TNM114 - Artificial Intelligence for Interactive Media
November 14, 2023

Comparing different GAN architectures to generate 2D character sprites

Julian Voith[1], Mario Impesi[2]

Abstract
This report covers a comparison between multiple types of Generative Adversarial Network architectures and explores different approaches to generate 2D character pixel sprites. The first approach is to generate 2D character sprites by using the image-to-image translation model U-GAT-IT, starting from real-life pictures. For the second approach, unconditional GANs like StyleGAN and Deep Convolutional GAN are used to generate 2D character sprites in four different orientations, without providing any input image. We show that all of the architectures used can produce 2D character sprites but StyleGAN is the only architecture that produces good quality 2D sprites in a predictable orientation. Lastly, we measure the quality of the results using the Fréchet inception distance and we discuss the lack of variety in the generated characters.

Source code:
`https://github.com/JulianVoith/TextureGAN-UGATIT-pytorch-`
`https://www.kaggle.com/code/julvoi/pixelcharatergen/notebook`
`https://github.com/NVlabs/stylegan3.git`

Authors
[1] *Computer Science at Linköping University, julvo429@student.liu.se*
[2] *Computer Science at Linköping University, marim694@student.liu.se*

Keywords: GAN — 2D character sprite

Contents

1. Introduction

In modern computer games, there is often a need for many different characters. A large set of different characters has a significant impact on the realism of a virtual environment. The creation of such video game characters is traditionally done by designers performing manual labor with multiple iterations which can result in a long and costly process.

Generative Adversarial Networks (GANs) have emerged as a groundbreaking technology. GANs can generate highly realistic and diverse images, making them an ideal candidate for applications in various creative domains. This project aimed to explore the capabilities of different GAN architectures to generate 2D pixel character sprites in a big volume by using different approaches and comparing the architectures based on their strengths, limitations, and the potential they could offer to game developers and designers.

The architectures that were tested are, CycleGAN (U-GAT-IT), Deep Convolutional GAN (DCGAN), and StyleGAN which is an extension to the classic GAN architecture.

2. Theory

In this project, different GAN networks were examined for their capability of creating 2D pixel character sprites. In the following sections, each architecture will be explained.

2.1 Generative Adversarial Networks

First proposed by Goodfellow et al. [2], Generative Adversarial Networks (GAN) consist of two different networks, a generator network and a discriminator network. Both networks are adversarial, which means that they work against each other. On one hand, the generator tries to produce an output that fools the discriminator. On the other hand, the discriminator tries to detect if the produced output is real or fake. This process generates two types of losses, a generator and discriminator loss.

The generator loss measures how well the discriminator is fooled by not detecting the generated fake image. It is calculated by comparing the discriminator's classification of the generated image with the label ”real”. To improve the loss, a technique called back-propagation is used. In terms of the generator, it means that the generator adjusts its parameters based on the output of the discriminator. It tries to improve itself over time.

The discriminator loss measures how well the discrimination can correctly classify the real and fake images. It is calculated by comparing the output of the discriminator for real and fake images to their respective labels. This loss and the generator's loss are optimized by back-propagating the information to the discriminator and its parameters.

Over time, the generator and discriminator will learn to generate and distinguish realistic images. Theoretically, a trained GAN should be able to generate realistic outputs, if it is given enough training time. However, GANs can suffer from an issue called ”mode collapse”. This issue causes the problem that the model produces the same output continuously independent from the given input. The CycleGAN technique can overcome this issue.

2.2 CycleGAN (U-GAT-IT)

A CycleGAN is a type of General Adversarial Network used for Image-To-Image translation [7]. Unlike traditional Image-To-Image translation models, like ”pix2pix”, it does not need paired image data to translate from domain X to domain Y [7]. The goal is to learn the mapping between two different image domains without preparing data accordingly. In a CycleGAN, there are two generator networks and two discriminator networks [7]. The generator networks are trained to translate the images from domain X to Y and Y to X. Each Discriminator is trained to classify the translations as ”fake” or ”real” in each domain [7]. In addition to the introduction of additional networks, Cycle GAN also introduces a cycle consistency loss [7]. This metric indicates as the name suggests, the consistency between the domain translations. If a translation from one domain into the other is done, the translation back to the initial domain should be similar to the initial state in the starting domain [7]. This metric mainly prevents both domains from contradicting each other [7].

In the project, a more advanced implementation of CycleGAN was used. It is called Unsupervised Generative Attentional Networks with Adaptive Layer-Instance Normalization for Image-to-Image Translation, or short U-GAT-IT [5]. It makes two main changes to the initial architecture. It adds an attention module for each generator and discriminator with different functions [5]. The module in the discriminator guides its generator to focus on regions vital for generating a realistic image [5]. The attention module in the generator, however, shifts the focus on regions that distinguish one domain from the other [5]. Apart from the attention module, U-GAT-IT also adds an adaptive layer-instance normalization (AdaLIN) function to give the model flexibility control in the amount of change in shape and texture [5].

2.3 Deep Convolutional GAN

Introduced by Radford et al. [6], Deep Convolutional GANs (DCGAN) bring the success of Convolutional Neural Networks to unsupervised learning applications. It achieves this by using convolutional and convolutional-transpose layers in the discriminator and generator, respectively.

The input to the generator is random noise, while the input to the discriminator is a 3x64x64 input image and the output is a probability that the input is from the real data distribution, instead of being originated from the generator. The discriminator and the generator are both made up of batch norm layers and LeakyReLU activations. However, the discriminator also contains strided convolution layers, whereas the generator has convolutional-transpose layers.

DCGANs can also be used to generate images with different conditions e.g. specific orientations on the generated output.

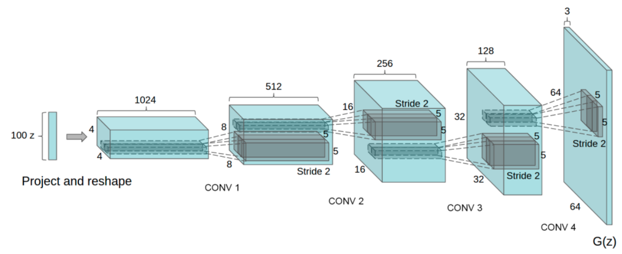


Figure 1. DCGAN generator architecture [6]

2.4 StyleGAN

StyleGAN is an extension to the traditional GAN architecture. The extension of the model is mainly aimed at the generator of the model. The architecture introduces a mapping network that maps the data into an intermediate latent space [1]. This space is used to specialize its content styles with the use of learned affine transformations [4]. It allows control of the style of the output at each point in the generator [1]. Furthermore, it introduces noise as a source of variation at each step in the generator [1]. The model also allows to set different conditions for the output. In the initial paper [4] it was used to generate high-quality photos of faces, but it can be trained on different types of styles. In our case, we generated 2D pixel character sprites with the condition on all orientations.

StyleGAN exists in three different versions. In addition, the second version (StyleGAN2) has an extension called StyleGAN2-ADA, introduced by Karras et al. [4]. This version implements Adaptive Discriminator Augmentation (ADA) to achieve better performance with limited training data. ADA is a data augmentation technique to avoid discriminator overfitting. The level of augmentation, reflected by the probability for an augmentation to occur, is related to an overfitting heuristic. The more the model is overfitting, the more the probability is increased. In this paper, StyleGAN2-ADA will be referred to as StyleGAN.

2.5 Fréchet inception distance

In order to measure the fidelity of the output relative to the original dataset, Fréchet inception distance (FID) is used. It was introduced by Heusel et al. [3], as a tool that "captures the similarity of generated images to real ones". The FID score is defined as:

$$d^2((m,C),(m_w,C_w)) = ||m-m_w||_2^2 + Tr(C+C_w-2(CC_w)^{1/2}) \quad (1)$$

where (m,C) is the mean of the Gaussian obtained from $p(.)$, the probability of generating model data, and (m_w,C_w) is the mean of the Gaussian obtained from $p_w(.)$, the probability of observing real world data. Tr is the trace of a matrix.

3. Method

Within the project, two different approaches were examined. An Image-to-Image translation from real-life images to 2D pixel character sprites with U-GAT-IT, and the unconditional generation of 2D pixel character sprites with DCGAN and StyleGAN.

3.1 Platforms

Kaggle is a platform and community for data science and machine learning. It was an invaluable tool for this project since all models presented in this paper were trained on the platform using a P100 GPU, which is available for free with a limit of 30 hours per week, per person. In addition, it contains a large repository of datasets made available to the community. The datasets used in this paper, which are described in the next paragraph, were obtained on Kaggle.

Google Colab is another platform that offers cloud computing, and it was used in this project for image pre-processing, which will be described in this chapter.

3.2 Datasets

For the project, two datasets were gathered from the platform on Kaggle. The first dataset, later referred to as dataset 1, included 1000 pictures of real-life persons with their segmentation masks. Those segmentation masks were especially important for the image-to-image translation approach. The second dataset, later referred to as dataset 2, included 3648 64x64 pixel character sprites with all possible orientations. The orientations are front, left, right, and back. Examples of both datasets can be found in figure 2.

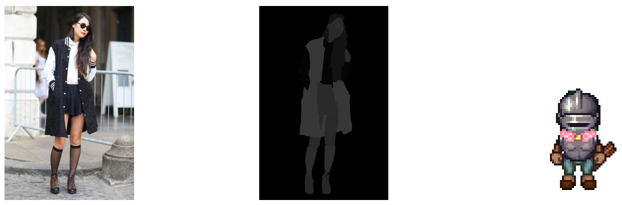

(a) Real-life picture (b) Segmentation mask (c) 64x64 pixel character

Figure 2. Examples of used datasets showing one real-life picture with its segmentation mask and a 64x64 pixel character sprite with frontal orientation.

3.3 Image-to-Image 2D character sprite generation

The first approach was to use U-GAT-IT to perform Image-to-Image transformation, from real-life pictures to 2D character sprites. The goal was to generate characters that resembled the original real-life picture, for example in the colors of clothing. The first step was to split the dataset into training and test subsets. This was only necessary for dataset 1. This paper will present multiple iterations of this model, based on the data preprocessing performed on dataset 1.

The first iteration was trained on the unaltered datasets. The generated heat map from the attention module showed that the model's focus was directed toward unrelated areas of the input image, for example, the background, as opposed to the person's body. This can be seen in Figure 3.

In order to better focus the model on the pertinent areas of the image, the background was removed using the clothing segmentation mask included in dataset 1. This results in a better heat map, as in Figure 4.

For the third iteration, the images from dataset 1 were resized so that a person's body would have the same dimensions as a character from dataset 2, and so that it would occupy the same area of the image. This model was trained for 120000 iterations.

3.4 Unconditional image generation

The second approach was to use different unconditional image generation architectures like DCGAN and StyleGAN. Those architectures do not use an image-to-image translation, hence only dataset 2 was necessary to train the models. The model for DCGAN was found on GitHub which was then transformed into a Kaggle Notebook. To use StyleGAN, the model was forked from the official implementation by NVIDIA and run on the platform Kaggle as well. The links to the corresponding repositories can be found at the beginning of the document.

The DCGAN model is capable of conditioning the output on all different orientations by default. Hence, the first approach was just to train the model with the dataset with different amounts of epochs, batch sizes, and learning rates (LR) for the generator and discriminator. Training parameters can be found in table 1.

Training parameters (DCGAN)				
Orienta. weight	Epochs	Batch size	LR Discr.	LR Gen.
0.5	2000	114	0.0004	0.0001
0.5	2000	64	0.0002	0.0002
0.5	5000	64	0.0002	0.0002
1	2000	64	0.0004	0.0001

Table 1. Used training parameters for the DCGAN model.

By only training the model, proper characters were produced which can be seen in Figure 5. Unfortunately, the consistency in regard to the orientation was not satisfactory.

In order to train a model capable of generating images in the specified orientation, the training loss calculation includes a factor based on the orientation. This loss component has an associated weight to decide how much importance it should have compared to other components. Table 1 shows that the weight was adjusted to try to improve the orientation accuracy. The results can be seen in Figure 8.

The StyleGAN model is, as well as the DCGAN model, capable of conditioning the output on all orientations of a character sprite. For this project mainly StyleGAN version 2 with adaptive discriminator augmentation has been used. There have been attempts to use StyleGAN version 3, but due to its time requirements for training, it was not feasible.

Before training the model, the dataset had to be adjusted for the model with the integrated dataset tool. StyleGAN also offers pre-trained models for face generation, but due to the different goal, none of them was useful. The network had to be trained from scratch. The different parameters used for training the model can be found in Table 2. The learning rate was not changed from the default 0.0025. In addition, when modifying training parameters, the training was resumed using the model from the previous run. The results can be seen in Figure 9.

Training parameters (StyleGAN)		
Gamma	Kimgs	Batch size
0.1024	1000	32
0.0512	2200	32
0.1024	500	64

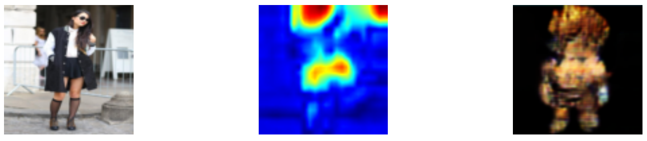
Table 2. Used training parameters for the StyleGAN model.

4. Result

This section shows the result of the tested architectures including various tested parameters.

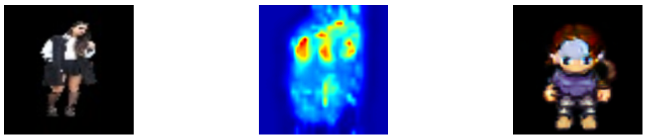
4.1 Real-life pictures to 2D character sprites

Results of the Real-life pictures to 2D character sprite translation without removing the background can be found in Figure 3 and with removed background in Figure 4.



(a) Real-life picture (b) Attention Map (c) Generated Character

Figure 3. Result of the image-to-image translation without removing the background.



(a) Real-life picture (b) Attention Map (c) Generated Character

Figure 4. Result of the image-to-image translation with removing the background.

4.2 Unconditional image generation with DCGAN

Results of the different training parameters of the unconditional image generation with DCGAN can be found in Figure 5, Figure 6, Figure 7. Furthermore, the result of the adjusted weights can be found in Figure 8.

4.3 Unconditional image generation with StyleGAN

The results of the unconditional image generation with StyleGAN can be found in Figure 9. The result of its orientational consistency can be found in Figure 10.

5. Discussion

The results from the Image-to-Image approach were not up to expectations. The two datasets are too different to make such an approach work, at least using U-GAT-IT. Dataset 1 contains high-quality images full of details, whereas dataset 2 contains very small images of characters with limited distinctive qualities. The amount of information contained in the input images cannot be successfully matched in the output images. In order to reduce this gap, the background was removed from the input images. This led to some improvement, which can be observed by examining the heat map that U-GAT-IT generates as an intermediate step in the transformation. The heat map in Figure 3, generated from an image with a background, shows that the model is assigning attention to parts of the background, and this leads to a lower-quality output image. When the background is removed, as shown in Figure 4, the

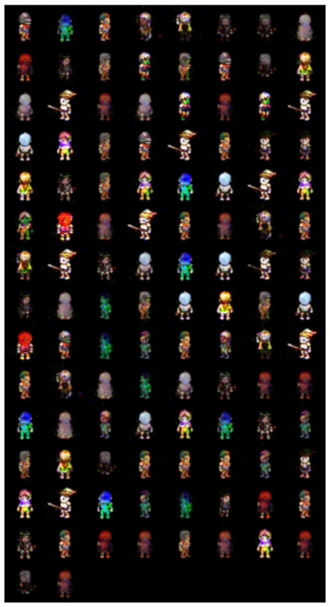


Figure 5. Results of the DCGAN with 2000 Epochs and a batch size 114.

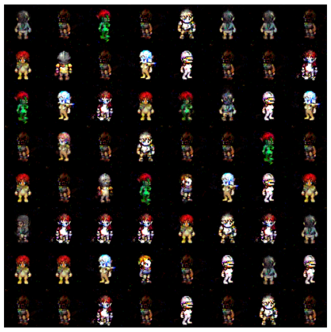


Figure 6. Results of the DCGAN with 2000 Epochs and a batch size of 64.

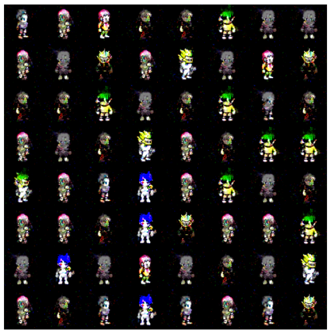


Figure 7. Results of the DCGAN with 5000 Epochs and a batch size of 64.

output image is of higher quality. However, the features of the real-life person are not well reflected in the output character.

DCGAN was able to generate characters that resembled the ones in dataset 2, achieving better results than the output

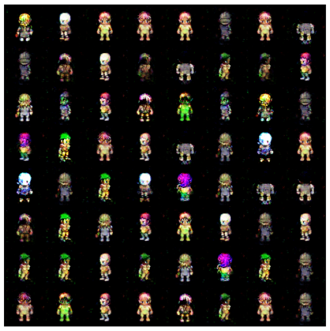


Figure 8. Results of the DCGAN with 2000 Epochs and batch size 64 with adjusted weights.

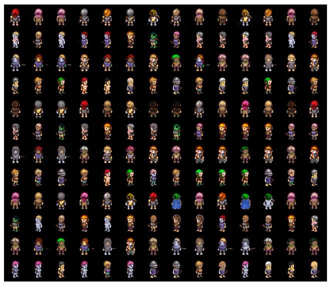


Figure 9. Results of the StyleGAN architecture.

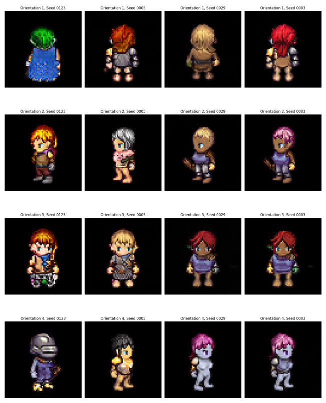


Figure 10. Orientational consistency of the generated characters with the StyleGAN architecture.

of U-GAT-IT. However, a big portion of the output characters have major visual artifacts. In addition, the output from DCGAN lacks diversity, in the sense that many characters share most of their appearance traits and some appear to be identical. The orientation selection is also lacking accuracy. More diverse data and adjustments to the trained model might improve the output.

StyleGAN gave the best results in terms of character generation and orientation accuracy. The generated character is always facing the correct direction, as shown in Figure 10, even though some of the generated characters present some visual artifacts. In terms of the diversity of the output, Style-

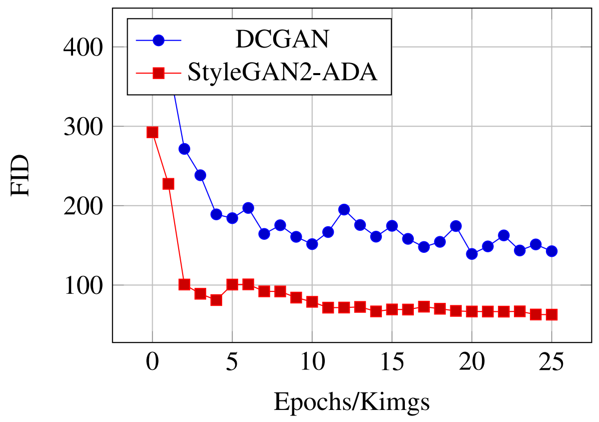


Figure 11. Comparison of DCGAN and StyleGAN2-ADA in terms of FID.

GAN achieves better results than DCGAN, although it is still possible to observe identical characters in the output. As indicated in Table 2, the gamma parameter for StyleGAN was halved for a portion of the training to increase the output diversity. Also here a bigger and more diverse dataset could improve the overall output.

The FID comparison in Figure 11 confirms that StyleGAN2-ADA achieves better performance than DCGAN. The same graph also shows that the FID converges very quickly and that the additional training after that point only contributes to minimal improvements.

6. Conclusion

In conclusion, the project showed interesting results for all of the applied GAN architectures. For the executed project and the available datasets, StyleGAN produced, with the given amount of training, the best results. DCGAN and U-GAT-IT also showed potential to solve the problem, but they require adjustment to the specific use case and/or different datasets. Further, the project also indicates the potential, generative AI can have for game studios in the overall character asset creation process of their video games. With some training and time, a variety of character assets can quickly be created to create new immersive worlds.

Acknowledgments

Special thanks go to Gabriel Eilertsen for supporting the execution of the project.

References

[1] A gentle introduction to stylegan the style generative adversarial network. URL https://machinelearningmastery.com/introduction-to-style-generative-adversarial-network-stylegan/.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[3] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf.

[4] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. *CoRR*, abs/2006.06676, 2020. URL https://arxiv.org/abs/2006.06676.

[5] Junho Kim, Minjae Kim, Hyeonwoo Kang, and Kwanghee Lee. U-gat-it: Unsupervised generative attentional networks with adaptive layer-instance normalization for image-to-image translation, 2020.

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016.

[7] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.