Understanding StyleGAN
Term Project Final Report - Team 11

Seong Woo AHN
20226042
Graduate School of AI
chrisahn99@kaist.ac.kr

Alexandre Petit
20226043
Graduate School of AI
alexandrepetit@kaist.ac.kr

Victor Osso
20226054
Graduate School of AI
osso.victor@orange.fr

1. Introduction

Over the past years, art has progressively become an interest in artificial intelligence. Because of the emergence of new algorithmic techniques, generative art is one of the most popular. The most famous example of this type of art is style blending. This technique is already used to mix paintings (example below 1). The user provides a content image and a style image. Then, the algorithm creates an image which has coarse features of the first one but with the style (e.g impressionism, cubism) of the second one.

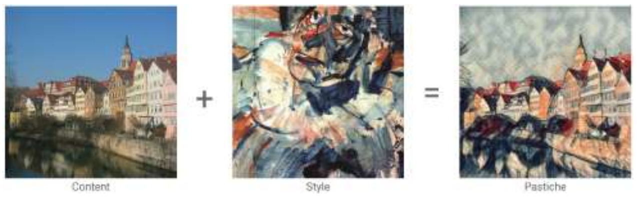


Figure 1. Example of transfer style with two paintings, the content one on the left and the style one one the right

However, art requires certifications and authenticity which are hardly ensured on the internet. That is why some projects are stored as NFTs on a blockchain. Thus, their unicity and proof of ownership is certified. CryptoPunk is an example of them, it is a set of 10,000 unique collectible 24x24 pixel characters stored on the Ethereum blockchain. They can be traded or purchased, the total value of the whole collection is estimated at 1,600,000 ETH (February 2022) or more than 5B USD. Here are the largest sale so far 2 and an example of CryptoPunk's profile page which describes all of its features 3.

We are particularly interested in CryptoPunks because it is a unique pixelized artistic style applied to human faces. First, since the generated image looks like the input image, everyone could have one's profile picture in pixel version. Moreover, this could be used as filters on social media. Indeed, filters are widely used on platforms such as Instagram and TikTok. The CryptoPunk filter could bring some unusual and unique features for casual users but also for influencers who could create and monetize different versions of themselves in CryptoPunk style. Eventually everyone could even have and own their personal CryptoPunk image stored on the blockchain.

In 2018, Nvidia developped a state-of-the-art model named StyleGAN (later followed by StyleGAN2) which is able to generate faces of people with astonishing realism.

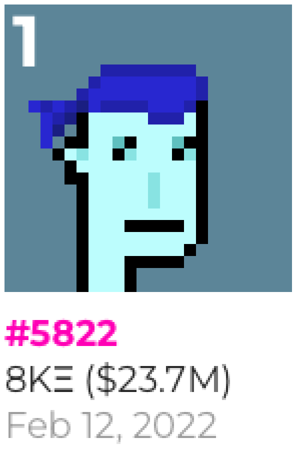
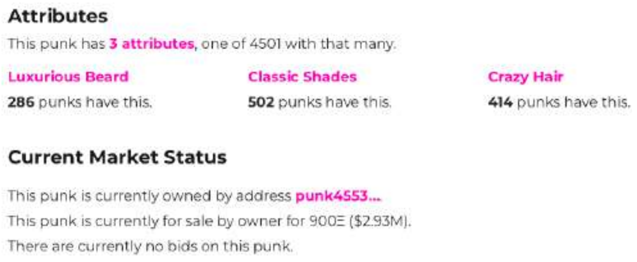


Figure 2. Largest CryptoPunk sale

2. Definition of the task

Our goal is to develop an algorithm which creates a CryptoPunk version of a given input image. Based on transfer learning, the algorithm would transfer the CryptoPunk style onto the input image, which will be someone's picture. We have chosen the CryptoPunk style because it is an unusual creative design and it allows people to have personalized cartoon versions of themselves. Here is an example of a hand-made profile photo with CryptoPunk style 5.



Attributes

This punk has 3 attributes, one of 4501 with that many.

Luxurious Beard	Classic Shades	Crazy Hair
286 punks have this.	502 punks have this.	414 punks have this.

Current Market Status

This punk is currently owned by address punk4553...

This punk is currently for sale by owner for 900Ξ ($2.93M).

There are currently no bids on this punk.

Figure 3. List of all the attributes of a given CryptoPunk, everyone can access it

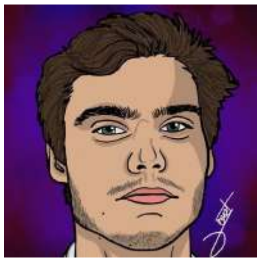


Figure 4. Original image

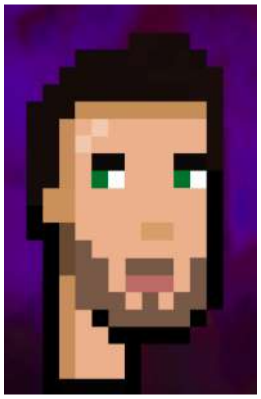


Figure 5. Hand made CryptoPunk design

Ideally, if we have enough time and computing resources, we would also like to make animations. Given a video input such as a Gif, the CryptoPunk would also copy the animation and have the same head, mouth or eyes movements. However we are well aware that animations are more complicated to implement, that is why we only consider it as an additional element, and not the core of our project.

3. Dataset

To train the model we will need some CryptoPunk pictures. We can easily access the 10,000 original CryptoPunks on Kaggle for example which will be our dataset. The dataset is composed of 10000 png images of size 24x24, which is a low quality. We will develop later the issues we faced related to this particularly small input size. They will be used in the training phase so the algorithm can generate pictures with this style. One of our motivations for using the CryptoPunks dataset was that it was clearly one of the most used NFTs currently in the crypto marketplace. Also, since the data is relatively simple (pixelized and low resolution), our initial intuition was that it would simplify the training phase, and require less training time. Furthermore, the presence of abundant data (Kaggle dataset) further motivated the use of CryptoPunks for our fine-tuning task.

4. Related work

For our project we were inspired by previous works. These articles “Online tools to create mind-blowing AI art” [2] and “What is generative art ?” [3] helped us discover different forms of art using AI to generate it. The article “StyleGAN network Blending” [4] is introducing the notion of “layer swapping” with StyleGAN models in order to mix real faces photos with Japanese artwork style as shown in the pictures below 6. It is based on the research done by NVIDIA [1].

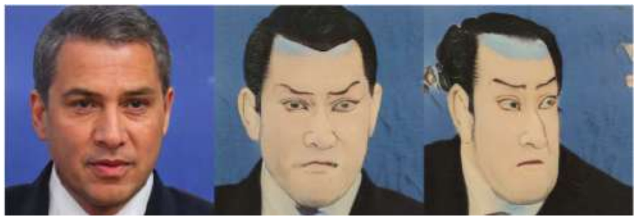


Figure 6. Example of style blending

The author tries to swap the layers of the StyleGAN2 models used to create the final picture to generate photorealistic photos with the style of japanese faces 7.

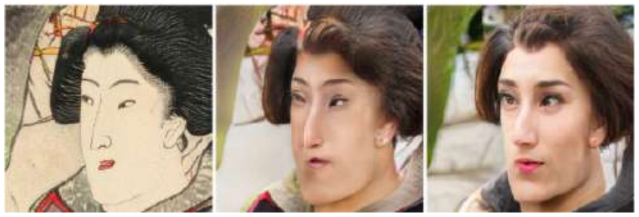


Figure 7. Example of style blending

In these articles “Blending StyleGAN2 models to turn faces into cats” [5] and “Fine-tuning StyleGAN2 for Cartoon Face Generation ” [6] they use the same method to mix human faces with cartoons or cats.

A previous work on the generation of NFT CryptoPunks “Generate NFT CryptoPunks with DGGAN ” [7] has already been done using a DGGAN model which is older and different from the StyleGAN2 model that we intend to use. Here are some CryptoPunks generated in the article 8.

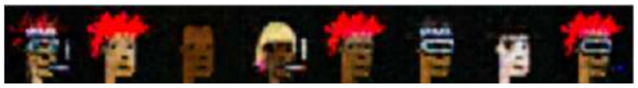


Figure 8. Example of style blending

5. Method

In order to implement this form of style blending to get personalized crypto-punk like NFTs, we use the architecture of StyleGAN2. Before going over the specific architecture of StyleGAN2, it is important to have a quick recap on how GAN, or *Generative Adversarial Networks* work, which are a type of neural network architecture particularly adapted to generating synthetic data.

GANs are not actually a single end-to-end style deep neural network, but actually are composed of two networks, a *discriminator* and a *generator* which are trained asynchronously. The generator here is what we will mainly preserve later on to generate fake images. The first step is to train the discriminator whose task is to classify between real images and fake images, and we do so by feeding it images of the training dataset (here it will be crypto punk images) as well as noise generated by the untrained generator. The discriminator will learn to differentiate which ones are real and which ones are fake. The next step is to train the generator while freezing the discriminator, and the generator will learn to create more realistic images until it has successfully "fooled" the discriminator. We then go to a new cycle, re-training the discriminator based on the real images and the new fake images given by the trained generator. This cycle is repeated until the generator is capable of creating sufficiently realistic images.

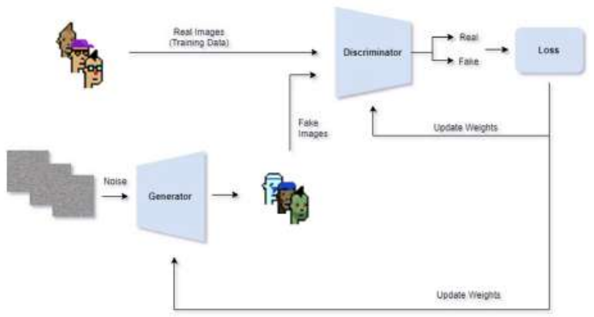


Figure 9. Architecture of a GAN

StyleGAN (NVIDIA 2018) is a new type of architecture that builds upon the foundation of GANs but also adds the possibility of modulating different levels of style in the image, such as pose, head shape, lighting and texture. This novelty is possible through the disentanglement of the latent space by inserting an additional mapping network in the architecture. The input (noise/random vector sampled from a normal distribution) is mapped into a separate vector that is then fed into different levels of the network layers.

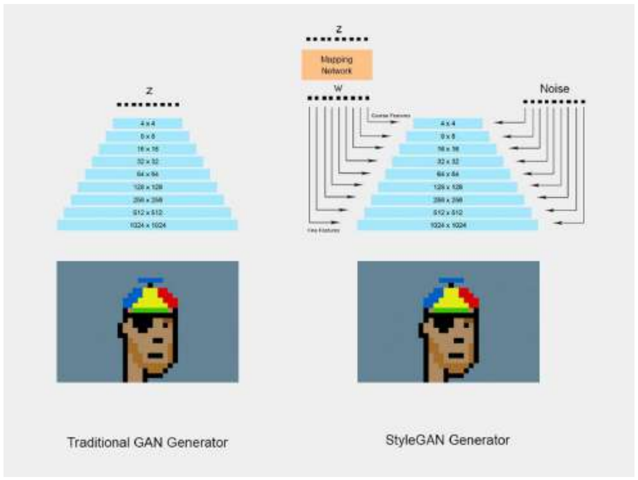


Figure 10. StyleGAN Generator Architecture

Now that we have control over visual attributes of the generated images at different levels, we can implement style blending by swapping layers of a StyleGAN network trained on one style of image with that of another trained on a different style. We could then get an image that preserves the recognizable base features of an input image (such as global facial features, posture, head orientation) and mixes the style of another at the higher level (texture, shading, coloring...). Indeed, given an input image, we want to generate its CryptoPunk version, this is the style mixing step. To do so, the pictures are decomposed in different resolution layers which are interchangeable. Depending on the switched layers, the generated image is modified at different levels: if the low resolution layers are swapped, it changes the structure of the image such as the haircut or the shape of the face whereas if the high resolution layers are swapped, it changes much finer details such as the texture or the colour of the image. For example if we consider two images, an content one A and a style one B. Switching the low layers would lead to a generated image very in between A and B whereas switching the high layers would lead to a generated image similar to A but with the tint of B [8]. Figure 12 exemplifies these differences.

For our implementation, we will train a StyleGAN network on the crypto punk dataset, then switch the final layers with that of a pretrained model, FFHQ (flickr-faces-HQ dataset - trained on images of faces) 15. There are slight differences between StyleGAN and StyleGAN2 [9] such as:

- Simplifying how the constant input is processed
- Removing the mean in the normalizing features
- Move the noise module outside the style module

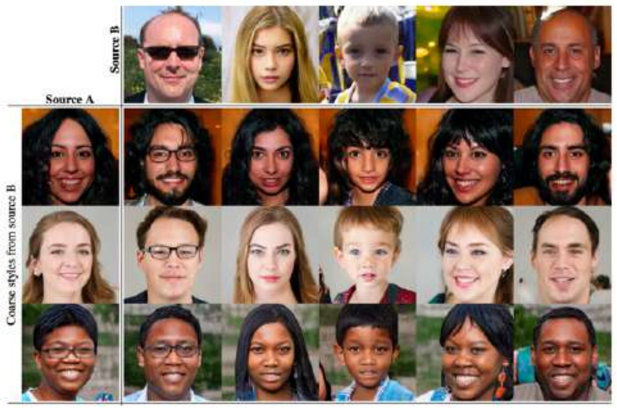


Figure 11. Style mixing: influence of the low resolution layer swapping

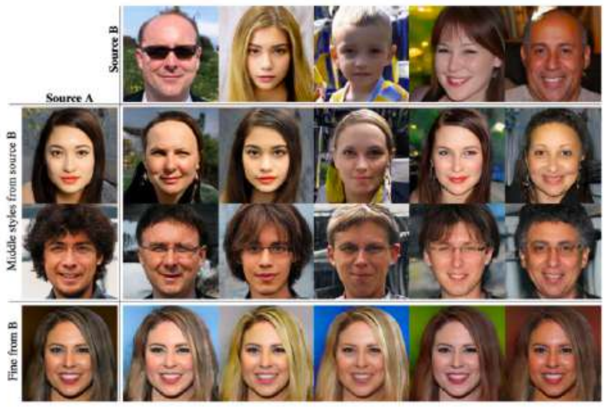


Figure 12. Style mixing: influence of the higher resolution layer swapping

Figure 13. FFHQ: Flickr-faces-HQ dataset

6. Experiment

6.1. Environment

All of our technical work was done in a Google Colab environment. By using Colab Pro account, we benefited from the following technical specs: 1 NVIDIA Tesla T4 GPU with 16GB of GDDR6 RAM 24 hours of total running time The reason why we used a Google Colab environment was because of its ease of use and direct compatibility with Google Drive storage. The Drive storage was leveraged to store our CryptoPunks dataset, as well as other useful data (saved .pkl/.pt files of trained networks). Furthermore, the prior experiences that most of the team members had with google colab further motivated its use for the project.

6.2. Understanding the concepts

6.2.1 Play with style mixing

Our first goal was to mix the style of a human face with another style. At this point there are two possibilities, you can either generate a face that does not exist or start from an existing picture which requires extra steps.

For the last option, the first step is to align the face in the existing image in order to center it perfectly and to crop the image to 1024x1024 pixel. It is very important because GANs and especially FFHQ were trained with 1024x1024 pixel centered faces and they could be troubled if you input different parameters.

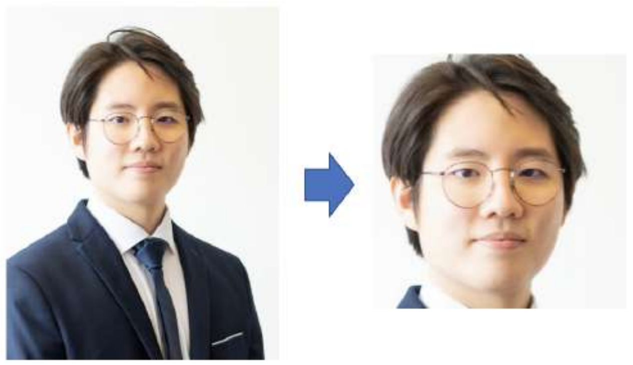


Figure 14. Align : center and crop the face

The second step is to adjust the settings of the generator of faces (FFHQ) so that it can generate the real face that you input. Indeed you can't directly use your image and mix it with another style, you have to generate the face using the generator and get a generated face that is similar to the input image as much as possible. To do that you have to project your own image and make an eigenvector in the latent space of the pretrained model (FFHQ). This vector will be the key to generate the image that you want using the face generator.

The last step is to swap layers of two pretrained models so that the style of the two models blend together and then generate your image. For our tests we tried to mix human faces with a Disney style using FFHQ and a Disney pre-trained StyleGAN2 models, the results presented bellow 15 are quite convincing.

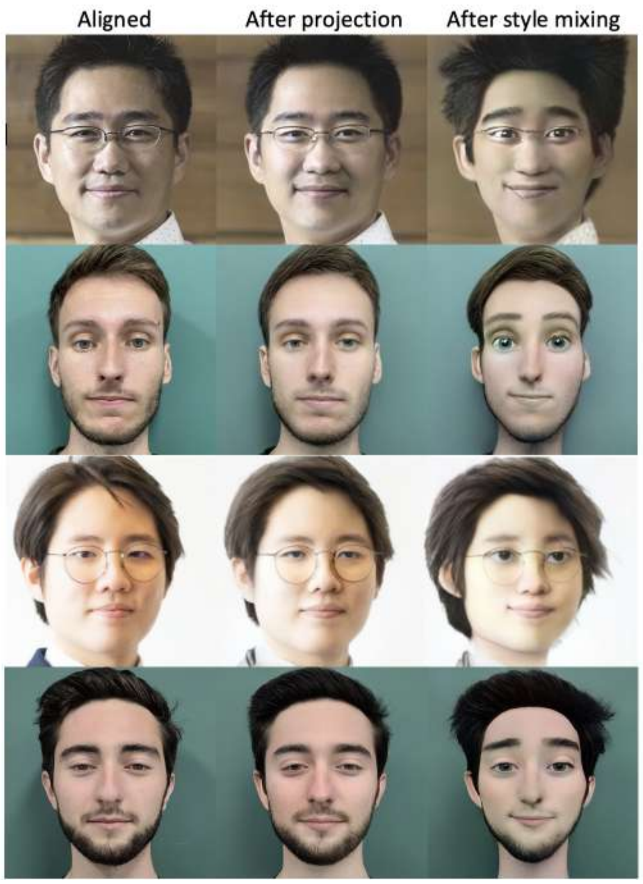


Figure 15. Mixing faces with a Disney style

6.2.2 Layer swapping

Our second goal was to understand and play with the impact of layer swapping. On Figure 16 are presented the results of our tests, we tried to swap the layers of a FFHQ styleGAN and a Naver Webtoon StyleGAN. We can see that if we swap the lasts layers (6 or 5) it will only impact small details like the color or the texture of the image. However, if we swap the first layers it will impact bigger components like the shape of the face or objects in the image.

6.3. Project journey

After testing out the tutorial notebooks and understanding how the training of the network worked, we moved on to the implementation phase: fine-tuning a StyleGAN2 network by using our custom CryptoPunks kaggle dataset. This implementation phase was non trivial, as it was difficult to find existing repositories where we could successfully execute the training. The repositories we used for implementation were the following:

- *Cartoon-StyleGAN: Fine-tuning StyleGAN2 for Cartoon Face Generation* by **happy-jihye**

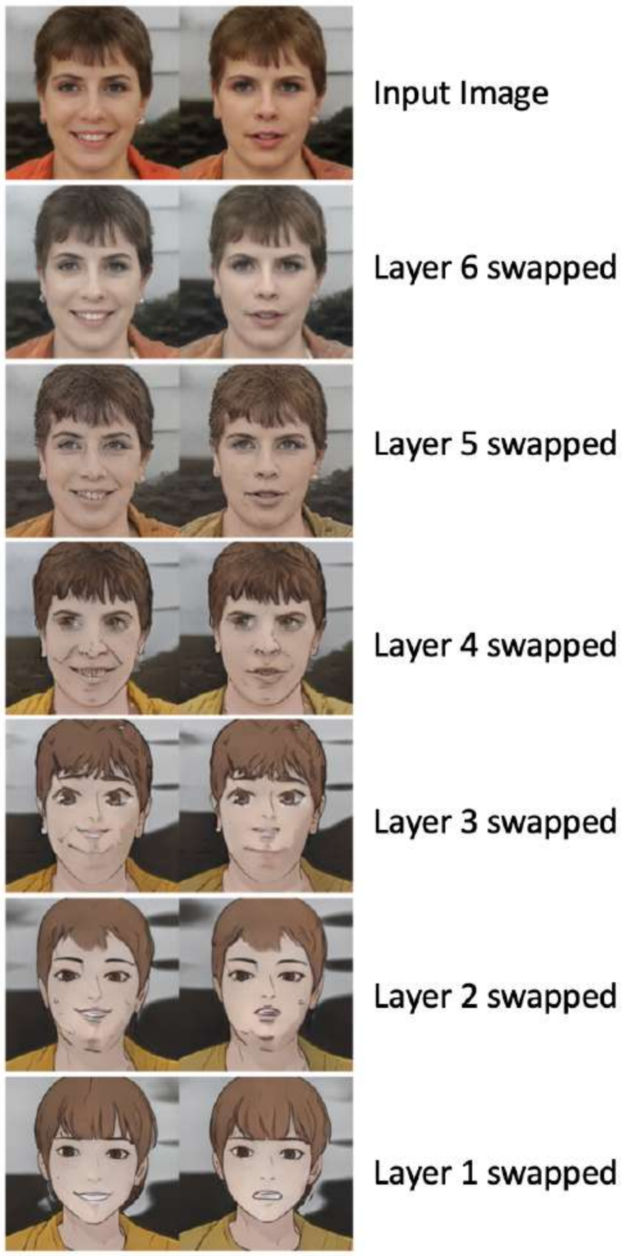
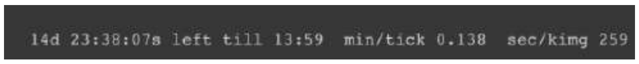


Figure 16. Style mixing by swapping different layers

- *StyleGAN2 for practice* by **eps696**
- *StyleGAN - Official Tensorflow Implementation* by **NVLabs**

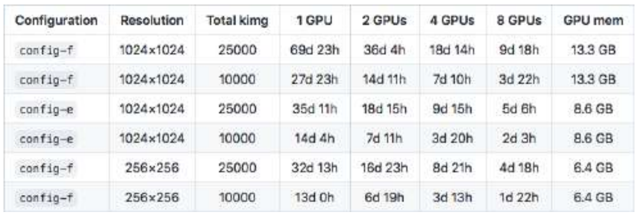
One of the main problem we faced was the training time and the computational resources required to run the model. One of the earliest tries at implementation was using eps696's tutorial. This tutorial, which used Tensorflow, presented the advantage of being very simple to use, as the author gave a notebook that was designed to facilitate direct training on your own dataset. Our very first try was to train a StyleGAN2 network from scratch, but we quickly came to realize that that wasn't feasible, as the training time required for such a task was clearly out of proportion 17, even on our T4 GPU.



```
14d 23:38:07s left till 13:59  min/tick 0.138  sec/kimg 259
```

Figure 17. Ridiculous training times

It is no surprise since GANs in general are known to be notoriously difficult and long to train, even more so with the StyleGAN2 architecture which is quite close to SOTA in terms of style-based image generation. Here's a comparison table which gives an idea of the training times required for StyleGAN2. The running times for Nvidia's implementation is given in figure 18.



Configuration	Resolution	Total kimg	1 GPU	2 GPUs	4 GPUs	8 GPUs	GPU mem
`config-f`	1024×1024	25000	69d 23h	36d 4h	18d 14h	9d 18h	13.3 GB
`config-f`	1024×1024	10000	27d 23h	14d 11h	7d 10h	3d 22h	13.3 GB
`config-e`	1024×1024	25000	35d 11h	18d 15h	9d 15h	5d 6h	8.6 GB
`config-e`	1024×1024	10000	14d 4h	7d 11h	3d 20h	2d 3h	8.6 GB
`config-f`	256×256	25000	32d 13h	16d 23h	8d 21h	4d 18h	6.4 GB
`config-f`	256×256	10000	13d 0h	6d 19h	3d 13h	1d 22h	6.4 GB

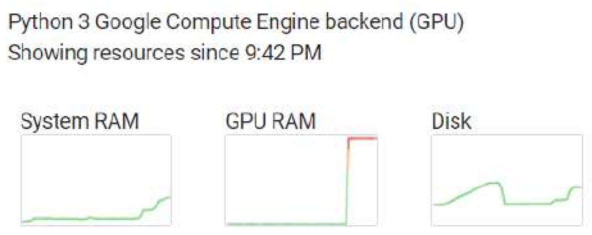
Figure 18. Running time for Nvidia's model if training from scratch

After realizing this bottleneck, we decided to turn towards fine-tuning an existing pre-trained model. Looking at recent literature, we came to the conclusion that using networks trained on the FFHQ (Flickr-Faces-HQ) dataset would be the simplest way, as it is both one of the most popular pre-trained networks for StyleGAN2 and is available in both Pytorch and Tensorflow format (at this point in time of our project, we still didn't know which backend we were going to use for the implementation). It should be noted that the version of the model available on both backends was trained on 1024x1024 size images, which imposed as well for the dataset pictures to be resized at 1024x1024.

Much to our demise, fine-tuning using eps69 still presented technical difficulties, mostly related to GPU issues 19. Here are some of the errors we got:

- *CUBLAS_STATUS_ALLOC_FAILED when calling* 'cublasCreate(handle)'
- *RuntimError: cuDN error: CUDNN_STATUS _EXECUTION_FAILED*

Through some debugging, we realized that the issue was that we didn't have sufficient GPU, which was a bit problematic. This same error occurred when trying to execute with happy-jihye's repository, which was a Pytorch version

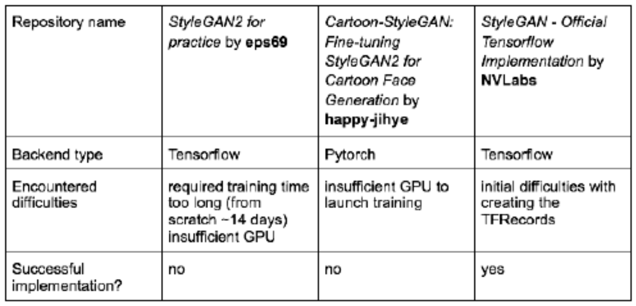


Python 3 Google Compute Engine backend (GPU)
Showing resources since 9:42 PM

Figure 19. GPU resources for fine tune training

of the implementation, and we came to the conclusion that even fine-tuning a StyleGAN2 model on our custom dataset was going to be difficult.

Our solution to this difficulty was to turn towards an older version of the architecture: StyleGAN. For this, we directly used NVLabs' official tensorflow implementation. Although not easy, we finally managed to get this implementation working. Here is a table summarizing the different trials and errors we went through.



Repository name	*StyleGAN2 for practice* by **eps69**	*Cartoon-StyleGAN: Fine-tuning StyleGAN2 for Cartoon Face Generation* by **happy-jihye**	*StyleGAN - Official Tensorflow Implementation* by **NVLabs**
Backend type	Tensorflow	Pytorch	Tensorflow
Encountered difficulties	required training time too long (from scratch ~**14** days) insufficient GPU	insufficient GPU to launch training	initial difficulties with creating the TFRecords
Successful implementation?	no	no	yes

Figure 20. Tables summarizing the repositories used

6.4. Training parameters

For the training implementation on StyleGAN, the parameters which were used for the first trial of training StyleGAN were the following:

- A total kimg of 3500, that is total number of iterations measured in thousands of real images shown to the discriminator
- Adam optimizer for both the Generator and Discriminator. Adam optimizer hyperparameters were $\beta_1 = 0.0$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$.
- Non-Saturating GAN Loss for the Generator. This loss is a modification to the generator loss to overcome the saturation problem, and involves the generator maximizing the log of the discriminator probabilities for generated images instead of minimizing the log of the inverted discriminator probabilities for generated images.

- For the discriminator, they use something called “logistic simplegp loss”, but documentation on this seems very poor. According to what we understood reading the paper [10], it seems to be correlated to WGAN-GP loss (Wasserstein GAN loss).
- Decaying learning rate using a custom scheduler. It varies from 0.001 to 0.003
- Decaying batch size using a custom scheduler. For our trial, the batch size started from 128 and decreased to 64 then 32 as kimg increased progressively.
- The metric used for calculating the distance between feature vectors for real and generated images was Frechet Inception Distance (FID).

7. Results

7.1. Low quality images

As mentioned previously, the images in the dataset have a very small size compared to the usual pictures used in the neural network training. Indeed, usual training datasets that are used in StyleGAN model such as ffhq use images of size 1024x1024. Our fine-tuning dataset must have the same dimensions as the pretrained one, therefore we needed to adapt the CryptoPunk dataset, especially sizing it up to 1024x1024 images. As we can see on the picture 21, resizing blurs the contrasts of the input image, which reduces the quality of the generated images.

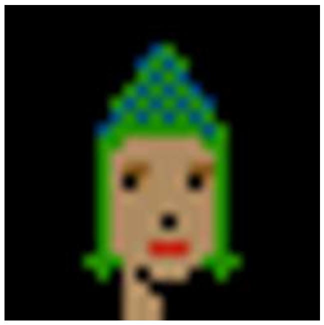


Figure 21. Resized CryptoPunk

Globally this has had a negative impact in our results, as the sharpness was affected and downgraded compared to the original image (22).

7.2. Fine-tuning results

For our fine-tuning implementation using the CryptoPunks dataset and a pre-trained FFHQ model, we ran the training for about 12 hours on a single NVIDIA Tesla T4 GPU. Additional trials at extending the training time were done, but due to technical difficulties and time constraints, 12 hours was the max duration we got for total training time.

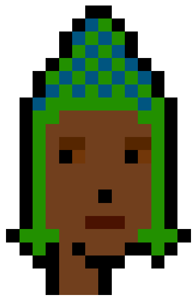


Figure 22. Original CryptoPunk

Figures 23 to 26 show the evolution of the training, with examples of outputs from the generator.

Figure 23. Original FFHQ images (baseline)

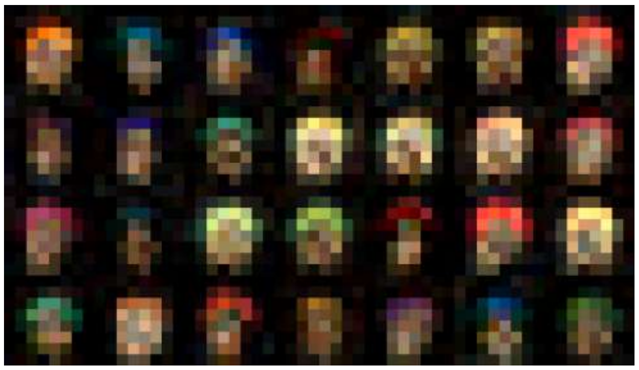


Figure 24. Early training phase result (kimg=280)

We can see overall that the network learns a representation of the custom dataset, as the final results are somewhat similar to the original distribution of training data (27). It is clear that extending the training for a longer period would have yielded even more realistic synthetic images. It should be noted that the model achieved the non-trivial task of learning this new representation from a network pre-trained from FFHQ dataset. This means that the out-

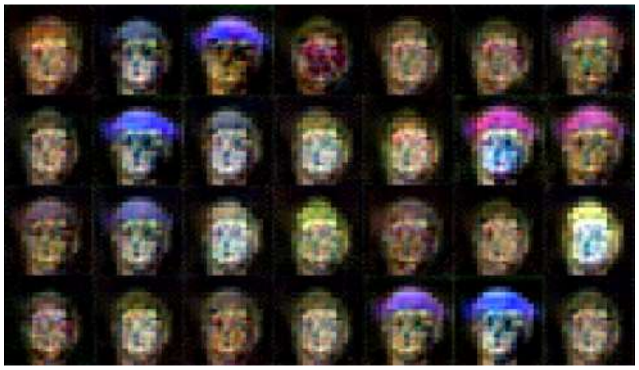


Figure 25. Mid training phase result (kimg=1864)

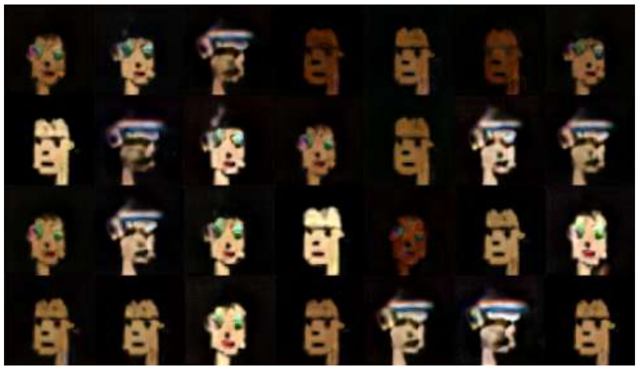


Figure 26. Final result (kimg=3500)

put model shares the same initial layers as the pre-trained network, implying that layer swapping would theoretically work without any problems.

Furthermore, although we didn't get the same level of sharpness as in [7], it should be noted that our implementation fine-tunes from a pre-trained StyleGAN network, and therefore has a more powerful representation than that of DGGAN, as is parametrizable in terms of style features.

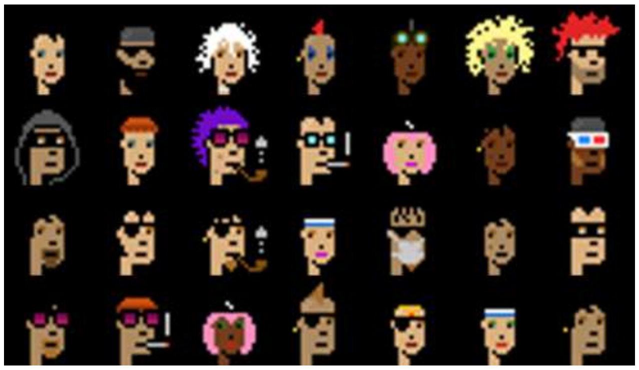


Figure 27. Original training images

8. Conclusion

Ultimately, we weren't able to achieve the original goal of doing style blending by swapping layers between our model and that of the FFHQ pre-trained model. Yet our initial work done on testing swapping layers between FFHQ StyleGAN and Naver Webstoon StyleGAN allowed us to understand the mechanisms involved in doing style blending. With more time in our hands, we would have succeeded in doing so with our own trained model, further extending it to test on real-life test images.

We weren't able to perform ablation studies on our trained model, as our initial goal was to precisely analyze the influence of different network parameters as a more profound academic contribution. Nevertheless, through this implementation project we really got to understand the inner workings of a highly popular and trending GAN architecture. The different manipulations of these networks helped us better understand the mechanisms behind StyleGAN. Without the temporal or technical constraints (reminder: StyleGAN is very GPU intensive and requires more sophisticated material than that we used), it would be possible to elaborate a more interesting study and more complete implementation of a StyleGAN2-based CryptoPunk-style blending network.

References

[1] Tero Karras, Samuli Laine, Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. NVIDIA. 2019. 2

[2] Poornima Nataraj. Online tools to create mind-blowing AI art. 2022 Link 2

[3] Amy Goodchild, What is generative art ? 2022 Link 2

[4] Justin Pinkney, StyleGAN network blending. 2020 Link 2

[5] Tarang Shah. Blending StyleGAN2 models to turn faces into cats, 2021 Link 2

[6] Jihye Back. Fine-tuning StyleGAN2 for Cartoon Face Generation. 2021 Link 2

[7] Bao Tram Duong. Generate NFT CryptoPunks with DGGAN. 2021 Link 2, 8

[8] Jonathan Hui. GAN — StyleGAN & StyleGAN2. 2020 Link 3

[9] Nvidia. StyleGAN2 pretrained models. 2021 Link 3

[10] Lars Mescheder, Which Training Methods for GANs do actually Converge ? 2018 Link 7