was trained in the ImageNet dataset, may not be applicable with consistent performance to other datasets.

2.1.2.2 *Fréchet Inception Distance Score*

Fréchet Inception Distance (FID) (Heusel et al., 2017) is one of the state-of-the-art metrics to compare the generative components of GANs. The FID score outperforms other metrics, such as the Inception Score, with respect to diversity and quality (Lucic et al., 2017). In FID, a hidden layer of Inception Net (Szegedy et al., 2015, 2016) (trained on ImageNet (Russakovsky et al., 2015)) is used to transform images into a feature space that is interpreted as a continuous multivariate Gaussian. This transformation is applied to a subset of the real dataset and samples created by the generative method. The mean and covariance of the two resulting Gaussians are estimated and the Fréchet distance between these Gaussians is calculated by:

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{1/2}), \tag{5}$$

where μ_x, Σ_x, μ_g, and Σ_g represent the mean and covariance estimated for the real dataset x and fake samples g, respectively, and Tr is the trace function applied on the resulting matrix of the operations with Σ_x and Σ_g. In summary, the score is given by the norm of the means and the trace of the covariances estimated for real and fake data.

2.1.2.3 *Skill Rating*

Another approach to evaluate GANs is to use an existing game rating system. In this context, the Skill Rating has been successfully used to evaluate GANs to assess the skill of generators and discriminators (Olsson et al., 2018). The approach is to consider each generator and discriminator as a player in a game. Thus, one step of the GAN training algorithm is designed as a match between one generator and one discriminator. The outcome of the matches is used as input to calculate the skill of each player, generating a ranking corresponding to the skills of them.

In games like chess, it is common to use a rating system to quantify the skill of players. In this context, the Glicko-2 (Glickman, 2013) rating system can be used to measure the performance of players given a history of matches, taking into account three variables: the rating r, the deviation RD, and the volatility σ. The rating, r, indicates the actual skill of the player after a sequence of matches with other players in a game. The volatility, σ, represents the expected variability on the rating of a player. The deviation, RD, represents the confidence in the player's rating. A system constant τ is also used to control the rate of change on the volatility σ. Different from r, RD, and σ, this parameter is associated with the whole rating system.

All players are initialized with the recommended values of 1500 for the rating r, 350 for the deviation RD and 0.06 for the volatility σ. These values can be adjusted according to the characteristics of the application. At a fixed time period, the results of all matches between players are stored and used to update the rating r, deviation RD, and volatility σ. It is recommended to use a time span large enough to contain at least 10 to 15 games for each player.

2.1.3 *Visualizing the Distribution of Samples*

Besides evaluation metrics, a tool to visualize the results is also important not only to provide insight for the coverage of the distribution achieved by the generative model but also to detect issues such as the mode collapse problem. In this regard, Zhang et al. (2018b) proposed the use of t-Distributed Stochastic Neighbour Embedding (t-SNE) to visualize the distribution of images produced by the generative model.

The t-SNE (Maaten and Hinton, 2008) is a technique used to produce a map of two or three dimensions that represents the data distribution. The t-SNE algorithm works iteratively. A set of pairwise affinities is calculated for the input data and the solution is randomly initialized using a probabilistic distribution. At each iteration, the gradient is calculated based on the Kullback-Leibler divergence between the high-dimensional input space and the corresponding lower-dimensional representation. The gradient is used to update the solution. After all iterations, t-SNE outputs the final solution, representing points in a two or three-dimensional grid for the input data. Figure 4 shows an example of the distribution of images from the MNIST dataset by t-SNE.

The number of iterations and perplexity are two important parameters for the t-SNE algorithm. Perplexity defines how the neighborhood of each data point is handled. Values between 5 and 50 are commonly chosen for perplexity (Maaten and Hinton,

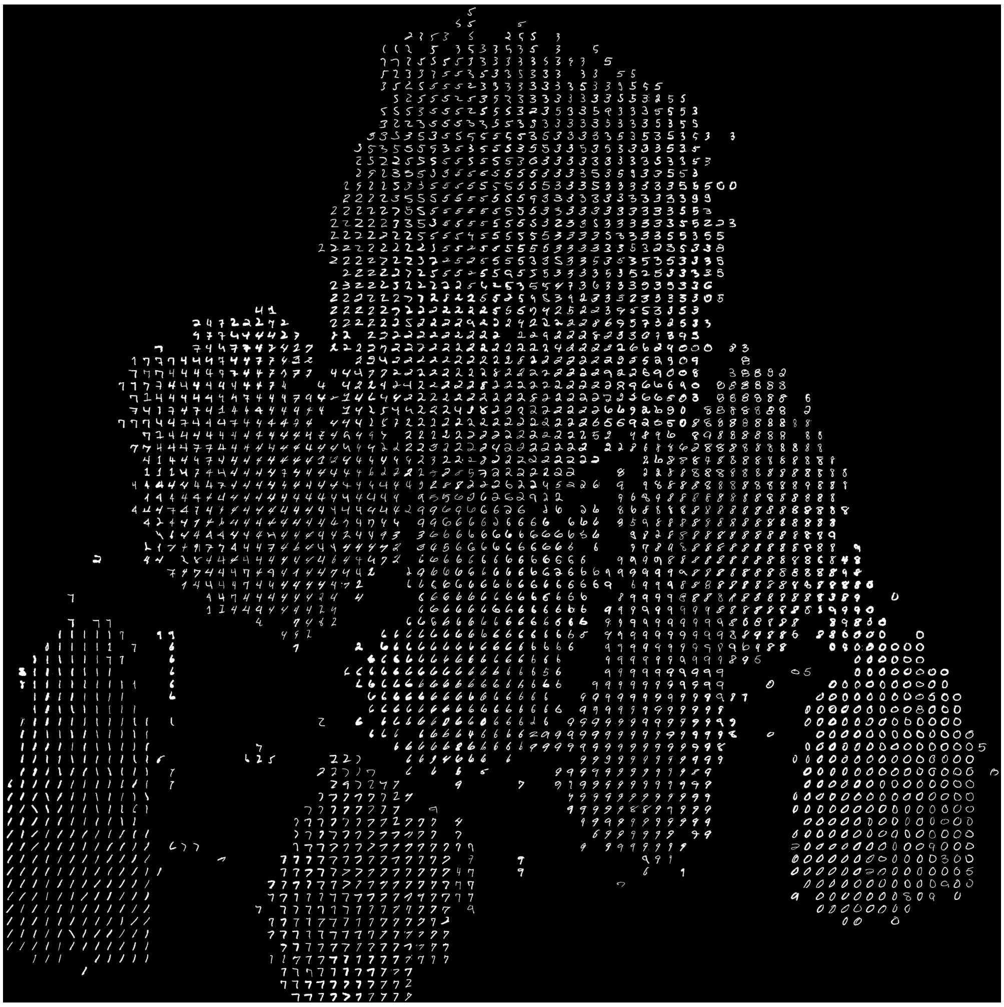


Figure 4: Example of the distribution of images from MNIST by t-SNE into a two-dimensional grid.

2008). The number of iterations limits the number of steps used to update the final solution.

Principal Components Analysis (PCA) can be used as a preprocessing step to reduce the dimensionality of the data, suppress noise, and achieve a faster computation (Kobak and Berens, 2019; Maaten and Hinton, 2008). For example, PCA was used to reduce the dimensionality of the data to 30 and 50 by Maaten and Hinton (2008) and Kobak and Berens (2019), respectively.

t-SNE provides a useful visualization of complex distributions by revealing the structure of the data. Regarding GANs, the visualization method proposed by Zhang et al. (2018b) was capable of coherently distribute the produced samples into a two-dimensional grid by grouping images considering their inner characteristics.

2.1.4 *Variations of GANs*

Several advances over the original GAN model were recently proposed (Arjovsky, Chintala, and Bottou, 2017; Berthelot, Schumm, and Metz, 2017; Durugkar, Gemp, and Mahadevan, 2016; Karras et al., 2018; Karras, Laine, and Aila, 2018). These proposals focused not only on the improvement of the quality of the created samples but also on the improvement of the training stability. In this section, we detail the main proposals that have been put forward to address and mitigate the issues discussed in Section 2.1.1. These proposals are divided into two main categories (Pan et al., 2019; Wang, She, and Ward, 2019): architecture improvements and alternative loss functions. We also included in this study a third category related to conditional GANs, which is another modification of the original model that has slightly different applications. As conditional GANs introduce other aspects into the model, modifying the original objective of GANs, we put these proposals in a separated category.

2.1.4.1 *Architecture Improvements*

The architectures for generators and discriminators initially used in GANs were based on Multi-Layer Perceptron (MLP) (Goodfellow et al., 2014). In 2015, Radford, Metz, and Chintala proposed Deep Convolutional Generative Adversarial Networks (DCGAN), which quickly became a reference architecture for the discriminator and the generator in GANs. This architecture is composed of a set of constraints and rules that were experimentally evaluated and that revealed performance gains over other generative methods. Some of these rules are:

- **Use batch normalization in the generator and discriminator:** helps to stabilize learning and the gradient flow;
- **Use the ReLU activation function in all hidden layers of the generator:** allows the model to learn the color space of the input distribution;
- **Use LeakyReLU in all layers of the discriminator:** achieves good performance in images with high resolution.

In the experiments conducted by Radford, Metz, and Chintala (2015), the training stability was improved with DCGAN, but this model still had issues such as the mode collapse problem in some executions.

Berthelot, Schumm, and Metz (2017) developed a model called Boundary Equilibrium Generative Adversarial Networks (BEGAN), where the most relevant contribution is the use of an autoencoder as the discriminator, instead of a classical neural network. BEGAN also uses a different loss function, based on the Wasserstein distance (Arjovsky, Chintala, and Bottou, 2017). The authors argued that BEGAN presents better training results with respect to the stability and quality of the created samples.

Durugkar, Gemp, and Mahadevan (2016) proposed Generative Multi-Adversarial Network (GMAN), a model that uses multiple discriminators in the training process. GMAN uses the set of discriminators to select the best two for training the generator. The authors argued that this strategy leads to an improvement in the samples quality and the necessity of fewer training iterations when compared to regular GANs.

Following this line of research, Ghosh et al. (2018) explore an approach called Multi-Agent Diverse GAN (MAD-GAN) that also uses multiple components in the GAN training. However, instead of multiple discriminators, they propose to use multiple generators. In MAD-GAN, all generators share the parameters of the initial layers and only the last layer is completely independent. The discriminator is also changed to include another output: the prediction of which generator created the input sample. The experiments evidenced that the use of multiple generators helped to increase the variability on the created samples and avoid the mode collapse problem.

Elgammal et al. (2017) propose a variation over the original GAN model to leverage the possibility of creativity called Creative Adversarial Networks (CAN). The authors claimed that the samples created in a regular GAN are not entirely innovative, and proposed changes to the model to improve the creation of novelty on the generated samples. In CAN, the discriminator outputs the probability of the sample to be an artwork and also classifies the sample into predefined possibilities of art styles. This is similar to the Conditional GAN model proposed by Odena, Olah, and Shlens (2017). The main difference is that in CAN the generator is not conditioned with additional input. The extra output of the discriminator is used only to help the generator training.

Karras et al. (2018) proposes a strategy to grow the model progressively during training, by increasing the layers in both discriminator and generator. This mechanism will make the model more complex while the training procedure runs, allowing it to generate higher resolution images at each phase. However, these layers are added progressively in a preconfigured way, i.e., they are not produced by a stochastic procedure. Karras et al. (2018) tested the proposal in different datasets, achieving better training performance and stability, and generating realistic high-resolution images.

Self-Attention Generative Adversarial Networks (SAGAN), proposed by Zhang et al. (2018a), introduces a self-attention module into GANs. Self-attention is a mechanism capable of representing the interactions between data points within a context (Vaswani et al., 2017). In SAGAN, this module was used in the generator and the discriminator, introducing the capacity to model the relationship between spatial regions of the input sample. The self-attention module works by adding a mechanism after convolutional layers that decompose the input obtained from this previous layer into two feature spaces. These feature spaces are used to learn how the model maps different regions of the input sample. When using the self-attention module, SAGAN was able to create more detailed samples when compared to the state-of-the-art.

Karras, Laine, and Aila (2018) propose a style-based architecture for the generator. In this model, the generator does not directly receive data from the latent space. First, a transformation is made to an intermediate latent space W that is feed to each convolution layer in the generator architecture. Adaptive Instance Normalization (AdaIN) is used to control the effect of the intermediate latent space. The experiments evidenced that the use of this architecture improves the quality of the created samples.

Brock, Donahue, and Simonyan (2019) presented the results for large-scale experiments with GANs, using in the generator and discriminator a higher number of parameters than in any previous experiments found in the literature. The authors found that using large batch sizes in training promotes better results concerning the quality, but made the system more prone to stability issues. Another evaluation was made about the use of a truncated normal as the latent distribution. This distribution was demonstrated to provide the best results regarding the sample quality, but with a reduction in the variety of created samples. For large models, using the truncated normal distribution affects the quality of the samples by introducing unwanted artifacts. To tackle this issue, Brock, Donahue, and Simonyan (2019) proposed the use of the following orthogonal regularization function:

$$R_\beta(W) = \beta||W^T W \odot (1 - I)||_f^2, \tag{6}$$

where W is the weights matrix of a layer, β is the hyperparameter controlling the effect of the regularization, 1 is a matrix filled with ones, and I is the identity matrix. Using the orthogonal regularization given by Eq. 6 improved the results for large models and made the truncated normal distribution viable. However, even with the application of these strategies, issues regarding training stability are still present. The authors argued that not only the individual aspects of the discriminator and the generator are paramount to the training stability, but also the interaction between them.

Other models propose to combine GANs with other neural network architectures. Makhzani et al. (2016) designed a model combining autoencoder with the adversarial mechanisms proposed for GANs. In their model, the latent code produced by the encoder is used as input for the discriminator to identify real or fake samples. This mechanism helps the decoder to produce realistic samples. Thus, in this case, it is the decoder that generates the samples and no generator is used in the solution. Other models using autoencoders were proposed in Bidirectional Generative Adversarial Networks (BiGAN) (Donahue, Krähenbühl, and Darrell, 2017) and Adversarially Learned Inference (ALI) (Dumoulin et al., 2017). In these cases, the encoder is introduced into the solution for mapping images to the latent space used in the discriminator. The generator is used instead of a decoder to produce the samples. VQGAN (Esser, Rombach, and Ommer, 2021) makes use of a transformer to enhance the GAN architecture, producing high-resolution images with better quality when compared to other models.

2.1.4.2 *Alternative Loss Functions*

Another strategy to overcome the training issues is to modify the loss functions used to guide the training of GANs. Since the original GAN proposal, researchers put forward a variety of alternative loss functions, attempting to minimize problems such as the vanishing gradient and the mode collapse problems, resulting in improved models such as Wasserstein Generative Adversarial Networks (WGAN) (Arjovsky, Chintala, and Bottou, 2017), Wasserstein Generative Adversarial Networks with Gradient Penalty (WGAN-GP) Gulrajani et al. (2017), Least Squares Generative Adversarial Networks (LSGAN) (Mao et al., 2017), and Relativistic Generative Adversarial Network (RGAN) (Jolicoeur-Martineau, 2019). A study from Lucic et al. (2017) compares the performance of alternative models such as WGAN, LSGAN, and WGAN-GP, with respect to the original GAN. In this empirical study, the experimental evaluation suggests that there are no relevant differences between the assessed alternatives and the original GAN model.

Arjovsky, Chintala, and Bottou (2017) proposed a new model derived from the original GAN called WGAN. Their approach uses a new loss function for the discriminator and the generator to improve the training stability. These loss functions are based on the Wasserstein distance between the real data distribution and the distribution created by the generator, given by:

$$W(p_x, p_g) = sup_{||D||_L \leqslant 1} \mathbb{E}_{x \sim p_x}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))], \tag{7}$$

where the loss function of the discriminator is:

$$\mathcal{L}_D^{WGAN} = \mathbb{E}_{x \sim p_x}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(x))], \tag{8}$$

and the loss function of the generator is:

$$\mathcal{L}_G^{WGAN} = -\mathbb{E}_{x \sim p_z}[D(G(z))], \tag{9}$$

where D represents the discriminator, G is the generator, p_x is the input distribution, p_g is the generated distribution given by $\mathbb{E}_{z \sim p_z}[G(z)]$ obtained from the latent space z. It is important to note that D, the discriminator, should be a 1-Lipschitz function, i.e., the gradients of the discriminator have an upper bound of 1. WGAN ensures this by clipping the weights of the discriminator to a predetermined range limited by a constant hyperparameter.

Arjovsky, Chintala, and Bottou (2017) found no occurrence of the mode collapse problems in their experiments. However, further experiments made by (Gulrajani et al., 2017) revealed that there are still stability issues, such as vanishing gradient, in the training method of WGAN.

To address some of these issues, Gulrajani et al. (2017) proposed WGAN-GP, an improvement over the original WGAN model that uses gradient penalty to enforce the Lipschitz constraint over the discriminator function. The loss for the discriminator is defined as:

$$\mathcal{L}_D^{WGAN-GP} = \mathbb{E}_{x\sim p_x}[D(x)] - \mathbb{E}_{z\sim p_z}[D(G(z))] + \lambda\mathbb{E}_{x\sim p_x}[(\|\nabla_x D(x)\|_2 - 1)^2], \tag{10}$$

where the λ parameter controls the effect of the gradient penalty in the loss. In spite of this enhancement, Mescheder, Geiger, and Nowozin (2018) showed that WGAN-GP does not always converge, i.e., the balance between the generator and the discriminator can be disrupted.

Mao et al. (2017) proposed another method called LSGAN. The contribution of this method is the use of the least squares loss function for the discriminator, given by:

$$\mathcal{L}_D^{LSGAN} = \mathbb{E}_{x\sim p_x}[D(x) - 1] + \mathbb{E}_{z\sim p_z}[D(G(z))^2]. \tag{11}$$

The loss function for the generator is given by:

$$\mathcal{L}_G^{LSGAN} = -\mathbb{E}_{z\sim p_z}[(D(G(z)) - 1)^2]. \tag{12}$$

Mao et al. (2017) claim that LSGAN is able to generate better samples and also has a more stable training when compared to the original GAN proposal.

Jolicoeur-Martineau (2019) introduced relativistic properties into the GAN model, defining new loss functions that produced the models Relativistic Standard Generative Adversarial Network (RSGAN) and Relativistic Average Standard Generative Adversarial Network (RaSGAN). The following loss functions is used in RSGAN:

$$\mathcal{L}_D^{RSGAN} = \mathbb{E}_{(x,z)\sim(p_d,p_z)}[f(D(x) - D(G(z)))], \tag{13}$$

$$\mathcal{L}_G^{RSGAN} = \mathbb{E}_{(x,z)\sim(p_d,p_z)}[f(D(G(z)) - D(x))], \tag{14}$$

The loss functions of RaSGAN are defined by:

$$\begin{aligned}\mathcal{L}_D^{RaSGAN} = {} & \mathbb{E}_{x\sim p_d}[f(D(x) - \mathbb{E}_{z\sim p_z}D(G(z)))] + \\ & \mathbb{E}_{x\sim p_z}[1 - f(D(G(z)) - \mathbb{E}_{x\sim p_d}D(x))],\end{aligned} \tag{15}$$

$$\begin{aligned}\mathcal{L}_G^{RaSGAN} = {} & \mathbb{E}_{x\sim p_d}[1 - f(D(x) - \mathbb{E}_{z\sim p_z}D(G(z)))] + \\ & \mathbb{E}_{x\sim p_z}[f(D(G(z)) - \mathbb{E}_{x\sim p_d}D(x))],\end{aligned} \tag{16}$$

where $f(x)$ is defined by $\log(sigmoid(x))$. Experiments with relativistic GANs showed that training is more stable and produced samples with better quality when compared to other solutions, such as the original GAN (Jolicoeur-Martineau, 2019).

Miyato et al. (2018) presented a new weight normalization strategy called spectral normalization. This strategy should be used in the architecture of the discriminator of existing GAN models and affects the calculation of the loss function. Models using this technique are called Spectrally Normalized Generative Adversarial Networks (SN-GANs) and obtain more diversity of samples created by the generator.

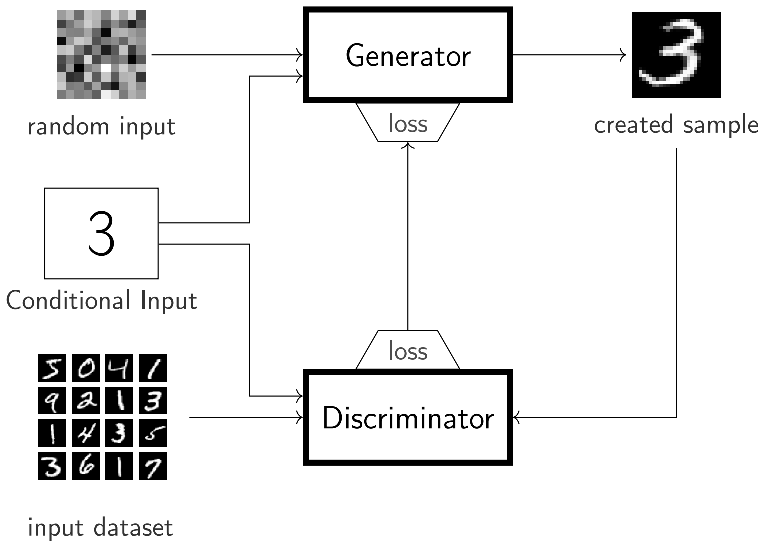


Figure 5: High-level interaction between the components of a Conditional GAN during the training process. Note the additional conditional input, indicating to the model that the number 3 should be represented.

2.1.4.3 *Conditional GAN*

Conditional GAN is a modification over the original GAN proposal that introduces a mechanism to condition the output sample created by the generator. In conditional GANs, the model usually receives another input representing the type of sample that should be created. Figure 5 shows the representation of a Conditional GAN. When compared to the representation of the original GAN model (Figure 1), we can see that a new input is supplied for the discriminator and the generator to conditioning the generated sample.

Conditional GANs have the same stability issues present in the original GAN model. Thus, they can also suffer from mode collapse and vanishing gradient. The contributions of the model are related to a new mechanism to influence the type of results created by the generator.

Mirza and Osindero (2014) presented the first proposal using this approach. In their work, the auxiliary information is provided to both the discriminator and the generator in addition to the regular inputs to conditioning the output sample.

Isola et al. (2017) developed an approach similar to the one used by Mirza and Osindero (2014) to translate from an input image to another image. For example, the additional input can be a map image, resulting in the translation of this map to a realistic aerial photo. Reed et al. (2016) also used a similar approach to generate images with conditional GANs. However, instead of an image, they used text as the conditioning input.

Odena, Olah, and Shlens (2017) proposed another strategy to condition the output created by the generator. The model, called Auxiliary Classifier Generative Adversarial Networks (AC-GAN), provided the conditioning input only to the generator. The discriminator receives the same input as the original GAN model (i.e., samples from the input dataset). However, instead of only one output representing the probability of the sample to be real (i.e., to be a sample from the input dataset), the output also includes the class of the given sample, as in Odena (2016). The generator also adds to its output the class of the created sample.

In Information Maximizing Generative Adversarial Networks (InfoGAN) (Chen et al., 2016), a latent code is added into the input to condition the output samples. An auxiliary distribution Q, represented by a neural network, is used in the regularization. The experiments conducted by Chen et al. (2016) showed that InfoGAN is able to learn disentangled representations on datasets such as MNIST and Street View House Number (SVHN). Therefore, the manipulation of the latent code is equivalent to a meaningful transformation of the output sample.

CycleGAN (Zhu et al., 2017a) uses unpaired images to train GANs for image translation. Different of other approaches using images as input, these unpaired images do not preserve the information about the corresponding pairs of samples from the input and the respective output. CycleGAN introduces a cyclic process of training for GANs. The algorithm uses two mapping functions: one that translates an image from a domain X to a domain Y and the reverse function ($Y \rightarrow X$). The discriminators D_X and D_Y are used in this process. Results show that the model can be used in applications such as style transfer and photo enhancement. The cyclic process was also used to design BycicleGAN (Zhu et al., 2017b). In this model, paired images are used to train a model in two cycles. One cycle uses an autoencoder to learn the latent code z and reconstruct the input samples. The other cycle reverses the process by using z to learn the samples and reconstruct the latent code. The result of this process is a hybrid model that is able to produce diverse results on tasks related to image translation.

Other approaches can be used to drive the training process of GANs. EAs can be applied to GANs, making use of selective pressure to achieve efficient models. We present solutions using this approach in Section 2.3. First, we introduce in the next section the concepts of EAs relevant to this thesis.

2.2 EVOLUTIONARY ALGORITHMS

Evolutionary Algorithms (EAs) are a family of algorithms inspired by the theory of natural selection proposed by Darwin (1859), simulating the evolutionary mechanism found in nature (Holland, 1992; Koza, 1992; Sims, 1994a,b). There are several variants and applications of EAs that aim to solve a diverse variety of problems. In EAs, a population of potential solutions is generated. This population is composed of individuals that represent a solution for a given problem, using a high-order abstraction to encode their characteristics called genotype. A solution derived from the genotype mapped to the problem domain is called phenotype.

Algorithm 2 represents the main steps of an EA (Goldberg, 1989; Yao, 1999). First, the population is initialized (line 1) and evaluated (line 2). After this, the algorithm works iteratively. Each iteration from the main loop (line 3) represents a generation of the population. The algorithm runs until a termination condition is met, such as the maximum amount of generations or a desired fitness value. In each generation, the individuals are evaluated using a fitness function. The outcome of the fitness functions is used on the selection of individuals for reproduction to compose the offspring. The offspring will compete with the parents for a place in the next population, and a new iteration starts. The fitness function is used in lines 2 and 6. It represents a score of each individual in solving the problem. Selection is used in line 4. Typically, a stochastic approach is used to select individuals based on their fitness. Those selected individuals are used in the reproduction step (line 5). Reproduction may use methods such as crossover between mates and mutation in the offspring generation. The last step is the replacement (line 7), where individuals in the current population are replaced by the new individuals (offspring) according to some rule (e.g., replace the least fit individuals).

Algorithm 2 General framework of an Evolutionary Algorithm (EA).

```
1: P ← initialize_population()
2: evaluate_population(P)
3: while not_terminated() do
4:     R ← select(P)
5:     C ← reproduce(R)
6:     evaluate_population(C)
7:     P ← replace(P, C)
8: end while
```

EAs can also be applied to solve multiobjective optimization problems, originating a new class of algorithms called Multi-Objective Evolutionary Algorithms (MOEAs) (Zhou et al., 2011). Nondominated Sorting Genetic Algorithm (NSGA) (Srinivas and Deb, 1994) is arguably the most popular MOEAs that uses an elitist method to implement a Pareto-based search approach. Nondominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al., 2002) is an improvement over the original NSGA that uses an algorithm to sort solutions and determine nondominated fronts to define the next populations. Besides, a crowding-distance computation is used as a second criterion to prioritize solutions in less explored spaces. This algorithm also uses an archive to keep the previously explored solutions and improve diversity. Individuals are usually inserted into the archive using a probabilistic approach.

Quality Diversity (QD) algorithms are a family of EAs aiming to improve the diversity of viable solutions discovered during the evolutionary process (Pugh et al., 2015). Novelty Search with Local Competition (NSLC) (Lehman and Stanley, 2011) uses a Pareto-based MOEA, such as NSGA-II, to promote the quality and diversity of solutions. NSLC uses as objectives the quality and novelty of individuals according to a local neighborhood. Therefore, when combined with NSGA-II, NSLC does not use the crowding-distance mechanism because the novelty criterion already produces the desired diversity.

MAP-Elites (Mouret and Clune, 2015) is another proposal in the class of QD algorithms. In MAP-Elites, an archive of phenotypes is kept to explore the diversity of high-performing solutions. Thus, at each step, an item of the archive is chosen to produce the offspring. Then, the performance is calculated and the newly generated individual is placed into the archive in the position determined by the feature space, replacing older individuals in case of better performance.

2.2.1 *Neuroevolution*

Neuroevolution is the application of EAs to automatically design a neural network. Neuroevolution can be used to evolve the weights, topology, and hyperparameters of a neural network (Stanley and Miikkulainen, 2002; Yao, 1999).

In neuroevolution, an abstract representation of the neural network is encoded in the genotype. On the other hand, the phenotype is the concrete neural network based on this abstract representation. Two approaches may be used to encode a neural network into the genotype: direct or indirect (Yao, 1999). The genotype using a direct encoding represents a neural network directly, i.e., all nodes and connections of the model are encoded into a data structure. An example of direct encoding is the use of a list of genes, where each element represents one node or a connection between nodes (Stanley and Miikkulainen, 2002). In the indirect encoding, the genotype specifies rules for the generation of the neural network. An example of indirect encoding is the use of a structured grammar to represent rules that will be derived in the process of neural network generation (Assunção et al., 2017; Lourenço, Pereira, and Costa, 2015).

When neuroevolution is used to generate a network topology, a substantial benefit is the automation of the architecture design and parameter decision, transforming a manual human effort into an automatic procedure. This automation is even more critical with the rise of deep learning that is producing deeper models and creating large search spaces (Miikkulainen et al., 2017). However, the size of the search space is also a challenge for neuroevolution (Martinez et al., 2021), such as the time-consuming executions that may turn their application unfeasible.

One of the first neuroevolution examples is the work by Moriarty and Miikkulainen (1997) Symbiotic, Adaptive Neuro-Evolution (SANE). SANE works by evolving individual neurons and blueprints. These neurons are the components of a hidden layer in a two-layered neural network. Each neuron is composed of a set of weights that connects it with inputs and outputs. A blueprint is a recipe that chooses a subset of components from the neurons population to form a complete neural network. Thus, SANE evolves both the population of neurons and the way they should be connected, i.e., the blueprint. In the evaluation phase, each element of the population of blueprints is transformed into a neural network by selecting individuals in the population of neurons. This network is evaluated against a parasite opponent to obtain a fitness value. This fitness is used in offspring generation for both neurons and blueprints.

NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002) is a neuroevolution algorithm that evolves the weights and topologies of neural networks. In NEAT, a Genetic Algorithm (GA) is used to encode and evolve neural networks. The genotype of the individuals contains two lists representing the nodes (neurons) of the neural network and the connections between them. The initial population contains individuals with small genomes, that will incrementally become more complex through generations by gradually adding more nodes and connections to the neural networks. The algorithm also proposes the use of a mechanism to promote novelty in the population. This mechanism works by assigning an innovation number to each gene to mark its origins. The population is divided into species based on the similarity of the genomes, that is calculated based on the innovation number of the genes. NEAT also uses fitness sharing to represent the fitness of each individual belonging to a species. Experiments demonstrated that the strategies used in NEAT were efficient and useful in the promotion of innovation. Stanley and Miikkulainen (2004) conducted further experiments, arguing that NEAT was also successfully applied in a coevolution context.

The NEAT model was also expanded to work on larger search spaces, such as deep neural networks, in the Deep NeuroEvolution of Augmenting Topologies (DeepNEAT) and Coevolution DeepNEAT (CoDeepNEAT) methods (Miikkulainen et al., 2017). In DeepNEAT, genes in the genome represent entire layers of a deep neural network and the connection between them. Furthermore, CoDeepNEAT combines the ideas used in SANE with DeepNEAT to introduce a two-level evolutionary mechanism that evolves modules and blueprints, promoting the evolution of repetitive structures.

Assunção et al. (2018, 2019) proposed a method to search for efficient topologies and also to tune the hyperparameters, called Deep Evolutionary Network Structured Representation (DENSER). DENSER uses a two-level algorithm to evolve the topologies and parameters of deep neural networks. The first level is concerned with the overall architecture of the network. The second level is responsible for giving the inner parameters of each layer. DENSER uses GA on the first level and grammatical evolution on the second level. The variation operators affect these two levels by specific types of mutation. Crossover was also used in the reproduction process. This proposal was evaluated on the MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100, achieving performance similar to other manually designed state-of-the-art proposals.

When compared to other automatic methods, DENSER was able to outperform their results in the experiments made by the authors.

2.2.2 *Coevolution*

The simultaneous evolution of at least two distinct populations (also denominated species) is called coevolution (Hillis, 1990; Rawal, Rajagopalan, and Miikkulainen, 2010). There are two types of coevolution algorithms: cooperative and competitive. In cooperative coevolution, individuals of different species cooperate in the search for efficient solutions, and the fitness function of each species is designed to reward this cooperation. Several works used cooperative coevolution to evolve neural networks, such as in García-Pedrajas, Hervás-Martínez, and Muñoz-Pérez (2003), García-Pedrajas, Hervás-Martínez, and Ortiz-Boyer (2005), Gomez, Schmidhuber, and Miikkulainen (2008), and Potter and De Jong (1995). In competitive coevolution, individuals of different species are competing between them in the search for better solutions. This competition is designed by the problem domain and can be, for example, players competing in a game (Ficici and Pollack, 2003) or robots dueling for resources (Nelson, Grant, and Henderson, 2004). In this case, their fitness function directly represents this competition in a way that scores between species are inversely related.

The work by Hillis (1990) is one of the first to address evolution with competitive coevolution. In his work, the adversarial populations are characterized as hosts and parasites. Following this terminology, a predator-prey approach was used to model the coevolutionary algorithm. Specifically, individuals from the population of predators (parasites) hunt the less-fitted individuals from the population of preys (hosts).

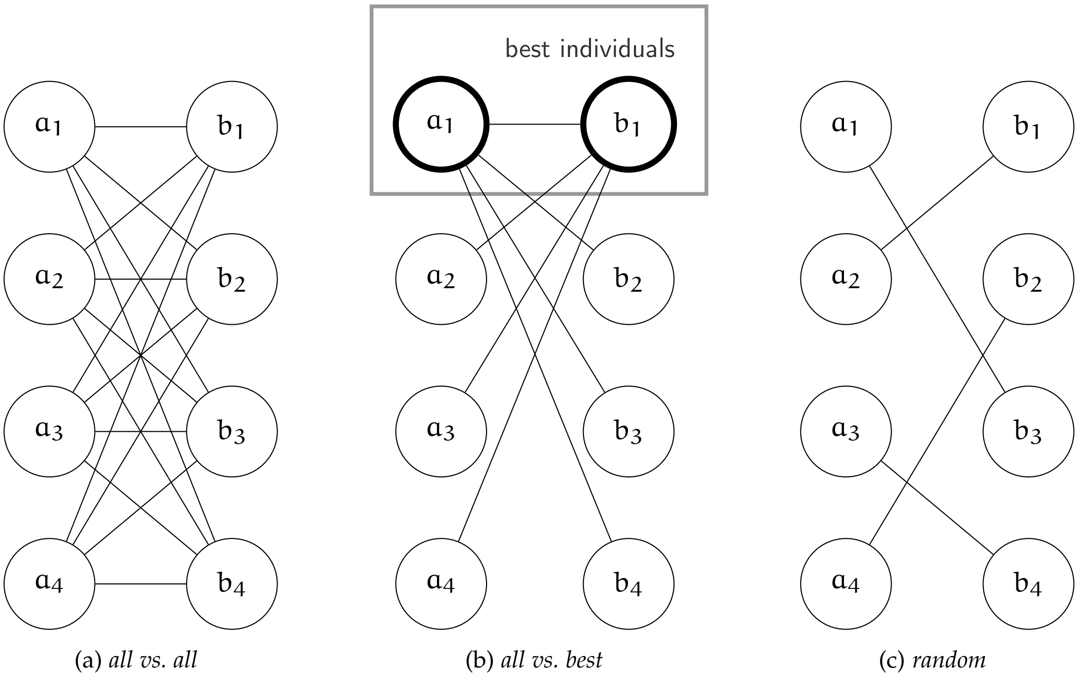
There are several other approaches to model the interaction from different populations in coevolution (Antonio and Coello, 2018; Sims, 1994a). Figure 6 shows examples of the following strategies:

- *all vs. all*: this approach pairs each individual from one population with each individual in the other population (Figure 6(a));

- *all vs. best*: this strategy pairs each individual from one population with the best-fitted individual in the other population (Figure 6(b));

- *random*: this approach randomly matches individuals across populations (Figure 6(c)).

The use of coevolution in EAs can cause some issues, such as intransitivity and disengagement (Antonio and Coello, 2018; Mitchell, 2006). The intransitivity occurs when a solution a is better than b and b is better than c, but this does not guarantee that a is better than c. This issue can lead to cycling between these solutions during the evolutionary process, breaking the progress of individuals toward optimal solutions. Disengagement occurs when the equilibrium between the populations is broken. In



(a) *all vs. all* (b) *all vs. best* (c) *random*

Figure 6: Representation of (a) the *all vs. all*, (b) *all vs. best*, and (c) *random* pairing strategies.

this case, individuals from one population are much better than individuals from the other, leading to ineffective progression.

There are some other approaches to apply coevolution in the context of EAs. Spatial coevolution was studied by Mitchell (2006) and Williams and Mitchell (2005). In spatial coevolution, individuals are spatially distributed and a spatial logic is used to pair individuals on the evaluation phase. For example, Mitchell (2006) made experiments distributing the individuals among a toroidal grid, where each cell holds one individual for each coevolutionary population. The fitness was calculated using individuals from each cell, and the selection used the neighborhood of these cells to rank the individuals. The crossover also follows these spatial rules defined by this strategy. This strategy was found to overcome the issues with coevolution strategies mentioned before. Williams and Mitchell (2005) found that spatial coevolution promotes diversity in the population. They also found that spatial coevolution promotes individuals that target specific weaknesses of individuals from the other population in the neighborhood.

2.2.3 *Competitive Coevolution and Neuroevolution*

In this section, we detail existing proposals of competitive coevolution algorithms in the evolution of neural networks in some adversarial domains, such as competitive robots and game playing.

Sims (1994a) proposed a model to evolve virtual creatures by competition in a simulated environment. The author used a coevolution algorithm to evolve the morphology and the neural controller systems of a virtual creature. The creatures may be seen as articulated robots with logical controllers performing autonomous actions. The simulated environment had a cube in its center that is the primary objective of the duel. The creature that reached the cube first within a certain amount of time was declared the winner. The fitness function was proportional to the time the creature had the cube under control. This promotes the evolution of individuals that not only win but also defeat an enemy by a large margin. A directed graph was used as the mechanism to represent the genotype of an individual. The genotype to phenotype transformation in respect to morphology is based on a hierarchy of articulated three-dimensional rigid parts. Each node in the directed graph of the genotype has information about the creature, such as the dimensions of the rigid part, the joint-type of a connection between parts, and a set of neurons that controls the part. This set of neurons is responsible for the behavior of creatures. Neurons receive as inputs the information sensed by rigid parts and output forces or torques to be used by joints between them. During the evolution of creatures, both the neural network structure and the weights are adjusted. Crossover and mutation are used to generate offspring based on the surviving parents of an iteration. In each iteration, two creatures are selected to enter into a duel in the simulated environment. The pairing alternative selected by the author was *all vs. best*: the best-performing individual is chosen to be paired with all other individuals. The model was evaluated both for single-species and two-species evolution. The results are better for two species, where creatures in each species developed their morphology and neural controllers efficiently to reach the cube and create counter strategies against the other species.

Lubberts and Miikkulainen (2001) proposed an approach to evolve a neural network to play the game of Go. Since it is difficult to find a good opponent to play against the proposed algorithm, the authors suggested replacing this opponent by using competitive coevolution. They used neuroevolution techniques along with a coevolutionary algorithm to evolve neural networks in the task of playing Go. To make competitive coevolution possible, they created two species, called hosts and parasites, using the terminology of Hillis (1990). Hosts have the objective of finding an optimal solution. Parasites try to defeat hosts by taking advantage of their weaknesses. The SANE (Moriarty and Miikkulainen, 1997) system was adapted to be used for evolving two adversarial populations simultaneously. For performance reasons, the authors limited the experiments to a 5x5 board. The overall performance was compared with a model using standard evolution to evolve a solution to Go, and the proposed method outperforms it by a large margin. Therefore, Lubberts and Miikkulainen (2001) concluded that coevolution is useful in generating solutions for adversarial problems, mostly when it is hard to have a good opponent for these problems to evaluate these evolving solutions.

Stanley and Miikkulainen (2004) described a method that used coevolution to evolve complex neural network architectures to be used as robot controllers. They used the NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002) method and applied it to a competitive environment: a robot duel. In this environment, two robots have to fight against each other and the last one standing wins. The robot controller is a neural network that receives several inputs (e.g., energy supplies position, energy left, enemy position, and enemy energy) and outputs the strength that should be applied to the motors that are responsible for the movement of the robot. As a competitive coevolution method, the fitness function depends on the competition between robots in this duel. This fitness was used in the regular NEAT method, and neural networks evolve following the rules defined in NEAT. In NEAT, individuals start small and become complex through generations. The hypothesis that this complexification process is important in competitive coevolution was validated experimentally by the authors. Thus, the proposed model was evaluated against the following variations: fixed-topology coevolution of large networks, fixed-topology coevolution of small networks, fixed-topology coevolution of best complexifying network, and simplifying coevolution. Stanley and Miikkulainen (2004) concluded that complexification, a characteristic inherited by using NEAT, is paramount to discover complex and optimized solutions in a coevolutionary domain.

Chong, Tan, and White (2005) proposed a coevolution model to develop solutions for playing Othello (also known as Reversi). The model evolves neural networks that evaluate solutions proposed by minimax search strategies. As the topology of the network is fixed, the model is focused on the evolution of weights. The neural network takes the board state as input and outputs a score representing the evaluation of this state. The selection process uses a tournament between pairs of networks, where the ten best networks are selected to be in the next generation. The selected networks will also be the parents used to generate the offspring, in which only the mutation operator will be used in the breeding process. The authors compared the solutions obtained by this process using external computer players. Solutions generated by the proposed model outperformed the external computer player in most of the games played. So, Chong, Tan, and White (2005) concluded that the use of coevolution is useful in the context of an evaluation function used in a minimax search.

Monroy, Stanley, and Miikkulainen (2006) proposed a combination of NEAT and Layered Pareto Coevolution Archive (LAPCA) (a Pareto coevolution strategy). In this method, LAPCA was used as the coevolution memory in order to direct the evolutionary path towards better solutions. Therefore, the EA used NEAT guided by LAPCA on the population control. The authors argued that the use of LAPCA reduced the number of iterations needed in the EA. Besides, the use of LAPCA in the experiments evidenced a better performance when compared to other approaches (e.g., random sampling).

Rawal, Rajagopalan, and Miikkulainen (2010) proposed a model that used both competitive and cooperative coevolution to evolve a neural controller for robots in a predator and prey domain. The model starts with a fixed architecture that was previously designed by hand, and they were only concerned with the evolution of the neural controller weights. In the first set of experiments, the authors found that the architecture was not enough to generate efficient solutions. Thus, they improved the architecture of the neural controller by hand, creating a hierarchical strategy using multiple neural networks. With this new architecture, the results were positive and complex solutions were generated to address the problem. Therefore, the authors concluded that the model proposed by them was able to create complex, cooperative, and competing solutions to the predator and prey scenario. However, it is important to note that the initial architecture problem might be automatically addressed by a model that evolves both the weights and topology of the neural network.

2.3 EVOLUTIONARY ALGORITHMS AND GENERATIVE ADVERSARIAL NETWORKS

Several aspects that compose the Generative Adversarial Networks (GAN) model can be subject to evolution in an Evolutionary Algorithm (EA). However, it is important to keep in mind that the EA should preserve the balance of these components to address the stability issues listed in Section 2.1.1. In this section we present and discuss the possibilities for the application of EA to optimize the GAN model. The options related to neuroevolution and the aspects of GANs are presented as possible choices to design an algorithm. Next, we describe works that use EA for training and evolution of GANs.

Other non-evolutionary techniques were also proposed to automatically search for efficient architectures of GANs. AutoGAN (Gong et al., 2019) applies a multi-level architecture search for generators. AdversarialNAS (Gao et al., 2020) uses a differentiable method to search for efficient architectures not only for generators but also for discriminators. However, these works are beyond the scope of this research and we focus specifically on the use of EA with GANs.

2.3.1 *Analysis of Evolutionary Aspects of GANs*

Neuroevolution can be fully applied in the context of GANs. The evolution of the topologies of the discriminator and the generator should take into account that the equilibrium between them is paramount to the convergence of the training process. Not only the structure (i.e., the number of layers and the connections between them) but also the internal characteristics of each layer composing a neural network can be the subject of evolution. For example, the type of a layer (e.g., convolution or fully connected), the number of output features, and the activation function (e.g., ReLU, ELU, Tanh). Other aspects relevant to the network can also be a variable of the individual, such as the choice for the optimizer used in the training, the learning rate, the batch size, and the number of training iterations.

We can also make use of other techniques regarding Evolutionary Computation (EC) in neuroevolution, such as coevolution. For instance, GANs can be modeled as a competitive coevolution problem, where we consider a population of discriminators competing with a population of generators. Therefore, an EA can make use of competitive coevolution concepts to match individuals from these two populations at the evaluation phase. Furthermore, we can relate problems that frequently affect the training of GANs (Section 2.1.1) to coevolution problems. For example, the vanishing gradient can be linked to the disengagement issue. Thus, the use of coevolution and strategies to avoid its common problems can be explored in combination with other techniques (e.g., neuroevolution) to solve the challenges of the GAN training process.

2.3.2 *Current Proposals*

In this section we present the state-of-the-art on the application of EA in GANs. This work builds upon the analysis of Costa et al. (2020b) expanding the scope to other models recently proposed. We describe the solutions, focusing on the choices concerning the aspects of the EA and the characteristics of GANs, showing the characteristics of the algorithms concerning the selection method, fitness functions, variation operators, evaluation, and experiments.

2.3.2.1 *Progressive Growing of GANs*

Karras et al. (2018) proposed a training method that uses a predefined strategy for progressive growing the architecture of a GAN. As the model progresses in a preconfigured way, the proposal of Karras et al. (2018) does not use EAs in the training process. This method shares characteristics with NEAT (Section 2.2.1), the neuroevolution algorithm proposed by Stanley and Miikkulainen (2004). Both methods rely on a simple start point and make use of complexification strategies in the search for better solutions. However, NEAT uses a coevolution approach, while Karras et al. (2018) uses a predefined procedure, relying on the previous knowledge about the problem to manually predefine a good start point and the progression path. Therefore, as the evolutionary path is predefined, Karras et al. (2018) do not use a population to search for different solutions. Their method is equivalent to the use of a single individual through the training process.

2.3.2.2 *E-GAN*

Evolutionary Generative Adversarial Networks (E-GAN)[1] was one of the first proposals to use EAs to optimize GANs (Wang et al., 2018). Their approach evolves GANs using a mutation operator that selects the loss function of the individuals between a set of predefined possibilities. They also use minimal populations of individuals, only to capture the possibilities of losses predefined in the definition of the model. The evolution occurs only in the generator, and a single-fixed discriminator is used as an adversary for the population of generators. The network architectures for the generator and the discriminator are fixed and based on DCGAN (Radford, Metz, and Chintala, 2015).

The possibilities for the loss functions are implemented through three mutation operators: minimax, heuristic, and least-squares mutation. In this case, the population of generators is composed of three individuals, each one representing one of the possible losses. The minimax mutation follows the original GAN objective given by Eq. (3), minimizing the probability of the discriminator to detect fake samples. On the other hand, the heuristic mutation aims to maximize the probability of the discriminator making mistakes regarding fake samples. The least-squares mutation is based on the objective function used in LSGAN (Mao et al., 2017). Each loss function in the predefined set focused on an objective to help in the GAN learning process.

Two criteria were used as fitness in the evaluation phase of the algorithm. The first, called quality fitness score, is defined as:

$$F_q = \mathbb{E}_{z \sim p_z}[(D(G(z)))], \tag{17}$$

that is similar to the loss function used in the generator of the original GAN model (Eq. (1)). The second criteria, called the diversity fitness score, is defined as:

$$F_d = -\log \|\nabla_D - \mathbb{E}_{x \sim p_x}[\log(D(x))] - \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]\|. \tag{18}$$

In Eq. (17) and Eq. (18), z, p_z, p_x, G and D represent the latent space, the latent distribution, the input dataset, the generator, and the discriminator, respectively. These two fitness criteria are combined as follows:

$$F = F_q + \gamma F_d, \tag{19}$$

where the γ parameter is used to regulate the influence of the diversity criteria on the final fitness.

At each generation, individuals are evaluated following their specific loss function, and only the best-fitted generator survives. In the next generation, the survivor individual is used to train the discriminator and to generate the three descendants for the next evaluation.

[1] Code available at https://github.com/WANG-Chaoyue/EvolutionaryGAN.

The E-GAN model was evaluated on the CIFAR-10, LSUN and CelebA datasets. The Inception Score was used as the metric to analyze the results. Wang et al. (2018) concluded that E-GAN improved the training stability and achieved satisfactory performance, outperforming other methods in specific scenarios.

Li et al. (2021) propose a framework C-GAN that incorporates a crossover operator that can be integrated into other evolutionary models for GANs. The operator is based on $\mathcal{Q}$-filtered distillation crossover (Bodnar, Day, and Lió, 2020). The process consists of using one parent as the basis of the offspring model and select the generated data of parents to be used on the learning process for the offspring. The authors included C-GAN in E-GAN to produce a new model called Evolutionary Generative Adversarial Networks with Crossover (CE-GAN)[2]. Experiments on CIFAR-10 show that CE-GAN is capable of achieve competitive results when compared to E-GAN with respect to Inception and FID scores.

2.3.2.3 *Pareto GAN*

A neuroevolution approach for training GANs was proposed by Garciarena, Santana, and Mendiburu (2018). Although not named by the authors, we refer to this solution as Pareto GAN[3]. The proposal uses a genetic algorithm to evolve the architecture of the neural networks used for both the generator and the discriminator. A single individual (G_i, D_i) is used to represent both the generator and the discriminator in the EA.

The crossover operator combines two parents exchanging the discriminator and the generator between them. For example, a crossover between the individuals (G_1, D_1) and (G_2, D_2) produces the children (G_1, D_2) and (G_2, D_1). The crossover operator does not change the internal state of the generator and the discriminator in each individual. To accomplish this, a set of possible mutations is applied to individuals when creating a new generation.

Regarding the architecture of the neural networks the mutation can change, add or remove a layer. When changing a layer, the operator can modify the weights, the activation function, and/or the loss function used in the GAN algorithm. Other modifications include changing the characteristics of the algorithm, such as the number of iterations for the generator and the discriminator when applying the GAN training algorithm to an individual.

A benchmark for GANs based on the problem of Pareto set approximations was also proposed (Garciarena, Santana, and Mendiburu, 2018). The comparison between the Pareto front of a solution and the real front is used to assess the quality of the samples and can also identify issues, such as the mode collapse problem. Therefore, the Inverted Generational Distance (IGD) (Coello and Sierra, 2004) was used as fitness

[2] Code available at https://github.com/AlephZr/CE-GAN.

[3] Code available at https://github.com/unaigarciarena/GAN_Evolution.

to drive the EA. The IGD measures the smallest distance between points in the true Pareto front and in the Pareto front approximation and is given by:

$$\mathrm{IGD} = \frac{1}{|R|}\left(\sum_{r\in R} min_{a\in A}\, d(r,a)^{p}\right)^{\frac{1}{p}}, \quad d(r,a) = \left(\sum_{k=1}^{m}(r_k - a_k)^2\right)^{\frac{1}{2}}, \tag{20}$$

where R is the real Pareto front, A is the Pareto approximation, and m is the number of vectors in R.

The evaluation phase will transform each individual (G_i, D_i) into a concrete GAN, composed of a discriminator and a generator, that will be trained according to the regular GAN algorithm. The fitness is calculated, and the selection uses the Pareto-dominance to compose the offspring that will form the next generation.

The proposed solution was evaluated using bi-objective functions as the input data, each one with 10 input variables. A population of 20 individuals, evaluated for 500 generations, was used in the experiments. The authors concluded that the algorithm was able to found architectures for the discriminator and the generator that improve the Pareto set approximation. The experiments do not include evaluations with image datasets. However, experiments using the same data dimension as the MNIST dataset, i.e., with 784 input variables, were also conducted. The authors demonstrated that the solution is scalable to this dimension, as the results showed that useful architectures were also found in this case. Furthermore, a study of transferability study demonstrated the capacity of generalization of the solution to different problems (Garciarena, Mendiburu, and Santana, 2020).

2.3.2.4 *Lipizzaner*

Al-Dujaili et al. (2018) proposed a model called Lipizzaner[4] that defines a coevolutionary framework to train and evolve GANs. In Lipizzaner, the evolution occurs only on the internal parameters of the generator and discriminator, such as the weights of their neural networks. Thus, the network architecture used in both the discriminator and generator is fixed and defined *a priori*. The architecture varies with the dataset used in the experiments.

The fitness used in Lipizzaner for the generators and discriminators is based on the GAN objective function, defined as:

$$\mathcal{L}(u,v) = \mathbb{E}_{x\sim p_{data}}[\phi(D_v(x))] + \mathbb{E}_{x\sim G_u}[\phi(1 - D_v(x))], \tag{21}$$

where ϕ is a concave function in the interval $[0,1]$, p_{data} is the input dataset, G_u is the generator with the parameters u, and D_v represents the discriminator with its parameters v.

[4] Code available at https://github.com/ALFA-group/lipizzaner-gan.

At the evaluation step, $\mathcal{L}(u_i, v_j)$ is calculated for each pair (G_i, D_j), and the fitness values are updated as $f_{u_i} \mathrel{-}= \mathcal{L}(u_i, v_j)$ and $f_{v_j} \mathrel{+}= \mathcal{L}(u_i, v_j)$ for generators and discriminators, respectively.

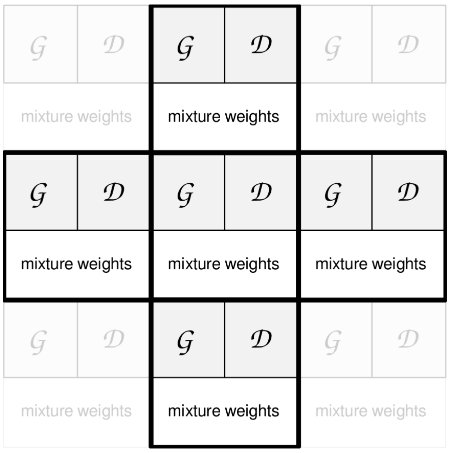


Figure 7: A 3×3 grid representing the spatial coevolution mechanism used in Lipizzaner. The neighborhood of the central cell includes the four-highlighted nodes in the grid. Each cell contains one discriminator, one generator, and a mixture of weights. Image adapted from Al-Dujaili et al. (2018)

Spatial coevolution (Mitchell, 2006; Williams and Mitchell, 2005) was used to design the algorithm that trains and evolves generators and discriminators. Individuals are allocated on a two-dimensional toroidal grid, where each cell contains individuals from the generator and discriminator populations. In the evaluation phase, the EA matches individuals in neighbor cells following a coevolutionary pairing approach. A five-cell neighborhood was used to determine these interactions. Figure 7 displays an example of a 3x3 grid with the spatial coevolution strategy used in Lipizzaner.

Lipizzaner uses two mutation operators. The first operator mutates the learning rates of the optimization method used in the generator and the discriminator. In this case, a normal distribution is used to change the learning rate at small steps in each generation. The second operator is a gradient-based mutation that updates the weights of the individuals in the populations of generators and discriminators. The generator is determined as a mixture of generators in this neighborhood. Furthermore, an evolution strategy combined with a performance metric (e.g., the Inception Score or FID) is used to update the mixture of weights.

The model was evaluated on the MNIST and CelebA datasets, using a 2×2 grid, forming a population of 4 generators and 4 discriminators. These populations were evolved through 400 generations and the Adam optimizer (Kingma and Ba, 2015) to update the weights. The authors found that Lipizzaner was able to avoid the mode collapse problem in most of the experiments. Moreover, the authors argued that the model can recover from the mode collapse issue and continue to improve as the training advances through next generations. Further experiments demonstrate the contributions of the components in Lipizzaner through ablation studies (Hemberg et al., 2021).

A further improvement proposes to optimize the spatial coevolution strategy, creating a new model called Lipi-Ring (Toutouh and O'Reilly, 2021). In Lipi-Ring, a ring topology is used instead of a toroidal grid. When compared to Lipizzaner, Lipi-Ring is able to achieve similar performance but with lower computational resources.

2.3.2.5 *Mustangs*

MUtation SpaTial gANs (Mustangs) combines E-GAN and Lipizzaner in a hybrid approach to train and evolve GANs[5] (Toutouh, Hemberg, and O'Reilly, 2019). As in Lipizzaner and E-GAN, the topologies of the generator and discriminator are kept fixed during the evolutionary process.

The Mustangs model uses the same fitness strategy used in Lipizzaner (Eq. (21)). At each evaluation step, the value $\mathcal{L}(u_i, v_j)$ is computed for each pair (G_i, D_j), and the fitness values are updated as $f_{u_i} -= \mathcal{L}(u_i, v_j)$ and $f_{v_j} += \mathcal{L}(u_i, v_j)$ for generators and discriminators, respectively.

The variation operators used in Mustangs are a combination of the ones used in Lipizzaner and E-GAN. Therefore, as in E-GAN, the loss function of the individuals can be changed. However, the strategy used here is to randomly select one of the three possibilities for the loss function, instead of evaluating the individuals using all losses. The goal is to increase the diversity of genomes in the population. In addition, Mustangs also uses the mutation operators and the strategy to update weights of Lipizzaner. Crossover is not used in this proposal.

The evaluation phase follows the same proposal as Lipizzaner. As in Lipizzaner, Mustangs uses spatial coevolution to pair discriminators and generators, using a toroidal grid to spatially distribute the individuals. Individuals are matched using the grid neighborhood to calculate the fitness and evaluate each individual. The generator is determined as a mixture of generators in this neighborhood, the same strategy used in Lipizzaner.

Mustangs was evaluated with the MNIST and the CelebA datasets. As the architectures of the neural networks that compose a GAN are fixed and predefined, the authors chose different topologies according to the dataset used in the experiments. A four-layer MLP network with 700 neurons and a DCGAN-based architecture was used for the experiments with the MNIST and the CelebA dataset, respectively. For MNIST, a grid size of 3x3 was used with a time limit of 9 hours. For CelebA, the experiments were executed with a 2×2 grid for 20 epochs. A comparison between standard GAN, E-GAN, Lipizzaner, and Mustangs was presented. The authors found that Mustang is able to generate the best results concerning the FID score. They also concluded

[5] Code available at https://github.com/mustang-gan/mustang.

that spatial coevolution is an efficient way to model the population of generators and discriminators to train GANs.

2.3.2.6 *CDE-GAN*

Cooperative Dual Evolution based Generative Adversarial Network (CDE-GAN)[6] (Chen et al., 2020) evolves a separated population of generators and discriminators to train GANs. A soft mechanism was proposed to promote a stable interaction between generators and discriminators in the model. This mechanism is paramount in CDE-GAN for the application of the GAN training and keeps the balance between generators and discriminators, avoiding problems such as the vanishing gradient. Thus, discriminators will be weakened to allow the generator to gradually learn the input distribution. In practice, the losses achieved by generators are softened by applying softmax weighted arithmetic average over the respective adversarial discriminators.

The architecture of generators and discriminators are fixed and do not evolve through generations. Mutation is used as the variation operator to switch the loss functions of generators and discriminators. Thus, CDE-GAN uses a mechanism similar to E-GAN, but also adding the possibility of using different loss functions not only on generators but also on discriminators. Individuals can use the loss functions of the original GAN model (the non-saturated and the minimax versions) and from LSGAN.

The offspring is created based on the best individuals in the generation. For this, the fitness of individuals is calculated based on the interaction between generators and discriminators.

Experiments were conducted using CIFAR10, LSUN, and CelebA datasets. The architectures used in the experiments were based on DCGAN and also on a simple architecture based on MLP. Results show that CDE-GAN is able to avoid mode collapse and promote stable training. It also shows consistent performance when trained on complex image datasets.

2.3.2.7 *Other Approaches*

Bharti, Biswas, and Shukla (2021) propose Evolutionary Multiobjective Cyclic GAN (EMOCGAN), a model that uses MOEA to tackle the stability problem of GANs. It is based on conditional GANs (Section 2.1.4.3), using cyclic GANs as the base model for the application of the EA. Evolutionary mechanisms of E-GAN are also used to provide variation and selective pressure for individuals. Experiments with EMOCGAN were conducted on datasets related to image-to-image translation. Results show that the proposed EA is able to outperform a non-evolutionary cyclic GAN model.

Category-aware GAN (CatGAN)[7] (Liu, Wang, and Liang, 2020) proposes the use of EA to drive the training of GANs for text generation. The model uses a hierarchical

[6] Code available at https://github.com/shiming-chen/CDE-GAN.

[7] Code available at https://github.com/williamSYSU/CatGAN.

EA to stabilize the GAN training, pursuing an equilibrium between quality and diversity on the generated text samples. The algorithm evolves a population of generators that are trained and evaluated in a dynamic environment given by a specific discriminator. For this, a new objective function was proposed to take into account categorical data. At each generation, individuals in the population of generators reproduce based on mutation. Hierarchical selection is applied to compose the next generation. Experiments were conducted using text datasets to compare the proposed evolutionary model with other non-evolutionary GAN models. The results show that CatGAN is able to generate text with quality and retaining diversity, outperforming most of the other models evaluated in the experiments.

Other models rely on different approaches to evolve GANs. Cruz, Acosta-Mesa, and Mezura-Montes (2021) use Particle Swarm Optimization (PSO) to design and train GANs to generate biomedical Chest X-Ray images of pneumonia. In this model, the architectures of the neural networks grow progressively and are evaluated by using the FID score.

2.3.3 *Discussion*

Section 2.3.2 presented the current proposals that apply EAs in the context of GANs. In recent years we have seen several proposals that rely on EAs to optimize the different components of GANs. Next, we present and discuss these characteristics regarding the aspects of the GAN model used in the proposals, the choices concerning the EA, and the experimental results.

2.3.3.1 *Characteristics of the GAN model*

Table 1 presents choices with respect to the GAN model used in each proposal. These proposals are compared under the perspective of four attributes: the number of discriminators used in the algorithm, the number of generators, the architecture of each component, and the loss function used to train the GAN.

Except for E-GAN, all proposals used multiple discriminators. For the generators, all proposals used multiple generators, with E-GAN using a fixed number of three generators, corresponding to the number of possible loss functions designed in the algorithm. Thus, E-GAN works with small populations, limiting the evolutionary options that can emerge through generations. With variation operators similar to E-GAN, CDE-GAN also works with small populations. On the other hand, Mustangs adapted successfully the E-GAN model in the context of a larger population, using the spatial coevolution approach of Lipizzaner to handle the individuals.

Regarding the architecture, only Pareto GAN allows it to be modified during the evolutionary process. The other proposals used a predefined and fixed architecture for the neural networks of generators and discriminators. As such, Pareto GAN works

Table 1: Aspects of the GAN used in the evaluated proposals.

Algorithm	Discriminator	Generator	Architecture	Loss Function
E-GAN	single-fixed	three	DCGAN-based	evolvable[1]
Pareto GAN	many	many	evolvable	evolvable[1]
Lipizzaner	many	many	MLP and DCGAN-based[2]	original GAN
Mustangs	many	many	MLP and DCGAN-based[2]	evolvable[1]
CDE-GAN	many	many	MLP and DCGAN-based[2]	evolvable

[1] The loss function is selected using a predefined set of possibilities.

[2] The DCGAN-based architecture was used with the CelebA dataset and a simpler approach was applied with the MNIST dataset (see Section 2.3.2.4 and Section 2.3.2.5)

with larger search spaces, as the architectures that can emerge from the EA have a high number of possibilities. It has also the potential to handle the balance between generators and discriminators, as the complexity of the architecture is determined by the algorithm.

Concerning the loss function, Lipizzaner uses a fixed one for the GAN training, whilst CDE-GAN, E-GAN, Pareto GAN, and Mustangs allow it to be modified by evolution. This approach uses a set of predefined possibilities to select and attribute a loss function to an individual. A more flexible approach can also be used instead of a predefined set, using genetic programming to discover better loss functions for GANs. However, the proposals analyzed in this chapter did not explore this approach.

2.3.3.2 *Components of the Evolutionary Algorithm*

Table 2 presents a comparison between the solutions presented in Section 2.3.2, focusing on the components of the evolutionary algorithm, namely: the pairing approach, the variation operators, the fitness function, and the selection method.

As multiple generators and/or discriminators are used in all proposals, and the GAN training occurs using generators and discriminators as adversarial, we need a mechanism to pair the individuals. With the exception of Pareto GAN, all other solutions use separated individuals to represent discriminators and generators. In E-GAN, as there are only a single discriminator and three generators, the policy for pairing is to use the discriminator to evaluate all three generators. Lipizzaner and Mustangs use a spatial coevolution strategy to match generators and discriminators. It is important

Table 2: Components of the evolutionary algorithm used in the evaluated proposals.

Algorithm	Pairing	Variation Operators	Fitness	Selection
E-GAN	one-vs-three	mutation (loss)	custom	best individual
Pareto GAN	–	crossover and mutation	IGD	Pareto dominance
Lipizzaner	spatial coevolution	mutation (weights)	GAN objective	spatial
Mustangs	spatial coevolution	mutation (weights, loss)	GAN objective[1]	spatial
CDE-GAN	all-vs-bests	mutation (loss)	custom	best individuals

[1] The FID score is used as the performance metric to evolve the mixture of weights in Mustangs.

to note that the spatial coevolution mechanism applied in Lipizzaner and Mustangs uses the mixture of weights from the neighborhood to compose the weights of the generator in each cell, taking advantage of multiple individuals to produce a single model. The other solutions do not apply an analogous mechanism to combine weights from different individuals. Pareto GAN has individuals with diverse architectural characteristics in the population, preventing the use of the mixture mechanism designed for Lipizzaner and Mustangs. CDE-GAN uses the best individuals from the generation to compose pairs on the breeding and evaluation of the offspring.

The variation operators are paramount to provide diversity in the search for good solutions in an EA. Pareto GAN uses crossover and mutation as operators. It is also the solution that provides the most variability regarding the elements that can be evolved through generations in the EA. As Pareto GAN models its individual as a representation of the entire GAN, i.e., encoding both the discriminator and the generator into the genotype, the crossover works exchanging the generator and the discriminator between two parents to form the offspring. The other solutions modeled the GAN with independent genotypes to represent the generator and the discriminator. Therefore, this approach is not applicable to them.

Pareto GAN has evolvable neural network architectures for discriminators and generators. The mutation operator is used to provide small changes in these architectures that are built through generations to produce strong discriminators and generators. E-GAN, Lipizzaner, and Mustangs use a restricted mutation strategy. In E-GAN, only the loss function can be evolved. CDE-GAN uses a similar approach but applying different loss functions not only to discriminators but also for generators. In Lipizzaner, gradient-based mutations are applied to update the weights of generators and discriminators. Furthermore, Lipizzaner uses an evolution strategy to update the mixture of weights used for generators. Mustangs combines the operators of E-GAN and Lipizzaner.

The choice for fitness is diverse among the proposals. E-GAN uses a custom function that represents the quality and diversity of the created samples. As only the generator is subject to evolution, the discriminator does not have a fitness associated. CDE-GAN uses a similar approach but also introduces the fitness for discriminators. Pareto GAN based its fitness on the concepts of the Pareto front, using the IGD to represent the fitness value. Lipizzaner and Mustangs use the GAN objective function to calculate the fitness for the individuals. In addition, the FID score was used as the performance metric to evolve the mixture of weights by Toutouh, Hemberg, and O'Reilly (2019).

The selection method used in E-GAN is based on the choice of the best generator. As E-GAN has only three generators, each one with a specific loss function, the fitness guide the evolution by selecting the function that fits the best generator for the current environment. The switch between functions through generations contributes to E-GAN with enough diversity in training to achieve convergence. CDE-GAN select the I best discriminators and J best generators to compose the next generation. In Pareto GAN, Pareto dominance is used as the strategy to select individuals to form the next generation. Lipizzaner and Mustangs have a selection strategy based on the spatial coevolution mechanism used in the evaluation phase. The neighborhood is used to evaluate and replace the individual in the center of a neighborhood according to the fitness.

2.3.3.3 *Experiments and Results*

Table 3 compares the proposals under the perspective of the experimental setup used to assess the contributions of each solution. Four experimental attributes are presented: the dataset used in the training, the number of generators and discriminators in the populations, the number of generations used in training, and the metric used to evaluate the results.

Except for Pareto GAN, all proposals used image datasets in the experiments. Pareto GAN uses bi-objective functions to validate the model, also including a function that simulates the data dimension of the MNIST dataset. In the category of images, MNIST is a simple dataset and should be used carefully to draw generic conclusions about the performance of a solution. The CelebA dataset is perhaps the most commonly used data to validate GANs. Therefore, it would be important to assess the performance of Pareto GAN in this dataset.

The populations used in the experiments vary a lot among the proposals. Except for E-GAN and CDE-GAN, the solutions used multiple individuals for both populations in the experiments. Although it is possible to use more individuals in E-GAN,

Table 3: Comparison of the experiments presented in the EA proposals to train GANs.

Algorithm	Dataset	Population (Discriminators × Generators)	Generations	Metric
E-GAN	CIFAR-10, LSUN, CelebA	1×3	200000	Inception Score
Pareto GAN	bi-objective functions	20^1	500	IGD
Lipizzaner	MNIST, CelebA	4×4	400	–
Mustangs	MNIST, CelebA	4×4, 9×9	time-limited, 20	FID score
CDE-GAN	CIFAR-10, LSUN, CelebA	$[1, 2, 4, 8] \times 1$	100k to 400k	Inception score

[1] In Pareto GAN one individual completely represents a GAN, i.e., it contains both a generator and a discriminator.

the experiments used only a single discriminator and three possibilities for generators (representing each possible loss function). CDE-GAN can also use multiple generators, but the experiments were limited to a single individual in order to limit the execution time. In Pareto GAN, one individual completely represents a GAN. Therefore, 20 individuals were used, meaning that 20 independent GANs with their own generator and discriminator were trained through generations. Lipizzaner and Mustangs use spatial coevolution to distribute the individuals in a grid of 2×2 for the MNIST dataset. For CelebA, Mustangs used a grid of 3×3. As these grids hold a single generator and discriminator in each cell, the population is composed of 4 and 9 individuals for the 2×2 and 3×3 setups, respectively. As a five-cell neighborhood is applied, spatial coevolution reduces the number of iterations needed to evaluate the individuals. Thus, a larger number of individuals can be used to evaluate Lipizzaner and Mustangs.

The number of generations used to evaluate each approach also present high variability. Each approach adapted the experiments to use a number of generations respecting their internal characteristics. For example, as E-GAN and CDE-GAN works with smaller populations, the number of generations needed to converge is much higher than the others. Mustangs used a time-limited strategy of nine hours for the experiments with MNIST and a limit of 20 generations for experiments with CelebA. The time-limited approach used in the MNIST experiments corresponds to more than 150 generations.

Mustangs use the FID score to report and analyze the results achieved by trained generators. As discussed in Section 2.1.2, the FID score is currently the state-of-the-art metric used to evaluate and compare GANs. The Inception Score, the former most used metric for GANs, was applied in the experiments for E-GAN and CDE-GAN. Pareto GAN adopted the IGD as the metric, which is adequate to its approach based on the Pareto set approximations. Lipizzaner analyzed the results through visual inspections and does not present an evaluation with respect to an objective measurement. As the proposals use different metrics, we can not directly compare the results of all proposals.

2.4 SUMMARY

In this chapter, we reviewed the background and the state of the art about Generative Adversarial Networks (GANs) and Evolutionary Algorithms (EAs). Section 2.1 presented the concepts about GAN, describing its challenges and variations of the original model. Despite the recent advances in the field, it is possible to see that there are still open problems. The stability of training remains a challenge, being tackled by researchers using different approaches, such as new loss functions and/or alternative architectures. Section 2.2 introduced EAs, focusing on neuroevolution and coevolution. In Section 2.2.3, we also described proposals that used competitive coevolution to evolve neural networks.

Finally, Section 2.3 presented state-of-the-art proposals that unifies concepts of EAs and GANs. We described proposals that train GANs using techniques from EC such as neuroevolution and coevolution.

The architectures of the generator and the discriminator are paramount to the equilibrium on the GAN training, and coevolution combined with neuroevolution can give the capabilities needed to balance their power. Furthermore, we can relate problems of GANs to coevolution problems. For example, the vanishing gradient problem (Section 2.1.1) can be linked to the disengagement issue (Section 2.2.2). Thus, we explore in the next chapter the use of EA to improve the training of GANs by using a combination of existing coevolution and neuroevolution techniques.

3 COEVOLUTIONARY GENERATIVE ADVERSARIAL NETWORKS

In this chapter, we introduce our proposal to use Evolutionary Algorithms (EAs) to train and evolve Generative Adversarial Networks (GANs) called Coevolutionary Generative Adversarial Networks (COEGAN). This work was first published in Costa, Lourenço, and Machado (2019), with a more detailed evaluation and validation published in Costa et al. (2019). The present chapter adapted and reproduced parts of these works.

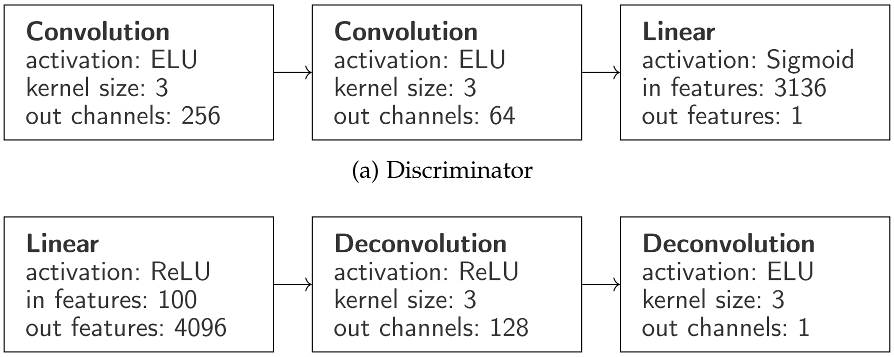
As described in Section 2.1.1, GANs can suffer from mode collapse and vanishing gradient, making them hard to train. To address these issues, we propose the use of EAs to evolve GANs, improving the training stability and, at the same time, discovering novel neural network architectures for the discriminator and generator.

This chapter provides a detailed description of COEGAN, validating the proposal experimentally in different scenarios. Section 3.1 describes the model and details each main component. Section 3.2 presents the results of the first experimental evaluation of COEGAN. In Section 3.3, the model is compared with other proposals presented in the literature. Finally, Section 3.4 summarizes the conclusions and points out the limitations of the proposal and directions for further improvement.

3.1 MODEL

Coevolutionary Generative Adversarial Networks (COEGAN) (Costa et al., 2019; Costa, Lourenço, and Machado, 2019) combines neuroevolution and coevolution in the coordination of the GAN training algorithm. Our approach is inspired by NeuroEvolution of Augmenting Topologies (NEAT) and DeepNEAT (Miikkulainen et al., 2017), extending and adapting them to the context of GANs adopting the same representation and selection mechanisms. Our main modification and contribution concerns the introduction of mechanisms to take into account a competitive coevolutionary environment, where generators and discriminators are paired as adversaries.

In COEGAN, the genome is represented as an array of genes, which are directly mapped into a phenotype that consists of a sequence of layers in a deep neural network. Each gene represents a linear, convolution, or transpose convolution layer. Moreover, each gene also has an activation function, chosen from the following set: ReLU, Leaky ReLU, ELU, Sigmoid and Tanh. From the specific parameters of each type of gene, convolution and transpose convolution layers only have the number of output channels as a random parameter. The stride and kernel size are fixed as 2 and 3, respectively. The number of input channels is calculated dynamically, based on the previous



(a) Discriminator

(b) Generator

Figure 8: Example of genotypes of a discriminator and a generator. The discriminator contains two convolution layers and one linear layer. The generator has one linear and two deconvolution layers. The parameters are listed for each gene (e.g., activation type, kernel size, and the number of channels).

layer. Similarly, the linear layer only has the number of output features as the random parameter. The number of input features is calculated based on the previous layer. Only the activation function, output features, and output channels are subject to the variation operations.

Figures 8(a) and 8(b) show an example of a discriminator and a generator genotype, respectively. The discriminator genotype is composed of a convolutional section and is followed by a linear section (fully connected layers). As in the original GAN approach, the output of discriminators is the probability of the input sample be a real sample drawn from the dataset. Similarly, the generator genotype is composed of a linear section, followed by a transpose convolutional section. The output of the generator is a fake sample, with the same characteristics (i.e., dimension and channels) of a real sample.

COEGAN evolves two separated populations: a population of generators, where each G_i represents a generator; and a population of discriminators where each D_j represents a discriminator. Inside each population, a speciation mechanism inspired in the strategy used in NEAT is applied to promote innovation. This mechanism ensures that individuals with new layers will have the chance to survive long enough to be as powerful as individuals from previous generations. The speciation mechanism divides the population into species based on a similarity function, comparing the genomes to compose groups of similar individuals. Thus, the innovation, represented by the addition of new genes into a genome, causes the creation of new species when the modified individuals do not fit into the current species. Therefore, as the selection mechanism take the species into account, these new individuals have the chance to survive through generations and reach the performance of older individuals in the population.

For the current proposal of COEGAN, we are only interested in the evolution of the neural network architecture. The parameters of the layers in the phenotype (e.g., weights and bias) are trained by the gradient descent method and will not be part of the evolution. The number of parameters to be optimized is too large and evolving them will increase the computational complexity.

3.1.1 *Fitness*

For discriminators, the fitness is based on the loss obtained from the regular GAN training method (Section 2.1), i.e., the fitness is equivalent to:

$$J^{(D)}(D, G) = -\mathbb{E}_{x\sim p_{data}}[\log D(x)] - \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))], \tag{22}$$

where p_{data} represents the dataset used as input to the discriminator, z is the latent space, p_z is the latent distribution, G is the generator, and D is the discriminator.

We have tried to use the same approach for the generator. However, preliminary experiments evidenced that the loss does not represent a good measure for quality in this case. The loss for generators, represented by Eq. 3 (Section 2.1), is unstable during the GAN training, making it not suitable to be used as fitness in an EA.

To address this issue, we selected the Fréchet Inception Distance (FID) (Heusel et al., 2017) as the fitness for generators, defined as:

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{1/2}), \tag{23}$$

where μ_x, Σ_x, μ_g, and Σ_g represent the mean and covariance estimated for the real dataset x and fake samples g, respectively, and Tr is the trace function applied on the resulting matrix of the operations with Σ_x and Σ_g.

As presented in Section 2.1.2.2, FID is the state-of-the-art metric to compare the generative components of GANs and outperforms other metrics, such as the Inception Score (Salimans et al., 2016). Using the FID score we put selection pressure in generators and direct the evolution of the population towards strong generators with respect to this metric.

3.1.2 *Variation Operators*

The variation operators used in COEGAN are defined by mutation operators that perform three main modifications to the networks layers:

- **Add**: In the addition operation, a new layer is randomly drawn from the set of possible layers. For discriminators, the available layers are linear and convolution. For generators, the available layers are linear and transpose convolution (also called deconvolution).

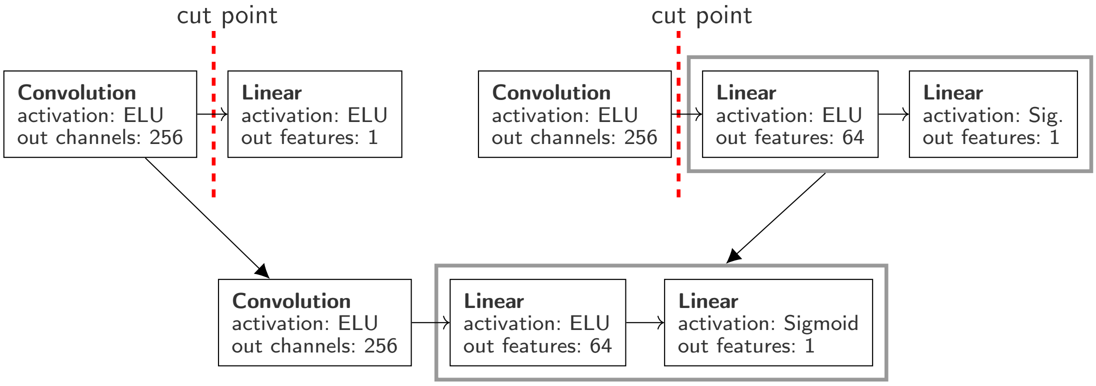


Figure 9: Example of crossover between discriminators.

- **Remove**: The remove operation randomly chooses an existing layer and excludes it from the genotype.
- **Change**: The change operation modifies the attributes and/or the activation function of an existing layer. In this case, the activation function is randomly chosen from the set of possibilities listed before. Other specific attributes can be changed depending on the type of layer. The number of output features and the number of output channels are mutated for the linear and convolution layers, respectively. The mutation of these attributes follows a uniform distribution, with a predefined range limiting the possible values.

It is important to note that during the breeding process the parameters (weights and bias) are copied when the genes involved in the mutation are compatible. This ensures that the new individual will carry the training information from the previous generation. However, when the attributes of a linear or convolution layer change, the trained parameters are lost. This happens because the setup of weights changes, becoming incompatible with the new layer. Consequently, the weights are not transferred from the parents and the layer will be trained from scratch.

Besides mutation, we have also defined and evaluated a crossover operator. The crossover process uses the transition between convolutional layers and linear (fully connected) layers as the cut point. Figure 9 represents an example of this process. However, preliminary tests evidenced that crossover decreases the performance of the system. The composition of new individuals using segments of different neural networks caused instability in training, leading to poor results.

3.1.3 *Pairing Strategy*

In the evaluation step of the EA, discriminators and generators must be paired to calculate the fitness for each individual in the population. As discussed in Section 2.2.2, the pairing strategy is paramount to coevolution.

As we want to train the GAN to avoid problems such as the mode collapse and vanishing gradient, we consider the use of the *all vs. best* strategy here. In this way, we avoid training individuals against adversaries with poor capacities. However, we select k individuals rather than only one to promote diversity in the GAN training, naming this strategy as *all vs.* k*-best*. We pair each generator with k best discriminators from the previous generation and, similarly, each discriminator with k best generators. For the first generation, we assume a random pairing approach since we do not have a ranking of the best individuals, i.e., k random individuals are selected for pairing in the initial evaluation.

Another approach is to apply the *all vs. all* pairing strategy, where each discriminator is paired with each generator, resulting in many competitions. In this case, the fitness for discriminators will be the average of the losses obtained by each training pair. The *all vs. all* strategy would also be interesting for our model as it will improve the variability of the environment for both discriminators and generators during the training. However, the trade-off is the time to execute this approach.

Figure 10 displays examples of the *all vs. all* and *all vs.* k*-best* pairing strategies for a population of generators and discriminators. The first column contains the generators $g_i \in \mathcal{G}$ and the second column contains the discriminators $d_j \in \mathcal{D}$. In Figure 10(a), all pairwise combinations are generated for the application of the GAN training algorithm:

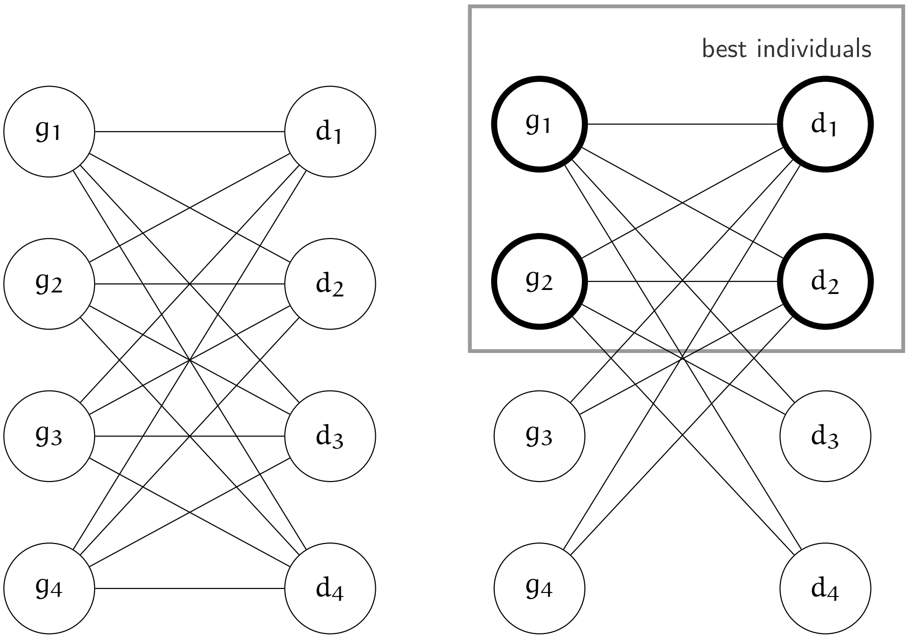
$$\mathcal{P} = \{(g_i, d_j) \mid g_i \in \mathcal{G}, d_j \in \mathcal{D}\}$$

Figure 10(b) represents an example of the *all vs. best* strategy approach with $k = 2$. The best individuals are highlighted and the set is composed of the individuals g_1, g_2, d_1, and d_2. In this case, after the initial generation, the GAN training algorithm is applied in the following set of pairs:

$$\begin{aligned}\mathcal{P} = \{&(g_1, d_1), (g_1, d_2), (g_1, d_3), (g_1, d_4), (g_2, d_1), (g_2, d_2),\\ &(g_2, d_3), (g_2, d_4), (g_3, d_1), (g_3, d_2), (g_4, d_1), (g_4, d_2)\}\end{aligned}$$

3.1.4 *Selection*

For the selection phase we used a strategy based on NEAT (Stanley and Miikkulainen, 2002), where we divided the population of generators and discriminators into subpopulations, following a speciation strategy. Each species contains individuals with similar network structures. For this, we define the similarity between individuals based



(a) *all vs. all*

(b) *all vs. k-best*

Figure 10: Representation of (a) the *all vs. all* pairing strategy and (b) the *all vs. k-best* competition pattern with $k = 2$.

on the parameters of each gene composing the genome. Different from NEAT, we do not use the weights of each layer to differentiate between individuals. Therefore, we calculate the distance δ between two genomes i and j as the number of genes that exist only in i or j. Each species inside the populations of generators and discriminators are clustered based on a threshold δ_t. This threshold is calculated to suit the desired number of species.

The selection occurs inside each species. The number of individuals selected inside each species is proportional to the average fitness of the individuals belonging to it. Given the number of individuals to keep in a species, a tournament between k_t individuals is applied to finally select the individuals to breed and compose the next generation.

3.2 EXPERIMENTS

In this section, we evaluate the performance of our method on the image dataset namely MNIST. Normally, the network would be training for several epochs using the whole dataset. However, this demands large-scale computational resources, and to reduce this we will use only a subset of the dataset per generation. This strategy, combined with the transfer of parameters between generations, ensures the evolutionary pressure towards efficient solutions and to promote the GAN convergence.

There is no consensus on the metric to represent the quality of samples generated by generative models. However, as stated in Section 2.1.2.2, it has been shown that FID is a good metric when comparing the quality of samples generated by GANs (Lucic et al., 2017). Therefore, we used the FID score, the same metric used as fitness for generators, to compare our results with the state of the art.

3.2.1 *Experimental setup*

Table 4 describes the parameters used in all experiments reported in this section.

Table 4: Parameters for the experiments.

Evolutionary Parameters	**Value**
Number of generations	100
Population size (generators)	20
Population size (discriminators)	20
Crossover rate	0%
Add Layer rate	30%
Remove Layer rate	10%
Change Layer rate	10%
Output features range	[32, 1024]
Output channels range	[16, 128]
k (all vs. best)	3
Tournament k_t	2
FID samples	1000
Genome Limit	4
Species	4
GAN Parameters	**Value**
Batch size	64
Batches per generation	20
Optimizer	RMSProp
Learning rate	0.001

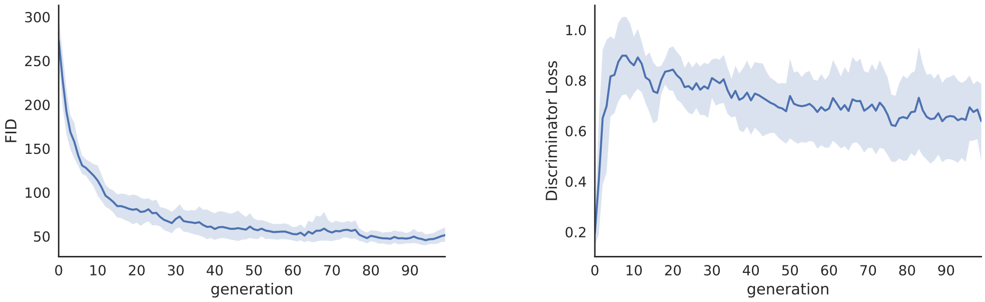
For the evolutionary parameters, we execute our experiments for 100 generations. We used 20 individuals for the population of both generators and discriminators. We acknowledge that a larger population would probably achieve better results, but the computational cost would be too large. The size of the genome was limited to four layers, also to reduce the computational cost. The number of species used was set to four, permitting an average of five individuals per species in each subpopulation (generators and discriminators). We empirically defined a probability of 30%, 10% and 10% for the add, remove and change mutations, respectively. As stated before, crossover was not used in the experiments reported in this section.

For the GAN parameters, we choose 64 as batch size, running 20 batches per generation. This amounts to 1280 samples per generation to train discriminators. The optimizer used in the training method was RMSProp (Tieleman and Hinton, 2012). We have also conducted preliminary experiments with Adam (Kingma and Ba, 2015), but the best results were achieved with RMSProp.

The MNIST dataset was used and we executed each experiment 10 times to achieve the results within a confidence interval of 95%.

3.2.2 *Results*

Figure 11 shows the progression of fitness for the best generators and discriminators. In Figure 11(a), we can see the fitness for generators reducing through generations with reduced variance, providing evidence that the FID score is a good metric to guide the evolutionary process. For discriminators (Figure 11(b)), we can see a high variance, which can harm the selection process in the EA, suggesting that there is room for improvement regarding the choice of the fitness function.



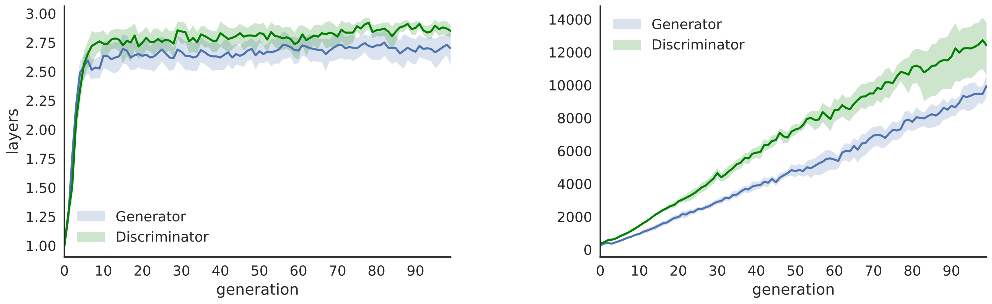
(a) Fitness for generators

(b) Fitness for discriminators

Figure 11: Fitness for discriminators and generators with a 95% confidence interval.

In the final generation, the mean FID was 49.2, with a standard deviation of 10.5. Inspecting the results obtained by our model, it is possible to see that the model did not collapse into a single point from the input distribution, which is a common problem in GANs.

Figure 12 shows the progression of the network through generations. We can see in 12(a) the average number of layers in the population of generators and discriminators. Because we have limited the genotype to a maximum of four genes, the number of layers rapidly saturates. This is an indication of premature optimization. We can overcome this issue by either increasing the limit or decreasing the growth rate (i.e., reduce the mutation probability). Figure 12(b) shows the number of genes with the parameters reused in each generation. The linear growth in the amount of reused genes is evidence of the transference mechanism. Because we use a strategy similar to transfer learning to keep the trained parameters, this reuse is important to pass the trained weights through generations. With this mechanism, we can use a subset of the input data in each generation, ensuring that weights that are kept through generations will have the chance to be trained with other subsets of the dataset.



(a) Layers per generation

(b) Reused genes

Figure 12: Progression of layers and the reuse of parameters with a 95% confidence interval.

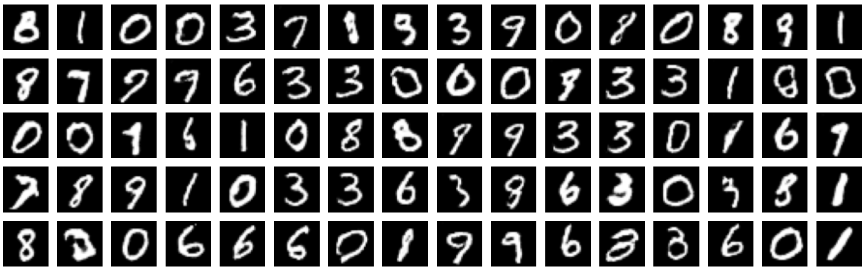
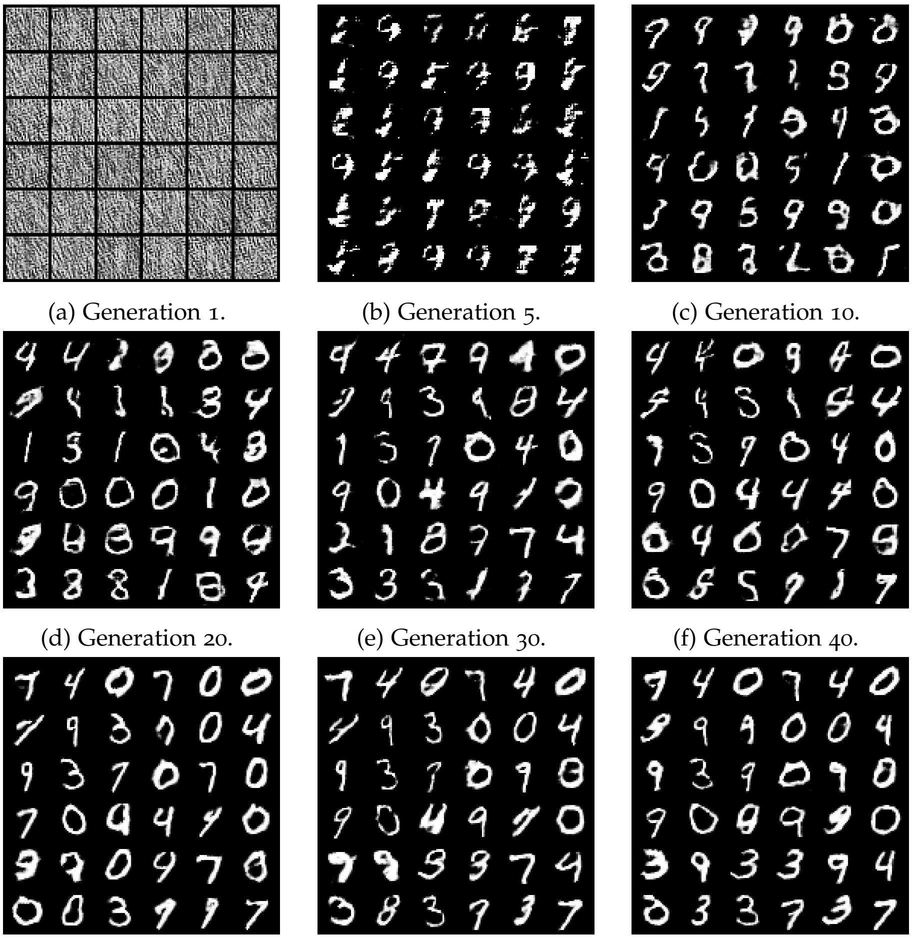


Figure 13: Samples created by a generator after the evolutionary process.

Figure 13 represents samples generated in one execution, evidencing that our model does not collapse into a single point of the input distribution. We can see this behavior occurring in all executions, suggesting that COEGAN is able to solve, at least partially, the mode collapse problem. Moreover, all executions reached convergence with the equilibrium between the discriminator and the generator, and the vanishing gradient never occurs. This evidences that our proposal brings stability to the training procedure of GANs.



(a) Generation 1.

(b) Generation 5.

(c) Generation 10.

(d) Generation 20.

(e) Generation 30.

(f) Generation 40.

(g) Generation 50.

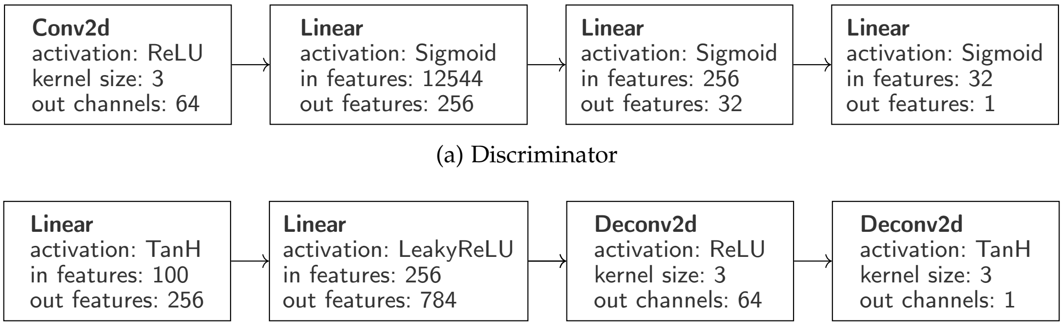
(h) Generation 70.

(i) Generation 100.

Figure 14: The progression of samples created by the best generator through generations.

Figure 14 shows the progression of the generator during the evolutionary process. We can see that in the first generation there are only noisy samples, without any structure resembling a digit. After five generations (Figure 14(b)), we can see some structure emerging to form the digits. From generation 10 onwards we can start to distinguish between the digits, with a progressive improvement of the quality.

Finally, figure 15 presents the best network architecture discovered after the last generation. We can see that both components reached the limit of four layers imposed



(a) Discriminator

(b) Generator

Figure 15: Best (a) discriminator and (b) generator found after the final generation.

in the experiments. Furthermore, both the generator and the discriminator were composed by a combination of convolutional and linear layers with different activation functions.

3.3 evaluation and comparison

To validate the performance of our method, we experiment COEGAN with the MNIST (LeCun, Cortes, and Burges, 1998) and Fashion-MNIST (Xiao, Rasul, and Vollgraf, 2017) datasets, comparing with other approaches. We also evaluate COEGAN in experiments with CelebA to assess the contributions of the method in a more complex dataset. This section reproduces the experiments made in Costa et al. (2019) and Costa et al. (2020b).

We evaluate COEGAN against a random search method and a reference architecture based on Deep Convolutional Generative Adversarial Networks (DCGAN) (Radford, Metz, and Chintala, 2015). The random search method is similar to COEGAN, but instead of the fitness described in Section 3.1.1, we use a random function as the fitness of individuals in the population, drawing values from a uniform distribution to assign fitness values for individuals. In this way, we ablate the fitness functions proposed for COEGAN to inspect their ability to promote selective pressure in the model. All other characteristics of the random method, such as the pairing strategy, remain the same as used in COEGAN. The DCGAN model is a well-defined set of architectural constraints, used as a reference in several works related to evaluations of GANs (Karras et al., 2018; Lucic et al., 2017). We follow this approach to build a reference architecture (based on DCGAN) to compare our results with commonly used models in the context of GANs.

3.3.1 *Experimental Setup*

Table 5 presents the parameters used in all experiments reported in this section.

Table 5: Parameters for the evaluation experiments.

Evolutionary Parameters	**Value**
Number of generations	50
Population size (generators)	10
Population size (discriminators)	10
Add Layer rate	20%
Remove Layer rate	10%
Change Layer rate	10%
Output features range	[32, 1024]
Output channels range	[16, 128]
Tournament k_t	2
FID samples	1000
Root mean squared error samples	1000
Genome Limit	6
Species	3
GAN Parameters	**Value**
Batch size	64
Batches per generation	20
Optimizer	Adam
Learning rate	0.001

The number of generations used in all experiments is 50. We used 10 individuals for each population of generators and discriminators. In order to limit the computational resources used in our experiments, the size of the genome was limited to six layers. To emulate a network with similar power, the DCGAN architecture used in the experiments also contains six layers. We use three species for each population of generators and discriminators, which allow an average of 3 individuals per species. We empirically defined a probability of 20%, 10%, and 10% for the add, remove and change mutations, respectively. A higher probability for these mutations causes the premature convergence of the system, leading to performance issues and instability on the GAN training process. Hence, the probability rates were kept low but sufficient to create diversity in the population through generations.

For each pair of (G_i, D_i), 20 batches were executed per generation, with the batch size of 64. Therefore, in our scenario of a population composed of 10 generators and 10 discriminators with the *all vs. all* pairing strategy, each individual will execute 200 batches per generation. Note that, as DCGAN is not an EA, the experiments with this model contain only one discriminator and one generator. Therefore, the evolutionary parameters do not apply to the experiments with DCGAN. In this case, we set the number of batches to 200 to keep it comparable with COEGAN and the random search method. The optimizer used in the training method was Adam (Kingma and Ba, 2015) with a learning rate of 0.001.

3.3.2 *Results*

To compare the results between COEGAN, Random Search and DCGAN, we use the FID score (Heusel et al., 2017), Inception score (Salimans et al., 2016) and the Root Mean Squared Error (RMSE). The RMSE is calculated between samples created by the generator and real samples randomly drawn from the input dataset. All figures in this Section contain plots with curves representing the average of the results from 10 runs, with a confidence interval of 95%.

3.3.2.1 *MNIST*

First, we present the results for the experiments on the MNIST dataset.

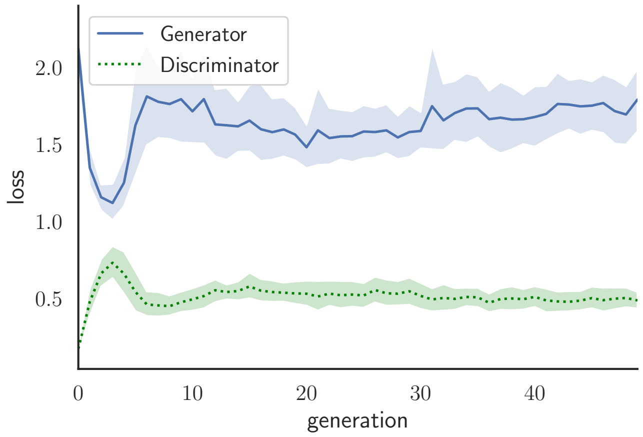


Figure 16: Losses of the discriminator and generator on the MNIST dataset.

Figure 16 displays the average of losses for the best generator and discriminator found by COEGAN in each generation. As stated in Section 3.1.1, this figure indicates that the use of the GAN loss function as the fitness for generators is not a good metric to assess the performance of an individual. We can see the value of the loss increasing with generations as well as some instability in the values.

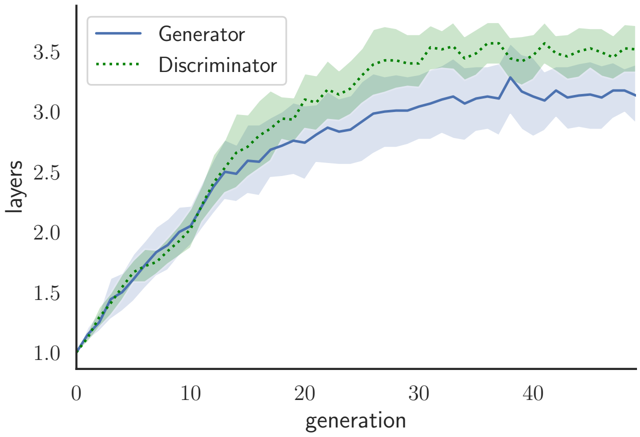


Figure 17: The average number of layers on the MNIST dataset.

Figure 17 presents the average progression of layers in the genome of individuals belonging to the population of generators and discriminators. The number of layers gradually increases with generations, demonstrating that the speciation mechanism used in COEGAN protects the innovation and creates a favorable environment for individuals with more layers.

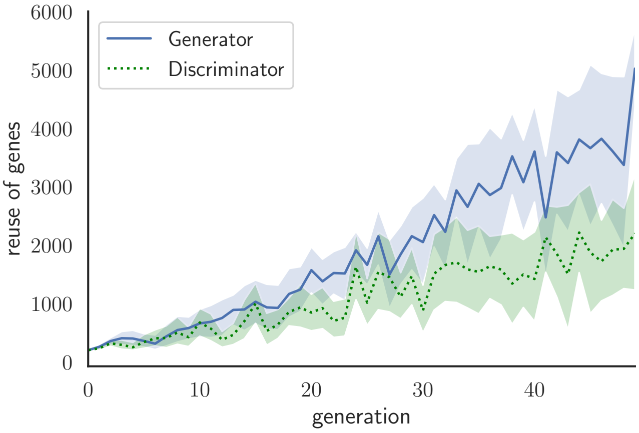


Figure 18: The average number of times a gene was reused when trained on the MNIST dataset.

Figure 18 displays the average number of times a gene was reused during the training process on the MNIST dataset. The results for this metric shows that the information is kept through generations.

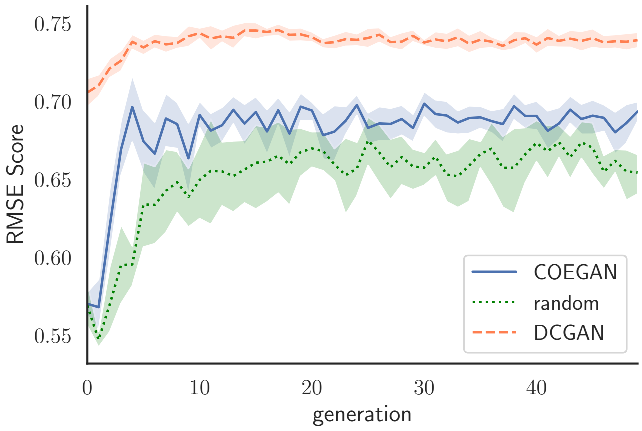


Figure 19: Root mean squared error on the MNIST dataset.

The RMSE is displayed in Figure 19, comparing the results for COEGAN, the random method, and DCGAN. We introduced this metric to evaluate the novelty of the samples created by generators in the different models analyzed in the experiments. In other words, we want to ensure that the samples are not a copy of the data from the input dataset. Thus, Figure 19 indicates that all methods exhibit some innovation in the new samples, with the DCGAN method being better for this metric. This means that the samples that are being created represent the data from the input dataset, without being an exact copy.

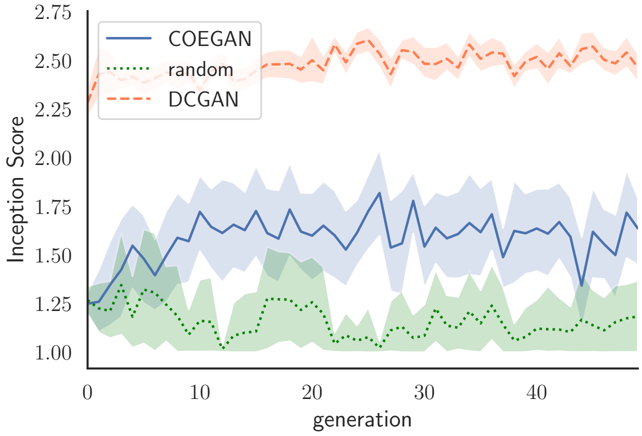


Figure 20: Inception Score on the MNIST dataset.

Figure 20 shows the average of the Inception Score (higher is better) for generators in COEGAN, Random Search, and DCGAN. For this metric, the DCGAN provides the best results. However, COEGAN is better than the random approach, demonstrating that our choice for fitness is relevant to the EA.

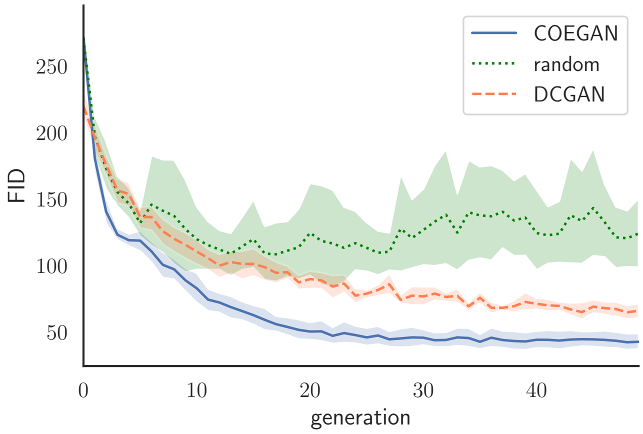


Figure 21: FID Score on the MNIST dataset.

Table 6: FID score of best generators in MNIST in the last generation.

Algorithm	**Last FID**
COEGAN	41.0 ± 7.4
Random	124.1 ± 42.2
DCGAN	66.0 ± 8.7

The FID of the generators in COEGAN, Random Search and DCGAN are displayed in Figure 21 and Table 6 (lower is better). We can see that the FID for COEGAN is better than the results of the random search and DCGAN. To support our analysis, we statistically test the significance of our findings. We used a non-parametric test (Mann-Whitney U, $\alpha = 0.05$) with Holm-Bonferroni correction to perform a pairwise comparison between the solutions. We found that the improvement of COEGAN over the random search and DCGAN is statistically significant ($p < 0.001$).

Moreover, the random search shows a high variability in the FID results, mainly caused by the stochastic process introduced by this approach. In what concerns the comparison between COEGAN and DCGAN, a study made by Lucic et al. (2017) found that the FID score is a good representation of diversity and quality of generated samples when compared to real samples. Thus, based on this study and the results displayed in Figure 21, the best generator found in COEGAN outperforms the generator in the reference architecture based on DCGAN.

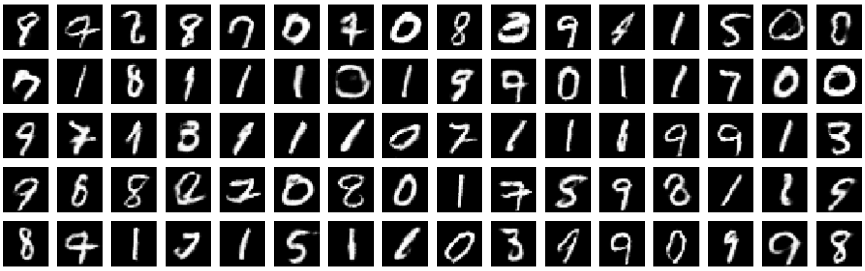


Figure 22: Samples generated by COEGAN when training on the MNIST dataset.

Figure 22 contains samples generated by the best generator found in COEGAN trained with the MNIST dataset after 50 generations. We can observe a good representation of the MNIST dataset in the generated samples. We found no evidence of the vanishing gradient and the mode collapse problem in all executions of COEGAN. As individuals with these issues perform worse than others, they will eventually not be selected by the EA, preventing these problems to persist through generations. Furthermore, a diverse population of generators and discriminators can increase the variability provided in the training process when compared to a regular GAN. This variation contributes to a stronger training algorithm, preventing the mode collapse and the vanishing gradient problems.

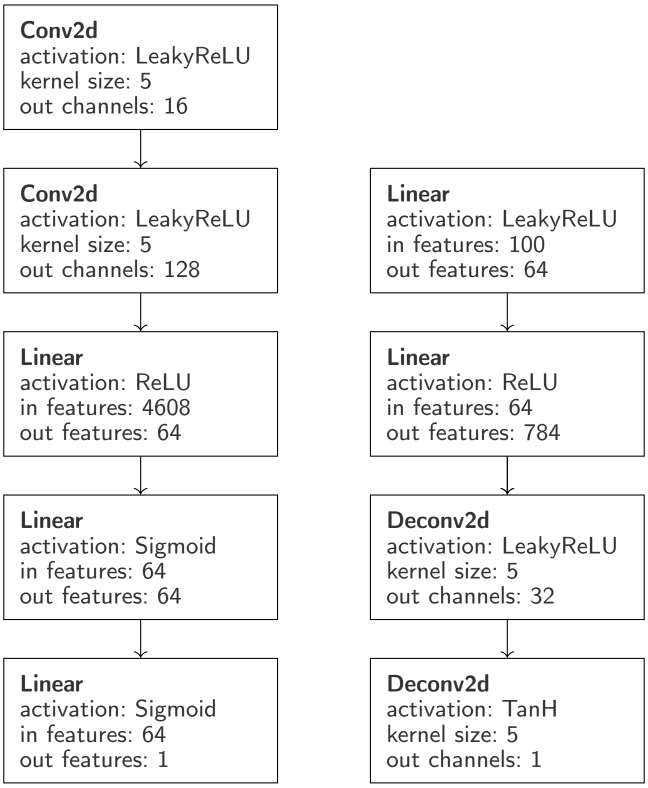
Figures 23(a) and 23(b) represent the best architecture found by COEGAN after 50 generations. Both architectures are composed of a combination of linear and convolutional layers (represented in the images by Conv2d and Deconv2d). It is relevant to note that not only the final architecture is important but also the process to construct the final models because of the mechanism of transference of the learned weights through generations. Therefore, COEGAN found models for the generator and the discriminator with fewer layers than the reference architecture based on DCGAN, but with better performance with respect to the FID metric.

3.3.2.2 *Fashion-MNIST*

The same methodology to assess the performance of COEGAN (used with the MNIST dataset in Section 6.2.2) was applied on the Fashion-MNIST dataset.

Figures 24, 25, 26, 27, 28 and 29 present results with similar characteristics of the previous results on the MNIST dataset.

As in the MNIST results, we can see in Figure 29 and Table 7 that the FID score for COEGAN outperforms the other methods. We used a non-parametric test (Mann-Whitney U, $\alpha = 0.05$) with Holm-Bonferroni correction to perform a pairwise com-



(a) Best Discriminator

(b) Best Generator

Figure 23: Best (a) discriminator and (b) generator found by COEGAN when training on the MNIST dataset.

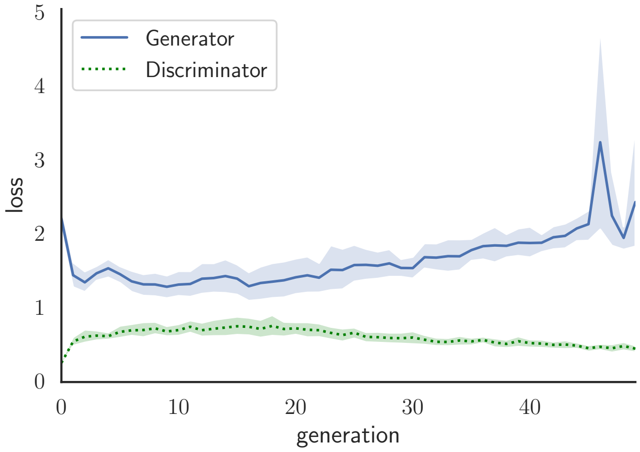


Figure 24: Losses of the discriminator and generator on the Fashion-MNIST dataset.

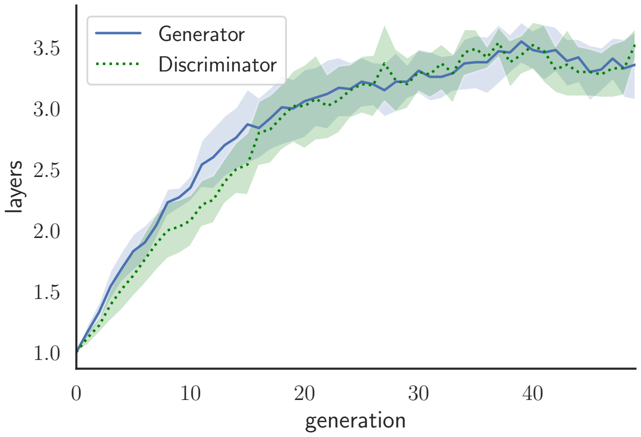


Figure 25: The average number of layers on the Fashion-MNIST dataset.

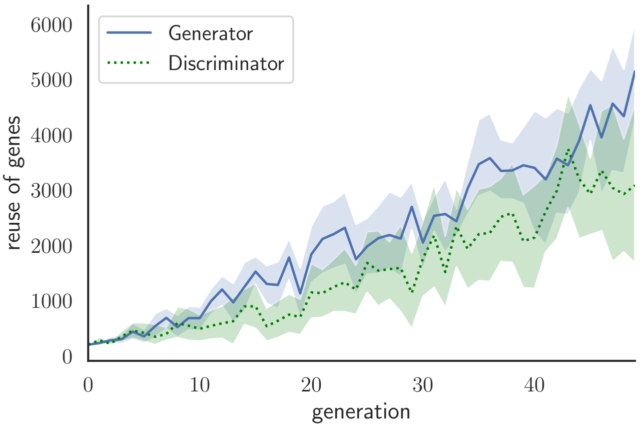


Figure 26: The average number of times a gene was reused when trained on the Fashion-MNIST dataset.

Table 7: FID score of best generators in Fashion-MNIST in the last generation.

Algorithm	**Last FID**
COEGAN	84.1 ± 17.5
Random	149.5 ± 29.3
DCGAN	140.8 ± 11.6

parison between the solutions. We found that the improvement of COEGAN over the random search and DCGAN is statistically significant ($p < 0.001$).

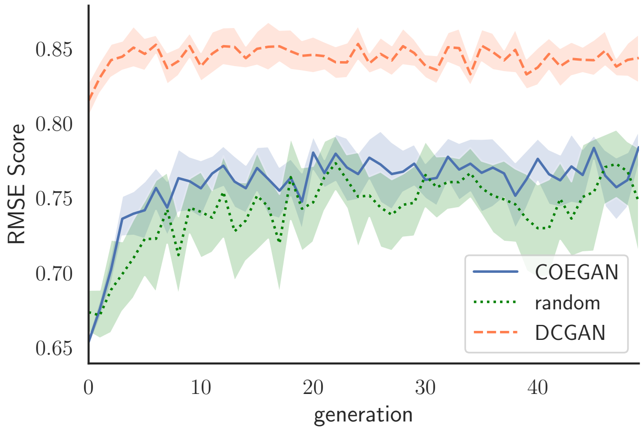


Figure 27: Root mean squared error on the Fashion-MNIST dataset.

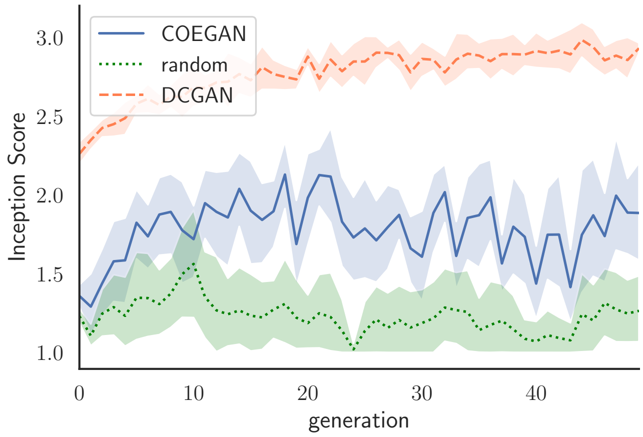


Figure 28: Inception Score on the Fashion-MNIST dataset.

Furthermore, the Inception Score is still better for DCGAN on the Fashion-MNIST dataset. The progression in the number of layers, presented in Figure 25 is similar to the MNIST results.

As the Fashion-MNIST dataset is more complex than MNIST, we can conclude that our method can be applied in more elaborated datasets. However, experiments with larger datasets such as CelebA (Liu et al., 2015) and CIFAR-10 (Krizhevsky and Hinton, 2009) can be conducted to support this statement. In Section 3.3.2.3, we present the results on the CelebA dataset.

Figure 30 contains samples generated by the best generator found in COEGAN trained with the Fashion-MNIST dataset after 50 generations. We can see a variety of images being generated, following the distribution of the input dataset. As in the

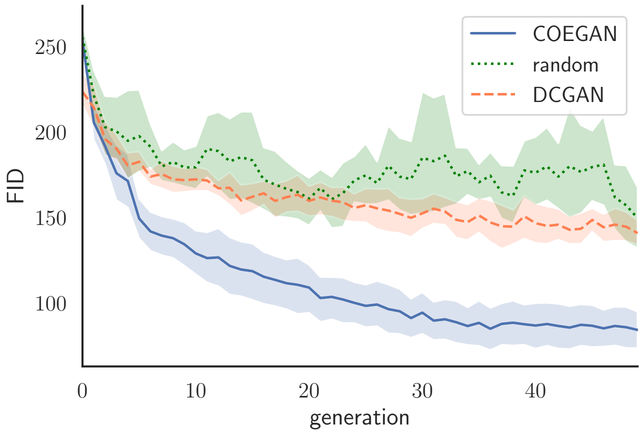


Figure 29: FID Score on the Fashion-MNIST dataset.

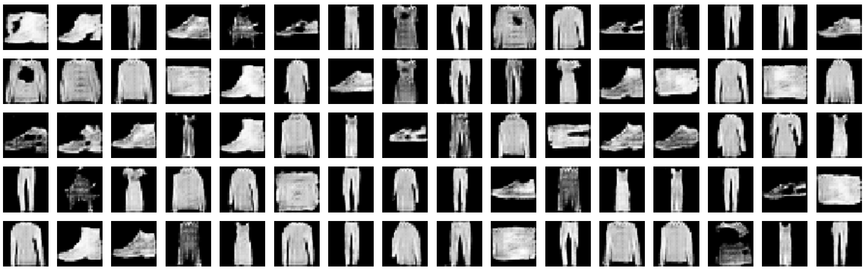


Figure 30: Samples generated by COEGAN when training on the Fashion-MNIST dataset.

results with the MNIST dataset, we also found no evidence of the vanishing gradient and the mode collapse problem in all executions.

3.3.2.3 *CelebA*

To assess the applicability of COEGAN in complex datasets, we expand the experiments to include the results with the CelebA dataset (Liu et al., 2015). For this, we use an experimental setup similar to the one described in Section 3.3.1, with some modifications to accommodate the new dataset. Concerning the architecture, we only use convolution and transpose convolution layers when adding a new gene, excluding the linear layer from the set of possibilities, following the directions of recent models for GANs Radford, Metz, and Chintala, 2015. Furthermore, we allow only ReLU and Leaky ReLU as possible activation functions in the mutation operators. The populations of generators and discriminators contain 10 individuals each, divided into three species. The *all vs. all* pairing strategy was applied, using 100 batches of 64 images to train each pair. The images from the CelebA dataset were rescaled to 64 × 64.

Figure 31 presents the FID score for COEGAN through generations. As expected, we can note the decreasing behavior of the FID score, resembling the behavior presented in the previous experiments. This is an indication of the generalization ability of COEGAN to effectively work with more complex datasets like CelebA. The average FID score achieved by COEGAN in the last generation is 89.8 ± 17.2. No evidence of the vanishing gradient and mode collapse was found in the experiments.

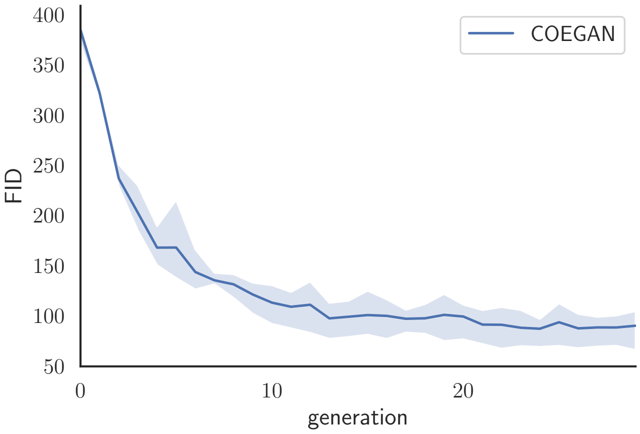


Figure 31: The FID score of COEGAN on the CelebA dataset. COEGAN achieves a FID score of 89.8 ± 17.2 at the last generation.

Figure 32 displays samples created by COEGAN at the final generation by one of the best models. We can clearly see the formation of faces in each created sample, with elements coherently positioned in each face. The variety achieved on samples also indicates that COEGAN achieved convergence when training, avoiding problems such as the mode collapse. However, the produced samples are not perfect. Undesired artifacts can be seen in some samples, affecting the quality of the outcome.

3.4 DISCUSSION AND LIMITATIONS

Generative Adversarial Networks (GAN) achieved important results for generative models in the field of computer vision. However, there are stability problems in the training method, such as the vanishing gradient and the mode collapse problems, as described in Section 2.1.1.

We presented in this chapter the first advances of this research, that originated a model called COEGAN (Costa et al., 2019; Costa, Lourenço, and Machado, 2019). This model uses neuroevolution and coevolution in the coordination of the GAN training process, making use of the adversarial characteristics of a GAN to apply a coevolu-

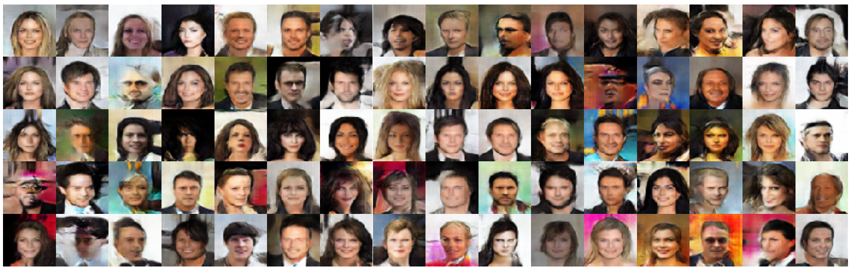


Figure 32: Samples generated by COEGAN when training on the CelebA dataset.

tion environment. The model was designed with inspiration on NEAT (Stanley and Miikkulainen, 2004) and DeepNeat (Miikkulainen et al., 2017), and also on recent advances in GANs, such as in Karras et al. (2018).

Moreover, we presented experiments made with the MNIST and Fashion-MNIST datasets to assess the efficiency of COEGAN. We found no evidence of the vanishing gradient and the mode collapse problem in all experiments with COEGAN. The selection process and the variation introduced by a diverse population of generators and discriminators contribute to the prevention of these issues, resulting in a more stable training solution than regular GANs. We compare our results with a random search method and also with a reference architecture based on DCGAN. The results displayed that COEGAN achieved a FID better than DCGAN and the random method for both datasets. However, the Inception Score of the DCGAN model was better than COEGAN. The Inception Score is a metric that has issues to represent the diversity and quality of the samples, being gradually replaced by the FID score in the analysis of the quality of GANs. We also show that COEGAN is better than a random search model, demonstrating the efficiency of the EA proposed by us. It is also important to note that COEGAN discovered models for generators and discriminators with fewer layers than the DCGAN used in our experiments. Therefore, the evolutionary process that leads to the final models in COEGAN also contributes to find more efficient neural networks regarding their architectural size. We argue that the mechanism of weights transference through generations promote the sufficient knowledge to smaller neural networks for achieving performance comparable to larger networks.

The use of an external evaluator, represented by the FID score, introduces a performance penalty in the proposal. The FID score uses the Inception Net to evaluate samples. Inception Net is a relatively big neural network, slowing down the process of achieving a metric for generators. Furthermore, the process of calculating the score is also CPU intensive, relevantly increasing the computational cost of the solution.

The diversity of individuals explored through generations can also be improved. The use of other mechanisms related to Evolutionary Computation (EC) can be studied in the context of the model proposed in this chapter to balance the trade-off between quality and diversity of individuals.

Several improvements that were proposed for GANs can also be explored in COEGAN. For example, the use of alternative loss functions as a replacement of the original loss functions for GANs. Besides, new architectural mechanisms can be incorporated to enrich the search space, permitting the discovery of better models.

Chapter 4 and Chapter 5 propose modifications into the original COEGAN model to address these limitations and improve the quality of the results. These modifications address not only aspects of the EA but also mechanisms related to GANs.

4 EXPLORING THE EVOLUTIONARY ASPECTS OF THE MODEL

Coevolutionary Generative Adversarial Networks (COEGAN), introduced in Chapter 3, proposes the use of neuroevolution and coevolution to orchestrate the training of GANs. Despite the advances in training stability, there is still room for improvement in the model.

This chapter presents our proposals to improve COEGAN concerning the evolutionary aspects of the solution. In concrete, we explore the use of alternative fitness functions and a different approach regarding the design of the Evolutionary Algorithm (EA). Each proposal is validated by experimental evaluation in different scenarios.

The remainder of this chapter is organized as follows. Section 4.1 proposes the use of the Skill Rating as fitness function. Section 4.2 explore the use of Quality Diversity (QD) to improve the exploration of the search space on the evolution of Generative Adversarial Networks (GANs). Finally, Section 4.3 summarizes the conclusions and points out directions for further improvement.

4.1 USING THE SKILL RATING AS FITNESS

The experimental evaluation of COEGAN identified that the fitness function is crucial to guide the evolution of the components, mainly regarding the discriminator. Currently, the discriminator uses the loss function of the respective GAN component. However, this function displayed a high variability behavior, disrupting the evolution of the population. The generator uses the Fréchet Inception Distance (FID) score, which introduces an external evaluator represented by a trained Inception Network (Szegedy et al., 2015, 2016). Fréchet Inception Distance (FID) showed good results when used as fitness, but there are drawbacks, namely the execution cost and the dependence of an external evaluator.

The FID score is currently the most used metric by the community, but several other metrics have been proposed to evaluate the performance of GANs (Borji, 2019; Xu et al., 2018). One example is the Skill Rating, which has been successfully used to evaluate GANs in some contexts (Olsson et al., 2018). Skill rating uses a game rating system to assess the skill of generators and discriminators. Each generator and discriminator is considered as a player in a game and the pairing between them is designed as a match. The outcome of the matches is used as input to calculate the skill of each player.

We took inspiration from the use of skill rating to quantify the performance of generators and discriminators in GANs to design a fitness function to be used within COEGAN. Specifically, we replace the regular fitness used in COEGAN with the skill rating, i.e., the discriminator and the generator use the skill rating metric instead of the loss function and the FID score. To analyze the impact of this modification, we present an experimental study, comparing the results with the previous approach used in COEGAN, a random search approach, and with a non-evolutionary model based on DCGAN.

4.1.1 *Model*

One way to apply the skill rating to measure the performance of players given a history of matches is the application of the Glicko-2 (Glickman, 2013) rating system. The system takes into account three variables: the rating r, the deviation RD, and the volatility σ (refer to Section 2.1.2.3 for more details). A constant τ is used for controlling the rate of change of σ.

Glicko-2 was previously used to compare evolutionary algorithms on a given problem, using the solutions found by them as input to the rating system (Veček et al., 2014; Veček, Mernik, and Črepinšek, 2014). Thus, the algorithms are ranked according to the rating score.

Another application of the Glicko-2 system was to evaluate the performance of GANs (Olsson et al., 2018). In this case, the rating was applied between discriminators and generators of different epochs to evaluate their progress. The authors found that skill rating provides a useful metric to relatively compare GANs.

We took inspiration from these use cases and applied the Glicko-2 system in COEGAN by changing the fitness function to use the skill rating metric computed using Glicko-2. Therefore, each generator G_i and discriminator D_j have an associated skill rating, represented by r, RD, and σ.

In the evaluation phase of the evolutionary algorithm, discriminators and generators are matched to be trained with the GAN algorithm and also to be evaluated for selection and reproduction. We modeled each evaluation step between a generator and a discriminator as a game and applied the skill rating calculation, composing a tournament of generators against discriminators. Therefore, as we use the *all vs. all* pairing strategy, each outcome of the match between (G_i, D_j) is stored and used to update the skill rating at the end of each generation. Inspired by the approach of Olsson et al. (2018), we use the following equations to calculate the outcome of a match for the discriminator:

$$D_j^{real} = \sum_{x \sim p_{data}} th\Big(D_j(x) > 0.5\Big) \tag{24}$$

$$D_{ij}^{fake} = \sum_{z \sim p_z} th\Big(D_j(G_i(z)) < 0.5\Big) \tag{25}$$

$$D_{ij}^{wr} = \frac{D_j^{real} + D_{ij}^{fake}}{m+n} \tag{26}$$

where D_j^{real} is the win rate of the discriminator with respect to the real data, D_{ij}^{fake} is the rate related to the fake data, D_{ij}^{WR} is the overall win rate of the discriminator D_j, th is a threshold function that outputs 1 when the threshold is met and 0 otherwise, D_j outputs the probability of the sample to be real, G_i is the generator, p_{data} is the input distribution, x is a sample drawn from the input distribution, p_z is the latent distribution, z is the latent input for the generator, m is the number of real samples, and n is the number of fake samples. In summary, the win rate for the discriminator is based on the number of mistakes made by it with the real input batch (Eq. 24) and fake data produced by the generator (Eq. 25).

For the generator, the result is calculated as:

$$G_{ij}^{wr} = 1 - D_{ij}^{wr} \tag{27}$$

where D_{ij}^{wr} is the discriminator win rate given by Eq. 26.

The win rates of each generator and discriminator are used as input to update the skill rate of the individuals. Each individual G_i and D_j has a set of outcomes T^{wr}, containing the win rate of each match and the skill of the adversary. Thus, a generator G_i has a set $T_{G_i}^{wr}$ containing each pair (G_{ij}^{wr}, D_j^{sk}) for a generation. A discriminator D_j has a set $T_{D_j}^{wr}$ containing each pair (D_{ij}^{wr}, G_i^{sk}). The sets $T_{G_i}^{wr}$ and $T_{D_j}^{wr}$ are used to calculate the new skill rating at the end of the generation, represented by G_i^{sk} and D_j^{sk}, respectively. It is important to note that the update of the skill rating of a player depends on the skill of the adversary, i.e., winning a game against a strong player is more rewarding than winning against a weak player.

We propose in this section the use of skill rating as fitness in COEGAN, represented by the use of D_j^{sk} instead of Eq. 1 for discriminators and G_i^{sk} instead of Eq. 5 for generators. Therefore, the fitness functions for discriminators and generators are defined as:

$$F_{D_j} = r_{D_j^{sk}}, \qquad F_{G_i} = r_{G_i^{sk}}, \tag{28}$$

where $r_{D_j^{sk}}$ and $r_{G_i^{sk}}$ are the rating r for discriminators and generators, respectively. At each generation, individuals update the skill rating following these rules. In the breeding process, the offspring carry the skill rating of their parent. In this way, we keep track of the progress of individuals through generations, even when mutations occur.

4.1.2 *Experiments*

To evaluate the use of skill rating with COEGAN, we conducted an experimental study using the Street View House Numbers (SVHN) dataset (Netzer et al., 2011). The SVHN dataset is composed of digits from 0 to 9 extracted from real house numbers. Therefore, it is a dataset with a structure similar to the MNIST dataset (LeCun, Cortes, and Burges, 1998) used in previous COEGAN experiments, but with more complexity introduced by the use of real images, presenting digits with a variety of backgrounds. The experiments compare the results of the original COEGAN approach (with the FID score and the loss function as fitness for generators and discriminators), COEGAN with skill rating applied as fitness, a random search approach, and a DCGAN-based architecture. We also present a comparison between the FID score and the skill rating metric in experiments with the MNIST dataset.

4.1.2.1 *Experimental Setup*

Table 8 lists the parameters used in our experiments. These parameters were chosen based on preliminary experiments and the results presented previously (Costa et al., 2019; Costa, Lourenço, and Machado, 2019). All experiments are executed for 50 generations. The number of individuals in the populations of generators and discriminators is 10. This number of individuals is enough to achieve the recommended matches to feed the Glicko-2 rating system. For the variation operators, we use the rates 20%, 10%, and 10% for the add layer rate, remove layer rate, and change layer rate, respectively. The number of output channels is sampled using the interval $[32, 256]$. A tournament with $k_t = 2$ is applied inside each species to select the individuals for reproduction and the algorithm self-adjust to contains 3 species for the population of generators and discriminators. The number of samples used to calculate the FID score is 2048. To make the experiments comparable, each individual has a genome limited to 4 genes, the same number of layers used in the DCGAN-based experiments. Besides, as the DCGAN-based model does not use an evolutionary algorithm, these evolutionary parameters described above are not applied to it.

The initial skill rating parameters used in the experiments are the same suggested by the Glicko-2 system (Glickman, 2013), i.e., the rating r, deviation RD, and the volatility σ are initialized with 1500, 150, and 0.06, respectively. The system constant τ was set to 1.0. We conduct previous experiments to choose the best τ for our context. We found no relevant changes with respect to this parameter.

All experiments used the original GAN model, i.e., the neural networks are trained with the classical loss functions defined by Eq. 1 and Eq. 3. The GAN parameters were chosen based on preliminary experiments and the setup commonly used on the evaluation of GANs (Gulrajani et al., 2017; Karras et al., 2018; Radford, Metz, and Chintala, 2015). The batch size used in the training is 64. The Adam optimizer (Kingma and Ba, 2015) is used with the learning rate of 0.001, beta 1 of 0.5, and beta 2 of

Table 8: Experimental parameters.

Evolutionary Parameters	Value
Number of generations	50
Population size (generators and discriminators)	10
Add Layer rate	20%
Remove Layer rate	10%
Change Layer rate	10%
Output channels range	[32, 256]
Tournament k_t	2
FID samples	2048
Genome Limit	4
Species	3
Skill Rating Parameters	**Value**
r, RD, σ	1500, 350, 0.06
constant τ	1.0
GAN Parameters	**Value**
Batch size	64
Batches per generation	20
Optimizer	Adam
Learning rate	0.001
Betas	0.5, 0.999

0.999. Each pairing between generators and discriminators is trained by 20 batches per generation. As the *all vs. all* is used, each generator and discriminator will be trained for a total of 200 batches. For the DCGAN-based experiments, we have a single generator and discriminator. Therefore, we train them for 200 batches to keep the results comparable with the COEGAN experiments.

The results are evaluated using the FID score and the skill rating. For the SVHN dataset, the FID score is based on the Inception Network trained with the SVHN dataset instead of the ImageNet dataset, the same strategy used in the experiments of Olsson et al. (2018). For the MNIST results, we use the Inception Network trained with the ImageNet dataset. All results presented in this section are obtained by the average of five executions, with a confidence interval of 95%.

4.1.2.2 *Results*

Figure 33 presents the results of the best FID score per generation for the experiments with the SVHN dataset. We can see that the results for the original COEGAN proposal, i.e., COEGAN guided by the FID and the loss as fitness functions, are still better than the results for COEGAN with the skill rating metric. However, COEGAN guided by skill rating presented better FID scores than the random search approach. Thus, this evidences that skill rating provides useful information to the system, presenting evolutionary pressure to the individuals in the search of efficient models. Moreover, COEGAN with the FID score as fitness outperforms the DCGAN-based approach, illustrating the advantages of COEGAN.

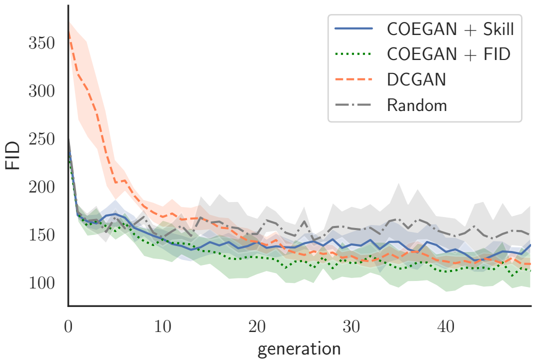


Figure 33: Best FID score for generators with a 95% confidence interval

We found in the experiments that skill rating sometimes overestimates the score for bad individuals, affecting the final results of the training. A dataset with the complexity of SVHN may require more training epochs to achieve better outcomes, and the variability introduced by the *all vs. all* pairing may be too large for complex datasets. Therefore, another approach such as spatial coevolution used by Al-Dujaili et al. (2018) and Toutouh, Hemberg, and O'Reilly (2019) will be considered in further experiments. Furthermore, the calculation of the match outcome, given by Eq. 24-27, can be improved to overcome this problem.

Table 9 shows the average FID of the best scores at the last generation for each experiment with the SVHN dataset. We can see the difference between the FID of the solutions evaluated in the experiments. As expected, the result for the random search approach is unstable and worse than the others.

Despite the slightly inferior results when compared to COEGAN with FID as fitness, the advantage of using the skill rating is that we can avoid the use of an external evaluator as in the FID calculation, represented by the Inception Network. The execution cost of the skill rating metric is also lower than the FID score and the FID score requires a high number of samples to have a good representation of the data. In our

Table 9: FID score of the algorithms used in the experiments with SVHN.

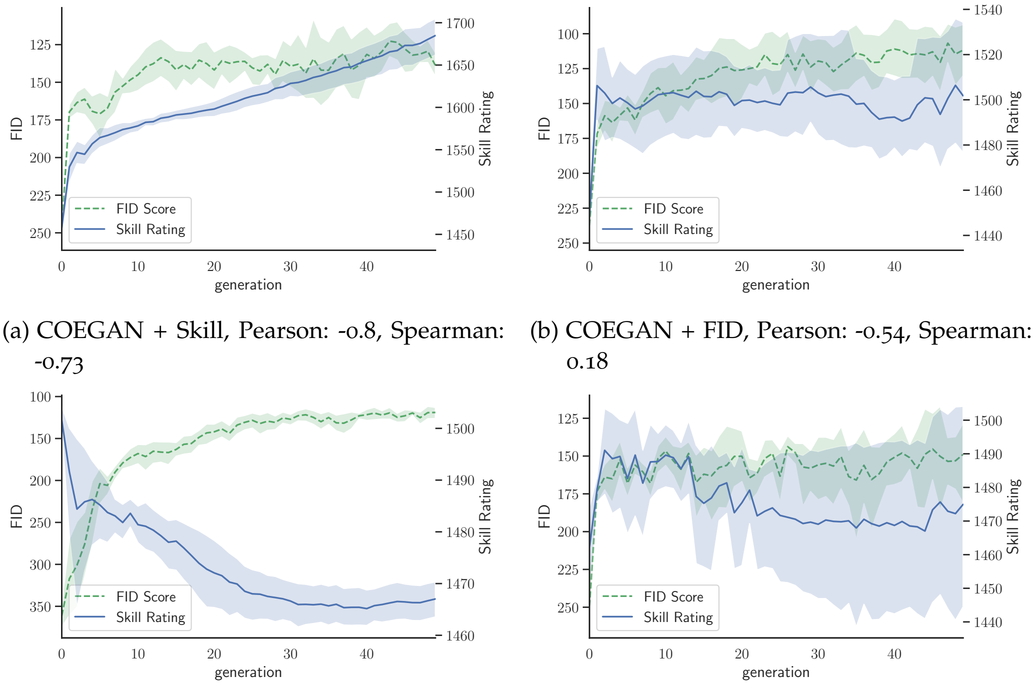
Algorithm	**FID Score**
COEGAN + Skill	135.1 ± 9.8
COEGAN + FID	111.7 ± 22.1
DCGAN-based	119.0 ± 10.1
Random search	148.9 ± 30.7

experiments, we use 2048 against 64 on the skill rating calculation (64 represents the batch size used in Eq. 26). Furthermore, the Inception Network has a complex architecture and the FID score uses slow procedures in the calculation. Skill rating uses the own neural network of individuals in the experiments, and the Glicko-2 system is fast to execute.

Figure 34 shows the progression of the skill rating through generations compared with the best FID scores. We can see in COEGAN guided by skill rating a clear improvement of the rating, as this is the same function used to provide evolutionary pressure in the individuals. In the experiments of COEGAN with FID, the progress also exists but is less relevant. The random approach presented an erratic behavior of the skill rating, showing that the individuals do not improve in this approach. In the DCGAN-based experiments, the skill rating behaves differently, showing a decreasing pattern. As there is only a single discriminator and generator, the number of matches per generation is only one. Therefore, we do not meet the recommendations of the Glicko-2 system of having at least ten matches per time period and the rating is not useful for this case.

Except for the DCGAN experiments, we can also see in Figure 34 some level of correlation between the best FID score and the respective skill rating among the generators in the populations. The results show that skill rating follows the tendency of the FID score, evidencing that it can be used to guide the evolution of GANs. We computed the Pearson correlation and the Spearman rank correlation between FID and skill rating to support this analysis. We found a relevant negative correlation for the experiments with COEGAN guided by skill rating, achieving a Pearson correlation coefficient of -0.8 and a Spearman rank correlation of -0.73. As FID is a distance measurement (lower is better) and skill rating is a score (high is better), a negative correlation is expected.

We experienced high variability on the FID score in the experiments with the SVHN dataset, both for the Inception Network trained with the ImageNet and SVHN datasets. Therefore, we conduct a study using the MNIST dataset to better understand the relationship between the FID score and skill rating. We used the same parameters



(a) COEGAN + Skill, Pearson: -0.8, Spearman: -0.73

(b) COEGAN + FID, Pearson: -0.54, Spearman: 0.18

(c) DCGAN-based, Pearson: 0.91, Spearman: 0.89

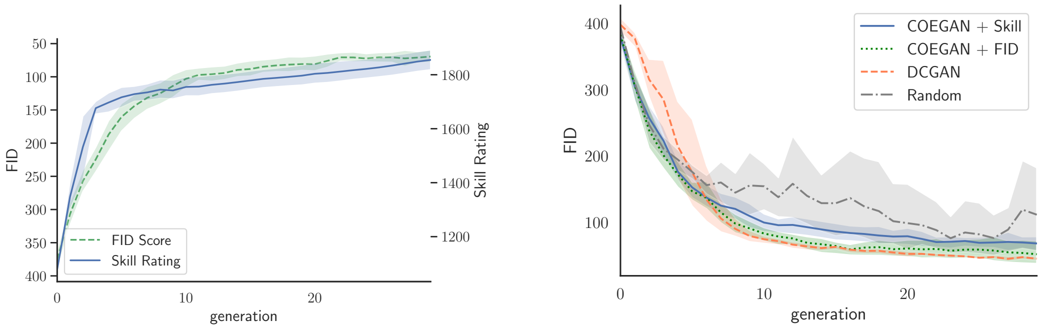
(d) Random search, Pearson: -0.16, Spearman: 0.02

Figure 34: Comparison between the best FID score and the respective skill rating of generators trained with the SVHN dataset.

presented in Table 8, but limiting the number of generations to 30. Figure 35(a) shows a smoother progression of skill rating and FID, illustrating a more clear relation between them, which is confirmed by the Pearson's correlation coefficient of −0.96 and the Spearman's rank correlation of −0.99.

We also show in Figure 35(b) and Table 10 that COEGAN guided by skill rating achieves performance similar to COEGAN guided by FID, outperforming the random search approach.

Figure 36 presents the average number of parameters in generators and discriminators from the experiments with the SVHN dataset. As there is no evolutionary algorithm applied to DCGAN, the number of parameters is constant. It is important to note that the average number of parameters on the individuals in the COEGAN experiments is much lower than the parameters in DCGAN. Despite this, the results of COEGAN are still better than DCGAN, showing that COEGAN is able to find more efficient models. We limited in the experimental setup the number of layers in the genome. Experiments with an expanded setup should be conducted to assess the possibility of even better results.



(a) Best FID score and the respective skill rating for COEGAN + Skill. Pearson: -0.96, Spearman: -0.99

(b) Best FID score for all solutions

Figure 35: Results for the experiments with the MNIST dataset.

Table 10: FID score of the algorithms used in the experiments with MNIST.

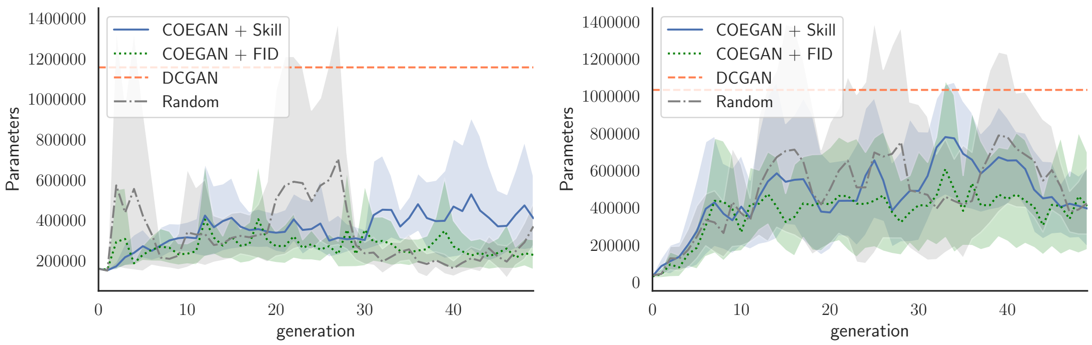
Algorithm	**FID Score**
COEGAN + Skill	67.7 ± 12.0
COEGAN + FID	51.5 ± 18.5
DCGAN-based	44.7 ± 3.1
Random search	111.2 ± 74.9

Figure 37 shows samples produced by the generator after the COEGAN training with FID and skill rating as fitness. In order to achieve better quality, we trained the algorithms using 200 batches at each generation (instead of 20). We can see that the quality of the samples is similar, with both strategies presenting variability on the samples.

4.1.3 *Discussion*

We propose the use of a game rating system, based on the application of Glicko-2 introduced by Olsson et al. (2018), to design a new fitness strategy for COEGAN. Specifically, we changed the fitness function used by discriminators and generators to use the skill rating metric instead of the loss function and the FID score. We conducted experiments to evaluate this proposal and compare the results with the previous COEGAN fitness proposal, a DCGAN-based approach, and a random search model.

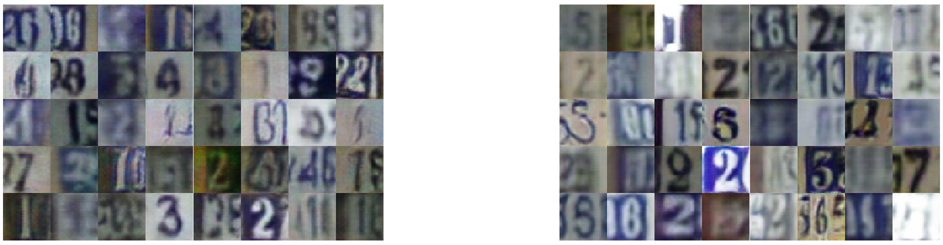
The results evidenced that, although the FID score as fitness provides better results, the skill rating method also contributes with useful information in the evolution of



(a) Number of parameters for generators

(b) Number of parameters for discriminators

Figure 36: Average number of parameters in the neural networks of generators and discriminators at each generation. Note that the number of parameters for the DCGAN-based experiments is constant, as there is not an evolutionary algorithm applied to this case.



(a) COEGAN with skill rating as fitness

(b) COEGAN with the FID score and loss function as fitness

Figure 37: Samples produced by the best generator after the COEGAN training.

GANs, while showing other advantages. The use of COEGAN with skill rating outperforms the random search approach, demonstrating the effectiveness of this fitness function. When compared to the FID score, the advantages when using skill rating are the lower execution cost and the self-contained solution, i.e., skill rating does not need to use an external component such as in the FID score. The calculation of the FID requires a trained Inception Network, making the score highly dependent on the context where it was trained and applied. Therefore, skill rating has the potential to be used in more domains. Besides, the skill rating does not require a neural network to interpret images produced by generators. Instead, the output of the discriminator is used in the calculation, resulting in a lower execution cost when compared to the FID score. We also show that there is a correlation between the FID score and the skill rating metric when using the latter as fitness with COEGAN. However, skill rating worked better with the MNIST dataset, making this correlation more evident. The SVHN dataset is more complex and sometimes leads to the disagreement between FID and skill rating.

The strategy to obtain the results of matches between generators and discriminators can be improved to better represent the player's skill. In this work, we use only individuals from the current generation in the skill rating calculation. However, there are common problems that occur in competitive coevolution algorithms that can affect this strategy, such as intransitivity. The intransitivity problem means that a solution a is better than other solution b and b is better than c, but it is not guaranteed that a is better than c, leading to cycling between solutions during the evolutionary process and harming the progress toward optimal outcomes (Antonio and Coello, 2018; Mitchell, 2006). Therefore, besides the matches between each pair (G_i, D_j), individuals in the current generation can also be matched against individuals from previous generations. The algorithm can keep an archive of the best individuals from previous generations to match them against the current individuals to ensure their progression. In this way, we can avoid the cycling between solutions and the intransitivity problem.

4.2 EXPLORING THE EVOLUTION OF GANS THROUGH QUALITY DIVERSITY

Experimental results with the original model show that COEGAN provides a more reliable training when compared to regular GANs in similar scenarios. However, the lack of diversity evidenced in the experimental evaluation (Section 3.2) affects the quality of the results leaving space for improvements. In this section we study the application of a novelty mechanism in COEGAN to improve the exploration of solutions. Quality Diversity (QD) algorithms are a class of solutions that can be used to enhance the population and produce a diversity of efficient individuals (Pugh et al., 2015).

We propose a new model combining concepts used on COEGAN and a QD algorithm for guiding the evolution of GANs. Instead of using strategies such as the speciation based on NeuroEvolution of Augmenting Topologies (NEAT) to support evolution, we propose the use of Novelty Search with Local Competition (NSLC) (Lehman and Stanley, 2011), a quality diversity algorithm that uses mechanisms of novelty in the search for efficient solutions. We aim to improve the exploration of the search space and achieve better models for generators and discriminators.

To validate our proposal, we conducted experiments using the MNIST (LeCun, Cortes, and Burges, 1998) and CelebA (Liu et al., 2015) datasets. We compare the results between the original COEGAN approach and two variations of our proposal: COEGAN with NSLC and with an alternative using a global competition strategy.

4.2.1 *Model*

We propose a different evolutionary approach to guide the COEGAN evolutionary process by replacing the NEAT-based evolutionary algorithm with an approach based on Novelty Search with Local Competition (NSLC) (Lehman and Stanley, 2011). As originally proposed by Lehman and Stanley (2011), we use Nondominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al., 2002) as the Multi-Objective Evolutionary Algorithm (MOEA) for NSLC.

The new proposal keeps some components from the original COEGAN model, namely the genotypic representation, the variation operators, and the competitive coevolutionary model (Section 3.1). Next we describe the differences and new aspects of the model proposed in this work.

We designed the pairing between individuals at the evaluation phase according to the *all vs. all* coevolution model used in COEGAN. However, as NSGA-II uses an elitist approach to select individuals to form the next generation, both the current population and the derived offspring should be evaluated. For this, we match generators from the current population with discriminators from the offspring, and discriminators from the current population with generators from the offspring population. This ensures the progress of all individuals through generations, making it possible to properly select them when targeting quality and diversity. The main drawback here is that now the double amount of individuals needs to be evaluated, increasing the execution cost of the algorithm. However, it is important to note that we keep the same number of training steps for each individual when compared to the original COEGAN approach.

Tournament is applied to select individuals for reproduction. As proposed by the NSGA-II algorithm, a dominance operator is used to determine the result of the tournament between a set of individuals. In our work, we use the constrained version of the operator, ensuring that the population does not deviate too much from the objective defined by the fitness function. Thus, we use not only the concept of dominance but also the feasibility of solutions.

NSGA-II designs the dominance operator using the ranking of solutions determined by the nondominated sorting algorithm, aiming to obtain solutions in the Pareto-optimal front. The feasibility concept ensures that, when comparing two solutions s_i and s_j, the fitness function meets the constraint $f(s_j) < 2f(s_i)$, otherwise s_j is considered unfeasible. In summary, the solution s_i constrained-dominates a solution s_j when s_i is feasible and s_j is not, or both solutions are feasible and s_i dominates s_j. Note that one of the solutions will always be feasible, i.e., the case that both solutions are unfeasible is not possible.

The definition of the neighborhood is paramount to the NSLC algorithm, being used in both the quality and diversity criteria. To determine the neighbors of each individual, we use the distance between the architectures of the neural networks of individuals, which is directly defined by the similarity between genomes. This distance is the same used originally in COEGAN to group individuals into species. Two individuals are considered equal if they have the same genome, i.e., the same sequence of genes, disregarding other characteristics like age or the number of samples currently used in the training. It is important to note that the neighborhood calculation considers not only the current population but also the archive of previous solutions. This archive is filled following a probabilistic approach, i.e., at each generation individuals are inserted into the archive with a certain probability.

In NSLC, n nearest neighbors of an individual are selected to calculate the innovation and the competition objectives. Innovation is defined by the average distance between the individual and the neighborhood. The competition objective is defined by the number of neighbors the individual outperforms with respect to the fitness value. In our proposal, the fitness is the same used in COEGAN: Eq. (1) for discriminators and Eq. (3) for generators.

The innovation criteria make it possible to better explore the available architectures characterized by the genotype representation. When combined with the strategy used to calculate the competition score, different niches can be efficiently explored to leverage the search space. Compared to the original COEGAN approach, we expect to improve the diversity of solutions and eventually find better results concerning the FID score. In COEGAN, the number of species is fixed and needs to be defined beforehand, being a limitation over the capacity of innovation for individuals that need to survive through generations to show consistent performances. In COEGAN guided by NSLC, the exploration of the search space is improved by the novelty criterion, using the quality definition to guide the population through the objective of obtaining better solutions.

4.2.2 *Experiments*

To validate our proposal, we present an experimental analysis of the application of Quality Diversity in the evolution of GANs[1]. Therefore, we conduct experiments using MNIST to evidence the performance of the algorithm proposed in this work compared with the original COEGAN model. We also design experiments with an alternative version of the solution which uses a global competition mechanism, i.e., the neighborhood is not limited by a constant n and uses all individuals available. We call this version of the algorithm Novelty Search with Global Competition (NSGC), inspired by the global competition approach experimented by Lehman and Stanley (2011). Therefore, we refer in our experiments to three variations of COEGAN:

- COEGAN: The original COEGAN proposal;
- COEGAN+NSLC: COEGAN trained with the NSLC algorithm;
- COEGAN+NSGC: COEGAN trained with the NSGC algorithm;

The FID score was used to measure the quality of the produced samples. Besides, the strategy proposed by Zhang et al. (2018b) was applied to present the visual distribution of image samples, using t-SNE (Maaten and Hinton, 2008) to embed samples into

[1] Code available at https://github.com/vfcosta/qd-coegan.

a two-dimensional space. Further experiments with the CelebA dataset were made to compare our method with a non-evolutionary GAN approach in a more complex dataset.

4.2.2.1 *Experimental Setup*

Table 11 lists the parameters used in our experiments. These parameters were selected based on the experiments presented in Chapter 3. The number of generations used in all experiments is 50. Each population of generators and discriminators contains 10 individuals. We use a probability of 30%, 10%, and 10% for the add, remove and change mutations, respectively. The genome of generators and discriminators was limited to four genes, representing a network of four layers in the maximum allowed setup. This setup is sufficient to discover efficient solutions for the experiments with the MNIST dataset.

For the original COEGAN, we use 3 species in each population of generators and discriminators. For COEGAN with NSLC, the number of neighbors n is limited to 3 and the probability to insert individuals into the archive is 10%. The global version of the QD algorithm does not limit the neighborhood, using all individuals when calculating the novelty and competition values.

Figures in this section display plots with curves representing the average of the results from 15 independent executions, with a confidence interval of 95%.

4.2.2.2 *Results*

We present in this section the results of the experimental analysis, comparing the solutions using the QD algorithm with the previously proposed COEGAN model. First, we present the results using the MNIST dataset. Then, we provide a further analysis with CelebA, a more complex dataset, comparing our proposal with a regular GAN approach.

Figure 38 shows the FID score of the best individuals for each generation when training with the MNIST dataset. We can see that the COEGAN+NSLC solution outperforms the original COEGAN model by a small margin until approximately half generations but has equivalent performance at the end. This effect is mostly due to the increased exploration capability given by the QD algorithm, which produces a more diverse population but causes a less focused evolution of more fitted individuals. Besides, COEGAN+NSGC, the global competition variation, has better performance than COEGAN+NSLC and the original COEGAN approach. The results obtained by comparing the global and local competition versions of the algorithm are similar to results presented by Deb et al. (2002), where the global version also achieved better fitness than NSLC.

We can see in Figure 39 and Table 12 that COEGAN+NSGC consistently achieved better results than the other approaches. COEGAN provides more unstable results

Table 11: Experimental parameters.

Evolutionary Parameters	**Value**
Number of generations	50
Population size (generators)	10
Population size (discriminators)	10
Add Layer rate	30%
Remove Layer rate	10%
Change Layer rate	10%
Output channels range	[32, 256]
Tournament k_t	2
FID samples	1024
Genome Limit	4
Species	3
Neighborhood size n	3
Archive probability	10%
GAN Parameters	**Value**
Batch size	64
Batches per generation	50
Optimizer	Adam
Learning rate	0.001

when compared to the global approach, presenting a higher standard deviation in FID values when trained with the experimental setup described in Table 11. This effect is also present in the experiments with COEGAN+NSLC, indicating that the high diversity produced by our experimental setup affects the results with respect to the FID score. Tacking into account these results, we conclude that the diversity provided by COEGAN+NSGC is sufficient to achieve better results in our approach for training GANs.

To support our analysis, we statistically test the significance of our findings. We assume that the results do not follow a normal distribution, as the normality test (Shapiro-Wilk, $\alpha = 0.05$) rejected this hypothesis for COEGAN and COEGAN+NSLC ($p < 0.001$). Then, we used a non-parametric test (Mann-Whitney U, $\alpha = 0.05$) with Holm-Bonferroni correction to perform a pairwise comparison between the solutions evaluated in this work. We found that the improvement of COEGAN+NSGC over CO-

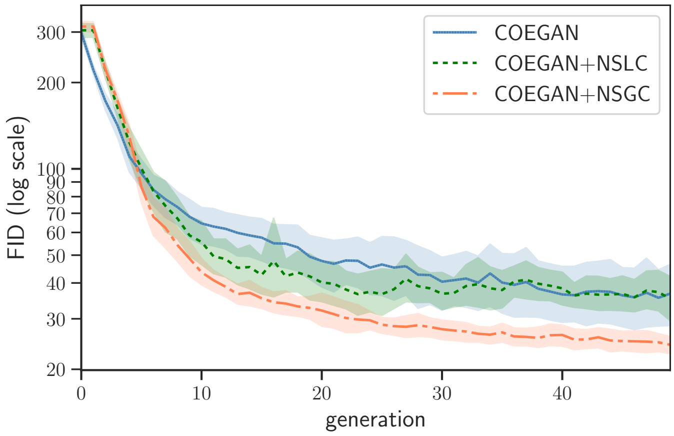


Figure 38: Best FID Score on the MNIST dataset.

Table 12: Average FID score of best generators after training with the MNIST dataset.

Algorithm	**FID Score**
COEGAN	36.8 ± 18.6
COEGAN+NSLC	35.2 ± 12.5
COEGAN+NSGC	24.3 ± 3.3

EGAN is statistically significant ($p = 0.03$). Moreover, the performance improvement of COEGAN+NSGC over COEGAN+NSLC is also statistically significant ($p < 0.001$). We found no statistical difference between COEGAN and COEGAN+NSLC ($p = 0.34$).

We also show in Figure 40 the FID scores of all individuals in the population of generators when training with MNIST. This result evidences that COEGAN+NSLC has a better exploration of the search space, increasing the diversity and leading to the discovery of not only good individuals but also less efficient solutions. The local competition approach used in NSLC has a stronger effect on the protection of innovation, as the competition calculation uses the fitness values from similar individuals, i.e., it uses the n closest neighbors concerning the architectural similarity. The effect of this is the discovery of niches that are not efficient in terms of fitness. In the scenario of global competition, individuals have to outperform a broader range of solutions to survive through generations, leading to the discovery of better models.

Figures 41 and 42 display the average number of samples used in the training process for all discriminators and generators in the population, respectively. In these charts, we confirm the effect of the novelty strategy applied in our solution, which is evident in Figure 41. These results evidence that newer individuals were more frequently selected through generations in the solutions based on the QD algorithm,

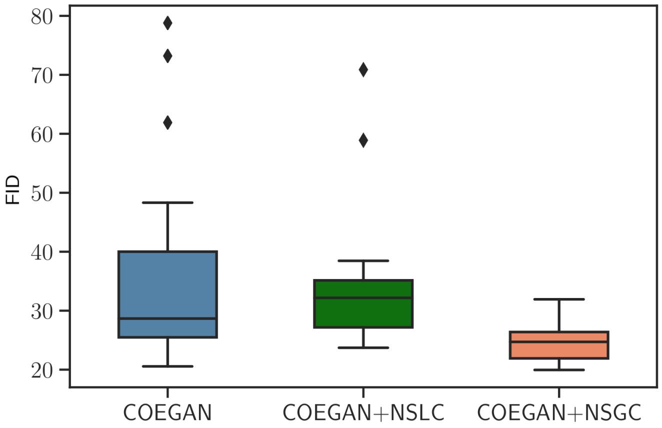


Figure 39: Boxplot of the FID score on MNIST dataset showing the performance of best generators computed for each independent run.

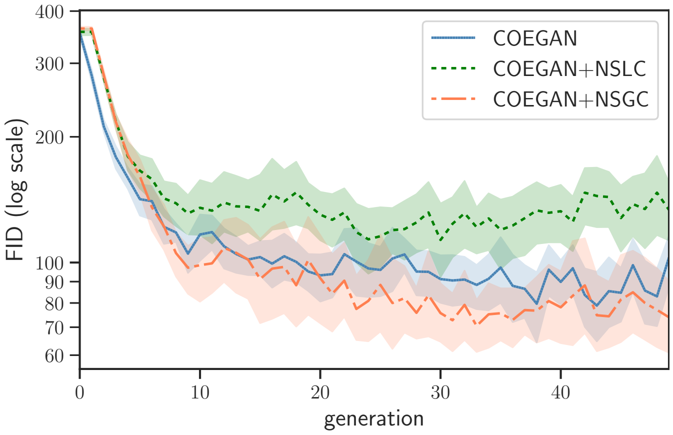


Figure 40: Average FID Score on the MNIST dataset.

resulting in fewer training samples directly seen by them (new individuals can have new genes introduced by variation operators). As expected, we also show that novelty is more present in the local competition solution when compared to the global competition version.

Figures 43 and 44 provide additional support for this analysis, showing the number of samples used in training of the best individuals in the population of discriminators and generators, respectively. The curves follow a similar behavior of Figures 41 and 42, evidencing that best individuals at each generation also present more innovation in the solutions using COEGAN+NSLC.

A difference in the novelty effect is evident when comparing Figure 41 to Figure 42 and Figure 43 to Figure 44. The effect of innovation is more evident for discriminators (Figures 41 and 43) than in the results with generators (Figures 42 and 44). We

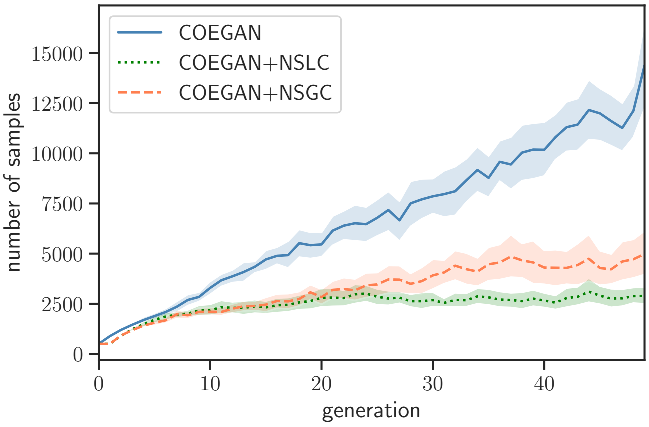


Figure 41: Average number of samples used to train all discriminators with MNIST.

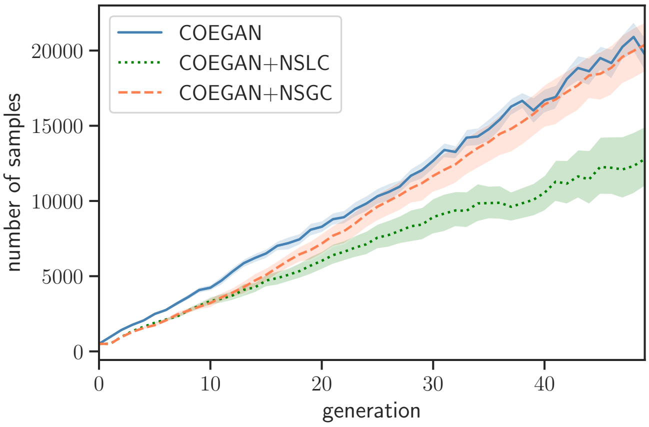


Figure 42: Average number of samples used to train all generators with MNIST.

attribute this difference to the choice of fitness functions. As concluded in Chapter 3, the FID score used in generators is a more reliable metric than the loss function used in discriminators. This affects the quality criterion used in the NSGA-II optimization method, making the selection of better individuals more assertive. However, further experiments are required to confirm this effect on innovation in the population.

To better study the quality and diversity achieved by our model, we present in Figure 45 the distribution of samples produced by the best generator at different steps of the training process in one execution using the COEGAN+NSGC approach. Samples are placed in a 40×40 grid, positioned according to t-SNE[2]. For this, 1600 samples from each case were used in the t-SNE training and a discretization function is applied

[2] An expanded version of these images using a 120×120 grid is available at https://github.com/vfcosta/qd-coegan.

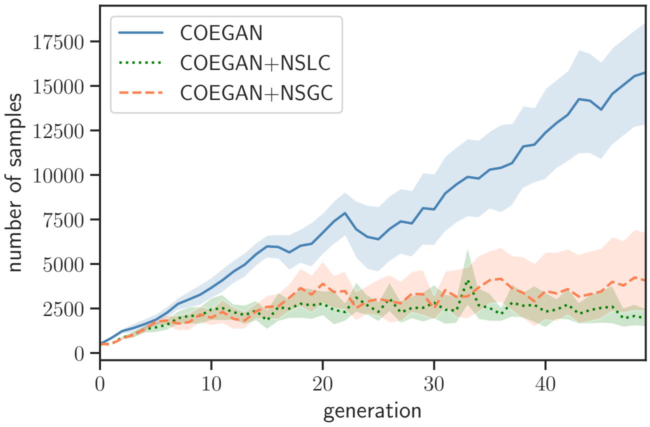


Figure 43: Average number of samples used to train best discriminators with MNIST.

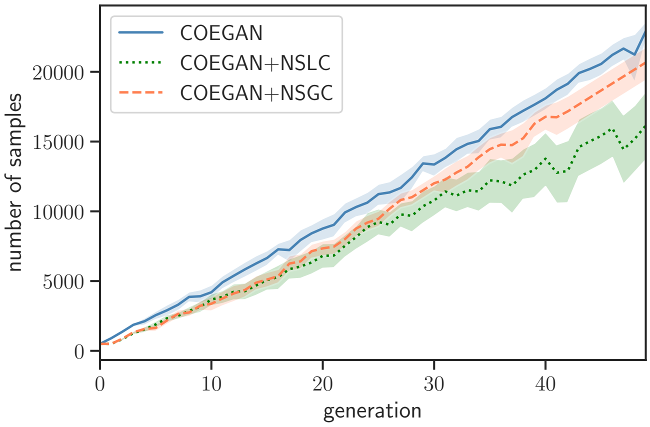
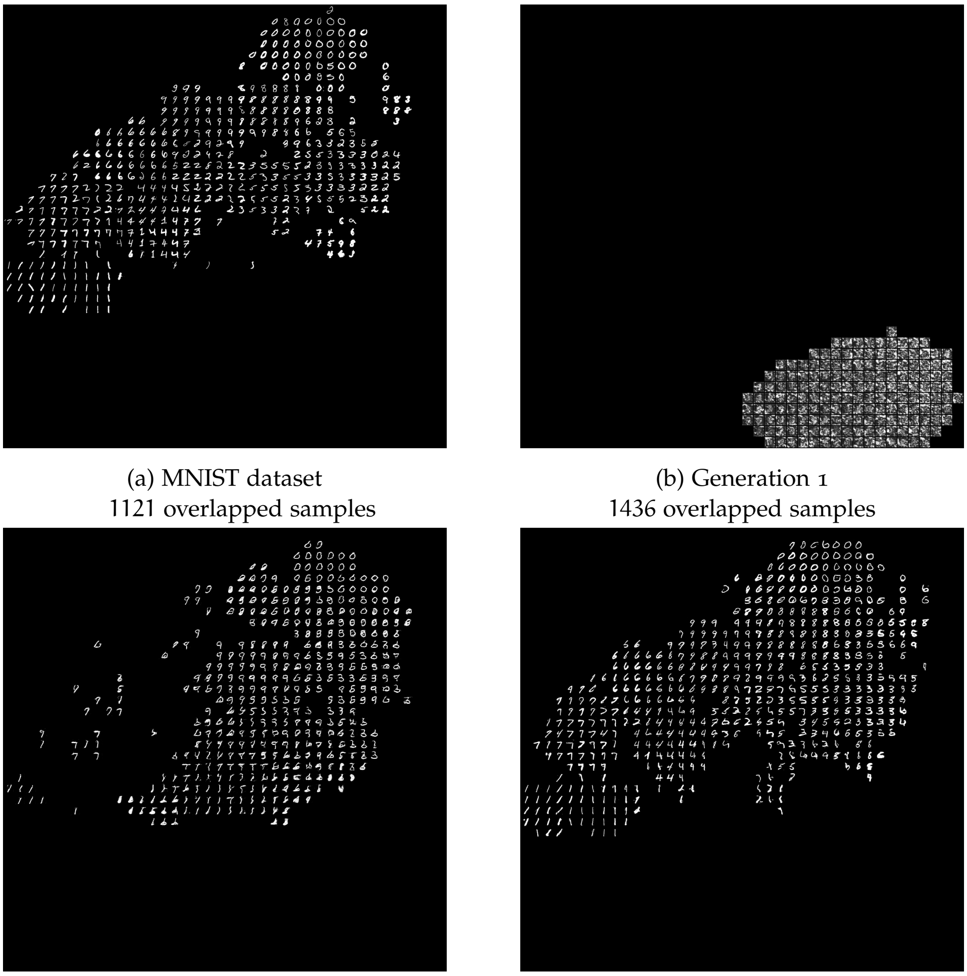


Figure 44: Average number of samples used to train best generators with MNIST.

to place these samples into the two-dimensional space. This method results in some overlapping samples, which indicates the level of diversity obtained by a model, i.e., fewer overlapping samples is evidence of better representation of the latent space.

Figure 45(a) represents the distribution of the MNIST dataset, i.e., the variety of samples used in training. We can see in Figure 45(b) that, at the initial stage, the samples are noisy and do not resemble images from MNIST, creating a high number of overlapping samples. In generation 10, represented by Figure 45(c), the distribution of samples is closer to the presented in Figure 45(a), although we can still see some lack of quality and under-representation of some digits. Figure 45(d) shows samples produced after the whole evolutionary process. These samples have better quality and preserve diversity, resulting in 1077 overlapping samples, even lower than the 1121 overlapping samples presenting in the MNIST dataset.



(a) MNIST dataset
1121 overlapped samples

(b) Generation 1
1436 overlapped samples

(c) Generation 10
1189 overlapped samples

(d) Generation 50
1077 overlapped samples

Figure 45: Distribution of samples using t-SNE with the MNIST Dataset. We show samples (a) from the input dataset, the best generator at the (b) first generation, (c) after ten generations, and (d) at the end of training. We fed t-SNE with 1600 samples from each scenario and used the results for positioning them into a two-dimensional space. The number of overlapped samples is displayed for each case.

To assess the efficiency of our solution in complex datasets, we used COEGAN+NSGC, the best performing version of the proposed algorithm, to conduct experiments with CelebA (Liu et al., 2015). For the sake of simplicity, we reduced the type of layers only to convolution and transpose convolution layers when adding a new gene, excluding the linear layer from the set of possibilities. Besides, the activation functions were restricted to ReLU and Leaky ReLU in the mutation operators. The populations of generators and discriminators contain 5 individuals each. The images from the CelebA

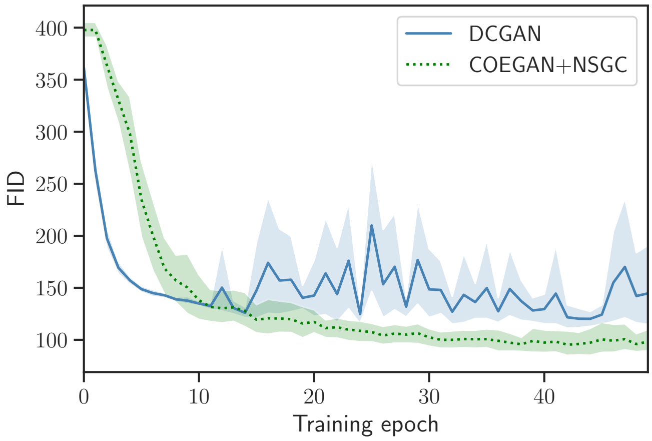


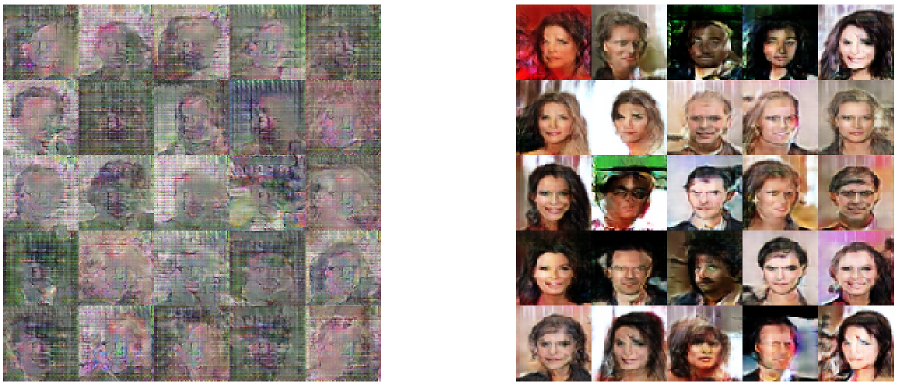
Figure 46: Best FID Score on the CelebA dataset.

dataset were rescaled to 64 × 64. To handle images of bigger sizes, we increased the genome limit to 5 and the number of batches per generation to 200. The remainder of the parameters is the same presented in Table 11.

We compare the results of our approach with a regular GAN that uses an architecture based on DCGAN (Radford, Metz, and Chintala, 2015). In the DCGAN-based experiments, the architecture of the generator and the discriminator is composed of four layers. Previous experiments with neural networks using five layers were conducted but the results were more unstable, making the four-layers version more suitable for comparison with COEGAN+NSGC. It is important to note that we ensure the DCGAN approach is trained with the same number of samples of an individual in COEGAN+NSGC. Therefore, we define one training epoch as the training of DCGAN with 1000 batches. For COEGAN+NSGC, one training epoch is equivalent to one generation of the evolutionary algorithm. As we use the *all vs. all* pairing approach, each individual is also trained with 1000 batches per generation (5 individuals times 200 batches).

Figure 46 presents the progression of the best FID score when training with CelebA. We can see a smooth progression of the FID in the COEGAN+NSGC approach, leading to a final result consistently better when compared to the DCGAN-based solution. In DCGAN, FID varies during the training epoch, demonstrating spikes during the process mostly due to the occurrence of common stability issues on the GAN training, such as the mode collapse problem (Brock, Donahue, and Simonyan, 2019). This is evidence that our approach provides more stability on GAN training when compared to regular GANs with similar architectures.

Figure 47 shows samples created by both approaches when trained with the CelebA dataset. Figure 47(a) displays samples created by the DCGAN approach after the final epoch when issues were observed in training. This is an example of the resulting effect



(a) DCGAN

(b) COEGAN+NSGC

Figure 47: Samples created by (a) DCGAN after collapsing in final training epoch and by (b) COEGAN+NSGC after training.

of a spike that occurred in the DCGAN training. In Figure 47(b) we can see samples produced after training with COEGAN+NSGC. Although spikes are not present, the quality of samples produced by COEGAN+NSGC is still not perfect, but better than DCGAN.

4.2.3 *Discussion*

We have investigated in this section the application of a QD algorithm to train and evolve GANs. Chapter 3 presents COEGAN, a model that uses coevolution with an EA inspired by NEAT to train GANs. However, the lack of diversity and premature optimization leave room for improvement of the solution.

We propose an extension of COEGAN to use a Quality Diversity (QD) algorithm in order to improve the exploration of the search space. Therefore, we design a new EA that combines COEGAN with the approach used in the Novelty Search with Local Competition (NSLC) algorithm.

The experimental results show that the use of QD to guide the evolution of GANs improved the diversity in the population, leading to the discovery of better models. Furthermore, experiments with a version of the algorithm using global competition evidence that we can consistently outperform the previous results of COEGAN in the MNIST dataset. Experiments with the CelebA dataset indicate that our proposal provides a more stable training when compared to a regular GAN based on the Deep Convolutional Generative Adversarial Networks (DCGAN) architecture, avoiding problems such as mode collapse and vanishing gradient.

4.3 SUMMARY

In this chapter, we revisit the COEGAN model to address limitations previously identified in Section 3.4. Thus, we explore evolutionary aspects of the algorithm to propose the incorporation of new mechanisms to improve the model. This originated two variations of the original model. The first, described in Section 4.1, proposes to replace the fitness with functions based on the Skill Rating. The second proposal, described in Section 4.2, focused on the structure of the EA. Therefore, recent advances in the field were investigated and we redesigned the algorithm using a QD approach.

The experiments with the Skill Rating in COEGAN shows that it contributes with useful information when used as fitness for evolving GANs. When compared to the original COEGAN proposal, that uses the FID score as fitness, COEGAN with the skill rating results in a lower execution cost and a self-contained solution, i.e., it does not depends on an additional neural network such as in the FID score. We also show that there is a correlation between the FID score and the skill rating as fitness in COEGAN.

The experiments using QD with COEGAN shows that the diversity in the population was improved, leading to the discovery of better models. Furthermore, the QD approach consistently outperformed the results of the original COEGAN and provides a more stable training when compared to a regular GAN, avoiding problems such as mode collapse and vanishing gradient.

Other strategies to further improve the model rely on the incorporation of recent advances related to GANs in the model. For example, using loss functions that were demonstrated to be more robust and provide better results with respect to the generative quality. Chapter 5 explores this approach and presents new variations of COEGAN that incorporate new mechanisms for GANs.

5 INCORPORATING GAN ADVANCES INTO THE EVOLUTIONARY MODEL

This chapter presents our proposals to improve Coevolutionary Generative Adversarial Networks (COEGAN) through the incorporation of recent advances proposed for Generative Adversarial Networks (GANs). Namely, we introduce new alternatives for the loss functions, different kinds of layers, and a new mutation operator. Experiments using different datasets were conducted to validate the effects of these proposals.

The remainder of this chapter is organized as follows. Section 5.1 proposes a mutation operator to switch the loss functions used in the GAN training. Section 5.2 incorporates recent advances proposed for GANs into the model. Finally, Section 5.3 summarizes the conclusions and points out directions for further improvement.

5.1 MUTATION OF LOSS FUNCTIONS

The loss functions are paramount for the success of the GAN training. Recently, several proposals of loss functions have emerged in the literature. Taking this into account, we propose a new variation operator that expands the possible loss functions used in discriminators and generators. In this way, we incorporate into COEGAN the improved loss functions developed for GANs to leverage the performance of the generative model. The objective of this study is to assess the contributions of alternative loss functions in our coevolutionary setup.

To evaluate our hypothesis, we devised and performed a series of experiments to compare the original COEGAN proposal with the new variation operator. We also compare COEGAN with a classical GAN approach with and without mechanisms to achieve better GAN training.

5.1.1 *New Mutation Operator*

We propose in this section an extension of the original COEGAN variation operators to introduce the mutation of the loss function, where the offspring have a probability to switch the current loss function to one randomly selected from a predefined list. With this new operator, we can have individuals with different losses, similarly to E-GAN.

To keep the performance of discriminators comparable, we always use Eq. (1) as the fitness, even when a different loss function is used for training the discriminator. Therefore, when comparing individuals for the selection phase, discriminators that were mutated to use a different loss function will still be evaluated using the loss function originally defined for GANs.

We selected the loss functions from WGAN, LSGAN, RSGAN, and RaSGAN as the set of possibilities for this mutation operator. Therefore, in addition to the original loss functions defined in Eq. (1) and Eq. (3), we use the alternative loss functions defined in Section 2.1.4.2 by Equations (8), (9), (11), (12), (13), (14), (15), and (16).

It is important to note that these alternative losses do not output probabilities as the discriminator result. Therefore, the sigmoid function is not applied in the final layer of discriminators. These variations output unbounded scores used as input in their specific loss functions.

5.1.2 *Experiments*

We design a set of experiments to assess the contributions of our model regarding the quality and stability in the GAN training using Fashion-MNIST to assess the performance of COEGAN under different scenarios. To evaluate the quality of the resulting models, we use the FID score, i.e., the same function used for fitness in generators.

First, we evaluate our new proposed mutation operator in comparison with COEGAN. Next, we compare COEGAN with a DCGAN-based model with and without batch normalization, similarly to the experiments conducted for WGAN by Arjovsky, Chintala, and Bottou (2017). With this study, we want to show that COEGAN is able to evolve robust models, even without using mechanisms to improve the training stability.

5.1.2.1 *Experimental Setup*

Table 13 lists the parameters used in our experiments. These parameters were selected based on the results of our previous experiments (Chapters 3 and 4). The number of generations used in all experiments is 100. Each population of generators and discriminators contains 10 individuals. We use a probability of 30%, 10%, and 10% for the add, remove, and change mutations, respectively. To make the comparison between COEGAN and DCGAN fair, the genome of generators and discriminators was limited to four genes for COEGAN, representing a network of four layers in the maximum allowed setup. We also limited the type of layers only to convolution and transpose convolution layers for the addition mutation. This setup is a compromise between a low computational cost and the capacity of discovering efficient solutions to the problems being tackled. More layers would be suitable only for datasets with larger dimensions. We use three species in each population of generators and discriminators. To evaluate generators, we apply the FID score with 5000 samples.

To apply the GAN training algorithm, we use a batch of 64 images and the Adam optimizer (Kingma and Ba, 2015) with 0.001 as the learning rate. COEGAN uses a

Table 13: Experimental Parameters

Evolutionary Parameters	**Value**
Number of generations	100
Population size (generators)	10
Population size (discriminators)	10
Add Layer rate	30%
Remove Layer rate	10%
Change Layer rate	10%
Mutate Loss function rate	30%
Output channels range	[32, 512]
Tournament k_t	2
FID samples	5000
Genome Limit	4
Species	3
GAN Parameters	**Value**
Batch size	64
Batches per generation	10
Optimizer	Adam
Learning rate	0.003

limited set of input data at each step. Therefore, we use 10 batches per training pair. Preliminary experiments evidenced that a bigger number of batches brings instability to the system. This happens because bad individuals can exist in the population, and the all vs. all pairing approach will make use of them extensively in training. Hence, small steps on training, i.e., a smaller number of batches, achieved better results in our model.

Figures in this section display plots with curves representing the average of the results from 20 independent executions, with a confidence interval of 95%.

5.1.2.2 *Results*

Figure 48 shows the progress of the FID score through evolution for the classical COEGAN proposal and COEGAN with mutation of the loss function enabled.

We can see that the curve of the original COEGAN decreases faster. This is evidence that COEGAN is already robust enough to solve the common stability issues in GANs. Therefore, the alternative loss functions used in our experiments do not contribute to

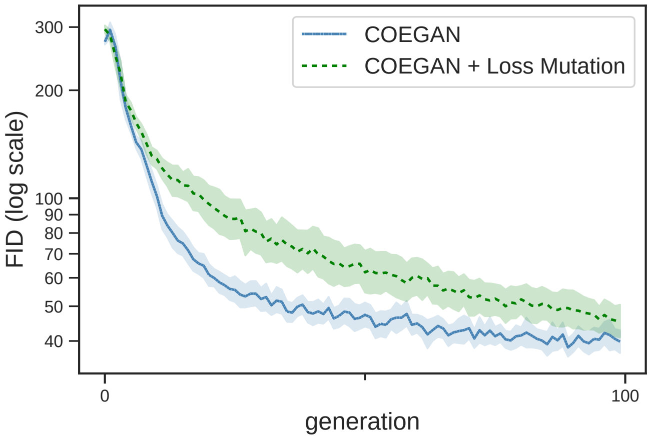


Figure 48: Best FID Score on the Fashion-MNIST dataset for COEGAN and COEGAN with mutation of the loss functions.

the improvement of the proposed model. Besides, mixing different loss functions in a population of individuals brings more complexity to the system, leading to the creation of more evolutionary paths that make the algorithm harder to achieve consistent results.

As suggested by Lucic et al. (2017), different alternatives for GANs can achieve similar results when in similar conditions and tuned properly. This is aligned with our results, showing that individuals in COEGAN could not benefit from these different approaches.

We present in Figures 49, 50, 51, and 52, how the individuals are using the possible loss functions in our experiments. In these charts, each row represents a loss function and the generations are represented in the horizontal axis. Each circle corresponds to an individual, and the darker circles are the best individuals for each specific generation. Following, we analyze the insights achieved through these charts.

In Figure 49 and Figure 50 we show the distribution through generations of individuals regarding the loss function in one execution of COEGAN. As mutation of the loss function is enabled in this case, we can see how individuals switch through loss functions.

For generators (Figure 49), we can see that all loss functions were used and became the best for some generations. However, for discriminators (Figure 50), we can clearly see a tendency to choose individuals with the loss function originally proposed for discriminators in Eq. (1). In the execution represented by Figure 50, individuals with this loss function performed better than others in almost all generations. It is also important to note that, to keep individuals comparable, we use Eq. (1) as fitness even when other losses are used. Therefore, this tendency is justified by the selection pressure determined by the use of a specific loss function as fitness.

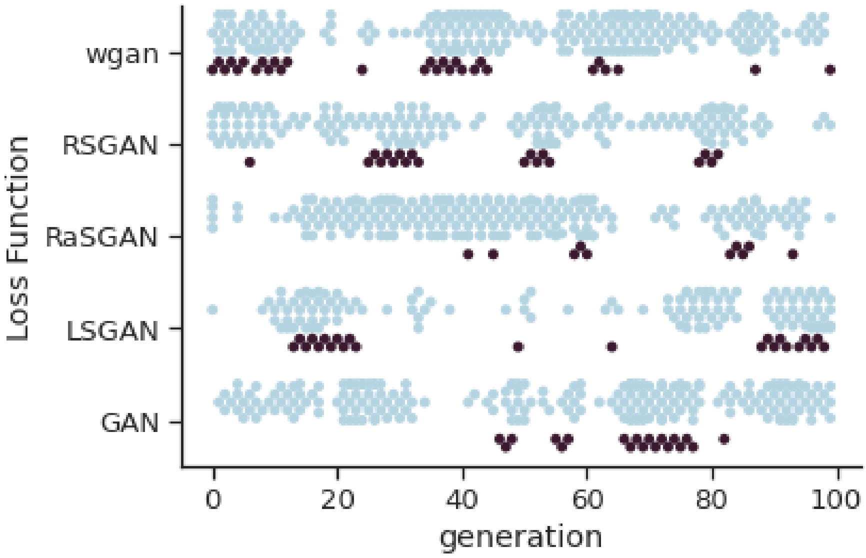


Figure 49: Distribution of individuals in the population of generators regarding the loss function. The best individual of each generation is darker.

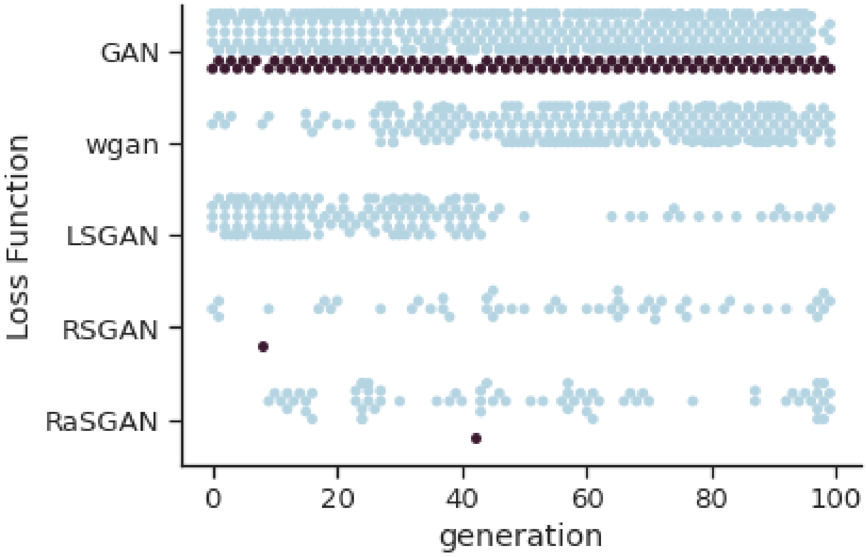


Figure 50: Distribution of individuals in the population of discriminators regarding the loss function. The best individual of each generation is darker.

Figure 51 shows the distribution of loss functions for discriminators in all executions. We can see that in almost all executions and generations the best individual uses the original loss function for discriminators. Therefore, the tendency to select individuals with the original GAN function is evident.

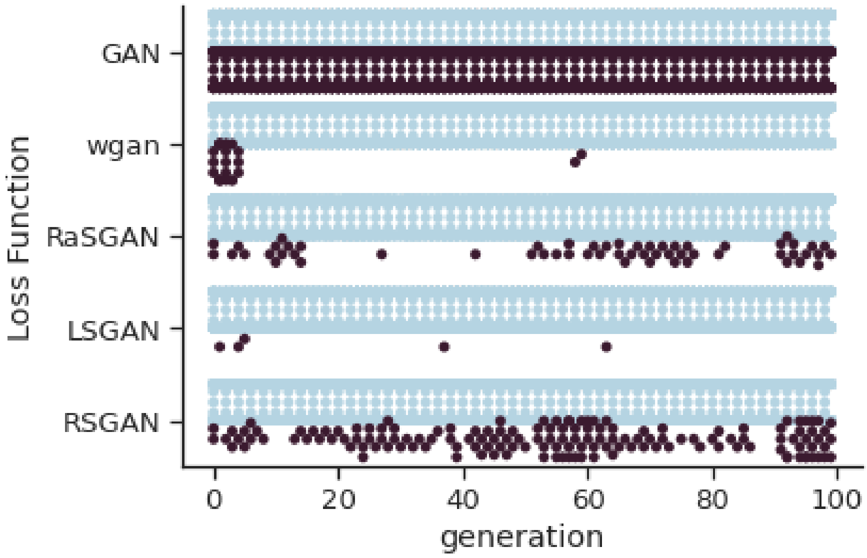


Figure 51: Distribution of individuals in the population of discriminators regarding the loss function for all executions (best individuals are darker).

For generators (Figure 52), we have better distribution of different loss functions in the population. As we use the FID score as fitness, the loss function is not used as fitness in this case.

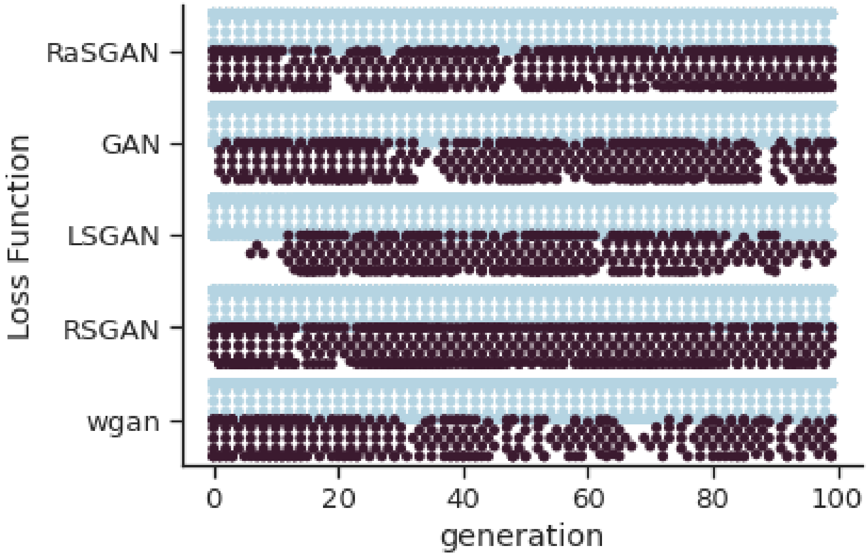


Figure 52: Distribution of individuals in the population of generators regarding the loss function for all executions. The best individual of each generation is darker.

Given these results, we conduct further experiments to assess how other strategies used to improve GANs impact COEGAN. Specifically, we compare the original proposal with a model based on DCGAN under different conditions. We disable batch normalization in COEGAN and DCGAN to assess the behavior of training in this scenario, resulting in the following variants: COEGAN+BN, COEGAN-BN, DCGAN+BN, and DCGAN-BN; where +BN/-BN indicates the presence or absence of batch normalization in the architecture, respectively.

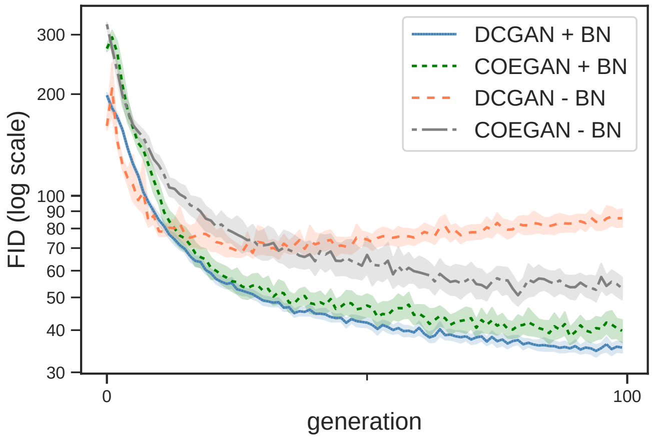


Figure 53: Best FID Score on the Fashion-MNIST dataset for COEGAN and DCGAN with and without batch normalization.

Figure 53 displays the progress of FID through generators. In this graph, we can clearly see that the solution based on DCGAN is not stable when batch normalization is disabled. However, we can see that COEGAN-BN is able to find robust models, as depicted by the decreasing behavior on the FID score.

Table 14: Average FID of best generators in Fashion-MNIST

Algorithm	**FID Score**
DCGAN+BN	35.5 ± 2.9
COEGAN+BN	39.8 ± 6.8
DCGAN-BN	85.8 ± 11.7
COEGAN-BN	53.2 ± 9.4

These results are confirmed by Figure 54, where the average FID between all executions is displayed. We can also see that COEGAN consistently outperforms DCGAN when batch normalization is disabled. When comparing COEGAN+BN and DCGAN+BN, the performance is similar, with COEGAN+BN achieving better results in some executions. The benefits of COEGAN are more visible in complex datasets such

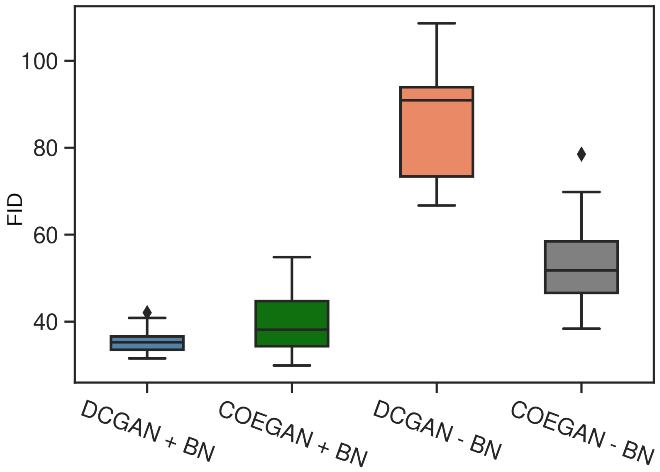


Figure 54: Boxplot of the FID score on Fashion-MNIST dataset showing the performance of best generators computed for each independent run.

as CelebA (Liu et al., 2015), as experimentally demonstrated in (Costa et al., 2020b). The results also evidenced that batch normalization contributes to the performance, although the effect is smaller in COEGAN compared with DCGAN.

When comparing COEGAN-BN and DCGAN-BN we provide evidence of the stability introduced by the evolutionary algorithm. COEGAN-BN has better training stability, leading to better performance concerning the FID score. On the other hand, DCGAN-BN shows unstable results, evidencing problems such as mode collapse in some executions.

We statistically test the significance of the results to support our analysis. Hence, the non-parametric test Mann-Whitney U ($\alpha = 0.05$) with Holm-Bonferroni correction was used to compare relevant pairs of solutions evaluated in our experiments. We found a statistical difference between COEGAN+BN and DCGAN+BN ($p = 0.04$) with a minor effect size. Besides, the improvement of COEGAN+BN over COEGAN-BN is statistically significant ($p < 0.0001$). The performance improvement of COEGAN-BN over DCGAN-BN is also statistically significant ($p < 0.0001$), evidencing that our proposal achieves better training stability in difficult conditions.

Figure 55 contains samples created by the generator after training COEGAN+BN on the Fashion-MNIST dataset. We can see that COEGAN successfully captured the distribution of Fashion-MNIST, displaying a variety of samples with corresponding quality.

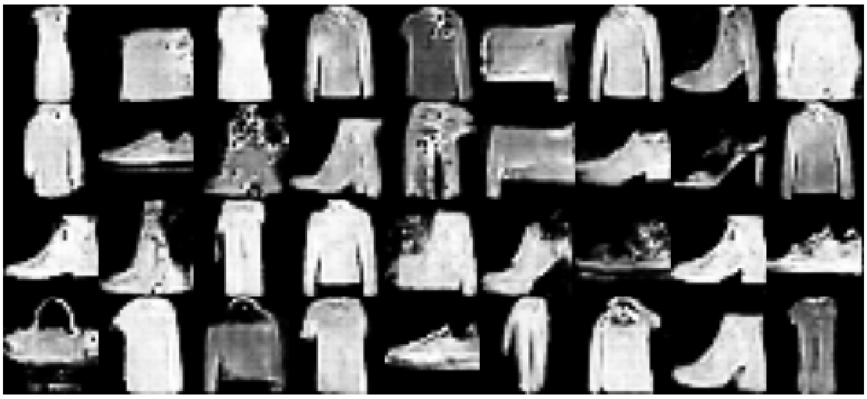


Figure 55: Samples created by COEGAN+BN after training on Fashion-MNIST.

5.1.3 *Discussion*

In this section, we investigated the inclusion of alternative loss functions into COEGAN by proposing a new mutation operator that switches the loss function of generators and discriminators. We also compare our model to the commonly used approaches to overcome stability issues when training GANs, such as alternative loss functions and new architectural mechanisms.

Experiments were conducted to compare COEGAN with and without mutation of loss functions. Results show that COEGAN does not benefit from the mixing of different loss functions into the evolutionary process when using the FID score and the original loss functions for discriminators to provide selective pressure. However, the use of other variations of GANs in a different setup, such as using other fitness functions, can be used in COEGAN to explore their improved capabilities to produce better generative models.

We also studied the behavior of COEGAN in the presence of other commonly used techniques for training GANs. Thus, COEGAN is compared with a DCGAN-based approach with and without batch normalization enabled in the architecture. We statistically showed that COEGAN is more stable than DCGAN when batch normalization is not enabled. This result indicates that the proposed evolutionary method achieved enough stability to train GANs, avoiding common issues even in hard scenarios.

The strategy to explore another variation of GANs is revisited in Section 5.2. Furthermore, other mechanisms were evaluated to assess the contributions to the design of a new variation of our evolutionary model for GANs.

5.2 COEGAN-V2

In this section we detail COEGAN-v2 (Costa et al., 2021b), which is an extension of the original model, incorporating recently proposed advances for GANs in the Evolutionary Algorithm (EA). Specifically, we improve the original COEGAN model (described in Chapter 3) by adding the possibility of having spectral normalization in layers and also incorporate a new kind of upsampling layer for generators (Brock, Donahue, and Simonyan, 2019; Miyato et al., 2018). Additionally, we propose the use of RaSGAN (Jolicoeur-Martineau, 2019) to guide the evolution of GANs in our model for generators and discriminators, using the loss functions not only for the GAN training but also as fitness functions. In this way, we avoid the costly computation of the FID score for every individual at each generation, making use of the same losses obtained during the GAN training process.

Experiments were conducted to further investigate the contributions of COEGAN-v2 to the stability of GAN training. Thus, we compare the results achieved by COEGAN-v2, the original COEGAN proposal, and non-evolutionary GAN with different stabilization mechanisms using the datasets Fashion-MNIST (Xiao, Rasul, and Vollgraf, 2017) and CelebA (Liu et al., 2015).

5.2.1 *Model Extensions*

The original COEGAN relies on the Fréchet Inception Distance (FID) metric to guide the evolutionary process. The main reason behind this is concerned with the fact that the loss functions from the original GAN model do not allow for a proper evolutionary pressure when applied for generators. However, computing the FID for every individual is computational expensive. To reduce this effort and keep a stable fitness function, we propose the use of the loss functions of RaSGAN to drive the evolution of individuals in both populations of generators and discriminators. Experiments with RaSGAN showed that training is more stable and produced samples with better quality when compared to other solutions, such as the original GAN (Jolicoeur-Martineau, 2019). Therefore, we include these losses in the training step for each pair of generators and discriminators in COEGAN-v2. The loss functions of RaSGAN are defined in Section 2.1.4.2 by Eq. (15) for discriminators and Eq. (16) for generators.

With this extension, the algorithm no longer uses the FID score to guide the evolution. The FID score is used only as the metric to report the quality of the best individuals through generations and to allow the comparison with other methods from the literature. It is also important to note that the FID score uses an external neural network (Inception Net) that leads to a high time-consuming task for the evaluation of individuals in the EA. Since COEGAN-v2 is guided by the loss functions of RaSGAN, it is substantially faster than the original COEGAN. The resulting fitness is directly calculated by the average of the losses obtained for each training pair of generators and discriminators, eliminating the need for extra steps to compute the fitness of individuals.

In this proposal, we also change the genome to introduce random normalization for layers in generators and discriminators. Therefore, layers can be created with none, batch, or spectral normalization. Spectral normalization is a kind of normalization used in the discriminator of SN-GAN. However, different from SN-GAN, we also include the possibility of using spectral normalization in generators to let the selective pressure identify the usefulness of this mechanism in these individuals.

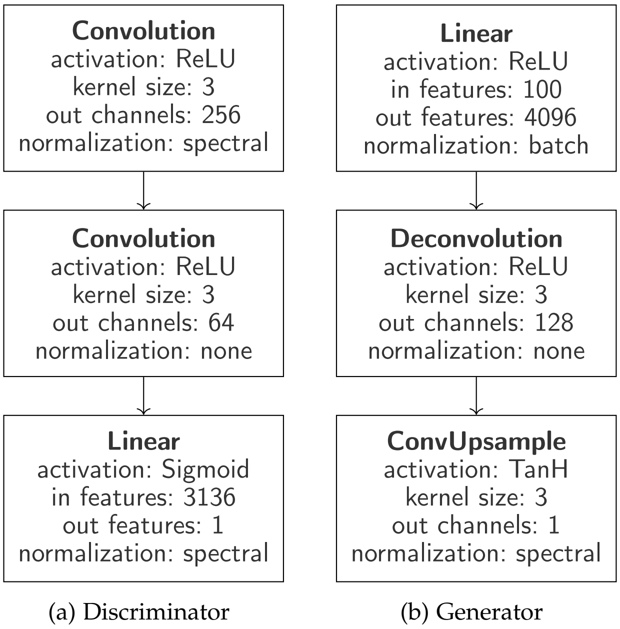
In the original COEGAN proposal, we use transpose convolution (also called deconvolution) in generators to produce the resulting samples. We incorporate in COEGAN-v2 the option to use convolution combined with upsampling (nearest neighbor). Therefore, new layers for generators are chosen between these two options: transpose convolution and convolution with upsampling, respectively represented as Deconvolution and ConvUpsample in the genome.

Figure 56 shows examples of the genotype of a discriminator and a generator for COEGAN-v2. We represent in each gene the attributes that can be randomly initialized by the EA, such as the normalization strategy, number of features, and number of output channels. To reduce the computational requirements, we limit the activation function to ReLU in all layers except the output layers of generators and discriminators. For the discriminator (Figure 56(a)), the genotype is composed of three layers. The first layer is a convolution layer with spectral normalization chosen as the normalization strategy. The second layer is also a convolution layer, but without normalization. The last gene represents a linear layer that outputs the probability of the input sample for being real (i.e., the probability that the sample belongs to the input dataset and was not produced by a generator). For the generator (Figure 56(b)), the first gene is a linear layer that receives the latent distribution and has batch normalization. The second gene is a Deconvolution layer that applies transpose convolution to upscale the image, having no normalization enabled in this case. The last gene is a ConvUpsample layer, which uses nearest-neighbor upsampling to upscale the image to produce the final outcome. In this case, this final layer uses spectral normalization.

5.2.2 *Experiments*

We design a set of experiments to assess the performance of our model regarding the quality and stability of the GAN training. We compare COEGAN-v2 with the original COEGAN and also with a non-evolutionary model based on DCGAN. We use the FID score to evaluate the quality of the results produced by generators at each generation of the training process.

For COEGAN, we evaluate the original approach and also a variation called COEGAN+loss, which uses the loss function of the original GAN model (Eq. (1) and Eq. (3)) as fitness. For the DCGAN approach, we use two variations to ensure a fair



(a) Discriminator (b) Generator

Figure 56: Example of genotypes of a discriminator and a generator. The discriminator contains two convolution layers and one linear layer. The generator has one linear, one deconvolution, and one convolution with upsampling as layers. Normalization, the number of output channels, and the number of output features are randomly chosen.

comparison with COEGAN-v2. The first variation uses DCGAN with batch normalization and the original loss functions of GANs. The second variation, called DCGAN+RG+SN, uses DCGAN with spectral normalization in the discriminator, as recommended by Miyato et al. (2018), and also the loss functions of RaSGAN. Both variations have the same architecture composed of four layers.

5.2.2.1 *Experimental Setup*

Table 15 presents the settings used in the experiments of this work. These parameters were selected based on the results of our previous experiments (Chapters 3 and 4). The number of generations is set to 50. In addition, we train the best individuals discovered in the last generation for 50 more steps. To achieve equivalence in the comparison of our experiments, the non-evolutionary models based on DCGAN are trained for 100 epochs. The number of batches used in each generation takes into account the pairing mechanism to have individuals trained for the same amount of data in both the evolutionary and the non-evolutionary approaches. For the evolutionary approaches, the populations of generators and discriminators contain 5 individuals each. We found that this amount is sufficient to provide the variability required for the proposed method. The probabilities for mutations to add, remove, and change genes are 30%, 10%, and 10%, respectively. In COEGAN and COEGAN-v2, the genome starts with only one input and output layer and can grow to a maximum of 5 layers. This setup is sufficient to discover efficient solutions and is able to encode models that deviate from the guidelines of the DCGAN architecture while keeping the computational load moderate. We use two species in each population of generators and discriminators. We apply the FID score with 10000 samples of the input dataset and 10000 samples produced by each evaluated generator.

To apply the GAN training algorithm, we use a batch of 64 images and the RMSprop optimizer (Tieleman and Hinton, 2012) with 0.0002 as the learning rate. COEGAN uses a limited set of input data at each step. Therefore, we use 50 batches per training pair. Experiments were repeated for 20 independent executions to report the average values with a confidence interval of 95% and allow for a sound statistical analysis.

The experimental study is divided into two main steps. The first step compares the performance of COEGAN-v2 with the original COEGAN and non-evolutionary GANs by using the Fashion-MNIST dataset. This dataset is composed of grayscale images of fashion items with dimension 28 × 28 and is commonly used in benchmarks of computer vision tasks. Next, we compare the performance of COEGAN-v2 with non-evolutionary GANs in a more complex scenario. Thus, we use CelebA, a dataset composed of RGB images of celebrity faces, resized in our experiments to 32 × 32 for the sake of computational resources to keep the same setup of Table 15. Experiments were performed on an Intel Core i9-9900K with 3.60GHz as CPU and an NVIDIA GeForce RTX 2060 (6GB RAM) as GPU.

5.2.2.2 *Results: Fashion-MNIST Dataset*

Figure 57 displays the average FID score for the approaches evaluated in the experiments using Fashion-MNIST on training. We can see that COEGAN-v2 is able to outperform the original COEGAN proposal and also the solutions based on DCGAN.

The vertical line in Figure 57 marks the generation where the evolutionary process ends and the training only phase starts for the COEGAN based approaches. After this mark, the best generator and discriminator were selected to continue the training for 50 more epochs. We can see that the performance of COEGAN degrades after the evolutionary phase, i.e., after the selective pressure is no longer working to guide individuals towards efficient solutions. On the other hand, individuals discovered by COEGAN-v2 were able to continue the improvement in the training phase. Individuals discovered by COEGAN-v2 have into their architectures aspects that help to achieve better training stability, such as different normalization mechanisms and the loss functions of RaSGAN.

In Figure 58 and Figure 59, we display a Boxplot representing the FID score after the last generation and the best score among all generations, respectively. The results show that COEGAN-v2 achieved better outcomes concerning the FID score of generators. Table 16 presents the average FID scores in the last generation and also the

Table 15: Experimental parameters.

Evolutionary Parameters	**Value**
Number of generations	50
Number of training epochs	50
Population size (generators)	5
Population size (discriminators)	5
Add Layer rate	30%
Remove layer rate	10%
Change layer rate	10%
Output channels range	[32, 256]
Tournament k_t	2
FID samples	10000
Genome limit	5
Species	2
GAN Parameters	**Value**
Batch size	64
Batches per generation	50
Optimizer	RMSprop
Learning rate	0.0002

average best FID value found during the 100 generations for all executions in our experiments. COEGAN-v2 achieved in the last generation an average of 24.8 ± 9.3, outperforming by 22% the second-best approach, represented by DCGAN+RG+SN (31.8 ± 3.2). Furthermore, COEGAN-v2 achieved 20.3 ± 4.5 as the average best FID score among generations, obtaining 13.2 as the best FID over all executions. Experiments of Lucic et al. (2017) reported the average best FID score of 18.0 ± 1.1 for WGAN in the Fashion-MNIST dataset when conducting a large-scale hyperparameter search. On average, COEGAN-v2 is able to achieve similar performance, outperforming in some executions the WGAN results reported by Lucic et al. (2017). However, some outliers executions in COEGAN-v2 obtain high FID scores, showing that there is still room for improvement in the solution.

To support our analysis, we validate statistically the significance of our findings. We apply the Shapiro-Wilk test with significance level $\alpha = 0.05$ to check the hypothesis of COEGAN-v2 results following a normal distribution, rejecting this hypothesis ($p = 0.002$). As such, we use the non-parametric test Mann-Whitney U ($\alpha = 0.05$) with

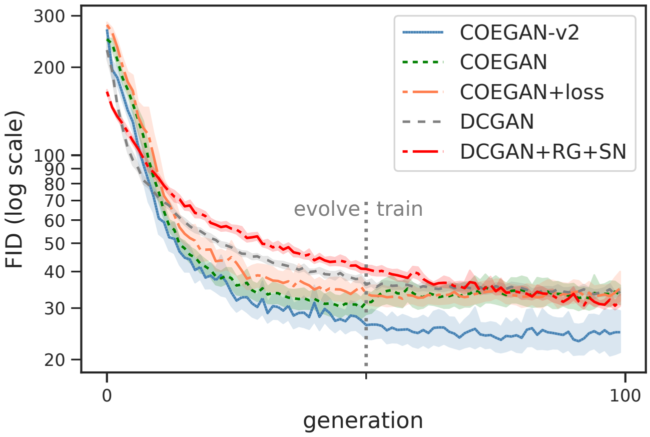


Figure 57: FID score on Fashion-MNIST dataset showing the performance of best generators computed for each independent run.

Table 16: FID score of best generators in Fashion-MNIST in the last generation and the best score among all 100 generations.

Algorithm	**Last FID**	**Best FID**
COEGAN-v2	24.8 ± 9.3	20.3 ± 4.5
COEGAN	33.3 ± 5.9	27.4 ± 5.0
COEGAN+loss	35.0 ± 10.2	28.6 ± 5.0
DCGAN	34.2 ± 2.1	30.7 ± 1.4
DCGAN+RG+SN	31.8 ± 3.2	28.8 ± 0.8

Holm-Bonferroni correction to perform a pairwise comparison. We found that the improvements of COEGAN-v2 over all other solutions are statistically significant with $p < 0.005$. These results reveal that our proposal consistently achieve better results concerning the FID score in our experimental scenario. When comparing the original COEGAN proposal and the DCGAN based models, we found no statistical significance. In this case, the benefits of the original COEGAN proposal are more relevant for harder scenarios.

Figure 60 displays a random sampling of images created by generators in one execution of COEGAN-v2. It is evidenced in these samples that our proposal was able to produce high-quality samples without getting undesired effects, such as the mode collapse problem.

In Figure 61 we present the architectures found by COEGAN-v2 for the generator and the discriminator that produced the samples from Figure 60. In this case, the maximum number of layers allowed in our setup was explored for both the discriminator

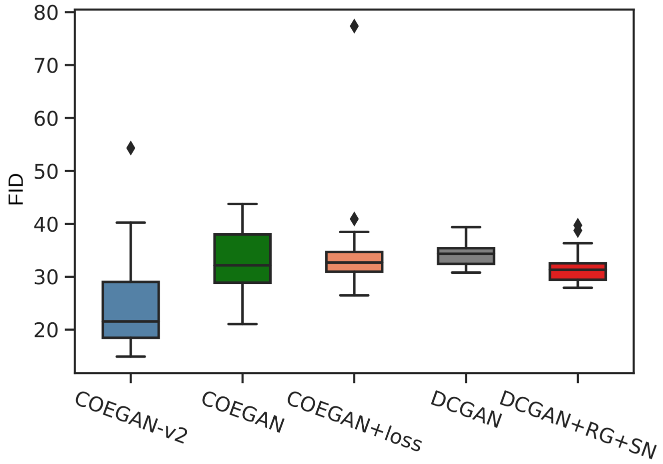


Figure 58: Boxplot of the FID score on Fashion-MNIST dataset showing the performance of best generators at the last generation computed for each independent run.

and the generator. The discriminator has three layers using spectral normalization, one with batch normalization and one without normalization. The generator has spectral normalization only in the output layer, alternating between batch normalization and no normalization in the remaining four layers. The generator also contains both Deconvolution and ConvUpsample layers, making use not only of the transpose convolution mechanism but also the nearest upsampling followed by convolution.

We also analyzed the distribution of the normalization strategies in hidden layers of generators and discriminators when applying COEGAN-v2 in the evolutionary phase (first 50 generations). In Figure 62, we can see the use of the different normalization strategies for individuals in the population of generators. The preference for a strategy is not evident in these results. Batch normalization and no normalization are slightly favorable in relation to spectral normalization. Figure 63 shows the use of the different normalization strategies for discriminators. In this case, the preference for batch normalization is clear, making spectral normalization the less used strategy in the hidden layers of discriminators. This result indicates that spectral normalization may not be paramount for COEGAN-v2.

Figure 64 shows the average number of hidden layers using either Deconvolution or ConvUpsample for individuals in the population of generators in COEGAN-v2. It is evidenced in this figure that the selective pressure produced a clear preference for ConvUpsample. This result indicates that the addition of this new possibility for generators was relevant for the improvement achieved by COEGAN-v2, showing that the gain obtained when using this type of layer is complementary to the improvement on the proposed EA for GANs.

We also compare the time spent to evaluate a single individual for COEGAN and COEGAN-v2. For this, we create individuals with the number of layers varying from

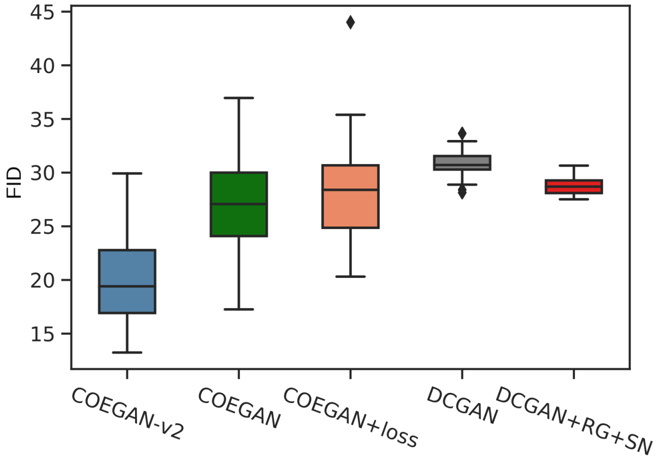


Figure 59: Boxplot of the FID score on Fashion-MNIST dataset showing the performance of best score among all generations computed for each independent run.

two to five to measure the time spent to calculate the fitness. We applied the settings described in Table 15 in this experiment. Thus, COEGAN calculates the FID score using 10000 samples and COEGAN-v2 uses 50 batches of 64 images to calculate the average losses to use as fitness. Results of 100 independent executions are displayed in Figure 65 for COEGAN and Figure 66 for COEGAN-v2. For both models, we can see that the execution time is slightly increasing within the number of layers. However, the cost for calculating the fitness in COEGAN is much higher. With five layers, COEGAN-v2 took 0.9022 ± 0.3972 seconds to execute the fitness function while COEGAN took 31.7696 ± 0.7011 seconds. Therefore, we evidence with these results that the FID score represents a significant impact on the performance of COEGAN and COEGAN-v2 overcomes this limitation by using a much faster approach.

5.2.2.3 *Results: CelebA Dataset*

To assess our contributions in a more complex scenario, we conduct experiments using the CelebA dataset (Liu et al., 2015) with the best evolutionary and non-evolutionary approaches evaluated in Section 5.2.2.2. Thus, we compare the performance of COEGAN-v2 and non-evolutionary GAN models based on DCGAN with respect to the outcome quality measured through the FID score. For our experiments, we rescale images from CelebA to 32×32 to have a similar setup to Table 15. As CelebA is a more complex dataset than Fashion-MNIST, we extend the number of training-only epochs to 100 to give enough computation time for the models. Furthermore, we learn from the architectures discovered in the experiments with Fashion-MNIST to reduce the search space by allowing only ConvUpsample layers in generators, leading to two versions of the solution: COEGAN-v2 and COEGAN-v2-up. We also applied the same

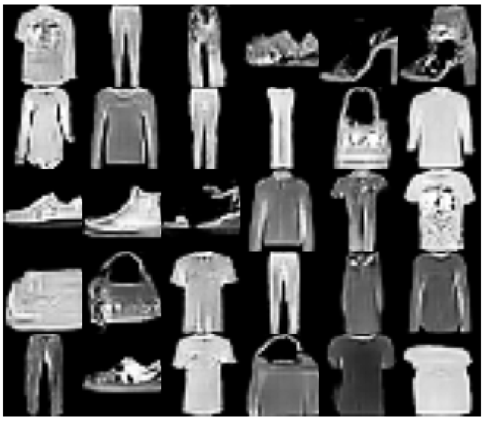


Figure 60: Samples produced by the best generator found by COEGAN-v2 when trained with Fashion-MNIST.

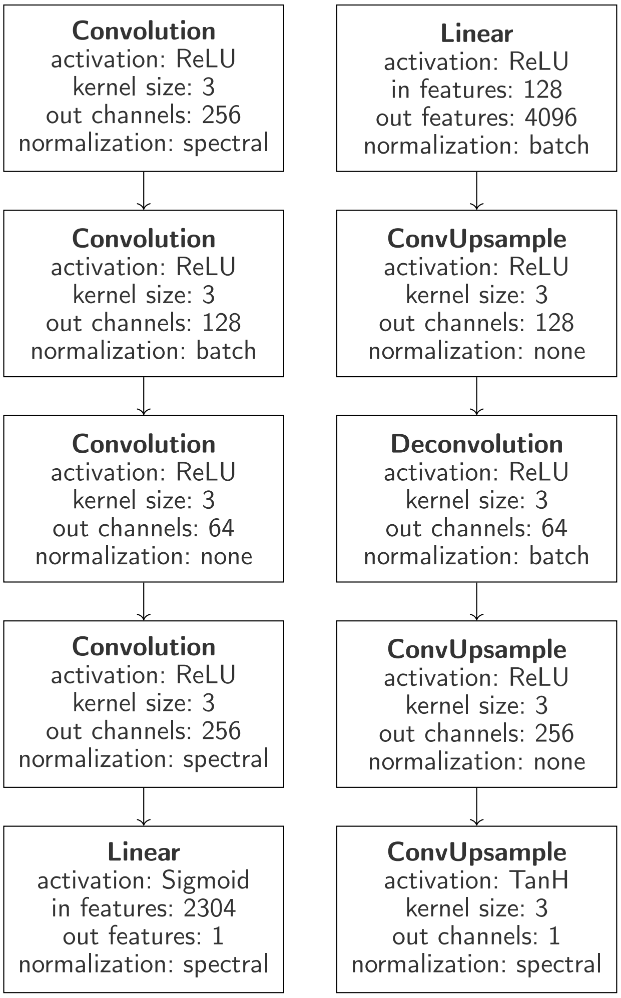
approach to produce two DCGAN models: DCGAN and DCGAN-up. DCGAN uses Deconvolution (transpose convolution) and DCGAN-up use ConvUpsample layers in generators. Besides, both DCGAN models follow the best performing model of the experiments with Fashion-MNIST (Section 5.2.2.2), using spectral normalization and the loss functions of RaSGAN.

Table 17: FID score of best generators in CelebA in the last generation and the best score among all 150 generations.

Algorithm	**Last FID**	**Best FID**
COEGAN-v2+up	11.6 ± 1.9	9.9 ± 1.4
COEGAN-v2	13.0 ± 2.8	11.4 ± 1.9
DCGAN+up	22.7 ± 1.4	20.9 ± 0.6
DCGAN	35.6 ± 3.0	33.9 ± 2.0

Figure 67 shows the average FID score for COEGAN-v2, COEGAN-v2+up, DCGAN, and DCGAN+up when training on CelebA. We can see that both variations of COEGAN-v2 are able to outperform by a large margin the non-evolutionary models based on DCGAN.

Table 17 displays the FID score in the last generation and also the best FID score among all generations. COEGAN-v2+up achieved an average FID of 11.6 ± 1.9 against 22.7 ± 1.4 for DCGAN+up (the best non-evolutionary models in our experiments), representing an improvement of 49%. COEGAN-v2 also outperforms DCGAN+up, achieving an FID 43% smaller.



(a) Discriminator

(b) Generator

Figure 61: Architectures of the best generator and the best discriminator found by COEGAN-v2 after the evolutionary process. Both the architecture of the discriminator (a) and the architecture of the generator (b) are composed of five layers, using different strategies for normalization. Furthermore, the generator contains layers with the two possibilities for upscaling designed in the model (transpose convolution and nearest upsampling followed by convolution).

We use the non-parametric test Mann-Whitney U ($\alpha = 0.05$) with Holm-Bonferroni correction to compare pairs of solutions from our results. We found that the improvement of COEGAN-v2 and COEGAN-v2+up over DCGAN models is statistically significant with $p < 0.001$. Therefore, the results evidence that COEGAN-v2 is also able to outperform a regular GAN in complex datasets such as CelebA. The smaller search space given by the experiments with COEGAN-v2+up leads to a better performance

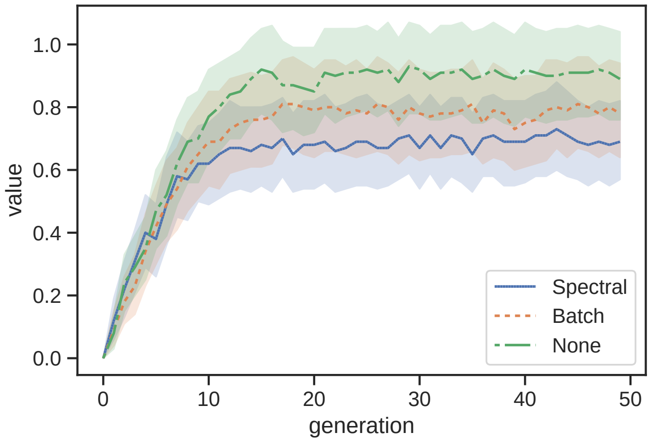


Figure 62: Average number of hidden layers using spectral, batch, and none normalization strategies for generators evolved in COEGAN-v2.

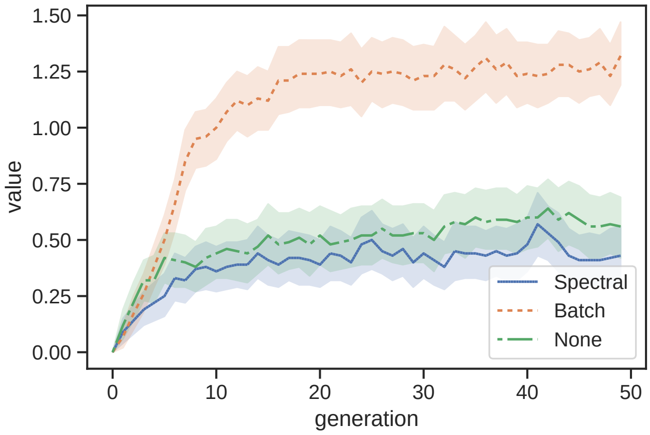


Figure 63: Average number of hidden layers using spectral, batch, and none normalization strategies for discriminators evolved in COEGAN-v2.

with respect to the FID score when compared to COEGAN-v2. However, these results are not statistically significant ($p = 0.12$).

During the evolutionary phase (first 50 generations), the variation on the FID score is evident for the COEGAN-v2 models. At this phase, the evolutionary algorithm is still searching for efficient architectures. At the train-only phase, we get a more stable behavior of the FID score, as the architectures are fixed, being trained to achieve maximum performance.

Figure 68 displays samples created by COEGAN-v2 in one execution. We can see the diversity and high quality of the produced images, following the characteristics of the CelebA dataset.

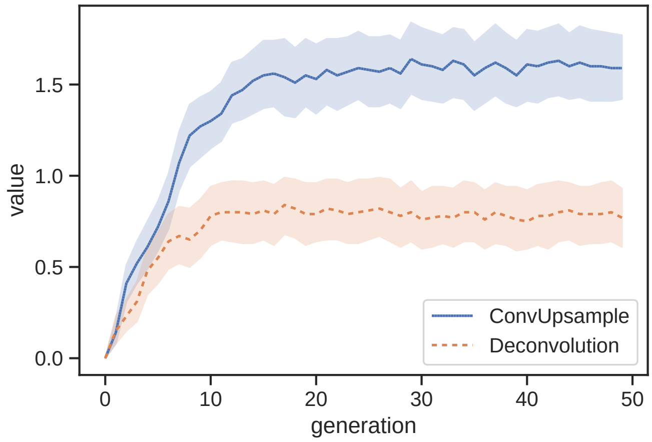


Figure 64: Average number of hidden layers of generators of types Deconvolution or ConvUpsample (Convolution with upsample) in COEGAN-v2.

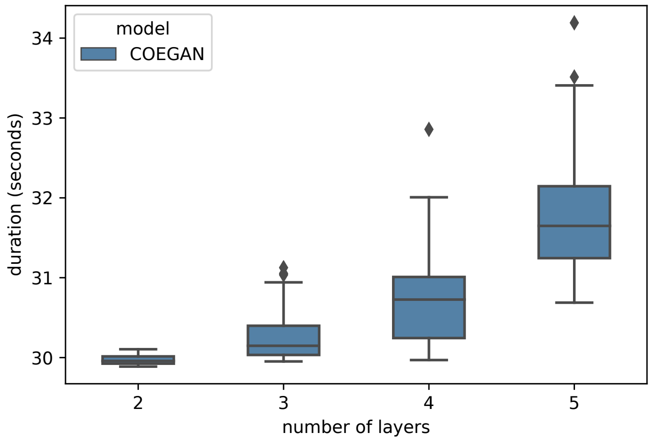


Figure 65: Boxplot of the execution time spent to calculate the fitness for individuals in COEGAN composed of two to five layers.

5.2.3 *Discussion*

In this section, we have incorporated into COEGAN new mechanisms to improve the performance and achieve better results in the training of GANs with the objective of achieving better training with respect to quality and stability. COEGAN uses competitive coevolution and neuroevolution to evolve and train GANs, using the FID score to guide the evolution of generators. However, the FID score is computationally expensive and introduces a relevant cost to performance. Furthermore, although COEGAN provides stable results in hard scenarios, the quality of the produced outcome is not as competitive as state-of-the-art results concerning GANs. To overcome this issue, we

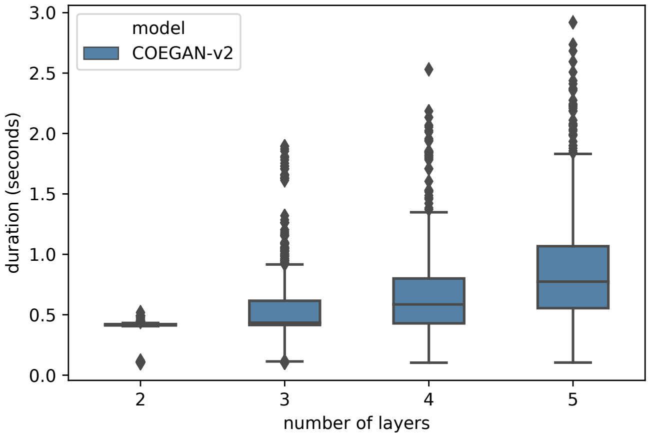


Figure 66: Boxplot of the execution time spent to calculate the fitness for individuals in COEGAN-v2 composed of two to five layers.

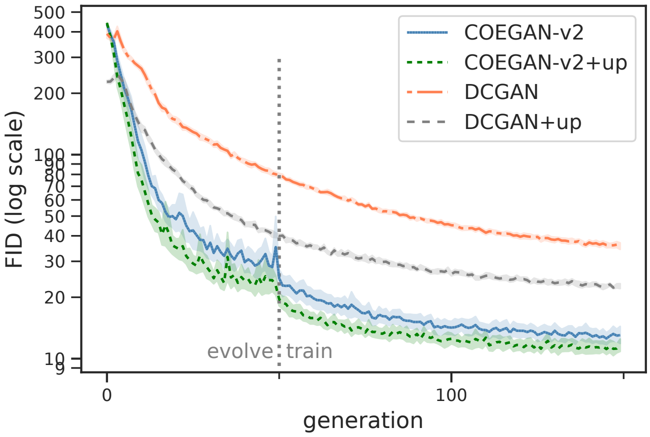


Figure 67: FID score on CelebA showing the performance of best generators computed for each independent run.

propose COEGAN-v2, an extension to COEGAN that enhances the model with the possibility of using spectral normalization, a new type of layer for the generator, and alternative loss functions not only for training but also for fitness assignment in the evolutionary process.

We compared COEGAN-v2 with the original model and also with a non-evolutionary strategy based on DCGAN. Results show that COEGAN-v2 is able to outperform the other approaches when training on the Fashion-MNIST dataset by at least 22%. Besides, we show that the improvement of COEGAN-v2 over DCGAN and the original COEGAN model is statistically significant. Experiments were also conducted using the CelebA dataset. Results revealed that COEGAN-v2 is able to consistently outper-

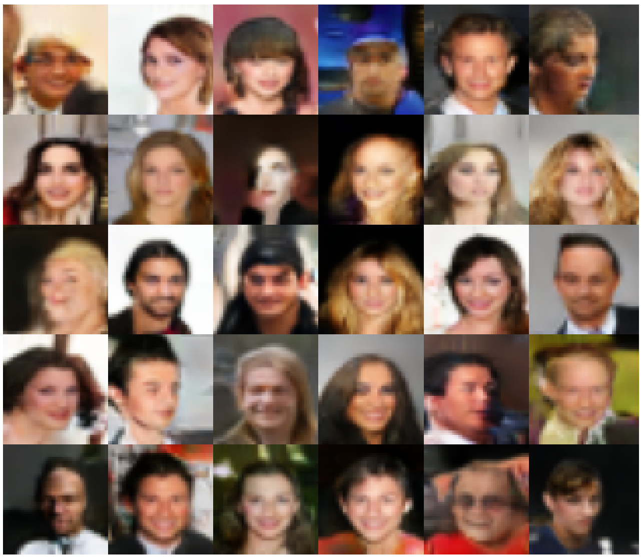


Figure 68: Samples produced by the best generator found by COEGAN-v2 when trained with CelebA.

form DCGAN, producing better results concerning the quality of generated images. We statistically showed that COEGAN-v2 achieved better results with respect to the FID score when compared to the original model and also non-evolutionary GANs. This result indicates that the proposed evolutionary method achieved enough stability to train GANs, discovering efficient models and avoiding common issues even in hard scenarios. A study on the execution cost of the evaluation phase of the EA shows that COEGAN-v2 is much faster than COEGAN.

5.3 SUMMARY

In this chapter, we extend the COEGAN considering the directions described in Section 3.4 regarding GANs. First, we propose in Section 5.1 a new mutation operator for COEGAN that switches the loss function of the individuals. Based on these results and further advances proposed for GANs, we propose COEGAN-v2 in Section 5.2.

The results show that the inclusion of a new mutation operator in COEGAN to select different loss functions does not improve the results. This provides us with strong evidence that using a coevolutionary approach to evolve GAN architectures minimizes the training problems. This is further confirmed by the study we conduct where we compare the performance of COEGAN whilst using a mechanism from the literature to stabilize the training of GANs. Therefore, we conclude that the aspects of evolutionary computation used in COEGAN are important to provide stability in training and discover efficient architectures for GANs.

Experiments with COEGAN-v2 show that it achieves better stability and outperforms the original COEGAN and non-evolutionary GANs with respect to the FID score when trained on the Fashion-MNIST dataset, achieving an average score of 24.8 ± 9.3 against 31.8 ± 3.2 for the best non-evolutionary approach (an improvement of 22%). A performance comparison between COEGAN and COEGAN-v2 evidenced that the evaluation strategy used in COEGAN-v2 is much faster than using the FID score. In the experiments with CelebA, we show that COEGAN-v2 is able to discover efficient models with at least 43% better FID when compared with the regular GAN models used in our experiments. We also show that the improvement of COEGAN-v2 is statistically significant. Besides, the results show that both COEGAN and COEGAN-v2 achieved better stability on training, leading to better results with respect to the quality of the produced images. Therefore, we conclude that the aspects of evolutionary computation used in COEGAN are important to provide stability in training and discover efficient architectures for GANs.

6 EVALUATING THE PROGRESS OF GENERATIVE MODELS THROUGH T-SNE

In Chapter 3 we proposed and detailed Coevolutionary Generative Adversarial Networks (COEGAN), which relies on neuroevolution and coevolution to build and train Generative Adversarial Networks (GAN) to overcome the stability issues. We showed through experimental analysis that the method was able to discover efficient models for GANs in different datasets. An improved version called COEGAN-v2 was proposed in Section 5.2 incorporating recent advances on GANs into the model, such as the use of loss functions from RaSGAN. Despite the results, further evidence is needed to show the progress of generators and discriminators during the evolutionary search.

In this chapter we propose a new method to evaluate the progress of GANs using the samples produced by generators, and discriminators to transform these samples into feature vectors. Then, we design an evaluation method that uses t-Distributed Stochastic Neighbour Embedding (t-SNE) (Maaten and Hinton, 2008) to visualize and quantify the performance of discriminators and generators. For this, we rely on the feature space produced by trained discriminators to analyze images produced by generators and also samples drawn from the input dataset. The t-SNE algorithm receives this feature space to distribute those images in a two-dimensional grid. A metric based on the Jaccard index in the resulting t-SNE maps is proposed to quantify the performance achieved by GAN models.

This evaluation method is used to analyze the evolution of discriminators and generators in the original COEGAN model trained on the Fashion-MNIST dataset (Xiao, Rasul, and Vollgraf, 2017). The experiments evidenced that the distribution of samples produced by our evaluation method is able to show how the evolutionary process in COEGAN progresses. The results provide additional evidence of the evolution of generators and discriminators achieved by COEGAN, depicting that it is able to overcome common limitations in the GAN training, such as the mode collapse problem.

We have also applied the evaluation method in a more complex scenario to assess our contributions. Thus, we designed experiments with COEGAN-v2 using the CelebA dataset (Liu et al., 2015). The results show that the proposed evaluation method is also suitable in this context. Besides, we provide further evidence of the evolutionary aspects achieved by COEGAN-v2.

The remainder of this chapter is organized as follows. Section 6.1 presents the method proposed in this chapter to evaluate the progress of discriminators and generators; Section 6.2 displays the experimental results using our evaluation method; finally,

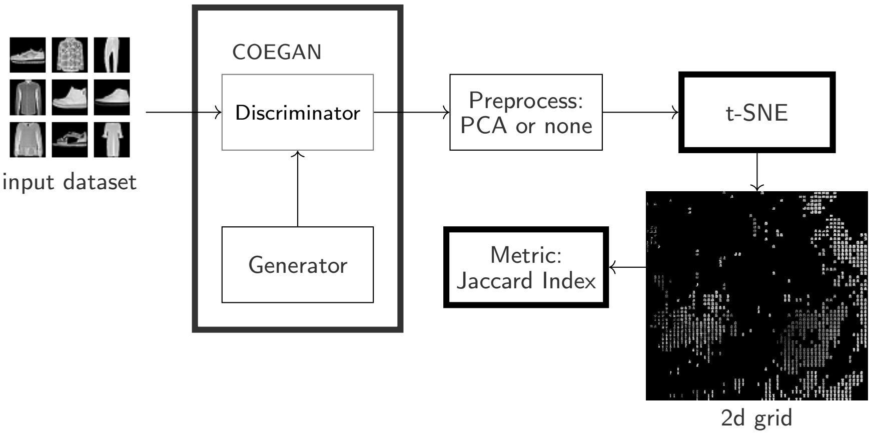


Figure 69: Overview of the evaluation method designed to analyze the progress of generators and discriminators in GANs.

Section 6.3 presents our conclusions, discussing the limitations and future work for the evaluation method.

6.1 EVALUATION METHOD

We proposed a new method that simultaneously provide visual information and metrics regarding the progress of discriminators and generators in GANs. The aim of this method is to provide further insights into how the evolution of GANs is progressing. To validate our proposal, we applied this method in COEGAN to give further evidence of its evolutionary contribution to the creation of strong generators and discriminators. Nevertheless, this method can also be applied in regular GANs.

Figure 69 presents an overview of the architecture of the evaluation method. We start by training the GAN model, in this case COEGAN, using the desired input dataset. After the training, we collect snapshots of the best discriminators and generators from different generations to analyze their performance. Instead of feeding t-SNE directly with image samples, we use discriminators to process images and extract features from them. Thus, discriminators receive samples from the input dataset and samples created by generators to construct a high-dimensional features matrix. This is achieved by using the resulting values of the last hidden layer of discriminators when processing these samples, i.e., the feature space produced by the neural network before applying the classification layer. The dimensionality of the features matrix depends on the architecture of the discriminator used in the process. To improve the performance, another dimensionality reduction technique such as Principal Components Analysis (PCA) is used to reduce the number of features. Depending on the complexity of the data, this preprocessing step can be skipped and the full features matrix can be used to feed t-SNE.

The resulting matrix contains data from the input dataset and also from all generators we want to evaluate. Thus, the resulting data of all inputs are jointly used for creating a lower-dimensional representation of the data. It is important to note that we need to provide all data at once for the t-SNE distribution, otherwise, we do not achieve correspondence between the resulting grids.

The next step transforms the output of t-SNE into a two-dimensional grid that spatially distributes the images. This grid represents a map revealing the distribution of samples according to their inner characteristics, allowing the visualization of problems such as mode collapse. This is achieved by inspecting the grid to ensure that the distribution of samples is not concentrated in a single region. We can also visually compare the distribution of samples from the input dataset with samples from the generator to assess how good the generative model is. Furthermore, by using discriminators to transform images into a feature space, we also assess their capacity to classify samples. Efficient discriminators are able to produce a feature matrix containing useful information to distribute images into the two-dimensional grid. In this way, we have an indirect measure of the performance of discriminators, making a complete assessment of both components of a GAN.

We also propose a metric to quantify the performance of the model. The same data used to produce the visualizations is used in this metric to keep the coherence with the qualitative and quantitative analysis. For this, given a map M^G of samples produced by a generator and a map M^d produced by the input dataset, we calculate the Euclidean distances $D_{(i,j)}$ between all samples in M^G and M^d:

$$\mathcal{D}_{i,j} = \|M_i^G - M_j^d\|. \tag{29}$$

Samples in M^G and M^d are not perfectly equal and distances in $\mathcal{D}_{i,j}$ are not zeroed. As such, we need to define a threshold for the similarity between samples. A global threshold τ is defined by the median of the minimum distances in $\mathcal{D}_i$ for maps M^G related to the last generation. This threshold defines the set of samples in M^G with corresponding samples in M^d as:

$$\mathcal{I}^G = \{M_i^G | \exists j, D_{i,j} < \tau\}. \tag{30}$$

The set $\mathcal{I}^G$ contains the samples in M^G that were successfully approximated by a sample in M^d, evidencing that this part of the input distribution was captured by the model. Thus, we consider this set as the intersection between these two grids and calculate the Jaccard index as:

$$J^G = \frac{|\mathcal{I}^G|}{|M^G \cup M^d|} \tag{31}$$

Using Eq. (31) we can quantify the quality of models. A high J^G indicates that the generator was able to capture the input distribution successfully. On the other hand, a perfect score in this metric indicates that the generative model is not able to produce innovative samples, i.e., it means that the model is memorizing the samples from the input dataset. It is important to note that these values can not be used to compare different executions of the evaluation method proposed in this work. The values achieved through Eq. (31) are related to a specific execution of the process. The comparison between different generative models is possible by jointly using data from them in one execution of the method.

The method described in this section can be generalized to work with other generative models that are not related to GANs. In this case, the feature matrix can be achieved by the use of an external component. For example, similarly to the strategy employed by the FID score, Inception Net (Szegedy et al., 2016) can be used to construct the feature matrix. Another option is to directly use the image samples to feed the t-SNE algorithm and extract the metrics. With this generalized version of the evaluation method, the values obtained through Eq. (31) will represent only the performance of the generative model, since no discriminative component is assessed.

6.2 EXPERIMENTS

Experiments were conducted to assess the evolution of generators and discriminators in COEGAN and COEGAN-v2 using the evaluation method proposed in this work. In concrete, we use the Fashion-MNIST dataset in the training of COEGAN to gauge the characteristics of the proposed evaluation method. To further assess how the evaluation method works in a more complex scenario, we use it in the CelebA dataset (resized to 32×32). For this, we make use of the more efficient method given by COEGAN-v2, verifying its performance in this dataset.

6.2.1 *Experimental Setup*

Table 18 describes the parameters used in our experiments. These parameters were chosen based on previous results and the characteristics of datasets used in this work (Chapter 3 and Chapter 4).

We train for 100 generations with a population of ten and five individuals for COEGAN and COEGAN-v2, respectively. The probabilities for mutations to add, remove, or change genes are 30%, 10%, and 10%, for both models. COEGAN uses the FID score as fitness (Eq. (5)) with 5000 samples in the calculation. In COEGAN, the genome was limited to four genes, representing a network of four layers in the maximum allowed setup. To tackle a more complex dataset, the genome was limited to five genes in COEGAN-v2. Only convolution layers were used as options when creating new genes by the addition mutation operator. We use three species in each population of generators and discriminators for COEGAN, reducing it to two species for COEGAN-v2 because of the smaller population used in its experiments.

Table 18: Experimental Parameters

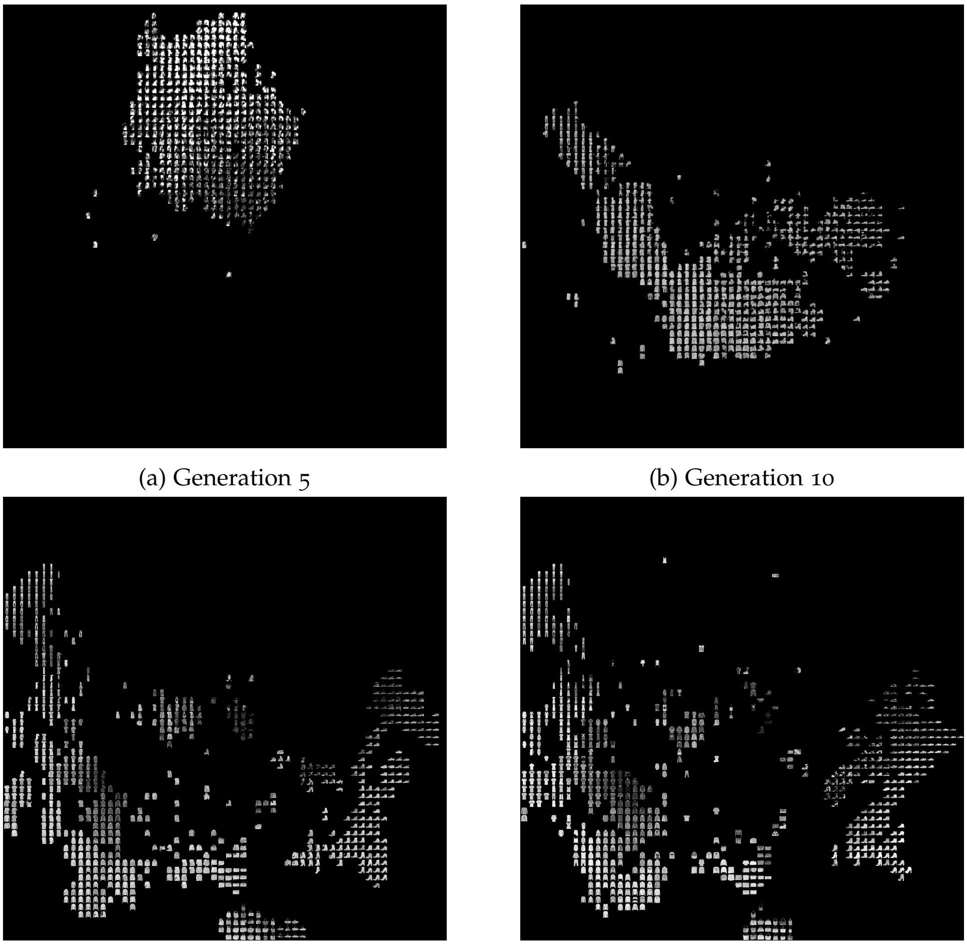
Evolutionary Parameters	**COEGAN**	**COEGAN-v2**
Number of generations	100	100
Population size (generators and discriminators)	10, 10	5, 5
Probabilities (add, remove, change)	30%, 10%, 10%	30%, 10%, 10%
Output channels range	[32, 512]	[32, 512]
Tournament k_t	2	2
FID samples	5000	not used
Genome Limit	4	5
Species	3	2
GAN Parameters	**COEGAN**	**COEGAN-v2**
Batch size	64	64
Batches per generation	10	50
Optimizer	Adam	RMSprop
Learning rate	0.003	0.0002
Evaluation Parameters	**COEGAN**	**COEGAN-v2**
PCA dimensions	50	not used
t-SNE Perplexity	30	30
t-SNE Iterations	1000	1000
Samples per model	1000	1000

In COEGAN, for each training pair of GANs, we use 10 batches of 64 images and the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.003. COEGAN-v2 uses 50 batches of 64 images and the RMSprop optimizer (Tieleman and Hinton, 2012) with a learning rate of 0.0002.

In Table 18 we also list the parameters used in the evaluation method. We rely on PCA to reduce the dimensionality of the data to 50. The original dimension of the data depends on the architecture of the discriminator. For the t-SNE algorithm, 30 and 1000 were used as perplexity and number of iterations, respectively. We use 1000 samples from each model and from the input dataset to obtain the two-dimensional map through t-SNE.

6.2.2 *Results: COEGAN on Fashion-MNIST*

This section describes the results of COEGAN trained with the Fashion-MNIST dataset.



(a) Generation 5

(b) Generation 10

(c) Generation 100

(d) Input Dataset

Figure 70: Two-dimensional grid revealing the distribution of images after applying t-SNE for generations 5 (a), 10 (b), 100 (c) of COEGAN, and for the Fashion-MNIST dataset (d).

First, we show the results of a single execution of COEGAN using the parameters defined in Table 18. Figure 70 presents the resulting map of images after the application of t-SNE with the feature map of the best discriminator at the last generation. In Figures 70(a), 70(b), and 70(c), we can see the distribution of samples created by the best generators at generation 5, 10 and 100, respectively. Figure 70(d) represents the distribution of the input dataset. Looking at the results, it is possible to see that, at generation 5, samples are concentrated in a compact region of the grid, indicating that the distribution has not been successfully captured yet. Looking at the distribution after 10 generations of the evolutionary process it is possible to see some improvements concerning the capture of the distribution of the input dataset. In the final generation, we can see that the distribution of samples resembles the input dataset, with Figures 70(c) and 70(d) presenting a similar structure regarding the two-dimensional grid. Furthermore, when comparing the distribution of samples from Figures 70(c) and 70(d) we can see that the mode collapse issue does not affect the GAN model.

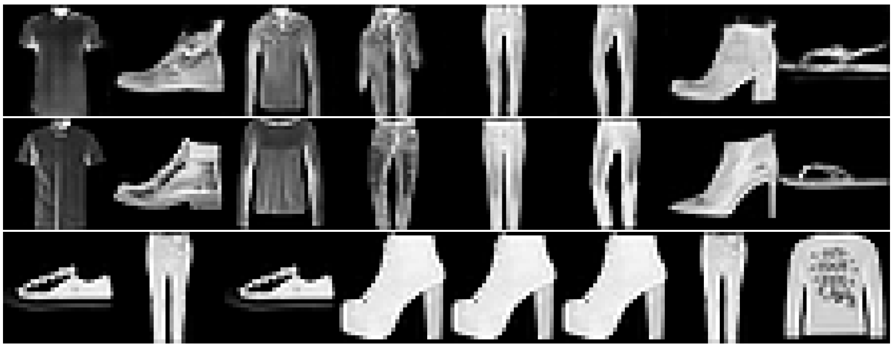


Figure 71: Comparison of samples created by the generator and samples from the input dataset using t-SNE. First row shows samples created by generators from the last generation of COEGAN. Second and third rows display the nearest and farthest samples from Fashion-MNIST using distances from the resulting t-SNE grid.

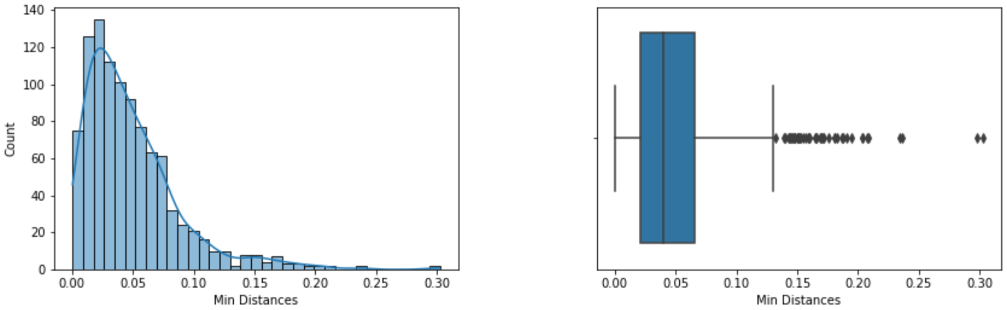
We show in Figure 71 examples of created samples and their respective nearest and farthest samples from the input dataset concerning the t-SNE map. We can see that the t-SNE map, calculated through the features trained for discriminators, is able to aggregate images based on similarity. Thus, the neighborhood of a sample in the t-SNE grid contains images with similar characteristics.

We use the outcome of t-SNE to extract some metrics to quantify and represent the observations we made by visual inspection. For this, we calculate the distances between each sample created at generations 5, 10, and 100 with the samples from the input dataset using Eq. (29).

Figure 72 shows the distribution of the minimum distances between each generated sample in the last generation and samples from the input dataset (Eq. (29)). This distribution is used to calculate the threshold for the next step. In this case, we use the value of the median (0.0394) as the threshold.

This threshold is applied to get the intersection between the map of the input dataset and maps of generations 5, 10, and 100 (Eq. (30)). We use the number of samples in this intersection to calculate the metric to quantify the progress of the model.

Figure 73 displays the Jaccard index (Eq.(31)) between created samples and the input dataset for generators and discriminators at generations 5, 10, and 100 for ten executions. The results corroborate that discriminators in all three generations were able to identify poor samples produced by generators in generation 5. This is shown



(a) Histogram of distances

(b) Boxplot of distances

Figure 72: Minimum distances between samples from Fashion-MNIST and samples created by generators at the last generation of COEGAN.

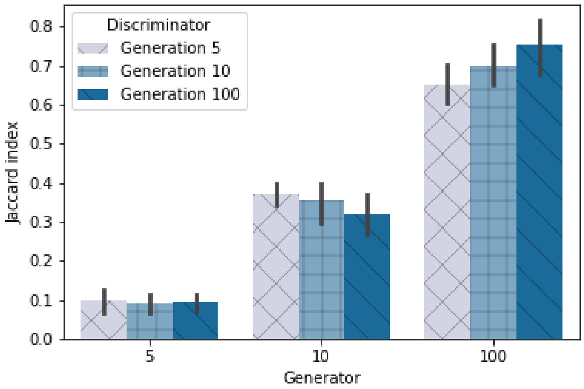
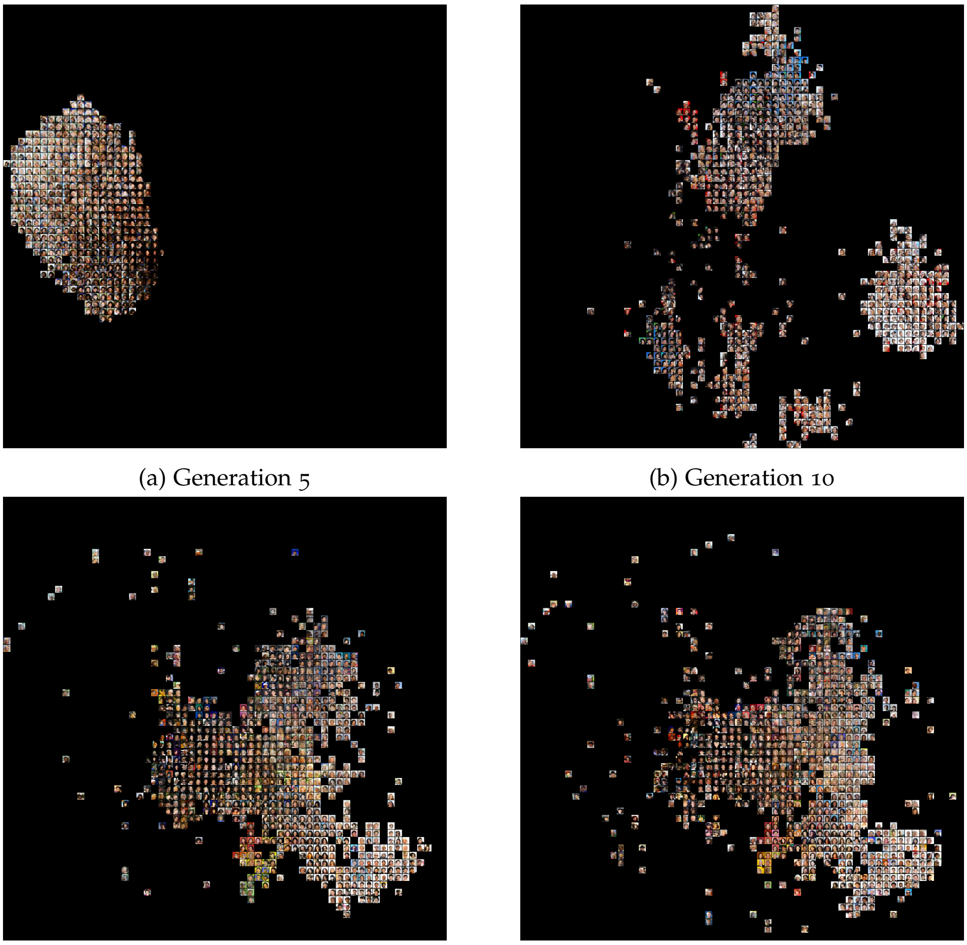


Figure 73: Average Jaccard index (10 executions) for COEGAN when comparing synthetic samples with samples drawn from Fashion-MNIST.

by the low Jaccard index, revealing that samples do not have strong similarities with the input dataset. For samples created at generation 10, we can see that the more evolved discriminators are slightly better at identifying the fake samples. Finally, all three discriminators were able to successfully distribute samples when evaluated with samples created at the final generation. Besides, more evolved discriminators are better at the distribution of samples, leading to better results concerning the proposed metric. The Jaccard index for generators in the last generation is 0.753 ± 0.112. As expected, this value is smaller than 1, revealing that generators are not memorizing the dataset, being able not only to capture the input distribution but also to produce innovative samples.

6.2.3 *Results: COEGAN-v2 on CelebA*

In this section we present the results of COEGAN-v2 trained with the CelebA dataset using our evaluation method. The CelebA dataset is more complex than Fashion-MNIST, composed of colorful faces that are not directly divided into classes.


(a) Generation 5

(b) Generation 10

(c) Generation 100

(d) Input Dataset

Figure 74: Two-dimensional grid revealing the distribution of images after applying t-SNE for generations 5 (a), 10 (b), 100 (c) of COEGAN-v2, and for the CelebA dataset (d).

Figure 74 presents the resulting map of images after the application of t-SNE with the feature map of the best discriminator at the last generation. In Figures 74(a), 74(b), and 74(c), we can see the distribution of samples created by the best generators at generation 5, 10 and 100, respectively. Figure 74(d) represents the distribution of the input dataset. In the first map (generation 5), samples are concentrated in a region of the grid, demonstrating a poor performance in capturing the characteristics of the input dataset. However, after 10 generations we can see improvements in the ability of the generator to capture the distribution. At the final generation, we can see that the distribution of samples resembles the input dataset. Therefore, Figures 74(c) and 74(d) presents similar structure regarding the two-dimensional grid. Furthermore, the wide distribution of samples in the final generation shows that the mode collapse problem is not present in this experiment.

Figure 75 presents a comparison between samples created by COEGAN-v2 and samples from the input dataset. The top row contains samples produced by the generator. The second and third rows contain the respective nearest and farthest samples from the input dataset with respect to the euclidean distance of the positions in the t-SNE maps. We can see that the nearest samples from the input dataset share common characteristics with the samples produced by generators. However, this is not as evident as in the experiments with Fashion-MNIST, since CelebA is a more complex dataset that is not directly separated into classes. Nevertheless, we can still see that t-SNE maps calculated using the features extracted by discriminators are able to aggregate similar images.

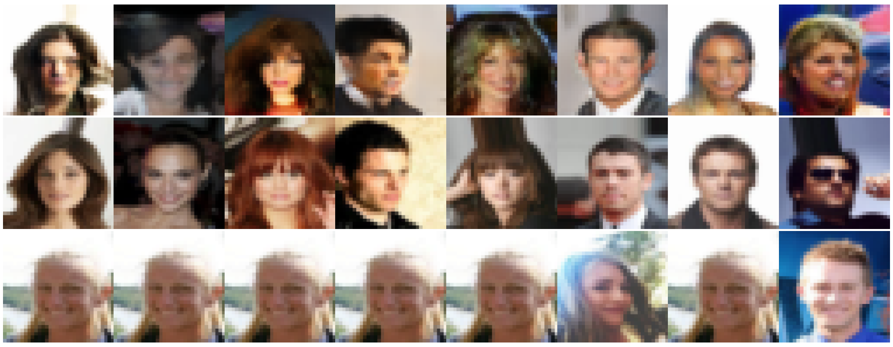
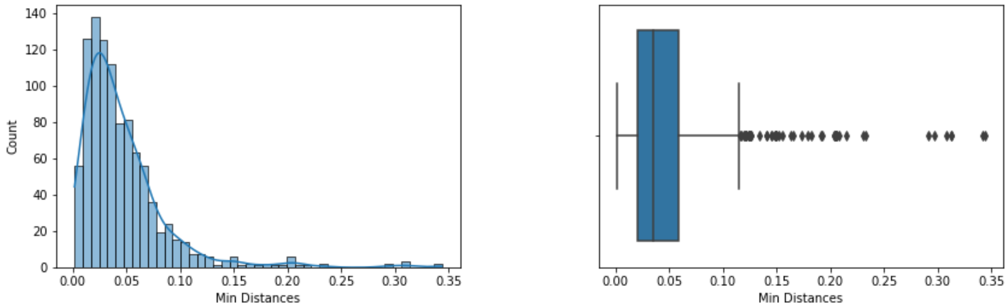


Figure 75: Comparison of samples created by the generator and samples from the input dataset using t-SNE. First row shows samples created by generators from the last generation of COEGAN-v2. Second and third rows display the nearest and farthest samples from CelebA using distances from the resulting t-SNE grid.

Figure 76 displays the distribution of the minimum distances when comparing the maps produced by generators with the map from the input dataset (Eq. (29)). The median value of this distribution is 0.0356. This value is used in Eq. (30) to determine the intersection between two maps.

Figure 77 shows the application of the Jaccard index (Eq.(31)) between samples created by COEGAN-v2 and the CelebA dataset at generations 5, 10, and 100 for ten executions. The results clearly show the progress of the GAN model through generations. First, in generation 5, the model is not capable to generate good samples. In generation 10, we can see better samples but without quality and variety. Finally, in generation 100, the distribution of samples created by COEGAN-v2 presents a similar structure with the input dataset. As with Fashion-MNIST (Section 6.2.2), the metric for



(a) Histogram of distances

(b) Boxplot of distances

Figure 76: Minimum distances between samples from the CelebA dataset and samples created by generators at the last generation of COEGAN-v2.

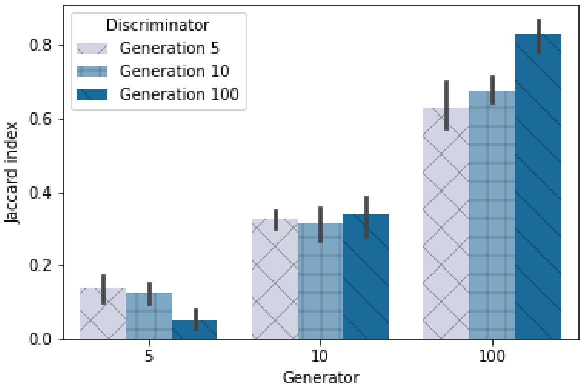


Figure 77: Average Jaccard index (10 executions) for COEGAN-v2 when comparing synthetic samples with samples drawn from CelebA.

generators in the last generation is smaller than 1 (0.8295 ± 0.0714). This is a demonstration that the discovered models are able to produce innovative samples.

6.3 SUMMARY

We design an evaluation method to visualize and measure the progress of generators and discriminators in Generative Adversarial Networks (GANs). In our proposal, t-SNE is used to distribute samples in two-dimensional grids to provide a visual inspection of the quality of discriminators and generators. Furthermore, a metric based on the Jaccard index between t-SNE maps was designed to quantitatively represent the aspects of the model, providing the relation between the visual characteristics of the grids with a concrete metric.

To validate our proposal and assess our contributions, we apply the evaluation method in the context of GANs and Evolutionary Algorithms. COEGAN combines competitive coevolution and neuroevolution on the evolution of GANs and is capable of avoiding stability issues on training, using selective pressure to guide the progress of generators and discriminators. A further improvement of this model was proposed to design COEGAN-v2. Therefore, we use COEGAN and COEGAN-v2 in our experiments to show the evolution of discriminators and generators through our evaluation method, providing further evidence of their evolutionary aspects.

Results show both by visual inspection and the proposed metric that COEGAN and COEGAN-v2 are able to gradually evolve GANs, avoiding problems such as mode collapse. We also show that the use of t-SNE proposed in this work can aggregate similar samples and provide their efficient distribution in a two-dimensional grid.

7 CONCLUSIONS AND FUTURE WORK

Generative models gained popularity in recent years mainly due to the impressive results in the image domain. In this context, Generative Adversarial Networks (GANs) represented a relevant advance in generative models. In GANs, two neural networks, a generator and a discriminator, are trained in an adversarial setup. Taking into account the modus operandi of GANs, the generative component plays a crucial role in the model. Specifically, the generator receives a vector of random numbers that is used to generate a set of synthetic samples. These samples are provided to the discriminator, along with samples from the input dataset. The goal of the discriminator is to identify which samples are real and which ones are synthetic. The training is performed by adjusting the parameters of both models, taking into account: 1) the ability of the generator to create samples that can deceive the discriminator; 2) the ability of the discriminator to distinguish the synthetic samples from the real ones. Despite the impressive results achieved so far, the training of a GAN is challenging and often requires a trial-and-error approach to obtain the desired outcome. Different techniques arise to tackle these issues, producing improved models and different approaches for training, such as alternative loss functions and architectural changes. One line of research relies on the use of Evolutionary Algorithm (EA) to train and evolve GANs, which is the focus of this work.

The research hypothesis of this thesis is that the training of GANs can be improved by using EA. Precisely, we investigate the application of neuroevolution in the context of GANs. Furthermore, we propose to take advantage of the adversarial characteristics of GANs and model the problem as a competitive coevolution environment. Our goal is to tackle the issues that affect GANs, such as the mode collapse and vanishing gradient problems. To assess our hypothesis, we had to find answers to the following research questions:

I How to define useful fitness functions for the discriminator and the generator of a GAN that induce evolutionary pressure to produce efficient models?

II Can neuroevolution be applied to GANs to create useful neural network architectures?

III Can the use of coevolution improve the training stability of GANs?

IV Does the proposed evolutionary algorithm solve common problems in GANs, such as the mode collapse and the vanishing gradient?

Before developing our proposal, we introduced the fundamental concepts required for this work. First, we describe the canonical GAN model, presenting its limitations and variations. We discuss the stability problems that affect these models, such as the vanishing gradient and the mode collapse problem, which make their training a challenging task. Several improvements over the original model are also described, detailing the mechanisms used to achieve a more robust solution. We also discuss metrics that can be used to evaluate the performance of GAN, such as the Inception and Fréchet Inception Distance (FID) scores. We also provide an introduction to EAs, focusing on neuroevolution and coevolution. EAs are computational models that simulate the evolutionary mechanism found in nature. Neuroevolution, the application of EAs in the evolution of neural networks, is described and its relevant models are presented. Coevolution is the simultaneous evolution of more than one population. Important works in these areas were described and are foundations for our work. Finally, we analyze the evolutionary aspects of GANs, presenting a survey of current works using EAs on the training of GANs. We detail in this survey the main characteristics of the evolutionary methods used to train GANs, showing different aspects of each solution regarding the evolutionary algorithm and also the GAN model.

The research conducted to answer these questions resulted in a new model called Coevolutionary Generative Adversarial Networks (COEGAN). We show that COEGAN is capable of overcoming the issues affecting GANs providing a stable training, which confirms our hypothesis that EAs can be used to improve the training process of GANs. This model is the main contribution of this thesis and, together with its variations and the experimental analysis, provided answers to the research questions enumerated above.

We start by proposing the COEGAN model, which combines neuroevolution and coevolution in the coordination of the GAN training process. We provide through COEGAN the first answers to the research questions of this thesis. Experiments were conducted to evaluate the performance of COEGAN and compare it with regular (non-evolutionary) GAN proposals. The results show that COEGAN is able to produce stable training and better results when compared to regular GANs. However, we found that the search space can be better explored by using new evolutionary mechanisms in the solution, such as Quality Diversity (QD) algorithms. Some aspects of the model could also be improved, such as the use of more efficient fitness functions. Furthermore, new variations of GANs were proposed in the literature and could be used to enrich the baseline model. Besides, we found no evidence of the vanishing gradient and the mode collapse problem in our experiments with COEGAN in the MNIST and Fashion-MNIST datasets. This result answers our question regarding the usefulness of EAs in GANs. Although this partially fulfills the research question IV, this is further enforced later in this thesis by the proposal of a new evaluation method.

Next, we revisited the original COEGAN model to provide efficient alternatives concerning the EA. Therefore, we investigate the use of alternative fitness functions to replace the ones previously proposed for COEGAN, giving an alternative answer to the research question I. The use of skill rating as fitness functions was introduced in COEGAN, replacing the GAN loss and the FID score used in the original model. Additionally, to explore the research question II, we also adapt the solution to use a QD algorithm to guide the evolution. In this case, Novelty Search with Local Competition (NSLC) was incorporated into the model to provide a better exploration of the search space.

After this, we decide to explore the inclusion of recent improvements proposed for the GAN model. Therefore, we explore the use of alternative loss functions for GANs through a new mutation operator. This operator can switch between predefined loss functions to have individuals with different configurations in the population. We also propose to incorporate relevant advances to build a new version of the model called COEGAN-v2, such as the use of spectral normalization and a new type of layer for the generator. In this model, we use RaSGAN not only as loss functions but also as fitness in the EA. Thus, we avoid using the external evaluator when calculating the FID score.

During the experiments conducted to evaluate COEGAN, we relied on the FID score to provide evidence not only for the quality of the samples but also to show that mode collapse did not affect the model. We also provide evidence of these aspects by visual inspection of the distribution of samples. However, there was no relation between the visual inspection and the metric used to assess the model. To fill this gap, we propose a new evaluation method in Chapter 6 to answer the research question IV. Experiments show that the evaluation method is useful for giving information about the performance of the generative model both by visual inspection and by a metric calculated based on the Jaccard index between distribution maps. It is also evidenced by the application of this evaluation method that COEGAN and COEGAN-v2 are able to produce efficient generative models, avoiding common issues such as the mode collapse problem.

The research and experimental analysis conducted in this work showed that COEGAN is capable of avoiding common issues that are present in the classical GAN training. Furthermore, COEGAN provides the discovery of efficient architectures through an extensible representation of neural networks in the genotype. Thus, it is possible to incorporate other mechanisms to produce even powerful generative models. This extension was applied to produce COEGAN-v2 and can be revisited to construct an updated model.

The contributions of this research provide new applications of EAs, proposing models that approximate the fields of Evolutionary Computation (EC) and Machine Learning (ML) through the use of GANs. During the course of our research, we show that EAs are in fact capable of building models that avoid the most common pitfalls of the GAN training, answering the research question III.

We summarize our answers to the research questions as follows:

I How to define useful fitness functions for the discriminator and the generator of a GAN that induce evolutionary pressure to produce efficient models?

We propose in COEGAN the use of FID and the loss function of the discriminator to be used as fitness functions (Chapter 3). Furthermore, we propose the use of different strategies for the fitness functions in Section 4.1. Our experimental evaluation evidence that the loss functions used in this work are useful to guide the evolution of GANs and produce efficient models.

II Can neuroevolution be applied to GANs to create useful neural network architectures?

The experimental evaluations of COEGAN in Section 3.2 and Section 3.3 evidence that the proposed model is capable of creating useful neural network architectures in the context of GAN. Besides, the evolutionary algorithm was assessed in Section 4.2 by the incorporation of QD to better explore the search space for the problem. We show that a better exploration of the search space through a more efficient evolutionary algorithm generates better architectures.

III Can the use of coevolution improve the training stability of GANs?

The models proposed in this work use competitive coevolution to train the discriminators and generators of GANs. Experiments with COEGAN (Sections 3.2, 3.3, 4.1, and 4.2) and COEGAN-v2 (Section 5.2) shows that the training process is more stable when compared to regular GANs.

IV Does the proposed evolutionary algorithm solve common problems in GANs, such as the mode collapse and the vanishing gradient?

Experiments with COEGAN (Sections 3.2, 3.3, 4.1, and 4.2) and COEGAN-v2 (Section 5.2) evidence through the FID score that the training process avoids the mode collapse and the vanishing gradient. Furthermore, we propose an evaluation method in Chapter 6 to provide a metric and a visual way to better evidence this result. We show through this evaluation method that our models are able to successfully capture the input distribution used in training, producing a diverse distribution of samples through the two-dimensional grid.

The work conducted in this thesis originated six international peer-reviewed conference papers (Costa et al., 2019, 2020a,c, 2021a,b; Costa, Lourenço, and Machado, 2019) and one book chapter (Costa et al., 2020b). We also make the source code of our models fully available on GitHub at:

- COEGAN: https://github.com/vfcosta/coegan
- COEGAN with QD: https://github.com/vfcosta/qd-coegan
- Evaluation Method: https://github.com/vfcosta/gen-tsne

7.1 FUTURE WORK

Research on generative models, particularly related to GANs, are improving fast and better models are being regularly proposed. The results achieved mainly in the image domain are impressive. It is possible to produce realistic samples with high resolution. Besides, advances are also being achieved in the generation of videos.

One research direction to improve the work presented in this thesis is to evaluate the incorporation of the new mechanisms proposed for GANs into COEGAN. For example, self-attention modules can be used to enrich the model. This creates the possibility to explore more complex datasets to produce strong generative models through EAs. In this case, datasets such as CIFAR-100, LSUN, and ImageNet can be used to train the models. However, one relevant challenge is the computational cost of the solution. Exploring the search space of large neural networks, such as modern solutions proposed for GANs, consumes a lot of computational power. Therefore, another research direction is to use strategies of EA to handle the computational complexity of the solution to make experiments feasible in complex models.

The evaluation method proposed in this work can be extended and assessed in other generative models. Furthermore, the method can be applied to compare different generative approaches in more complex datasets, studying the relationship between the proposed metric with other measurements commonly used to evaluate generative models, such as the FID score. The evaluation method could also be used to create common baseline for researchers to validate and compare their solutions with others. Regarding real-world applications for the evaluation method, the proposal can be extended to provide a interface for intuitive generation of samples, improving the capacity to explore the latent space of generative models.

This thesis focused on the image domain to assess our contributions. However, GANs are also used in other domains, such as text and audio. Therefore, this work can be expanded to assess the contribution in these domains. Besides, other variations of GANs such as Conditional GAN can be used as the target models of the EA.

BIBLIOGRAPHY

Al-Dujaili, Abdullah, Tom Schmiedlechner, Erik Hemberg, and Una-May O'Reilly (2018). "Towards distributed coevolutionary GANs." In: *AAAI 2018 Fall Symposium*.

Antonio, Luis Miguel and Carlos A Coello Coello (2018). "Coevolutionary Multiobjective Evolutionary Algorithms: Survey of the State-of-the-Art." In: *IEEE Transactions on Evolutionary Computation* 22.6, pp. 851–865.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou (2017). "Wasserstein generative adversarial networks." In: *International Conference on Machine Learning*, pp. 214–223.

Assunção, Filipe, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro (2017). "Automatic Generation of Neural Networks with Structured Grammatical Evolution." In: *IEEE Congress on Evolutionary Computation (CEC)*.

Assunção, Filipe, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro (2018). "Evolving the Topology of Large Scale Deep Neural Networks." In: *European Conference on Genetic Programming*. Springer, pp. 19–34.

Assunção, Filipe, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro (2019). "DENSER: deep evolutionary network structured representation." In: *Genetic Programming and Evolvable Machines* 20.1, pp. 5–35.

Barratt, Shane and Rishi Sharma (2018). "A note on the inception score." In: *arXiv preprint arXiv:1801.01973*.

Berthelot, David, Thomas Schumm, and Luke Metz (2017). "BEGAN: Boundary equilibrium generative adversarial networks." In: *arXiv preprint arXiv:1703.10717*.

Bharti, Vandana, Bhaskar Biswas, and Kaushal Kumar Shukla (2021). "EMOCGAN: a novel evolutionary multiobjective cyclic generative adversarial network and its application to unpaired image translation." In: *Neural Computing and Applications*, pp. 1–15.

Bodnar, Cristian, Ben Day, and Pietro Lió (2020). "Proximal distilled evolutionary reinforcement learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04, pp. 3283–3290.

Borji, Ali (2019). "Pros and cons of GAN evaluation measures." In: *Computer Vision and Image Understanding* 179, pp. 41–65.

Brock, Andrew, Jeff Donahue, and Karen Simonyan (2019). "Large Scale GAN Training for High Fidelity Natural Image Synthesis." In: *International Conference on Learning Representations*.

Chen, Shiming, Wenjie Wang, Beihao Xia, Xinge You, Zehong Cao, and Weiping Ding (2020). "CDE-GAN: Cooperative dual evolution based generative adversarial network." In: *arXiv preprint arXiv:2008.09388*.

Chen, Xi, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel (2016). "InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets." In: *Advances in neural information processing systems*, pp. 2172–2180.

Chong, Siang Y, Mei K Tan, and Jonathon David White (2005). "Observing the evolution of neural networks learning to play the game of Othello." In: *IEEE Trans. on Evolutionary Computation* 9.3, pp. 240–251.

Coello, Carlos A Coello and Margarita Reyes Sierra (2004). "A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm." In: *Mexican International Conference on Artificial Intelligence*. Springer, pp. 688–697.

Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2019). "COEGAN: Evaluating the coevolution effect in generative adversarial networks." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, pp. 374–382.

Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2020a). "Exploring the evolution of GANs through quality diversity." In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 297–305.

Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2020b). "Neuroevolution of Generative Adversarial Networks." In: *Deep Neural Evolution*. Springer, pp. 293–322.

Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (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.

Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2021a). "Demonstrating the Evolution of GANs Through t-SNE." In: *Applications of Evolutionary Computation*. Ed. by Pedro A. Castillo and Juan Luis Jiménez Laredo. Cham: Springer, pp. 618–633. ISBN: 978-3-030-72699-7.

Costa, Victor, Nuno Lourenço, João Correia, and Penousal Machado (2021b). "Improved Evolution of Generative Adversarial Networks." In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*.

Costa, Victor, Nuno Lourenço, and Penousal Machado (2019). "Coevolution of Generative Adversarial Networks." In: *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, pp. 473–487.

Creswell, Antonia, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath (2018). "Generative adversarial networks: An overview." In: *IEEE Signal Processing Magazine* 35.1, pp. 53–65.

Cruz, Juan-Antonio Rodríguez-de-la, Héctor-Gabriel Acosta-Mesa, and Efrén Mezura-Montes (2021). "Evolution of Generative Adversarial Networks Using PSO for Synthesis of COVID-19 Chest X-ray Images." In: *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 2226–2233.

Darwin, Charles (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favored Races in the Struggle for Life*. London: John Murray.

Deb, Kalyanmoy, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II." In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 182–197.

Donahue, Jeff, Philipp Krähenbühl, and Trevor Darrell (2017). "Adversarial Feature Learning." In: *International Conference on Learning Representations*.

Dumoulin, Vincent, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville (2017). "Adversarially Learned Inference." In: *International Conference on Learning Representations*.

Durugkar, Ishan, Ian Gemp, and Sridhar Mahadevan (2016). "Generative multi-adversarial networks." In: *arXiv preprint arXiv:1611.01673*.

Elgammal, Ahmed, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone (2017). "CAN: Creative adversarial networks, generating "art" by learning about styles and deviating from style norms." In: *arXiv preprint arXiv:1706.07068*.

Esser, Patrick, Robin Rombach, and Bjorn Ommer (2021). "Taming transformers for high-resolution image synthesis." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873–12883.

Fedus, William, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M. Dai, Shakir Mohamed, and Ian Goodfellow (2018). "Many Paths to Equilibrium: GANs Do Not Need to Decrease a Divergence At Every Step." In: *International Conference on Learning Representations*.

Ficici, Sevan G and Jordan B Pollack (2003). "A game-theoretic memory mechanism for coevolution." In: *Genetic and Evolutionary Computation Conference*. Springer, pp. 286–297.

Gao, Chen, Yunpeng Chen, Si Liu, Zhenxiong Tan, and Shuicheng Yan (2020). "AdversarialNAS: Adversarial neural architecture search for gans." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5680–5689.

García-Pedrajas, Nicolás, César Hervás-Martínez, and José Muñoz-Pérez (2003). "COVNET: a cooperative coevolutionary model for evolving artificial neural networks." In: *IEEE Transactions on Neural Networks* 14.3, pp. 575–596.

García-Pedrajas, Nicolás, César Hervás-Martínez, and Domingo Ortiz-Boyer (2005). "Cooperative coevolution of artificial neural network ensembles for pattern classification." In: *IEEE transactions on evolutionary computation* 9.3, pp. 271–302.

Garciarena, Unai, Alexander Mendiburu, and Roberto Santana (2020). "Analysis of the transferability and robustness of GANs evolved for Pareto set approximations." In: *Neural Networks* 132, pp. 281–296.

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: ACM, pp. 434–441.

Ghosh, Arnab, Viveka Kulharia, Vinay P Namboodiri, Philip HS Torr, and Puneet K Dokania (2018). "Multi-agent diverse generative adversarial networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8513–8521.

Glickman, Mark E (2013). "Example of the Glicko-2 system." In: *Boston University*, pp. 1–6. URL: http://www.glicko.net/glicko/glicko2.pdf.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Gomez, Faustino, Jürgen Schmidhuber, and Risto Miikkulainen (2008). "Accelerated neural evolution through cooperatively coevolved synapses." In: *Journal of Machine Learning Research* 9, pp. 937–965.

Gong, Xinyu, Shiyu Chang, Yifan Jiang, and Zhangyang Wang (2019). "AutoGAN: Neural architecture search for generative adversarial networks." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3224–3234.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). "Generative Adversarial Nets." In: *NIPS*. Curran Associates, Inc., pp. 2672–2680.

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville (2017). "Improved training of Wasserstein GANs." In: *Advances in Neural Information Processing Systems*, pp. 5769–5779.

Hemberg, Erik, Jamal Toutouh, Abdullah Al-Dujaili, Tom Schmiedlechner, and Una-May O'Reilly (2021). "Spatial Coevolution for Generative Adversarial Network Training." In: *ACM Transactions on Evolutionary Learning and Optimization* 1.2, pp. 1–28.

Heusel, Martin, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter (2017). "GANs trained by a two time-scale update rule converge to a local nash equilibrium." In: *Advances in Neural Information Processing Systems*, pp. 6629–6640.

Hillis, W Daniel (1990). "Co-evolving parasites improve simulated evolution as an optimization procedure." In: *Physica D: Nonlinear Phenomena* 42.1-3, pp. 228–234.

Holland, John H (1992). "Genetic algorithms." In: *Scientific american* 267.1, pp. 66–73.

Isola, Phillip, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros (2017). "Image-to-image translation with conditional adversarial networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134.

Jolicoeur-Martineau, Alexia (2019). "The relativistic discriminator: a key element missing from standard GAN." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=S1erHoR5t7.

Karras, Tero, Timo Aila, Samuli Laine, and Jaakko Lehtinen (2018). "Progressive Growing of GANs for Improved Quality, Stability, and Variation." In: *International Conference on Learning Representations*.

Karras, Tero, Samuli Laine, and Timo Aila (2018). "A style-based generator architecture for generative adversarial networks." In: *arXiv preprint arXiv:1812.04948*.

Kingma, Diederik P and Jimmy Ba (2015). "Adam: A method for stochastic optimization." In: *International Conference on Learning Representations (ICLR)*.

Kobak, Dmitry and Philipp Berens (2019). "The art of using t-SNE for single-cell transcriptomics." In: *Nature communications* 10.1, pp. 1–14.

Koza, John R (1992). *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press.

Krizhevsky, Alex and Geoffrey Hinton (2009). *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning." In: *nature* 521.7553, pp. 436–444.

LeCun, Yann, Corinna Cortes, and Christopher J.C. Burges (1998). "The MNIST database of handwritten digits." In: *http://yann.lecun.com/exdb/mnist/*.

Lehman, Joel and Kenneth O Stanley (2011). "Evolving a diversity of virtual creatures through novelty search and local competition." In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pp. 211–218.

Li, Junjie, Junwei Zhang, Xiaoyu Gong, and Shuai Lü (2021). "Evolutionary Generative Adversarial Networks with Crossover Based Knowledge Distillation." In: *arXiv preprint arXiv:2101.11186*.

Liu, Zhiyue, Jiahai Wang, and Zhiwei Liang (2020). "CatGAN: Category-aware generative adversarial networks with hierarchical evolutionary learning for category text generation." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05, pp. 8425–8432.

Liu, Ziwei, Ping Luo, Xiaogang Wang, and Xiaoou Tang (2015). "Deep learning face attributes in the wild." In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3730–3738.

Lourenço, Nuno, Francisco B Pereira, and Ernesto Costa (2015). "SGE: A structured representation for grammatical evolution." In: *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer, pp. 136–148.

Lubberts, Alex and Risto Miikkulainen (2001). "Co-evolving a go-playing neural network." In: *Proceedings of the GECCO-01 Workshop on Coevolution: Turning Adaptive Algorithms upon Themselves*, pp. 14–19.

Lucic, Mario, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet (2017). "Are GANs Created Equal? A Large-Scale Study." In: *arXiv preprint arXiv:1711.10337*.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.Nov, pp. 2579–2605.

Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow (2016). "Adversarial Autoencoders." In: *International Conference on Learning Representations*. URL: `http://arxiv.org/abs/1511.05644`.

Mao, Xudong, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley (2017). "Least squares generative adversarial networks." In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, pp. 2813–2821.

Martinez, Aritz D, Javier Del Ser, Esther Villar-Rodriguez, Eneko Osaba, Javier Poyatos, Siham Tabik, Daniel Molina, and Francisco Herrera (2021). "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.

Mescheder, Lars, Andreas Geiger, and Sebastian Nowozin (2018). "Which training methods for GANs do actually Converge?" In: *arXiv preprint arXiv:1801.04406*.

Miikkulainen, Risto, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat (2017). "Evolving Deep Neural Networks." In: *arXiv preprint arXiv:1703.00548*.

Mirza, Mehdi and Simon Osindero (2014). "Conditional generative adversarial nets." In: *arXiv preprint arXiv:1411.1784*.

Mitchell, Melanie (2006). "Coevolutionary learning with spatially distributed populations." In: *Computational Intelligence: Principles and Practice*. Piscataway, NJ: IEEE Computational Intelligence Society, pp. 137–154.

Miyato, Takeru, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida (2018). "Spectral Normalization for Generative Adversarial Networks." In: *International Conference on Learning Representations*.

Monroy, German A, Kenneth O Stanley, and Risto Miikkulainen (2006). "Coevolution of neural networks using a layered pareto archive." In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, pp. 329–336.

Moriarty, David E and Risto Miikkulainen (1997). "Forming neural networks through efficient and adaptive coevolution." In: *Evolutionary computation* 5.4, pp. 373–399.

Mouret, Jean-Baptiste and Jeff Clune (2015). "Illuminating search spaces by mapping elites." In: *arXiv preprint arXiv:1504.04909*.

Nelson, Andrew L, Edward Grant, and Thomas C Henderson (2004). "Evolution of neural controllers for competitive game playing with teams of mobile robots." In: *Robotics and Autonomous Systems* 46.3, pp. 135–150.

Netzer, Yuval, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng (2011). "Reading digits in natural images with unsupervised feature learning." In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Odena, Augustus (2016). "Semi-supervised learning with generative adversarial networks." In: *arXiv preprint arXiv:1606.01583*.

Odena, Augustus, Christopher Olah, and Jonathon Shlens (2017). "Conditional image synthesis with auxiliary classifier GANs." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 2642–2651.

Olsson, Catherine, Surya Bhupatiraju, Tom Brown, Augustus Odena, and Ian Goodfellow (2018). "Skill rating for generative models." In: *arXiv preprint arXiv:1808.04888*.

Pan, Zhaoqing, Weijie Yu, Xiaokai Yi, Asifullah Khan, Feng Yuan, and Yuhui Zheng (2019). "Recent Progress on Generative Adversarial Networks (GANs): A Survey." In: *IEEE Access* 7, pp. 36322–36333.

Potter, Mitchell A and Kenneth A De Jong (1995). "Evolving neural networks with collaborative species." In: *Summer Computer Simulation Conference*, pp. 340–345.

Pugh, Justin K, Lisa B Soros, Paul A Szerlip, and Kenneth O Stanley (2015). "Confronting the challenge of quality diversity." In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 967–974.

Radford, Alec, Luke Metz, and Soumith Chintala (2015). "Unsupervised representation learning with deep convolutional generative adversarial networks." In: *arXiv preprint arXiv:1511.06434*.

Rawal, Aditya, Padmini Rajagopalan, and Risto Miikkulainen (2010). "Constructing competitive and cooperative agent behavior using coevolution." In: *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pp. 107–114.

Reed, Scott, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee (2016). "Generative adversarial text to image synthesis." In: *arXiv preprint arXiv:1605.05396*.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. (2015). "Imagenet large scale visual recognition challenge." In: *International Journal of Computer Vision* 115.3, pp. 211–252.

Salakhutdinov, Ruslan and Geoffrey Hinton (2009). "Deep boltzmann machines." In: *Artificial Intelligence and Statistics*, pp. 448–455.

Salimans, Tim, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen (2016). "Improved techniques for training GANs." In: *Advances in Neural Information Processing Systems*, pp. 2234–2242.

Sims, Karl (1994a). "Evolving 3D morphology and behavior by competition." In: *Artificial life* 1.4, pp. 353–372.

Sims, Karl (1994b). "Evolving Virtual Creatures." In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. ACM, pp. 15–22.

Srinivas, Nidamarthi and Kalyanmoy Deb (1994). "Muiltiobjective optimization using nondominated sorting in genetic algorithms." In: *Evolutionary computation* 2.3, pp. 221–248.

Stanley, Kenneth O and Risto Miikkulainen (2002). "Evolving neural networks through augmenting topologies." In: *Evolutionary computation* 10.2, pp. 99–127.

Stanley, Kenneth O and Risto Miikkulainen (2004). "Competitive Coevolution through Evolutionary Complexification." In: *Journal of Artificial Intelligence Research* 21, pp. 63–100.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna (2016). "Rethinking the inception architecture for computer vision." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826.

Tieleman, Tijmen and Geoffrey Hinton (2012). "Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural networks for machine learning* 4.2, pp. 26–31.

Toutouh, Jamal, Erik Hemberg, and Una-May O'Reilly (2019). "Spatial Evolutionary Generative Adversarial Networks." In: *arXiv preprint arXiv:1905.12702*.

Toutouh, Jamal and Una-May O'Reilly (2021). "Signal propagation in a gradient-based and evolutionary learning system." In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 377–385.

Tu, Zhuowen (2007). "Learning generative models via discriminative approaches." In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, pp. 1–8.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need." In: *Advances in Neural Information Processing Systems* 30.

Veček, Niki, Matej Črepinšek, Marjan Mernik, and Dejan Hrnčič (2014). "A comparison between different chess rating systems for ranking evolutionary algorithms." In: *2014 Federated Conference on Computer Science and Information Systems*. IEEE, pp. 511–518.

Veček, Niki, Marjan Mernik, and Matej Črepinšek (2014). "A chess rating system for evolutionary algorithms: A new method for the comparison and ranking of evolutionary algorithms." In: *Information Sciences* 277, pp. 656–679.

Wang, Chaoyue, Chang Xu, Xin Yao, and Dacheng Tao (2018). "Evolutionary Generative Adversarial Networks." In: *arXiv preprint arXiv:1803.00657*.

Wang, Zhengwei, Qi She, and Tomas E. Ward (2019). "Generative Adversarial Networks: A Survey and Taxonomy." In: *arXiv preprint arXiv:1906.01529*.

Williams, Nathan and Melanie Mitchell (2005). "Investigating the success of spatial coevolution." In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, pp. 523–530.

Xiao, Han, Kashif Rasul, and Roland Vollgraf (2017). "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms." In: *arXiv preprint arXiv:1708.07747*.

Xu, Qiantong, Gao Huang, Yang Yuan, Chuan Guo, Yu Sun, Felix Wu, and Kilian Weinberger (2018). “An empirical study on evaluation metrics of generative adversarial networks.” In: *arXiv preprint arXiv:1806.07755*.

Yao, Xin (1999). “Evolving artificial neural networks.” In: *Proceedings of the IEEE* 87.9, pp. 1423–1447.

Zhang, Han, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena (2018a). “Self-attention generative adversarial networks.” In: *arXiv preprint arXiv:1805.08318*.

Zhang, Han, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas (2018b). “StackGAN++: Realistic image synthesis with stacked generative adversarial networks.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.8, pp. 1947–1962.

Zhou, Aimin, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang (2011). “Multiobjective evolutionary algorithms: A survey of the state of the art.” In: *Swarm and Evolutionary Computation* 1.1, pp. 32–49.

Zhu, Jun-Yan, Taesung Park, Phillip Isola, and Alexei A Efros (2017a). “Unpaired image-to-image translation using cycle-consistent adversarial networks.” In: *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232.

Zhu, Jun-Yan, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman (2017b). “Multimodal Image-to-Image Translation by Enforcing Bi-Cycle Consistency.” In: *Advances in neural information processing systems*, pp. 465–476.