Exploring Evolutionary Generators within Generative Adversarial Networks

Francisco Baeta
University of Coimbra, CISUC, DEI
Coimbra, Portugal
fjrbaeta@student.dei.uc.pt

João Correia
University of Coimbra, CISUC, DEI
Coimbra, Portugal
jncor@dei.uc.pt

Tiago Martins
University of Coimbra, CISUC, DEI
Coimbra, Portugal
tiagofm@dei.uc.pt

Penousal Machado
University of Coimbra, CISUC, DEI
Coimbra, Portugal
machado@dei.uc.pt

ABSTRACT

Since their introduction, Generative Adversarial Networks (GANs) have represented the bulk of approaches used in image generation. Before GANs, such approaches used Machine Learning (ML) exclusively to tackle the training problems inherent to GANs. However, in recent years, evolutionary approaches have been making a comeback, not only across the field of ML but in generative modelling specifically. Successes in GPU-accelerated Genetic Programming (GP) led to the introduction of the TGPGAN framework, which used GP as a replacement for the deep convolutional network conventionally used as a GAN generator. In this paper, we delve further into the generative capabilities of evolutionary computation within adversarial models and extend the study performed in TGPGAN to analyse other evolutionary approaches. Similarly to TGPGAN, the presented approaches replace the generator component of a Deep Convolutional GAN (DCGAN): one with a line-drawing Genetic Algorithm (GA) and another with a Compositional Pattern Producing Network (CPPN). Our comparison of generative performance shows that the GA used manages to perform competitively with the original framework. More importantly, this work showcases the viability of other evolutionary approaches other than GP for the purpose of image generation.

CCS CONCEPTS

• **Computing methodologies** → **Generative and developmental approaches**; **Genetic programming**.

KEYWORDS

Evolutionary Computation, Genetic Programming, Generative Adversarial Networks, TGPGAN

ACM Reference Format:
Francisco Baeta, João Correia, Tiago Martins, and Penousal Machado. 2024. Exploring Evolutionary Generators within Generative Adversarial Networks. In *Genetic and Evolutionary Computation Conference (GECCO '24 Companion), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3638530.3654348

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
GECCO '24 Companion, July 14–18, 2024, Melbourne, VIC, Australia
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0495-6/24/07.
https://doi.org/10.1145/3638530.3654348

1 INTRODUCTION

In recent years, with the ever-increasing interest in Artificial Intelligence (AI), the field of Machine Learning (ML) has been the subject of more research. Specifically, with the current widespread popularity of generative AI systems such as ChatGPT [1], generative modelling has been in the limelight, not just of the scientific community, but of the general public.The allure of Generative Models (GMs) stems from their capability to generate novel data samples that follow the probabilistic distribution of pre-existing instances.

The origin of this new-found fame can be traced to the introduction of Generative Adversarial Networks (GANs) by Goodfellow et al. in 2014 [7]. Essentially, a GAN consists of two modules, a generator and a discriminator, that are trained mutually by optimizing opposing metrics, allowing for both components to learn from each other. Despite recent trends, the history of generative modelling spans far back to a time before GANs where evolutionary approaches, mainly Genetic Algorithms (GAs) and Genetic Programming (GP), were the main force driving behind artefact generation. [11]. Over time, the hardware industry specialized in parallelization tasks, which benefitted the evaluation of large neural networks instead of evolutionary representations that needed to be sequentially evaluated. Consequently, most of the early EC approaches were dropped in favour of hardware-friendly models such as DCGANs that took advantage of GPU vectorization technologies.

Nevertheless, GANs remain plagued by training issues such as instability, mode collapse, amongst other problems [7]. Attempts to increase the robustness of the learning process saw the proposal of variants based on the vanilla GAN model. To address this matter, a new branch of research has been focused on enhancing generative models using Evolutionary Machine Learning (EML), a recent field aimed at combining the discriminative power of ML with the flexibility of representation provided by EC. Our previous work saw the proposal of TGPGAN, a framework that modifies a conventional DCGAN with a convolutional discriminator to use an evolutionary GP run serving as the generator module. Motivated by the successes in generative performance achieved with TGPGAN, we consider it timely to investigate the capabilities of other Evolutionary Algorithms (EAs) in modern GAN models, a topic that remains

[1] https://chat.openai.com/

largely unexplored in the literature. Hence, this paper extends the study made with TGPGAN using a GP generator to a Compositional Pattern Producing Network (CPPN) and a line-drawing GA.

The outline for this paper is as follows. Section 2 provides an overview of the existing literature concerning EML and generative models in the context of image generation. Section 3 introduces the remaining frameworks to compare as well as the associated representation for each framework. Section 4 details the experimental setup and analyses results. Finally, section 5 compiles a summary of the main conclusions taken from this study as well as some pointers for future work.

2 RELATED WORK

This section gives a brief insight into the history of EC in generative modelling and highlights the most impactful body of work coupling EML with generative models and with GANs in particular. In this paper, we adopt the term Evolutionary Generative Models (EGMs) introduced by Correia et al. [9] to identify generative models that use EC in any part of the generative process. Looking at the inner workings of EAs, the practicality of applying EC to generative models should not come as a surprise. Every EC algorithm continuously evolves a population of solutions, each of which can be thought of as a new data sample generated by the model. Therefore, every EA in its simplest form can be classified as a generative model, even when applied directly. Early EGMs were diverse, not only in the type of EA used but also in the role that EC played in the generative pipeline. Most generative models driven by EC at this time used either GP or GA [9].

The first efforts in image generation can be traced back to Karl Sims, who pioneered the use of symbolic expressions as genotypes for the artificial evolution of generative art [14]. These early approaches contrast with modern neural network-based adversarial models which are trained on small-resolution images - mostly due to computational cost limitations - and can therefore only generate images of the resolution they were trained on [1]. GP presents a crucial advantage over these approaches as its expression-based representation allows the generation of images at any resolution desired. For the task of evolving CPPNs, the NEAT framework is by far the most common approach as it enables the evolution of not only the weights but also of the topology of the network itself .Stanley was the first to merge these two concepts with the proposal of the CPPN-NEAT framework [15]. Ha followed by applying CPPN-NEAT to generative art through the latent space exploration of the network, resulting in the creation of sequences of image morphs [8] and by incorporating the CPPN model within the adversarial pipeline using a Variational Autoencoder GAN [10]. In Ha's model, however, the CPPN was still trained through back-propagation. Later, In 2018, Wang et al. proposed Evolutionary GANs (E-GANs), which evolved a population of neural network generators [16]. Although not using NEAT, Costa et al. extended the E-GAN architecture by introducing the coevolution of sub-populations of both discriminators and generators [5]. More recently, Ekern and Gambäck generated images based on several datasets by employing CPPN-NEAT and replacing the generator with a CPPN, which was then trained using neuroevolution [6]. The core difference between this approach and the one presented in this paper is that while the authors only performed fixed-topology neuroevolution, we utilize the full capabilities of NEAT and let the network topology evolve as well.

As for the use of geometric primitives as representation in EGMs, we can trace the use of line-drawings to the work of Baker and Seltzer in 1993 using evolutionary agents [4]. Nevertheless, to the best of our knowledge, the use of line-drawing GAs in adversarial models that combine EC and ML is unexplored in literature. Hence, this paper also represents a contribution in this direction.

3 GENERATOR APPROACHES FOR TGPGAN

This section starts by introducing a new definition of TGPGAN, followed by the frameworks to be used as generators. To avoid confusion and the introduction of two additional frameworks in addition to the original TGPGAN, here we shall refer to TGPGAN simply as the original framework but without the generator definition. Algorithm 1 shows the pseudocode for a training step in the newly defined TGPGAN.

Algorithm 1: Main training loop of TGPGAN (without a defined generator)

```
foreach training step do
    Generate batch of images using a given evolutionary
      method for n generations;
    Get real batch from dataset;
    Train discriminator with real batch;
    Train discriminator with generated batch;
    Calculate losses;
```

The algorithm can be explained as follows. The generator runs a given EA for n generations after which the individuals of the final population are converted to their image phenotypes. These images will constitute the fake batch that will later be fed to the discriminator. Fitness assessment is then performed as a forward pass on the discriminator network followed by a sigmoid normalization. This ensures that whatever solutions the generator comes up with, generated solutions will be optimized to fool the network as we are maximizing over the direct discriminator's evaluation. While individuals in the first training step are randomly initialized by the EA, in subsequent training steps a given portion of the best-fitted individuals are taken from the last population of the previous training step, simulating elitism across training steps. After retrieving and converting the fake batch, another batch of images is fetched from the original dataset and both batches are passed to the discriminator for weights updating using standard backpropagation and the Adam Optimizer.

TGPGAN-GP is the first approach studied for the generator module of TGPGAN. This is achieved using TensorGP, an engine developed to accelerate the GP evaluation phase by parallelizing operations with tensor arithmetic and to save evaluation results corresponding to sub-trees that appear more often within the population [2]. TGPGAN-Pylinhas resorts to a GA to serve as the generator module of our base model. To this end, we make use of Pylinhas[1], a renderer specifically developed for the evolution of line drawings and which is part of a larger Covariance Matrix Adaptation Evolution Strategy (CMA-ES) system. Pylinhas genome

representation uses, each encoded as six values corresponding to the two pairs of (x, y) coordinates that define the end and start of a line, grayscale colour and the stroke.

As mentioned in Section 2, the question of whether evolving the topology can further enhance the generative capabilities of the model remains. Given the already-known success of GP in serving as a GAN generator and the similarities between this approach and CPPNs, it is reasonable to assume that enabling the topological exploration of CPPNs will lead to an evolutionary advantage. With this motivation, in TGPGAN-CPPN we also compare the evolution of CPPNs using NEAT, more specifically using the neat-python framework publicly available on GitHub as a generator [13].

4 EXPERIMENTATION

In this section, we start by detailing the experimental setup used along with the parameterization for the three generators analysed. We then provide the results from our experimentation along with a comprehensive analysis. To be in conformity with the known baseline of previous experiments using TGPGAN with TensorGP, and due to the novelty of the approaches used, training was performed over the MNIST dataset for all experiments.

Concerning the base setup, all digits were run separately for each approach and for a total of 15 runs per digit. For fitness assessment, all generators assess individuals according to a forward pass on the discriminator, *i.e.* its prediction of whether the generated sample belongs to the dataset or not. The number of generations performed by the EA in each training step is given by the function $f(n) = round(n/7) + 5$, where n is the current training step. This function takes advantage of the fact that, during the first training steps, the accuracy of the discriminator is lower, leading to higher randomness in the prediction of generated samples. This technique represents an optimization over simply using a constant number of generations at each training step and was used in previous experiments carried out in the context of adversarial image generation using GP [12]. In TGPGAN-GP, at the end of a training step, an elite of individuals is selected to be fed into the initial population of the next step. In TGPGAN-Pylinhas every individual is carried over and a periodic reset to the CMA strategy is implemented to compensate. Similarly, TGPGAN-CPPN also passes the whole final population but without a restart mechanism. Framework-specific setup is shown in Table 1. For TGPGAN-GP, the rest of the GP parameterization is the same as used in the original paper [3].

Figure 2 showcases best-fitted individuals from the end of a training step in the last epoch for all three approaches. Moreover, the samples were picked within the last training epoch and were chosen to demonstrate the variety of solutions found in each approach. As depicted, while by epoch 10 both the TGPGAN-GP and TGPGAN-Pylinhas approaches manage to generate images that resemble digits of the MNIST dataset, TGPGAN-CPPN clearly fails this task by outputting seemingly random imagery. We can also verify that the images resulting from the Pylinhas generator, albeit digit-like, do not seem to mimic the handwritten MNIST style in the way that the GP generator does. Intuitively this makes sense, as GP expressions allow for the simulation of curves and other features in a way that line drawings cannot, especially when restricted to

[1]https://github.com/lmagoncalo/ES_Engine

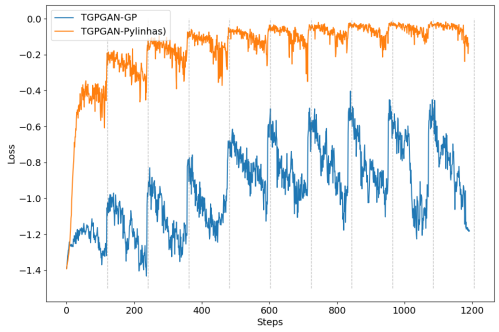

Figure 1: Average generator loss across all digits for the TGPGAN-GP and TGPGAN-Pylinhas approaches. Vertical lines represent the end of an epoch.

Table 1: Parameterization of each framework.

TGPGAN	Parameter	Value
GP	Tournament Size	2
	Mutation probability	0.3
	Crossover probability	0.8
	Initial depth	[3, 6]
	Allowed depth	[3, 14]
	Generation method	RHH (population)
Pylinhas	Initial depth (max)	6
	Genome size	13
	CMA (mu, sigma)	(25, 0.1)
	CMA restart	10
CPPN	Max Stagnation	10
	Survival Threshold	0.2
	Starting hidden neurons	10
	Starting connection	50
	Default activation	sigmoid
	Activation Set	All available

13 lines. We should note, however, that despite the comparatively lower generative performance, TGPGAN-Pylinhas runs faster for the same number of generations. In Figure 1 we provide the evolution of the generator loss across all digits for both approaches. As shown, Pylinhas tends to reach an equilibrium with the discriminator after the fifth epoch as the loss gets closer to 0, pointing to a halt in the learning process. We observe the increase in generator loss that follows a higher discriminator update frequency at the beginning of each epoch.

Nevertheless, we cannot conclude this analysis without addressing the first observation concerning the inadequacy of the CPPN generator. In hopes of better generative performance, we let the TGPGAN-CPPN evolve not for 10, but for 300 epochs and towards a simple digit, 1 in this case. Still, after being met with no noticeable improvements and after testing different parameterizations, we must acknowledge the limitations of the setup and implementation used for this approach.

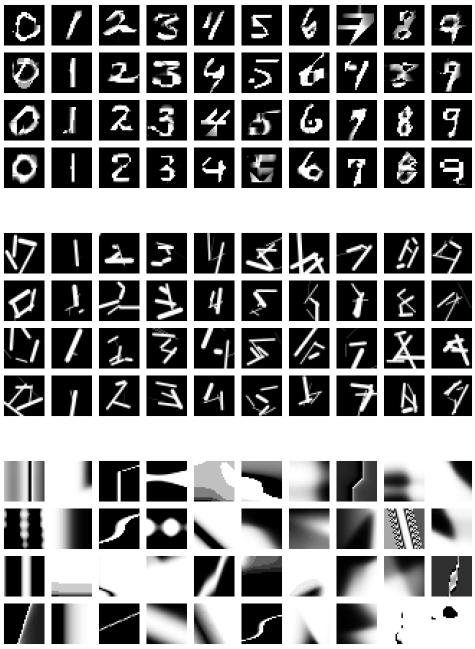

Figure 2: Samples of best-fitted individuals within the last epoch. From top to bottom: TGPGAN-GP, TGPGAN-Pylinhas, and TGPGAN-CPPN.

5 CONCLUSION

In this study, we compare three EGMs based on the DCGAN model by replacing the generator module with different EAs. This comparison is timely not only due to the previous success of the TGPGAN but also given the currently narrow body of research on hybridizing evolutionary techniques with adversarial models. The original TGPGAN framework used a GP engine to drive image generation. While we do not deny the popularity of applying GP as a GM prior to the emergence of GANs, the literature is self-evident in the use of other evolutionary approaches. With this motivation, we implement and test three generators: a GP engine, a line drawing GA named Pylinhas and a CPPN evolved by NEAT.

Results from our comparative study have demonstrated different levels of generative performance for the approaches presented after training on the MNIST dataset. In general, the TGPGAN-GP variation is shown to achieve the best performance across the board, mostly due to its ability to mimic the handwritten style of MNIST digits. Despite its limitation, the Pylinhas generator also manages to output visually appealing digits while trading some of the generative performance of TGPGAN-GP for speed of evolution and generation. Nevertheless, results for the CPPN-NEAT approach proved to be underwhelming as it failed to evolve towards imagery with any digit-like resemblance.

We should note that this paper only addresses a minimal set of the possible interplay between EC and GANs and, as such, several avenues for future work are to be considered. More importantly, the results of this work should not taken as indicative of the applicability of the respective EAs to adversarial generative models but rather of the specific techniques and setups employed. For instance, despite the substantial number of configurations tested, it is likely that the key to increasing the performance of the CPPN-NEAT generator lies in its parameterization. In addressing this issue, hyperparameter optimization might prove effective. Another aspect to improve upon has to do with the evolution of multiple classes simultaneously, which could be achieved by dividing the generating into sub-populations for each image class.

ACKNOWLEDGMENTS

This work is partially funded by the FCT - Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC RD Unit - UIDB/00326/2020 or project code UIDP/00326/2020, by Project "NEXUS Pacto de Inovação – Transição Verde e Digital para Transportes, Logística e Mobilidade". ref. No. 7113, supported by the Recovery and Resilience Plan (PRR) and by the European Funds Next Generation EU, following Notice No. 02/C05-i01/2022.PC645112083-00000059 (project 53), Component 5 - Capitalization and Business Innovation - Mobilizing Agendas for Business Innovation, by the Portuguese Recovery and Resilience Plan (PRR) through project C645008882-00000055, Center for Responsible AI, by the FCT, I.P./MCTES through national funds (PIDDAC) and by Fundação para a Ciência e Tecnologia (FCT), Portugal, under the grant SFRH/BD/08254/2021.

REFERENCES

[1] Francisco Baeta, João Correia, Tiago Martins, and Penousal Machado. 2021. Speed benchmarking of genetic programming frameworks. In *GECCO*. ACM, 768–775.
[2] Francisco Baeta, João Correia, Tiago Martins, and Penousal Machado. 2021. TensorGP - Genetic Programming Engine in TensorFlow. In *EvoApplications (Lecture Notes in Computer Science, Vol. 12694)*. Springer, 763–778.
[3] Francisco Baeta, João Correia, Tiago Martins, and Penousal Machado. 2022. TGPGAN: towards expression-based generative adversarial networks. In *GECCO Companion*. ACM, 216–219.
[4] Ellie Baker. 1993. Evolving Line Drawings. In *ICGA*. Morgan Kaufmann, 627.
[5] Victor Costa, Nuno Lourenço, and Penousal Machado. 2019. Coevolution of Generative Adversarial Networks. In *EvoApplications. (Lecture Notes in Computer Science, Vol. 11454)*. Springer, 473–487.
[6] Erlend Gjesteland Ekern and Björn Gambäck. 2021. Interactive, Efficient and Creative Image Generation Using Compositional Pattern-Producing Networks. In *EvoMUSART (Lecture Notes in Computer Science, Vol. 12693)*. Springer, 131–146.
[7] Ian J. Goodfellow. 2017. NIPS 2016 Tutorial: Generative Adversarial Networks. *CoRR* abs/1701.00160 (2017).
[8] David Ha. 2016. Generating Abstract Patterns with TensorFlow. *blog.otoro.net* (2016). https://blog.otoro.net/2016/03/25/generating-abstract-patterns-with-tensorflow/
[9] João Correia and Francisco Baeta and Tiago Martins. 2023. *Evolutionary Generative Models.* Springer Nature Singapore, Chapter 10, 283–329.
[10] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. 2016. Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning*. PMLR, 1558–1566.
[11] Penousal Machado and Amílcar Cardoso. 2002. All the Truth About NEvAr. *Appl. Intell.* 16, 2 (2002), 101–118.
[12] Machado, Penousal and Baeta, Francisco and Martins, Tiago and Correia, João. 2023. GP-Based Generative Adversarial Models. In *Genetic Programming Theory and Practice XIX*, Leonardo Trujillo, Stephan M Winkler, Sara Silva, and Wolfgang Banzhaf (Eds.). Springer Nature Singapore, Singapore, 117–140.
[13] Alan McIntyre, Matt Kallada, Cesar G. Miguel, Carolina Feher de Silva, and Marcio Lobo Netto. 2015. *neat-python.* https://github.com/CodeReclaimers/neat-python/
[14] Karl Sims. 1991. Artificial evolution for computer graphics. In *SIGGRAPH*. ACM, 319–328.
[15] Kenneth O. Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genet. Program. Evolvable Mach.* 8, 2 (2007), 131–162.
[16] Chaoyue Wang, Chang Xu, Xin Yao, and Dacheng Tao. 2018. Evolutionary Generative Adversarial Networks. *CoRR* abs/1803.00657 (2018).