CS230

Generating Digital Images in the style of Fortnite skins

Dwight Moore
Department of Computer Science
Stanford University
dmbm0830@stanford.edu

Abstract

This paper spawned from an appreciation of video game art styles and the design process behind cosmetics within certain games. This project initially began on a separate game entirely, but due to a dire lack of data available for a project of this scale, the project's focus had to be redirected to Fortnite, a game with a much wider variety of cosmetics throughout its history.

1 Introduction

If you were to ask a random person on the street if they were familiar with the game Fortnite, there is a high chance they will be familiar with what you are talking about. Whether they have played the game themselves or have bought Fortnite gift cards for their younger relatives, the game has managed to thrust itself into the mainstream in recent years. However, the game doesn't necessarily thrive off of its innovative gameplay driving people to purchase the game. In fact, the game itself is entirely free to play! Instead, people are typically drawn to the game due to the wide variety of cosmetic options available for players. It is possibly the only game in the world where one can (without violating any licensing issues) have a team of Thanos, Master Chief, Ariana Grande, and a second Ariana Grande. The financial success of this game has been built off the back of the skin creators, and this project came about as a way to see just how replicable their art style is.

Initially, my project was going to be based off of the video game Destiny 2, however, the amount of artwork available in that game does not make for a good input size for a Generative Adversarial Network. I was only able to get an input size of a few hundred images, and some of those images were simply duplicates of the same items. With Fortnite, I had a wider variety of items to work with, leading to better results.

2 Related work

As this is a project based on Generative Adversarial Networks, I decided to first go through existing work within the CS230 program itself for any inspiration. I knew that I wanted to pursue a project in this field, and seeing a project from a prior student about Pokemon generation is what initially inspired me to work on video game skins[1]. Google Scholar additionally provided many articles on the subject, however not too many pertained to the exact field, video games, that I wanted. A very interesting article I read spoke about the usage of generative machine learning models to create facial textures for use in video games, which was the most applicable paper to what I desired to do[2]. Another paper discussed the usage of Deep Convolutional GANs for the creation of assets for use in video games, as to lighten the load of artists within game development companies [3]. Due to these papers being so recent, I knew that if I followed similar methodologies to these projects, I would be able to start off on a good foot.

CS230: Deep Learning, Winter 2022, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

The issue now came for discovering which particular GAN I wanted to use. Deep Convolutional GANs, or DCGANs, had been mentioned in the papers I had read up to this point, but the papers also had very specific use cases (facial texture construction). There is not currently existing work out there that is trying to utilize GANs to create video game skins and outfits, so I was having to apply a lot of laterally-related research to the question I had. I was not convinced that this would be the best application for my project. I read another paper that used StyleGAN, but this also was based on specifically facial features, when I wanted to focus on a full-body art generation [4]. Another paper on Conditional GANs had a focus on level generation, but it brought up a good use case for Conditional GANs: being able to tell the generator to create based on certain features. Due to reasons which will be explained below, the Conditional GAN became my primary method of pursuing this project.

3 Dataset and Features

Initially, this project focused on Destiny 2, so my dataset consisted of images of weapons and armor from that game. However, my dataset was never able to crack the thousand mark, as the game is relatively recent and has not been putting out much content in the recent year or two. The switch of focus to Fortnite made this problem disappear. I was able to go to an online database that displays every single item ever sold within the game and scrape the images from the site, allowing me to have an input dataset well into the thousands. [6]. However, I ran into a bit of a problem. Despite the database having around 3,000 image links for me to download, there had only been approximately 1300 skins ever released for the game. Thus, I had to manually go through and delete all images that were simply placeholders for future content, and these were dispersed throughout the dataset. In the end, I was able to cut down to approximately 1600 images (as some skins had multiple official screenshots released in-game).

Once I had these images, I had to figure out how to best make them work with my algorithm. One issue I quickly noticed with these screenshots is that the skins came in two distinct advertising styles. One was a close-up shot of the skin, while the other was distanced away to show off the entire body.

(a) Example of a Close-Up Skin Photo

(b) Example of a Full-Body Skin Photo

Since these are two very distinct styles, I imagined that the generator should be aware of which type of photo it is dealing with, hence the usage of Conditional GAN. Splitting up each photo into either a Close-Up or Full-Body skin took a while, but it made the algorithm work much better.

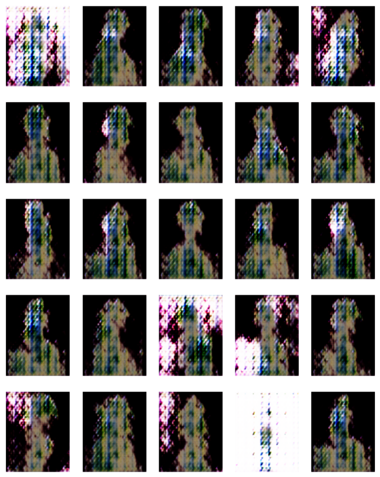
4 Methods

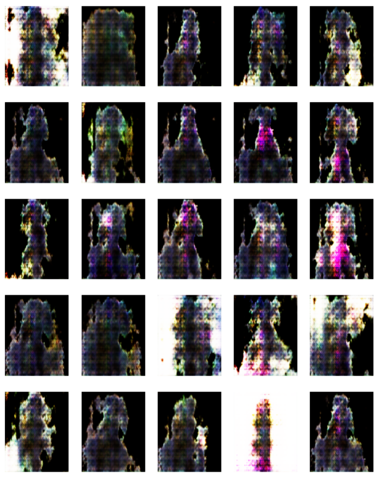
As stated prior, this project was performed using a Conditional GAN. As such, all images were downloaded along with a label indicating what kind of image it was: Close-Up or Full-Body. For this part of the project, I followed a lot of conventions typically utilized in Conditional GAN implementations [7][8][9][10]. Specifically, in the generator, I utilized Conv2DTransposes layers followed by Batch Normalization and then ReLU layers. And, in the discriminator, I did similar but replaced ReLU with LeakyReLU layers after a few failed runs with just ReLU. All of these runs were performed on my personal computer, which houses a GeForce 3070, and a computer that is back home, which houses a GeForce 2080Ti. (I could have used the cloud computing while back home, but Internet is so awful there that the input dataset would take ages to upload). For both the generator and discriminator, I utilized Categorical Cross Entropy loss, as that best suited the program I was working with.

5 Experiments/Results/Discussion

On the first few runs of this program (before I settled on the final implementaion), the learning rate was set a bit too low, at 0.0002. My first results ended up with black and white static images after 100 epochs with a batch size of 64, so I realized that I had to make the most of the relatively small dataset that I had at my disposal. In order to cover more ground on each epoch, I bumped the epoch size up to 128 and set the learning rate to 0.001. However, these results also did not converge in the way I desired, so I tuned down the learning rate to the value that it sits at now, 0.00045. I also fiddled with the beta1 values of both the generator and discriminator, and they both settled at 0.7 by the end.

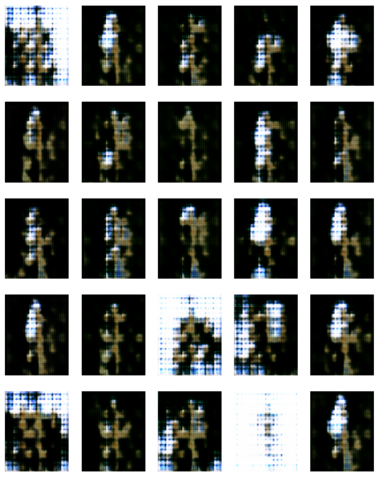
The end results of this experiment were not as detailed as I imagined when I first picked the project idea, but the generator was still able to capture some features better than I imagined. As I ran the algorithm for 100 epochs, I also had the training algorithm produce images representing the generator's predictions for an image of a certain label type based on the parity of the current epoch.

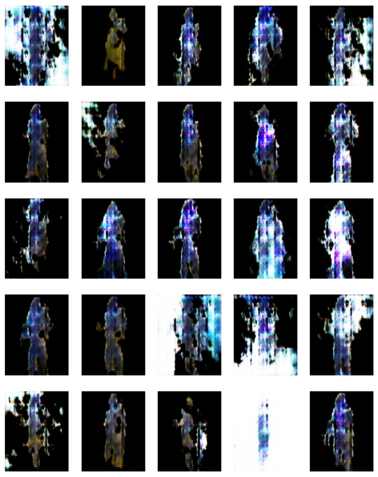

(a) Close-Up Skin Outlines at Epoch 50


(b) Close-Up Skin Outlines at Epoch 100

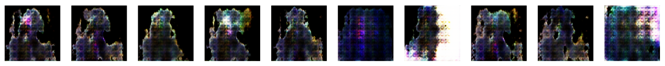
Here, you can see that by Epoch 50, the generator was able to get a good general outline of a human shape. By Epoch 100, the generator was picking up on some of Fortnite's specific features of skins, that being unique pieces of headwear and "back bling", or items that are worn on character's backs.

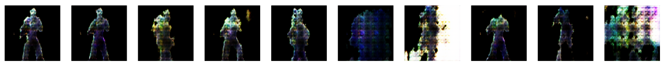
Here, the full-body outlines also begin to pick up on features around the later Epochs, which is exactly what was desired.

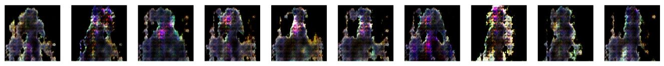

(a) Full-Body Skin Outlines at Epoch 25

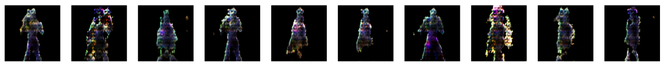

(b) Full-Body Skin Outlines at Epoch 97

After training the model, the generator was run to produce images of both classes, and some of the results are here:









On our generated images, we see that the full-body images seem to turn out better, on average, than their close-up counterparts. Additionally, I showed these images to friends and asked them to indicate what game they thought they were based off of, and the majority were able to accurately guess Fortnite (11/17, to be exact). While the generated images were not as colorful or detailed in design as initially hoped, the fact that key Fortnite-esque characteristics, such as character stances and "back-bling", were captured makes me satisfied, especially considering that I worked on this alone.

6 Conclusion/Future Work

In conclusion, my algorithm was able to capture key characteristics from Fortnite skins and create images that were obviously inspired by their character designs. One problem that I can attribute to the lack of more distinct features is the fact that Forntite itself tends to emulate OTHER games in their character designs, at times. When they utilize characters from other game or media franchises, they attempt to stay true to their styles (see the Dragon Ball skins or Naruto skins for reference). As a result, the only constants amongst all skins are the things captured by the algorithm (stances, pickaxe, back bling, etc). Another is the format of the images post-processing. I realized very late into the process that .png files were causing the backgrounds to be black instead of white when read by the Python program, leading to darker images in the results. With more time, I would be able to account for that and produce brighter images.

Future work on this algorithm would consist of adding more labels to indicate if the skin was an original design or emulation, so that the algorithm could better design characters with Fortnite's specific art style in mind. Additionally, there are several other classes of items that could be explored, which I had to discard when I downloaded the images. Finally, I would like to be able to input and image and have it be re-created in the style of a Fortnite skin. This would take a LOT more time, but I believe that it is possible.

7 Contributions

I did this all by myself.

References

[1] $http : //cs230.stanford.edu/projects_{f}all_{2}021/reports/103146257.pdf$

[2] Christian Murphy, Sudhir Mudur, Daniel Holden, Marc-André Carbonneau, Donya Ghafourzadeh, Andre Beauchamp, Artist guided generation of video game production quality face textures, Computers & Graphics, Volume 98, 2021, Pages 268-279, ISSN 0097-8493, https://doi.org/10.1016/j.cag.2021.06.004. (https://www.sciencedirect.com/science/article/pii/S0097849321001199)

[3] A. Tilson and C. M. Gelowitz, "Towards Generating Image Assets Through Deep Learning for Game Development," 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), 2019, pp. 1-4, doi: 10.1109/CCECE.2019.8861965.

[4] G. Hermosilla, D. -I. H. Tapia, H. Allende-Cid, G. F. Castro and E. Vera, "Thermal Face Generation Using StyleGAN," in IEEE Access, vol. 9, pp. 80511-80523, 2021, doi: 10.1109/ACCESS.2021.3085423.

[5] R. Rodriguez Torrado, A. Khalifa, M. Cerny Green, N. Justesen, S. Risi and J. Togelius, "Bootstrapping Conditional GANs for Video Game Level Generation," 2020 IEEE Conference on Games (CoG), 2020, pp. 41-48, doi: 10.1109/CoG47356.2020.9231576.

[6] https://www.fortniteskin.com/all

[7] https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/

[8] https://learnopencv.com/conditional-gan-cgan-in-pytorch-and-tensorflow/

[9] $https : //keras.io/examples/generative/conditional_{g}an/$

[10] https://medium.com/analytics-vidhya/step-by-step-implementation-of-conditional-generative-adversarial-networks-54e4b47497d6

Repositories Cited:

https://github.com/spmallick/learnopencv/tree/master/Conditional-GAN-PyTorch-TensorFlow
https://github.com/keras-team/keras-io/blob/master/examples/generative/conditional_gan.py

Packages Used:

Tensorflow, Keras, Numpy, Pandas, OpenCV, Selenium, URLLib, Matplotlib