University of Reading

Department of Computer Science

Using three GAN-based models to provide modeling inspiration

Linfeng Jia

Supervisor: Dr. Varun Ojha

September 16, 2022

Declaration

I, Linfeng Jia, of the Department of Computer Science, University of Reading, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of UoR and public with interest in teaching, learning and research.

Linfeng Jia
September 16, 2022

Abstract

Following the development of open world games in recent years, some of the landscape and building modelling styles in the game reference real world styles in order to enhance the immersion of the player. However, this is a great challenge for game modelers. To provide game modelers with inspiration for different styles of models, this paper uses the GAN (generative adversarial net), StyleGAN2-ADA and StyleGAN3 models to train the same style of realistic images for comparison. The training data were obtained from Flickr, Google and Baidu images of landscapes and buildings in major Chinese provinces. The results of the experiments show that the GAN converges slowly and crashes easily, and the results are not very satisfactory. StyleGAN2-ADA is faster in training and convergence and uses less data to obtain good quality images. StyleGAN2-ADA is a little slower. In summary, the comparison of the various aspects shows that StyleGAN2-ADA is the most suitable model to provide inspiration to game modelers.

Acknowledgements

Firstly, I would like to thank my supervisor Dr. Varun Ojha for helping me with the structure of my thesis and for recommending a lot of knowledge about GAN which has helped me to expand my thinking and my horizons. Secondly, I would like to thank my classmates for discussing some of the problems we encountered at the same time, which saved me some time in problem solving. Finally, I would like to thank my friends for allowing me to concentrate on my thesis during the time I was writing it, without receiving any outside distractions.

Contents

List of Figures

List of Table

List of Abbreviations

GAN	Generative adversarial net
ADA	Adaptive discriminator augmentation
bCR	Balanced consistency regularization
EMA	Exponential Moving Average
FID	Fréchet inception distance
AdaIN	Adaptive Instance Normalization
ReLU	Rectified linear unit
CNN	Convolution Neural Network
RNN	Recurrent Neural Network
ANN	Artificial Neural Network

Chapter 1 Introduction

1.1 Background

With increasing research in the field of deep learning, the methods based on deep learning are constantly being developed. The algorithms represented in this are CNN (Convolution Neural Network),, ANN (Artificial Neural Network), RNN (Recurrent Neural Network) etc (Pouyanfar et al., 2018). However, the Generative Adversarial Network (GAN), proposed by Ian J. Goodfellow in 2014 shows an important and novel theory for image processing (Goodfellow et al., 2014). This method trains two model simultaneously: a generative model which generate the realistic image with random seed, and a discriminative model to estimate whether the training image from generative model is real or not. GAN is also known as one of the coolest ideas in recent years.

Since GAN was published, various practical GAN-based models have been studied. For instance, Tero Karras et al. used PG-GAN to generate excellent realistic faces using the faces of celebrities as input in their paper published in 2017 (Karras et al., 2017). Moreover, in 2017 the paper by Junyan Zhu et al. presents the well-known CycleGAN technique and a large number of examples of image transformation (Zhu et al., 2017). Furthermore, StackGAN, proposed by Zhang et al. in 2017, can convert natural language into corresponding pictures, including colour, size as well as species (Zhang et al., 2017). In 2019, NVIDIA engineers have proposed a StyleGAN architecture based on PG-GAN. Where the output realistic image will have a similar style to the input image. This structure controls the main style, identity features through ‘style’ (style means the key attributes), and noise to control the detailed parts to make image more realistic (Karras et al., 2019).

During these years, as the GAN algorithm has continued to become more widely known, researchers in other fields also aim to use GAN to complement their research or applications. As Sbai et al. point out that clothing designers can use specially trained GAN-generated clothing images as a source of design inspiration, and because the GAN-generated images are random, the resulting clothing photos are original and highly varied in style, this would be a great help in providing inspiration and improving creativity. (Sbai et al., 2018).

According to the above, researchers believe that the GAN network could also provide inspiration for the modeling of numerous games on the market today. Several games are now gradually introducing the concept of open worlds. In order to increase the immersive effect of the game experience, the style of the in-game architectural models will be very close to the style of real-life architecture. For example, ‘Genshin Impact’ is the most popular game in the world with three different cities, each with a distinctly European, Chinese and Japanese architectural style. In the future there will be more cities that are similar in architectural style to real countries (Hoyoverse, 2020).

1.2 Problem Statement

In the field of game design, architectural modelers may suffer from a lack of inspiration in designing architectural models for games and this will cause failure to produce high quality and stylish models. It will not only have an impact on the company's projects, but also leaves the designer with a lack of confidence. Nevertheless, GAN could provide designers with inspiration by generating realistic images of buildings through unsupervised learning. Thus, the research question is how to use appropriate GAN models generate landscape and building images to provide inspiration to game landscape and architectural modelers.

1.3 Aims and Objectives

The aim of this study is to find a suitable GAN architecture to generate images of Chinese landscapes and buildings in the same style.

The objectives are:

- Understanding different GAN structures and choose three of them that are suitable to generate landscape and building images.
- To explore the strength and weakness of the previous work on the choose structures.
- Finding a Chinese landscape and building database and resize them to square images.
- Adjusting the hyperparameters of three different models and training the model with previous database as input.
- Comparison of the structure, overall training time, training time per epoch, accuracy of trained images and number of trainable parameters of the three different models.
- Thinking what practical applications these GANs models could be used in.
- To assessment the society and ethical impact of these GANs models.
- Self-assessment of the limitations of the method and where improvements can be made in the future.

1.4 Solution Approach

This article selects three GAN structures in order to generate better images of Chinese landscapes and architecture: the original GAN, StyleGAN2-ADA and StyleGAN3. Original GAN is the most basic GAN structure proposed by Goodfellow in 2014, most other GAN structures are based on Original GAN (Goodfellow et al., 2014). Therefore, it is chosen as the first solution in this article. StyleGAN2-ADA and StyleGAN3 are two GAN structures that can be trained to produce images with the same style as the input image, and their features are well suited to solve the problem presented in this article, as worked out by NVIDIA engineers. Among them, StyleGAN2-ADA could show a low

FID (Frechet Inception Distance) score training results even when the data size in the training dataset is not as sufficient (Karras et al., 2020). StyleGAN3 builds on StyleGAN2 and solves the problem of unnatural image transitions that make the pixels look like they are sticking together on the coordinate not on the surface of object when morphing (Karras et al., 2021). In the following article we will train all three methods using the same dataset and compare the results of all aspects.

1.5 Organization of report

This paper is organised into seven chapters. Chapter 2 will show the literature review of previous work of GAN. Chapter 3 focuses on the structure of the three GANs and how the three methods are trained with the dataset. Chapter 4 describes the results obtained by the three different GANs after training and the comparison of the performance of the three GANs with the same dataset. Chapter 5 will conclude the experiments, summarise the advantages and disadvantages of the three GANs for the dataset in this paper and look at how they can be improved in the future. It will also discuss what real-world applications the three methods can be applied to. Chapter 6 will provide a reflection that summarizes what the authors have learned out of GAN. Chapter 7 is the reflection of my learning experience during write the dissertation.

Chapter 2 Literature Review

2.1 Image processing methods regarding deep learning

2.1.1 Generative Adversarial Network

In order to provide inspiration to game modelers and to stimulate the desire to create is the research question of this article. Image generation techniques using deep learning models would be a favoured option. Deep learning models can be divided into generative models and discriminative models (Bengio, 2009). The discriminative model has been greatly developed because of the Back propagation algorithm proposed by Seppo Linnainmaa (Linnainmaa, 1976), the Dropout algorithm invented by Nitish Srivastava et al. to solve overfitting problem when training a model (Srivastava et al., 2014). However, the development of generative models has been slower because they are more difficult to model. It was not until 2014 when Ian Goodfellow et al. proposed the Generative adversarial network that it received increasing attention from academia and industry (Goodfellow et al., 2014). In the following years, GAN has also gained rapid development in theory and models.

Generative adversarial network is a deep learning model as well as an unsupervised learning model. Its main structure is a generative model and a discriminative model. The two models game and learn from each other to produce a better output. Although the original GAN theory does not specify that the two models must be neural networks, in general both the generative and discriminative models will be deep neural networks. In a generative adversarial network, the generative network is responsible for generating the image, and it input is a random noise z, generative network uses this input to generate a random image noted as G(z). The discriminative network is to determine whether the image is real or fake. Its input parameter is x (x represents an image generate by generator) and the output of the discriminative network noted as D(x), represents the probability that image x is a real image. D(x) ranges from 0 to 1, with closer to 1 meaning that the discriminator considers the image to be real and vice versa. From the above description, it is clear that GAN is a process in which two models

play against each other. The generative network tries its best to generate real images to deceive the discriminative network, while the discriminative network tries its best to differentiate the images generated by the generative network from the realistic ones.

The generator in GAN takes an input of an n-dimensional vector and it will output an image of a specified image pixel size. The generator can be any model that outputs an image, such as a deconvolutional network, a fully connected neural network, etc. Since GAN is only designed to generate realistic images, there is no requirement for specific information about the image output by the generator. Thus, study fellows just use a randomly generated vector as input. The random input should ideally satisfy common distributions (Gaussian, mean distribution, etc.).

After both the generative and discriminative models have been set up, the GAN network start training. In the first step, a random sample is taken from a noisy data distribution and input into the generative model to output a set of fake data z. In the second step, a random sample is taken from a real data distribution and recorded as real data x. The third step uses the data obtained in the first two steps as input to the discriminator model (the input to the discriminator model is real and fake images) The output value is the probability that the generate image is a real image. In the fourth step the algorithm calculates a loss function based on the probability values output from the discriminator model in the previous step. It should be noticed that the discriminator model and the generator model do not have the same loss function. In the fifth step, the parameters of the model are updated using the backpropagation algorithm based on the generated loss function. Generally, the parameters of the discriminator model are updated first, and then the generator parameters are updated by sample the noise data. At this point, the GAN network has completed one training loop. After several loops of training, the GAN will output realistic images (when the discriminator's output is close to 0.5).

2.1.2 StyleGAN

After the introduction of GAN by Ian Goodfellow in 2014, an increasing number of models based on GAN methods have been researched. Among the various models, the StyleGAN model, developed by NVIDIA engineer Karras et al. has received widespread acclaim and application (Karras et al., 2019). 'Style' in StyleGAN refers to the main attributes of the image data (in the StyleGAN paper this refers specifically to expression, skin colour, hair style, etc. of the face). In addition to these main attributes, StyleGAN also uses noise to control the details of the image (hair, wrinkles, etc.) in order to increase the realism of the image.

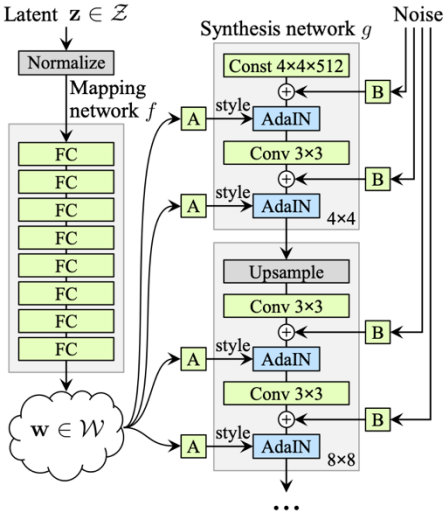


Figure 1: StyleGAN generator architecture (Karras et al., 2019)

The main structure of StyleGAN generator is in two main parts. One part is the Mapping Network, and the other part is the Synthesis Network. StyleGAN still maintains the structure of the PG-GAN (Progressive Growing GAN) proposed by Karras et al (Karras et al., 2017). The main role of the Mapping Network in StyleGAN is to decouple the latent space. In the process of generating images, the features of the image data need to be represented to better generate or classify the image data. However, image data generally has multiple features and have high coupling between them. It makes difficult for the model to identify the connection between them. This could make the model inefficient in learning. Therefore, in order to improve the learning efficiency of the model, researchers find the deep relationships under the features of the image data, and

decouple these relationships to extract the hidden features, which is the latent code. Latent space is the space composed of latent code. The Mapping Network consists of eight fully connected layers, where the input value latent z is transformed by a series of affine transformations to give the output value intermediate latent w. Different elements of w control different visual features. The reason for converting the variable z to w via Mapping Network is that in general the variable z is a random vector that fits a uniform or Gaussian distribution. However, this is not the case. In StyleGAN paper, Karras et al. propose a combined distribution of hair length and masculinity features in the training set, where the longer the hair length the weaker the masculinity, which leaves the part about the longer the hair the stronger the masculinity missing (Karras et al., 2019). The results obtained would be inaccurate if the uniform distribution and Gaussian were used for sampling. However, after Mapping Network, the model could generate a vector w that does not need to follow the distribution of the training data. This step also reduces the correlation between image features so that synthesis network for individual control of each feature of the image data.

The second part of the StyleGAN generator is the synthesis network, where the output values of the Mapping Network w are input into each layer of the generator via the AdaIN (adaptive instance normalization) (Huang & Belongie, 2017) style transformation method of each convolutional layer to control the style of the image, so that each convolutional layer could adjust the style of the image according to the input values. The transformed noise is added before each AdaIN layer to enrich the image with detail. For the discriminator the authors used the same structure as PG-GAN, the Adam (Kingma & Ba, 2014) optimizer and minibatch sizes.

Based on the structure described above, the researchers also propose in the paper a method Style mixing. This method allows the combination of two picture styles to generate a new realistic picture. The authors obtained two outputs, w1 and w2, representing the styles of two different images, by inputting two different latent codes, z1 and z2, into the mapping network. The synthesis network then randomly selects a

layer before which w1 is used as input and after which w2 is used as input. The resulting image will then have both the style of the first image and the style of the second image. Depending on the choice of convolutional layers, the images are generated differently. Because StyleGAN's convolutional layers are structured from low to high resolution, the choice of different layers will result in different characteristics of the w-controlled image data.

2.1.3 StyleGAN2

Since the introduction of StyleGAN, several applications have been developed based on its ability to generate images of the same style or a mixture of different styles, as well as to change the feature of an image according to different latent codes. In the article by Bermano et al. (2022) points out that StyleGAN can change the pose of an image character, gender hair length, etc. Moreover, it can also change the overall style of an image, from realistic to anime or oil painting style. It could even change the facial expressions of the people in the picture (Shen et al., 2022). However, as more and more people use StyleGAN to generate images, some problems with the model have gradually emerged. Therefore, Karras et al. further improved the model based on these findings and analysed it to improve the quality of the images generated by StyleGAN model which means the second generation of the StyleGAN model was developed (Karras, Laine, et al., 2020).

According to experiments by Karras et al. it was found that StyleGAN has the potential to generate water droplet-like imperfections when generating images. The researchers finally found that the reason that this problem would arise was a problem with AdaIN. Because AdaIN is normalising each feature map, it may destroy the information between the features of the image. To solve this problem the authors adjusted the network structure of StyleGAN. Firstly, the author removed the noise and normalisation Mean/Standard Deviation from the initial input of the synthesis network. Secondly, they remove Standard Deviation from normalisation and the third step removes all noise inputs from the style module. After the above steps the water droplet problem was

solved. Another highlight in StyleGAN is Style Mixing, and in StyleGAN2 this approach is a step closer to accurate control of image features. However, changing only the structure of the network is not enough to improve accuracy, so the researchers have modified weight demodulation by combining Modulation Standard Deviation with the convolutional layer, with mod std scaling the weights of the convolutional layer and then normalisation Standard Deviation as weight demodulation. This approach not only improves the accuracy of Style Mixing for image features, but also improves the training speed of the model.

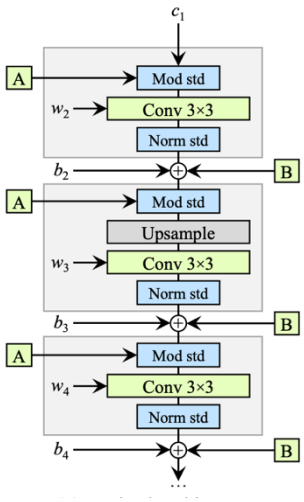


Figure 2: StyleGAN2 generator architecture (Karras, Laine, et al., 2020)

Furthermore, the authors also found that the way to generate high resolution in StyleGAN is to use a network of progressive growing GAN (Karras et al., 2017). This network is trained to generate a 4*4 resolution image and then gradually transition to a higher resolution image when the training is stable. However, researchers find out this network is very unstable when generating high resolution images, and the discriminator can easily separate whether an image is real or fake, which can lead to the weights of the generator not being updated effectively during training. Thus, Karras et al. have redesigned the StyleGAN2 image generation architecture inspired by the MSG-GAN published by Karnewar & Wang (Karnewar & Wang, 2020). The MSG-GAN approach is to first train images from 4*4 resolution and then map features from the generated low-resolution images to the final generated image via Resnet-style skip

connections. This enables each training step to influence the final output, increasing the stability of the network. The authors also used lazy regularization in StyleGAN2 to reduce the computational cost and not have a significant impact on the final result (regularization every 16 minibatches).

2.2 Relationship of the three methods to the research question

The research question in this article is: how to use suitable GAN structures to provide relative inspiration to game architectural landscape modelers. In the above review of the paper, models based on the GAN, StyleGAN and StyleGAN2 model structures might be appropriate to provide inspiration to game architectural landscape modelers, the reasons are as follows:

GAN is an unsupervised learning image model that automatically learns the data distribution of a sample set so that it can be easily used. User could put some landscape and building images into the model, and it will output a realistic image for the modeler to use as a reference. It's like a black box, the game modeler doesn't need to know the internal structure, he just needs to put in the images to get the output he wants.

StyleGAN improves on Progressive growing GAN by allowing it to influence the style (various attributes of the image) of the output image. In StyleGAN the authors propose and implement the idea of style mixing. This idea also coincides with Ali & Lee. They propose that the styles of two different buildings could be combined and used as a reference for new architectural designs (Ali & Lee, 2021). However, this idea is better reflected in the architectural and landscape modelling of the game. In Monster Hunter World, an open world game, the map has a series of designs combining rainforest and mountains, sand dunes and lava (Officially called ‘Ecosystem’), and different building designs in the town depending on the map area (Capcom, 2018). Because StyleGAN is highly effective at generating the same style or a mix of styles as the input image,

the images generated with it could be a valuable reference for game modelers.

StyleGAN2 improves on the first generation with structural changes and enhancements to make it more suitable for generating buildings and landscapes in HD resolution, and StyleGAN2 improves on the first generation's potential for water droplet artifacts problems and unrealistic images where some parts of the image may remain in place when image rotated. Moreover, StyleGAN2 also enhances the style mixing function by changing the network structure and weight demodulation to give more detailed control over 'style'. This change also further expands the possibilities for game modelers to use StyleGAN2 to generate different styles of images, allowing them to combine more detailed sections from different styles of architecture and landscapes to provide design inspiration.

2.3 The critique of three methods

Although the three methods mentioned above are all useful in solving the research problem, there is still potential for improvement in each method. This section will compare the network structures of the three methods and identify their weaknesses by comparing the experimental data and the final output images.

Original GAN is considered to be the first generative adversarial network, which has the advantages of unsupervised learning, automatically learning the distribution without knowing the data distribution of the original sample and producing better image samples than other generative models. However, GAN still has some problems that need to be solved. For instance, when both the generative and discriminative networks in a GAN are composed of neural networks, there is a risk that the entire network may not converge due to the constant adjustment of its own network weight. Moreover, because the GAN needs to update the discriminative network several times before updating the generative network during training, it is difficult to know how far the training has progressed. At the same time, the GAN is a bit overwhelmed with training

high-resolution images, and the training time could become very long.

StyleGAN is a model that learns the 'style' (image data attributes) of an image to enhance the image learning capability of the GAN and make the style of image generation controllable. It also provides a significant improvement in stability for high-resolution image generation compared to GAN. However, it has the potential to produce water droplet problems when generating images, which in severe cases may affect the integrity of the image subject. Secondly, because of the use of progressive growth, the more frequent details of the output in StyleGAN may be fixed in place and not change as the subject changes (rotates, moves, etc.), which may make the image look less realistic.

StyleGAN2 solves the water droplet problem and the image detail non-following problem in StyleGAN by changing the network structure and Weight Demodulation. And StyleGAN2 has higher accuracy for image style control than StyleGAN. According to the authors' experiments using the FFHQ dataset, the FID of the images generated by the second-generation algorithm is also reduced compared to that of the first generation (the lower the FID value, the more realistic the image is). At the same time, some shortcomings of StyleGAN2 were identified in the course of the experiments. For example, some details of the image (human beard, eyelashes, etc.) are shifted according to the axes rather than according to the content of the image during the style transformation, which makes the generated images look like they are stuck together, and the details are not displayed in high definition. In addition, a database of at least 30,000 images is required for better training results and collecting this number of images would be a major undertaking.

2.4 Conclusions

In summary, in order to address the research questions raised in Chapter 1, this section reviews the structure of three GAN models (GAN, StyleGAN, StyleGAN2) and the

various approaches to image processing proposed by the authors based on the different structures. Moreover, the paper analyses the weaknesses of the three models and the problems that may be encountered. However, it could see that each of the three models also has its own strengths in solving the research problem. GAN is more scalable in the network structure among the three models and can adapt the network to different needs. This approach is more suitable for people who have some basic knowledge of generative adversarial networks. The advantage of the StyleGAN model is that it focuses more on the 'style' of the image, which can be of great help to modelers who want to generate landscape images in the same style or a mixture of styles to provide inspiration. StyleGAN2 builds on the first generation with enhanced control over image detail style and fixes the water droplet problem that could occur in the first generation. The code of this model is similar to a black box, so the user does not need to know what is inside, but only needs to input hyperparameters for training, making it more user-friendly.

Chapter 3 Methodology

3.1 Algorithm description of three different GANs

3.1.1 Generative adversarial net

In the previous chapter, we reviewed the general structure of GAN, in this section we will explain each step of GAN in more detail.

The main structure of the generative adversarial network is consist of a generator and a discriminator, where the input to the generator is the latent space and random noise, after which the generator will get a set of image outputs G(z) based on the input noise, and this set of output data and the real image are input into the discriminator to get the probability D(x) that G(z) is the real image. After obtaining D(x) the loss function is calculated to adjust the weights of the two networks, with the final aim of making the discriminator output as close as possible to 0.5. (The discriminator cannot determine whether the image is generated by the generator)

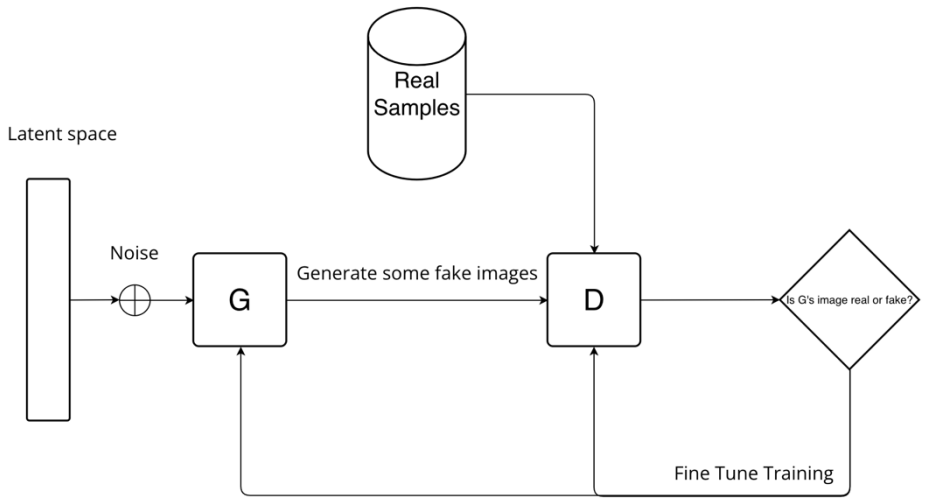


Figure 3: The structure of Generative adversarial network

In GAN, the generator needs to be a model of the output image, while the discriminator can be any discriminator model. Since the purpose of the discriminator is to discriminate whether the image is a real image (the similarity of the data distribution), the GAN needs to calculate a loss function to update the model parameters, and the mathematical expression of the loss function is Equation (1)

$$L(G,D) = -\mathbb{E}_{x\sim P_r} logD(x) - \mathbb{E}_{z\sim P_z} log(1-D(G(z))) \quad (1)$$

The task of the discriminator is to maximise L(G,D) and the task of the generator is to minimise L(G,D) so that the discriminator cannot distinguish between the real and the fake generated images. Generally, the discriminator model is updated after the loss function has been calculated, and then the generator model is updated. Furthermore, we could find that the discriminator is a classifier in the network. So, we can generally use cross entropy (de Boer et al., 2005) to discriminate the similarity of the image data distribution with the equation

$$H(p,q) \coloneqq -\sum_i p_i log q_i \quad (2)$$

where p_i and q_i in the equation are the distribution of the real samples and the distribution generated by the generator, respectively. Since the function of the discriminator in GAN is to determine whether the image is a real image or not (a binary classification problem), the equation can be further expanded as

$$H((x_i,y_i)_{i=1}^N, D) = -\sum_{i=1}^N y_i logD(x_i) - \sum_{i=1}^N (1-y_i) log(1-D(x_i)) \quad (3)$$

According to the derivation of the equation of cross entropy, Goodfellow et al. proposed the objective function of the optimization function in GAN (Goodfellow et al., 2014), which is also considered to be the core of GAN

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))] \quad (4)$$

This equation summarises everything that generators and discriminators do in a network. You could see from this equation that the generators and discriminators are playing a game, in a process of continuous learning and playing to finally reach a balance.

Having obtained the structure and loss function of the GAN, the authors also propose an algorithm for the GAN, as shown below:

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 4: Generative adversarial network Pseudo-algorithms (Goodfellow et al., 2014)

Combining the algorithm with figure 3 GAN architecture diagram we could see that the algorithm samples randomly in the noise to get a set in the data $z_1 - z_m$, and then randomly in the real data set to get a set of data $x_1 - x_m$. Furthermore, one of the data from $z_1 - z_m$ and $x_1 - x_m$ is input into the discriminator, which outputs the probability that the input value is a real image. Next step, the loss function could be calculated according to the loss function equation and the probability value. Finally, the algorithm updates the discriminant model first according to the loss function and the back

propagation algorithm, then updates the generative model afterwards. This completes one loop of GAN training.

3.1.2 StyleGAN2-ADA

The StyleGAN2-ADA (StyleGAN2-adaptive discriminator augmentation) algorithm is an improved algorithm based on the structure of StyleGAN2 reviewed in the literature review. Its main purpose is to solve the problem that StyleGAN2 may produce overfitting of the discriminator during training if the amount of training sets is small (Zhang & Khoreva, 2018), which may lead to the generator not being able to effectively adjust the model parameters thus leading to a reduction in model training efficiency. In general, the solution to this problem would be to use image augmentation. However, this method may result in affecting the output image. How to use image augmentation without affecting the output image is the thing that StyleGAN2-ADA does.

The core algorithm of StyleGAN2-ADA is the addition of image augmentation to the structure of StyleGAN2, which is based on the bCR method (balanced consistency regularization) proposed by Zhao et al. (Zhao et al., 2021).

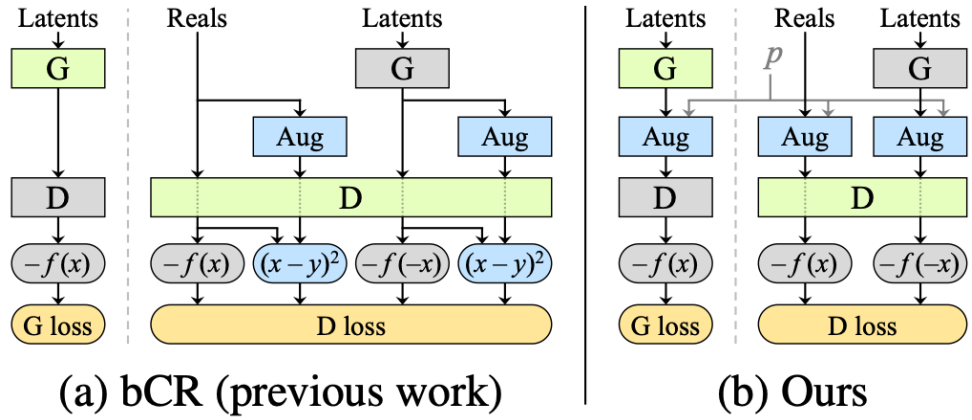


Figure 5: (a) is the image augmentation structure of bCR (Zhao et al., 2021) and (b) is the image augmentation structure of StyleGAN2-ADA (Karras, Aittala, et al., 2020)

As shown in Figure x, the main idea of the bCR method is to add a consistency regularization term to the discriminator. If bCR performs two different image augmentations on the same image, its output should be consistent. However, in StyleGAN2-ADA, the authors argue that bCR only performs image augmentation on the discriminator cannot constrain the output of the generator. Therefore, the authors add image augmentation to StyleGAN2 generator model and remove the consistency regularization term added to the discriminator loss function. For the loss function, the authors use a non-saturating logistic loss (Goodfellow et al., 2014), which is formulated as: $f(x) = log(sigmoid(x))$. In terms of transfer learning, among the transfer learning with respect to GAN studied by other authors (Wang et al., 2020) (Wang et al., 2018), Karras et al. found that transfer learning with the freeze discriminator layer studied by Mo et al. (Mo et al., 2020) has the best performance in StyleGAN2-ADA.

Generally, when doing image augmentation, a probability p is set which controls the intensity of the image augmentation. p ranges between 0 and 1 and is generally an artificially set hyperparameter. Researchers have found that the value of p can greatly affect the final output image. However, the optimal value of p varies depending on the augmentation method and the data set. Different datasets and augmentation methods have different p. This could make it difficult to determine the p -value before the start of the experiment. Since image augmentation is used to ensure that the discriminator is not overfitted to the image during training. Therefore, the authors propose two heuristic indicators for overfitting, which dynamically adjust the p-value according to the degree of fit (Karras, Aittala, et al., 2020). The two heuristics are r_v and r_t respectively, and their equation expressions are

$$r_v = \frac{\mathbb{E}[D_{\text{train}}] - \mathbb{E}[D_{\text{validation}}]}{\mathbb{E}[D_{\text{train}}] - \mathbb{E}[D_{\text{generated}}]} \qquad r_t = \mathbb{E}[\text{sign}(D_{\text{train}})] \tag{5}$$

The $D_{train}, D_{validation}, D_{generated}$ represents the output of the discriminator in the network for the training set, the validation set and the generated image set, respectively. After obtaining the results of the three parameters and bringing them into Eq. (5), the

discriminator results are averaged for each set of consecutive batches. The authors used 4 batches of batch size 64 in the algorithm to find the mean value. r values range from 0 to 1, with 0 representing no overfitting and 1 representing complete overfitting. However, this method has one drawback, as the data set is relatively small, this method also requires the data set to be divided into a training set and a validation set, further reducing the size of the training set, which is not what we would like to see. Therefore, the authors further propose a second heuristic r_t. The output of r_t is the proportion of D_{train} that gets a positive value. From equation (5) it could be seen that r_t only needs one parameter D_{train}. In the ideal case, D_{train} should converge to around 0 then there is no overfitting and vice versa. The r_t also used by the authors in the StyleGAN2-ADA algorithm for dynamically adjusting p, decreasing the p value as rt approaches 1 and increasing it as it approaches 0.

3.1.3 StyleGAN3

StyleGAN3 builds on the second generation in order to solve the problem of image texture sticking that arises during image deformation (e.g., transformation from one face to another). This problem can lead to unnaturalness and blurring in the image deformation process and in the final generated image. According to Karras (Karras et al., 2021), this problem is caused by the fact that the existing StyleGAN2 network has some positional reference information in the intermediate layer, which causes the pixels of the image to be fixed at the same coordinates during the deformation process. Among the causes of this are: image borders, noise input, and Aliasing, which is the effect of different signals becoming indistinguishably overlapped when sampled. This effect usually occurs when the sample rate is too low.

To address these issues, the authors redesigned the network structure of StyleGAN2. The researchers found that image texture sticking would be greatly reduced if a position code that would affect the transformation process was not inserted during the operation. However, this operation is difficult to implement in a normal neural network

layer (which is a discrete domain). Therefore, the authors switch the operation in the network from the discrete domain to the continuous domain for the operation. At the end of the operation the continuous domain is then transformed into the discrete domain for the other steps. In StyleGAN3, the discrete domain is converted to the continuous domain according to the Whittaker-Shannon interpolation formula (Shannon, 1949), which is given by: $z(x) = (\emptyset_s * Z)(x)$. where * stands for continuous convolution and $\emptyset_s$ stands for low-pass filter, Z means the discrete feature map. After the convolution, up/down sampling and ReLU operations, the network converts the continuous domain into a discrete domain by means of $two - dimensional\ Dirac\ comb$, which is given by:

$$III_s(x) = \sum_{X \in \mathbb{Z}^2} \delta(x - (X + \frac{1}{2})/s) \tag{6}$$

Converting a continuous domain to a discrete domain is simply a process of sampling from a continuous feature map using the above formula. Based on the above transformation approach, the researchers redesigned the convolutional layers, upsampling, downsampling and non-linearity based on the network structure of StyleGAN2. The authors represent F as an operation on a discrete feature map Z and f as an operation on a continuous feature map z as

$$f(z) = \emptyset_{s'} * F(III_s \odot z) \qquad F(Z) = III_{s'} \odot f(\emptyset_s * Z) \tag{7}$$

This makes it feasible to see the operations of a different domain corresponding to another domain. In the convolution layer, considering a discrete kernel K and a convolution with sampling rate s, we could derive different expressions in the discrete and continuous domains respectively according to the above equations

$$F_{conv}(Z) = K * Z \qquad f_{conv}(z) = \emptyset_s * (K * III_s \odot z) = K * (\emptyset_s * III_s \odot z) = K * z \tag{8}$$

At the upper sampling level, the continuous domain upsampling in the ideal situation is not modified. So, the expressions for the discrete and continuous domains in upsampling are respectively

$$F_{up}(Z) = III_{s'} \odot (\emptyset_s * Z) \qquad f_{up}(z) = z \tag{10}$$

In the continuous domain downsampling operation, $\psi_s \coloneqq s^2 \cdot \emptyset_s$ is the interpolation filter normalized, which can be used as a low-pass filter to filter out high frequencies to avoid Aliasing. the expression for the discrete domain downsampling can be obtained from equation (7). The continuous and discrete domain downsampling expressions are

$$F_{down}(Z) = III_{s'} \odot (\psi_{s'} * (\emptyset_s * Z)) \qquad f_{down}(z) = \psi_{s'} * z \tag{11}$$

In the nonlinearity layer, we can obtain the nonlinearity in the discrete domain by using equation (7), however the ReLU operation in the continuous domain may lead to Aliasing problems, so the authors use a low-pass filter to solve this problem. The expressions for the nonlinearity in the discrete and continuous domains are

$$F_\sigma(Z) = s^2 \cdot III_s \odot (\emptyset_s * \sigma(\emptyset_s * Z)) \qquad f_\sigma(z) = \psi_s * \sigma(z) \tag{12}$$

Based on the above theoretical formulations for the convolutional, upsampling/downsampling and nonlinearity layers, the network structure of StyleGAN3 is illustrated below:

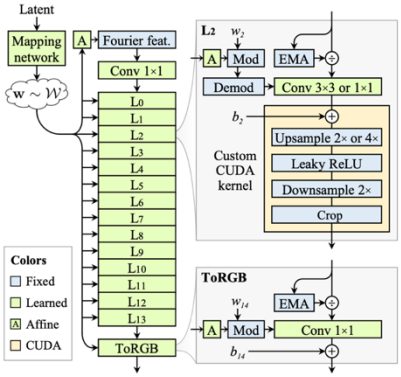


Figure 6: The StyleGAN3 architecture (Karras et al., 2021)

In contrast to the network structure of StyleGAN2, StyleGAN3 firstly transforms the input constants into Fourier transformations to ensure accurate translation of z Secondly, the noise input is removed to eliminate the introduced position reference information. Thirdly, the output skip connections are removed, which solves the problem of gradient disappearance. Also, each convolution operation is normalised by dividing by the EMA (Exponential Moving Average) before it. Fourth, the 2x upsampling filter in the network structure CUDA kernel is replaced with an ideal low-pass filter. Fifth, a fixed margin is maintained around the target canvas and then cropped to this canvas size after each layer. Finally, the Leaky ReLU is wrapped in the middle of the upsampling and downsampling layers.

3.2 Implementation of three different GANs

3.2.1 Chinese Architecture and Landscape Dataset

To generate images of Chinese architecture and landscapes through three different GAN-based models to provide inspiration for game modelers, this database contains 17,000 images from Flickr, Google Images and Baidu Images. The database includes photos of attractions, landscapes, and cities from all major provinces in China. The images are all in JPEG format, and the original resolution varies in size. In order to train the model more efficiently, all images are resized to 64*64.

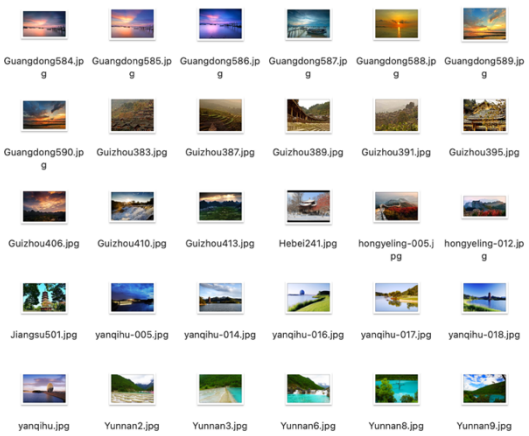


Figure 7: Chinese landscape and architecture dataset

3.2.2 Implementation of Generative adversarial net

GAN, as the most basic generative adversarial network structure, will be used as a baseline against other models in this experiment. To increase the efficiency of the code, GAN will be run on Google colab pro (Google, 2022). In this implementation, only the CPU used for training the GAN. The GAN code was built using TensorFlow platform. The original database (in the format of JEPG) was first uploaded to Google Drive, and then the whole database was converted to npy format and saved according to the requirements of TensorFlow. The second step was to build the generator model and the discriminator model using the code, both of which were constructed exactly according to the structure of the GAN paper. After building the models and the loss function, I write a function 'save_images' to store the images on Google Drive in order to compare the images generated by each epoch. In the training function, I write a checkpoint to prevent the program from crashing suddenly and causing the problem of starting from scratch. A checkpoint is generated every 5 epochs and there will only be a maximum of 3 checkpoints. Finally, after running the training function, each epoch will output the loss values of the generator and discriminator models as well as the training time after the training is completed.

```
def build_generator(seed_size, channels):
    model = Sequential()

    model.add(Dense(4*4*256,activation="relu",input_dim=seed_size))
    model.add(Reshape((4,4,256)))

    model.add(UpSampling2D())
    model.add(Conv2D(256,kernel_size=3,padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))

    model.add(UpSampling2D())
    model.add(Conv2D(256,kernel_size=3,padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))


    model.add(UpSampling2D())
    model.add(Conv2D(128,kernel_size=3,padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))

    if GENERATE_RES>1:
      model.add(UpSampling2D(size=(GENERATE_RES,GENERATE_RES)))
      model.add(Conv2D(128,kernel_size=3,padding="same"))
      model.add(BatchNormalization(momentum=0.8))
      model.add(Activation("relu"))


    model.add(Conv2D(channels,kernel_size=3,padding="same"))
    model.add(Activation("tanh"))

    return model
```

```
def build_discriminator(image_shape):
    model = Sequential()

    model.add(Conv2D(32, kernel_size=3, strides=2, input_shape=image_shape,
                     padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(64, kernel_size=3, strides=2, padding="same"))
    model.add(ZeroPadding2D(padding=((0,1),(0,1))))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(128, kernel_size=3, strides=2, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(256, kernel_size=3, strides=1, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Conv2D(512, kernel_size=3, strides=1, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))

    return model
```

Figure 8: The code of generator and discriminator model

After the generator and discriminator structures were constructed, the final generator trainable parameters obtained were 2,043,011 and the discriminator trainable parameters were 1,613,889.

3.2.3 Implementation of StyleGAN2-ADA

In implementing the StyleGAN2-ADA model, I used colab pro to run the code either. Unlike GAN, the StyleGAN2-ADA code has been integrated by NVIDIA engineers and uploaded to the NVlabs on GitHub (Karras et al., 2020). This makes it easier for users to use the StyleGAN2-ADA model code. As the model structure needs to use both CPU and GPU, the GPU assigned in the colab is Tesla P100-PCIE-16GB. After obtaining the GPU, I used the dataset_tool.py file in the code to convert the original database into a tfrecords file for the next operation The version of code I used was based on TensorFlow 1.X. However, after August 1, 2022, colab no longer supports Tensorflow 1. In order to make the code work as usual, I reinstalled Tensorflow 1.15.2 and tensorflow-gpu1.15 and specified NumPy=1.19.4 for proper output.

The StyleGAN2-ADA code faithfully implements the model structure of the thesis, the main bodies of which are the mapping network, the synthesis network and the ToRGB. The code shows the network structure as shown in the figure below:

G	Params	OutputShape	WeightShape
latents_in	-	(?, 512)	-
labels_in	-	(?, 0)	-
epochs	1	()	()
epochs_1	1	()	()
G_mapping/Normalize	-	(?, 512)	-
G_mapping/Dense0	262656	(?, 512)	(512, 512)
G_mapping/Dense1	262656	(?, 512)	(512, 512)
G_mapping/Broadcast	-	(?, 10, 512)	-
dlatent_avg	-	(512,)	-
Truncation/Lerp	-	(?, 10, 512)	-
G_synthesis/4x4/Const	8192	(?, 512, 4, 4)	(1, 512, 4, 4)
G_synthesis/4x4/Conv	2622465	(?, 512, 4, 4)	(3, 3, 512, 512)
G_synthesis/4x4/ToRGB	264195	(?, 3, 4, 4)	(1, 1, 512, 3)
G_synthesis/8x8/Conv0_up	2622465	(?, 512, 8, 8)	(3, 3, 512, 512)
G_synthesis/8x8/Conv1	2622465	(?, 512, 8, 8)	(3, 3, 512, 512)
G_synthesis/8x8/Upsample	-	(?, 3, 8, 8)	-
G_synthesis/8x8/ToRGB	264195	(?, 3, 8, 8)	(1, 1, 512, 3)
G_synthesis/16x16/Conv0_up	2622465	(?, 512, 16, 16)	(3, 3, 512, 512)
G_synthesis/16x16/Conv1	2622465	(?, 512, 16, 16)	(3, 3, 512, 512)
G_synthesis/16x16/Upsample	-	(?, 3, 16, 16)	-
G_synthesis/16x16/ToRGB	264195	(?, 3, 16, 16)	(1, 1, 512, 3)
G_synthesis/32x32/Conv0_up	2622465	(?, 512, 32, 32)	(3, 3, 512, 512)
G_synthesis/32x32/Conv1	2622465	(?, 512, 32, 32)	(3, 3, 512, 512)
G_synthesis/32x32/Upsample	-	(?, 3, 32, 32)	-
G_synthesis/32x32/ToRGB	264195	(?, 3, 32, 32)	(1, 1, 512, 3)
G_synthesis/64x64/Conv0_up	1442561	(?, 256, 64, 64)	(3, 3, 512, 256)
G_synthesis/64x64/Conv1	721409	(?, 256, 64, 64)	(3, 3, 256, 256)
G_synthesis/64x64/Upsample	-	(?, 3, 64, 64)	-
G_synthesis/64x64/ToRGB	132099	(?, 3, 64, 64)	(1, 1, 256, 3)

D	Params	OutputShape	WeightShape
images_in	-	(?, 3, 64, 64)	-
labels_in	-	(?, 0)	-
64x64/FromRGB	1024	(?, 256, 64, 64)	(1, 1, 3, 256)
64x64/Conv0	590080	(?, 256, 64, 64)	(3, 3, 256, 256)
64x64/Conv1_down	1180160	(?, 512, 32, 32)	(3, 3, 256, 512)
64x64/Skip	131072	(?, 512, 32, 32)	(1, 1, 256, 512)
32x32/Conv0	2359808	(?, 512, 32, 32)	(3, 3, 512, 512)
32x32/Conv1_down	2359808	(?, 512, 16, 16)	(3, 3, 512, 512)
32x32/Skip	262144	(?, 512, 16, 16)	(1, 1, 512, 512)
16x16/Conv0	2359808	(?, 512, 16, 16)	(3, 3, 512, 512)
16x16/Conv1_down	2359808	(?, 512, 8, 8)	(3, 3, 512, 512)
16x16/Skip	262144	(?, 512, 8, 8)	(1, 1, 512, 512)
8x8/Conv0	2359808	(?, 512, 8, 8)	(3, 3, 512, 512)
8x8/Conv1_down	2359808	(?, 512, 4, 4)	(3, 3, 512, 512)
8x8/Skip	262144	(?, 512, 4, 4)	(1, 1, 512, 512)
4x4/MinibatchStddev	-	(?, 513, 4, 4)	-
4x4/Conv	2364416	(?, 512, 4, 4)	(3, 3, 513, 512)
4x4/Dense0	4194816	(?, 512)	(8192, 512)
Output	513	(?, 1)	(512, 1)

Figure 9: Generator and Discriminator of StyleGAN2-ADA

In StyleGAN2-ADA, the trainable parameters for the generator are 22, 243, 610, and the trainable parameters for the discriminator are 23, 407, 361. Before model training can begin, the train.py file needs to be run and the hyperparameters set artificially. Among the StyleGAN2-ADA hyperparameters are:

Settings	Explanation
--outdir	The file path to save result
--gpus	How many GPUs used in training
--snaps	Number of ticks will have an image and network snapshot
--seed	Random seed
--data	The path of training dataset
--res	The resolution of dataset images
--mirror	Augment dataset with x-flips
--mirrory	Augment dataset with y-flips
--use-raw	Use raw image dataset
--cfg	Base configuration
--lrate	Learning rate
--ttur	Use two time-scale update rules
-gamma	R1 gamma
--nkimg	The number of starting counts
--kimg	Training duration
--aug	Augmentation mode (default: ada)
--p	Augmentation probability (default: 0.6)
--target	Override ADA target for --aug=ada and --aug=adarv

--augpipe	Augmentation pipeline
--resume	Resume from network pickle

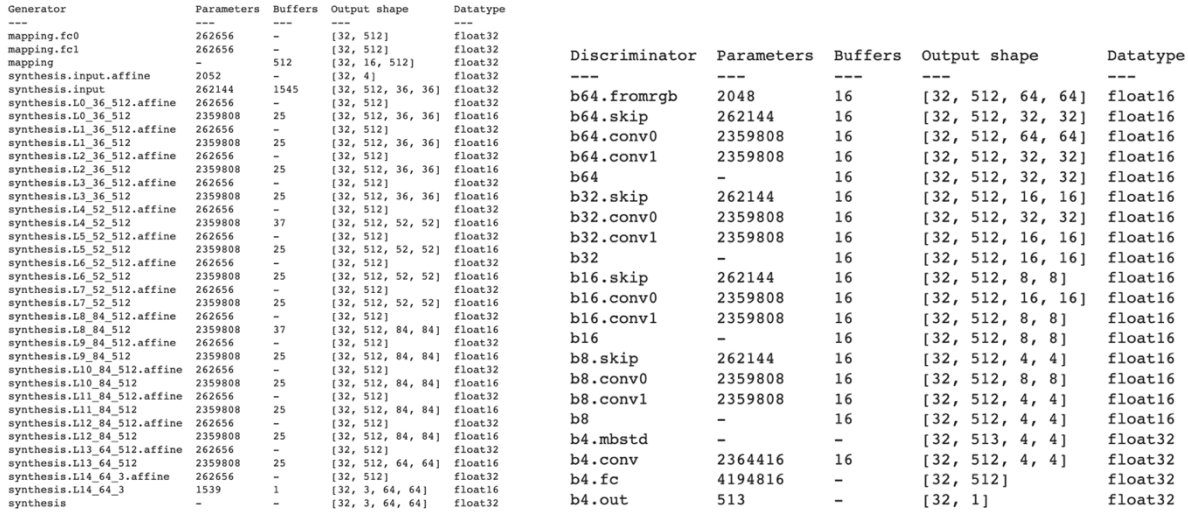
Table1: Hyperparameters of StyleGAN2-ADA

After setting the above hyperparameters, the entire network of the model is fully generated and ready for training.

3.2.4 Implementation of StyleGAN3

In terms of generating the StyleGAN3 model, it is more similar to the StyleGAN2-ADA approach. Its code has also been put into NVlabs. The difference is that StyleGAN3 is written using the Pytorch library. Model training is still run on both the GPU and CPU. The GPU used to train StyleGAN3 is a Tesla P100-PCIE-16GB, and the original dataset can be stored in Google Drive as a zip file after using the Pytorch library with dataset_tool.py, which saves space in the storage database. Since the model code uploaded by the author a year ago did not match the current Pytorch library, I reinstalled the Pytorch version with the following version information: torch==1.10.0+cu113 torchvision==0.11.1+cu113 torchaudio===0.10.0+cu113. Meanwhile, for proper output of log, training etc. files. I have set numpy to version 1.19.4 and TensorFlow to version 1.15.2.

The code of StyleGAN3 reproduces exactly the theoretical structure of the paper, and the network structure of its code output is shown in the following figure:



Generator	Parameters	Buffers	Output shape	Datatype
mapping.fc0	262656	-	[32, 512]	float32
mapping.fc1	262656	-	[32, 512]	float32
mapping	-	512	[32, 16, 512]	float32
synthesis.input.affine	2052	-	[32, 4]	float32
synthesis.input	262144	1545	[32, 512, 36, 36]	float32
synthesis.L0_36_512.affine	262656	-	[32, 512]	float32
synthesis.L0_36_512	2359808	25	[32, 512, 36, 36]	float16
synthesis.L1_36_512.affine	262656	-	[32, 512]	float32
synthesis.L1_36_512	2359808	25	[32, 512, 36, 36]	float16
synthesis.L2_36_512.affine	262656	-	[32, 512]	float32
synthesis.L2_36_512	2359808	25	[32, 512, 36, 36]	float16
synthesis.L3_36_512.affine	262656	-	[32, 512]	float32
synthesis.L3_36_512	2359808	25	[32, 512, 36, 36]	float16
synthesis.L4_52_512.affine	262656	-	[32, 512]	float32
synthesis.L4_52_512	2359808	37	[32, 512, 52, 52]	float16
synthesis.L5_52_512.affine	262656	-	[32, 512]	float32
synthesis.L5_52_512	2359808	25	[32, 512, 52, 52]	float16
synthesis.L6_52_512.affine	262656	-	[32, 512]	float32
synthesis.L6_52_512	2359808	25	[32, 512, 52, 52]	float16
synthesis.L7_52_512.affine	262656	-	[32, 512]	float32
synthesis.L7_52_512	2359808	25	[32, 512, 52, 52]	float16
synthesis.L8_84_512.affine	262656	-	[32, 512]	float32
synthesis.L8_84_512	2359808	37	[32, 512, 84, 84]	float16
synthesis.L9_84_512.affine	262656	-	[32, 512]	float32
synthesis.L9_84_512	2359808	25	[32, 512, 84, 84]	float16
synthesis.L10_84_512.affine	262656	-	[32, 512]	float32
synthesis.L10_84_512	2359808	25	[32, 512, 84, 84]	float16
synthesis.L11_84_512.affine	262656	-	[32, 512]	float32
synthesis.L11_84_512	2359808	25	[32, 512, 84, 84]	float16
synthesis.L12_84_512.affine	262656	-	[32, 512]	float32
synthesis.L12_84_512	2359808	25	[32, 512, 84, 84]	float16
synthesis.L13_64_512.affine	262656	-	[32, 512]	float32
synthesis.L13_64_512	2359808	25	[32, 512, 64, 64]	float16
synthesis.L14_64_3.affine	262656	-	[32, 512]	float32
synthesis.L14_64_3	1539	1	[32, 3, 64, 64]	float16
synthesis	-	-	[32, 3, 64, 64]	float32

Discriminator	Parameters	Buffers	Output shape	Datatype
b64.fromrgb	2048	16	[32, 512, 64, 64]	float16
b64.skip	262144	16	[32, 512, 32, 32]	float16
b64.conv0	2359808	16	[32, 512, 64, 64]	float16
b64.conv1	2359808	16	[32, 512, 32, 32]	float16
b64	-	16	[32, 512, 32, 32]	float16
b32.skip	262144	16	[32, 512, 16, 16]	float16
b32.conv0	2359808	16	[32, 512, 32, 32]	float16
b32.conv1	2359808	16	[32, 512, 16, 16]	float16
b32	-	16	[32, 512, 16, 16]	float16
b16.skip	262144	16	[32, 512, 8, 8]	float16
b16.conv0	2359808	16	[32, 512, 16, 16]	float16
b16.conv1	2359808	16	[32, 512, 8, 8]	float16
b16	-	16	[32, 512, 8, 8]	float16
b8.skip	262144	16	[32, 512, 4, 4]	float16
b8.conv0	2359808	16	[32, 512, 8, 8]	float16
b8.conv1	2359808	16	[32, 512, 4, 4]	float16
b8	-	16	[32, 512, 4, 4]	float16
b4.mbstd	-	-	[32, 513, 4, 4]	float32
b4.conv	2364416	16	[32, 512, 4, 4]	float32
b4.fc	4194816	-	[32, 512]	float32
b4.out	513	-	[32, 1]	float32

Figure 10: Generator and discriminator structure of StyleGAN3

In StyleGAN3, there are 37,768,199 trainable parameters for the generator and 26,488,833 trainable parameters for the discriminator. StyleGAN3 also requires hyperparameters to be set before the training of the model:

Settings	Explanation
--outdir	Where to save the results
--cfg	Base configuration
--data	Training data
--gpus	Number of GPUs to use
--batch	Batch size
--gamma	R1 regularization weight
--cond	Train conditional model
--mirror	Enable dataset x-flips
--mirrory	Enable dataset y-flips
--aug	Augmentation mode (default: ada)
--augpipe	Augmentation pipeline

--resume	Resume from given network
--freezed	Freeze first layer of discriminator
--p	Probability for augmentation
--batch-gpu	limit batch size per GPU
--cmax	Max feature maps
--glr	Generator learning rates
-dlr	Discriminator learning rates
--map-depth	Mapping network depth
--metrics	Quality metrics
--kimg	Total training duration
--tick	How often to print progress
--snap	How often to save snapshots
--seed	Random seed
--workers	Dataloader worker process
--nobench	Disable cuDNN benchmarking

Table 2: Hyperparameters of StyleGAN3

After setting the above hyperparameters, the StyleGAN3 model is ready to start training. In the next experiment design section, I will explain in detail how each model is set up with hyperparameters and compare the three models by setting different experiment times.

3.3 Experiment Design

The purpose of this experiment is to compare the performance of GAN, StyleGAN2-ADA and StyleGAN3 with the Chinese landscape and architecture database, so I

designed the experiment with one node for one day of training, one node for 3000kimg training and one node for 5000kimg training to compare the performance of the three models. Because of the relatively simple network structure, I set the hyperparameters batch_size=32, buffer_size=60000, generate_res=64*64 image_channels=3, epoch=300 in GAN.

The hyperparameters of styleGAN2-ADA are set as follows: --cfg= 11gb-gpu-complex --aug=ada --outdir=./results --snap=4 --data=./datasets/StyleGAN2-ADA_64 ---augpipe=bgc --kimg=5000 --mirror=True --mirrory=false --metrics=None --target=0.7.

The hyperparameters of StyleGAN3 are set as follows: --outdir=./results –cfg=stylegan3-t (translation equivariance) --data=./datasets/stylegan3_64.zip --gpus=1 --batch=32 --batch-gpu=32 –gamma=0.5 --mirror=True --kimg=5000 --snap=4 --metrics=None. This is because metrics take a lot of time to compute, and if they are computed once after each training session, it will take much longer to train. In the StyleGAN3 settings, I set both the batch size and batch gpu to 32, because the RAM size allocated by colab pro is about 51GB, and according to the test, the training speed is the fastest when the batch gpu=32 without exceeding the RAM limit.

3.4 Summary

At the beginning of the methodology section, the network structure of GAN is reviewed and how the key optimisation formulations are converted from cross-entropy. Secondly, this section writes about how the structure of the image augmentation part in StyleGAN2-ADA differs from its reference bCR, in addition to reviewing the method of adaptive adjustment of p-values proposed by the authors in their paper, which readily adjusts according to the values obtained from the two heuristic r_v and r_t. Moreover, this section reviews the network architecture of StyleGAN3, where the authors redesigned the convolutional, upsampling, downsampling, and nonlinearity layers of the overall network based on the StyleGAN2 network architecture in order to reduce the problems of Aliasing and images sticking. Low-pass filters, Dirac comb, Whittaker-

Shannon interpolation formula, and Fourier transform are used to transform the discrete and continuous domains. Following a review of the theory of the algorithm, a detailed description of how the three models were implemented is given. This includes the use of the colab platform for reproduction, how many hyperparameters the three models have, what they are, the number of trainable parameters, the various libraries used and version information. After reproducing the three models, section describes how the experiments were designed in order to compare the performance of the three different models in order to select the most suitable model to address the research questions in this paper.

Chapter 4 Result

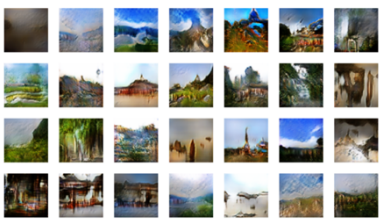
4.1 The result of experiment

After the settings in section 3.3, each of the three models was run in the colab for 24 hours. The various parameters output during the training process are the training epochs/ticks, training time for each epoch/tick, test time for generate images, augmentation and GPU memory usage as shown in the table below:

Model / Type	Original GAN	StyleGAN2-ADA	StyleGAN3
Epoch/tick	150 epochs	593 ticks	456 ticks
Training time per epoch or tick	570s (without GPU)	119.7s	179.5s
Time for generate 100 images	56s	93s	87s
Augmentation	N/A	0.04-83.091	0.02-0.73
GPU memory	N/A	8.1GB	10.84GB

Table 3: Training information of GAN, StyleGAN2-ADA and StyleGAN3

After one day of training, I used the resulting trained model for image generation. Original GAN used the 'build_generator' function, use of stored h5 format files to generate random images. StyleGAN2-ADA and styleGAN3 run generate.py and gen_images.py respectively to generate the images generated by the corresponding models. The result of the generated image is shown below:



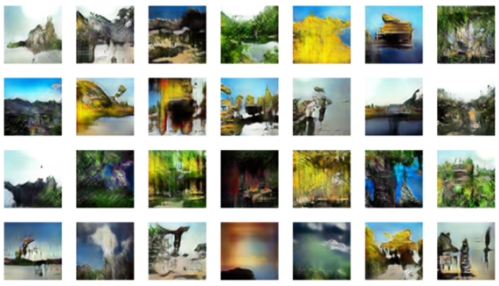
(a)

(b)

(c)

Figure 11: The generate images of GAN (without GPU) (a) and StyleGAN2-ADA (b) and StyleGAN3 (c) in one day

After a day of training, we continued with the three models. When the training volume reached 3000kimg we again performed the image output for the three trained models.



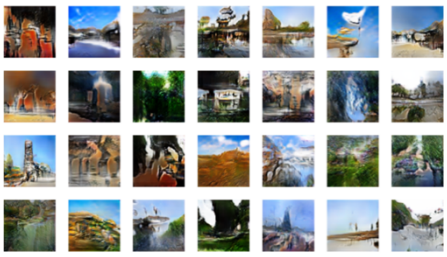
GAN

StyleGAN2-ADA

StyleGAN3

Figure 12: The Generate images of three models after 3000kimg training

Eventually, the experiment proceeded to the final step. In the 5000kimg training, I not only used the images generated by the completed models, but also further confirm the quality of the images generated, I performed the FID (Fréchet inception distance) calculation on each of the three models. FID is a metric used to judgement the quality of images created by a generative model ((Heusel et al., 2018)). A smaller value of FID means that the quality of the image generated by the generator is better. The FID value of GAN, StyleGAN2-ADA and StyleGAN3 after 5000kimg training are: 182.726, 54.011, 11.508. The images generated by the three models after training are shown below:



GAN

StyleGAN2-ADA

StyleGAN3

Figure 13: The Generate images of three models after 5000kimg training

4.2 Summary

This section shows the experimental results of three different models trained for one day, 3000kimg and 5000kimg, including the number of training epochs/ticks, training time, generate image time, augmentation, GPU usage, FID and output images of the three models. In the next section, I will compare and analyse the results obtained.

Chapter 5 Analysis and discussion

5.1 Analysis GAN, StyleGAN2-ADA and StyleGAN3 by result

After obtaining various information about GAN during the training process it can be found that his training speed is relatively slow, however he is somewhat faster than the other two models in terms of image generation speed. In terms of the generated images, the difference between the three generated images in one day of training, 3000epoch, and 5000epoch was not very much and did not produce good quality images. To find out the cause of the problem, I looked at the generator loss and discriminator loss in the code output and found that the generator loss remained around 5.839 and the discriminator loss around 0.3019 in the initial one-day training. However, after 1000 epochs of training, there is a gradual increase in the generator loss output, accompanied by a gradual decrease in the discriminator loss output. This usually happens because the generator starts to diverge gradually, and the model starts to crash. Although the model did not crash completely during this training, as the training epoch increased, eventually the generator would completely diverge. based on the environment and the circumstances of this experiment, I found that the problem that caused the generator to diverge was that the database was small. The database used for this experiment was only 17,000 images, which is a small amount of data. This could lead to overfitting problems during training and a rapid crash of the generator. Secondly, before the training, the GAN set the batch size to 32, which seems to be a bit too large from the results. Because using a large batch size may affect the performance of the network, many training examples will be input to the discriminator, which may overwhelm the generator and have a negative impact on the model training. Since the generator gradually collapses after a high number of epochs, it could be found that the generator works best when the epoch is around 1000, which is when the GAN model is best trained using the Chinese landscape and architecture database, based on the loss path of the generator and the discriminator. However, the quality of the images generated using the model at this time is still unsatisfactory.

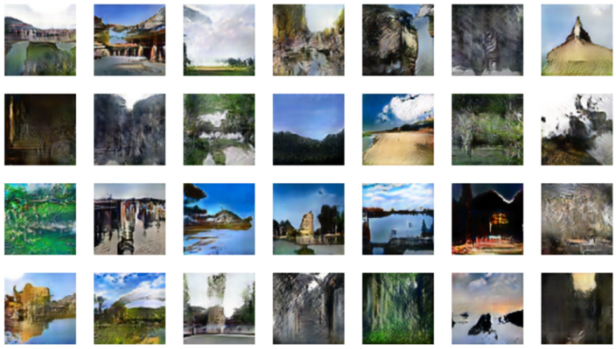


Figure 14: The images generated by GAN after 1000 epochs

StyleGAN2-ADA can be trained almost 5 times faster than a CPU-only GAN thanks to the addition of a GPU. However, due to the complexity of the network, it is a little slower when generating images. Since StyleGAN2-ADA uses adaptive image augmentation, the augmentation increases gradually as the training tick increases (because of the small amount of data, the r_t diverges relative to 0 at the beginning of training, and the augmentation increases gradually to ensure that the model does not overfitting. In terms of the generated images, it could be seen that after one day of training, the generated images have a lot of repetitive clouds, cut-off peaks and distorted buildings. This is clearly not the image quality we expected. After 3000kimg training the quality of the graphics improved significantly. However, the problem of image fragmentation and repetition show again in the generated images after 5000kimg. Moreover, the FID of StyleGAN2-ADA reached 54.011 after 5000kimg training, which is different from the FID of around 5 proposed by StyleGAN2-ADA in the paper. Based on the above it can be obtained that the model has reached the optimal point before 5000kimg. Therefore, I calculated the FID of a model with a training volume between 3000kimg and 5000kimg (3424kimg) which the FID is: 20.326 and also calculated one model's FID before 3000kimg (2424kimg) training volume which the FID is 24.754. Comparing the FIDs at 5000kimg, the model after 3424kimg train is more than twice as small as 5000kimg.It is clearly found that the model reaches its optimal point around 3000kimg and then collapse gradually.

Figure 15: The image generated by StyleGAN2-ADA after 3424kimg train

StyleGAN3 is the most complex of the three models, so it takes longer to train than StyleGAN2-ADA, which also uses GPUs, and uses the most GPUs. However, its image generation time is faster than that of StyleGAN2-ADA. In terms of images, the quality of the landscape images obtained by StyleGAN3 after one day of training time is much better than that of the buildings, which still exhibit rotation and are partially missing. After a training volume of 3000kimg these problems gradually disappear and the images look very realistic, with only less images generated that may show distortion of the buildings. After 5000kimg training, StyleGAN3 output the best quality of images with FID=11.508. StyleGAN3 is also the only one of the three models where the generator does not crash after 5000kimg of training.

5.2 Comparing the strength and drawbacks of the three models

The network structure of GAN is the simplest and easiest to reproduce among the three models, using only Python and the TensorFlow library, which also means that the layer structure of the GAN model is the easiest to tune. However, the simple model structure results in the least trainable parameters of the three models, and the GAN model is difficult to converge. In this experiment, the model did not converge well, and the discriminator loss was only around 0.3 at best. Overfitting due to less data could also be a cause of this situation. This is still a long way from the 0.5 discriminator (which is unable to determine whether an image is true or false) that we want to achieve. Therefore, the quality of the images generated by the trained model is not very

satisfactory.

StyleGAN2-ADA has the fastest training speed among the three models, and still produces good quality images (FID=20.326) when the data volume is small, and the number of training ticks is low. However, the generator model gradually diverges after reaching the optimum in the model training. The generator diverging is not as obvious as it could have been because the metrics had not been set to True in the hyperparameter settings (In order to save the training time). Furthermore, it was found that the StyleGAN2-ADA model still suffers from image blurring due to image sticking problem during training from the beginning to the 5000kimg generated images.

StyleGAN3 has the most trainable parameters and hyperparameters of the three models due to the redesign of some of the layers in the model. As a result, it generates the best quality images at the end of training (FID=11.508) and is the only one of the three models whose generator does not crash after 5000kimg of training. Due to the complex structure of the model, it is slower to train than StyleGAN2-ADA, and the convergence of the model is slower, requiring a training volume of 5000kimg to achieve the best image quality, which typically takes 11-12 days to complete, depending on the speed of the P100-PCIE-16GB GPU on the colab.

The research problem of this thesis is to find a way to provide landscape and architectural modelling inspiration for game modelers. Of the three methods GAN produces poor quality images, StyleGAN3 produces the best quality images, but it takes too long to train and converges slowly, and due to its complex network structure, it takes a lot of time to train and tune the model parameters if the modeler wants different styles of landscape and architecture(Chinese to British), StyleGAN2-ADA has the shortest training time and faster convergence among the three models, reaching the optimum at around 3000kimg in the database of this experiment. In the experimental environment of this colab, it only takes 5-6 days to train, and good image quality was obtained. Generally, modelers cannot wait too long on images that provide

inspiration, and there is a high probability that different styles exist in the same game (Capcom, 2018). This could lead to models needing to be retrained. Therefore, I think StyleGAN2-ADA is the most suitable model (assistant) to inspire modelers among the three models.

5.3 Some real-world application using StyleGAN architecture

As well as generating stylistically similar images to help modelers, StyleGAN has also made good contributions in other areas. In medicine, according to Nguyen et al (Nguyen et al., 2022), some young doctors are unable to accurately identify the degree of erosion in esophagitis. In order to help doctors to make more accurate judgements, the authors used StyleGAN2-ADA image augmentation on the RGB channel to significantly enhance the recognition accuracy after performing classification and extraction of esophageal disease features. The StyleGAN model also has good results in engineering applied sciences. According to a paper by Situ et al. it is difficult to obtain good training results for automatic sewer defect detection because of the small number of samples. The authors used the StyleGAN2 and StyleGAN2-ADA models to train the classifier by generating synthetic images from a small number of sample images. Excellent recognition results were obtained after training.

5.4 Limitations

In this experiment, only three models, GAN, StyleGAN2-ADA and StyleGAN3, were used to generate landscape and building images, and the number of models used was small. This may lead to problems such as clutter and lack of clarity in classification, which may affect the training results. Moreover, this experiment does not use transfer learning, which might lead to a lower convergence rate of the model. Finally, the parameters and hyperparameters of the model were relatively fixed in this experiment and were not adjusted or compared.

5.5 Society and ethical impact

There are no ethical issues as the database of training images are all from open-source images. The trained images are all realistic images and do not infringe on the rights of any person or place. In terms of social impact, the goal of this experiment is to find a way to provide inspiration to modelers, thus for modelers the landscape and building images generated by StyleGAN2-ADA can be used to improve design efficiency.

5.6 Summary

In this section, it begins by analysing the results of GAN, StyleGAN2-ADA and StyleGAN3 and comparing the benefits and drawbacks of the three results, concluding that StyleGAN2-ADA is the most suitable as an inspirational aid to modelers. Furthermore, the section presents some real-world applications in medical and engineering applications using the StyleGAN structure and get the excellent result. Finally, this section writes about some of the shortcomings of this experiment.

Chapter 6 Conclusion and Future work.

6.1 Conclusions

In the recent years, as more and more open world games are being developed, many of the landscapes and buildings in these open worlds reference the real-world style to increase the immersion of the player. However, this style poses a significant challenge to modellers. Based on recent developments in deep learning and GAN, the research problem of this thesis is to use suitable GAN-based models to provide inspiration to game modellers for landscapes and buildings. In this experiment, Generative adversarial net, StyleGAN2-ADA and StyleGAN3 were chosen as the models for experimental comparison. In the methodology part, the structure of each of the three models is described and analysed in detail. GAN first introduced the idea of letting generators and discriminators play each other to learn automatically, which also laid the foundation for later GAN-based models. StyleGAN2-ADA builds on StyleGAN2 by proposing two heuristics for adaptively adjusting image augmentation to reduce overfitting. StyleGAN3 redesigned the convolutional layer, upsampling layer, downsampling layer and non-linear layer in order to solve the problem of image sticking. The image processing is converted from the discrete domain to the continuous domain and then back to the discrete domain when the processing is complete. Furthermore, section describe the source of the Chinese landscape and architecture database and the detailed parameters for this experiment. At the end of the methodology part, the implementation of the three models in colab, how many trainable parameters, types of hyperparameters, the setting of hyperparameters, GPU model and the design of the experimental steps are presented. At the beginning of the experiment, information was obtained on the training time, image generation time, augmentation level and GPU usage for each of the three models at three training time points (one day train, 3000kimg,5000kimg). The images generated by the models after the three training time points were also obtained. After the results were obtained, the paper analyzed the results and found that the GAN model is slow and easy to crash during the training

process and cannot be effectively converged, the quality of the generated images is not quite well and adjusting the batch size may improve the training effect. but slower than GAN. According to the calculation of FID, we can find that StyleGAN2-ADA reaches the optimum around 3000kimg training length and converges faster. The quality of the generated images is also good (FID=20.326). StyleGAN3 has the most complex model structure, so the training speed is slower than StyleGAN2-ADA, and the convergence speed is also relatively slow. However, it produces the best quality images of the three models, with FID=11.508. Based on a comparison of the three models and how well they matched the work of game modellers, StyleGAN2-ADA was ultimately considered the most suitable model to provide inspiration to game modelers

6.2 Future work

Although it was found that StyleGAN2-ADA is more suitable as an inspirational method for modelers, based on the steps and results of this experiment, it could be found that more models can be added to compare various aspects of StyleGAN2-ADA in the future to find a much better solution (For instance: StyleMapGAN (Kim et al., 2021)). Secondly, multiple sets of hyperparameters can be set in future experiments. For example, changing the batch size value, gamma value (changing gamma will have an effect on batch normalization), and target value may have a significant impact on the training results of the model (Ioffe & Szegedy, 2015). In the future model training, transfer learning and freeze discriminator layer could be used to improve the training efficiency and convergence speed of the model. In terms of databases, it is also possible to use multiple databases for training to compare cross-sectionally whether different methods perform differently in different databases. In order to provide a better service to modelers, I will consider making the training process available as a software or interface for modelers to use in the future.

Chapter 7 Reflection

In the course of this thesis, I initially searched through a large body of literature for problems that could be solved using GAN-related models in order to select a research question for the thesis. During the literature search I learnt how to quickly understand the main content of a piece of literature. Reading the abstract and introduction allowed me to quickly understand the subject matter of the paper. After identifying the research question, I read the papers GAN, StyleGAN, StyleGAN2, StyleGAN2-ADA and StyleGAN3 in detail. In order to fully understand these papers, I also searched for relevant reading material online to deepen my understanding of the various models. This has taught me that if I encounter a problem in the learning process that I cannot understand, I can supplement it by reading related knowledge, and when I read and understand more, I sometimes find that the knowledge is in fact all interlinked. I also had the most problems with reproducing the code, as the model code was slow to update and sometimes could not keep up with the latest version of the library. For example, colab no longer supports TensorFlow 1 as of August 1, but the model code only runs on TensorFlow 1, which requires adjustments to the code or the environment. In addition, the various library versions (numpy, pytorch, jaxlib) need to be adjusted to run the model code properly. There were times when I ran into unsolvable problems when tuning the code and had to try to find other ways to train around the problem. In this experiment with the StyleGAN2-ADA model code, the model code run using the pytorch library failed to load the training plugin due to the CUDA version and CUDA could not be tuned for versioning on the colab. As this could not be resolved, I ended up using the TensorFlow library to train StyleGAN2-ADA and was able to get the completed model. With the experience of this experiment, any future projects that are similar could learn from the experience of this project in terms of information gathering and code replication. Moreover, I think the project need to be more logically rigorous.

References

Ali, A. K., & Lee, O. J. (2021). Facade Style Mixing using Artificial Intelligence for Urban Infill. *Preprints*. https://doi.org/10.20944/preprints202104.0690.v1

Baidu. (2022). *Baidu Images*. Baidu Images. https://image.baidu.com/search/index?tn=baiduimage&ct=201326592&lm=-1&cl=2&ie=gb18030&word=%D6%D0%B9%FA%B7%E7%BE%B0%D5%D5&fr=ala&ala=1&alatpl=normal&pos=0&dyTabStr=MCwzLDEsNiw0LDUsMiw3LDgsOQ%3D%3D

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, *2*(1), 1–127. https://doi.org/10.1561/2200000006

Bermano, A. H., Gal, R., Alaluf, Y., Mokady, R., Nitzan, Y., Tov, O., Patashnik, O., & Cohen-Or, D. (2022). State-of-the-Art in the Architecture, Methods and Applications of StyleGAN. *Computer Graphics Forum*, *41*(2), 591–611. https://doi.org/10.1111/cgf.14503

Capcom. (2018, January 16). *MONSTER HUNTER: WORLD*. Monsterhunterworld-Ecosystem. http://www.monsterhunterworld.com/us/

de Boer, P.-T., Kroese, D. P., Mannor, S., & Rubinstein, R. Y. (2005). A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, *134*(1), 19–67. https://doi.org/10.1007/s10479-005-5724-z

Flickr. (2022). *Flickr Search — "Chinese%20Landscape."* Flickr. https://www.flickr.com/search/?text=Chinese%20Landscape

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative Adversarial Nets*. https://arxiv.org/pdf/1406.2661.pdf

Google. (2022a). *chinese landscape - Google Search*. Google. https://www.google.com/search?q=chinese+landscape&rlz=1C5CHFA_enGB978GB978&sxsrf=ALiCzsa8cnW_lZaPR6GNLZ5k4E9Nzb7_Cg:16622369496

94&source=lnms&tbm=isch&sa=X&ved=2ahUKEwi35KyXu_n5AhXhnFwKHQjyAswQ_AUoAnoECAEQBA&biw=1792&bih=940&dpr=2

Google. (2022b). *Google Colaboratory*. Google. https://colab.research.google.com

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2018). GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *Advances in Neural Information Processing Systems*, *30*. https://arxiv.org/abs/1706.08500

Hoyoverse. (2020). *Genshin Impact – Step Into a Vast Magical World of Adventure*. Genshin.hoyoverse.com. https://genshin.hoyoverse.com/en/home?utm_source=HoYoverse&utm_medium=products

Huang, X., & Belongie, S. (2017). Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. *CoRR*. https://arxiv.org/abs/1703.06868

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *In International Conference on Machine Learning*. https://arxiv.org/abs/1502.03167

Karnewar, A., & Wang, O. (2020). MSG-GAN: Multi-Scale Gradients for Generative Adversarial Networks. *The IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7799–7808. https://arxiv.org/abs/1903.06048

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. ArXiv.org. https://arxiv.org/abs/1710.10196

Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., & Aila, T. (2020a). *NVlabs/stylegan2-ada*. GitHub. https://github.com/NVlabs/stylegan2-ada

Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., & Aila, T. (2020b). Training Generative Adversarial Networks with Limited Data. *ArXiv:2006.06676 [Cs, Stat]*. https://arxiv.org/abs/2006.06676

Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J., & Aila, T. (2021a). *NVlabs/stylegan3*. GitHub. https://github.com/NVlabs/stylegan3

Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J., & Aila, T. (2021b). Alias-Free Generative Adversarial Networks. *ArXiv:2106.12423 [Cs, Stat]*. https://arxiv.org/abs/2106.12423

Karras, T., Laine, S., & Aila, T. (2019). A Style-Based Generator Architecture for Generative Adversarial Networks. *ArXiv:1812.04948 [Cs, Stat]*. https://arxiv.org/abs/1812.04948v3

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2020). Analyzing and Improving the Image Quality of StyleGAN. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8110–8119. https://arxiv.org/abs/1912.04958v2

Kim, H., Choi, Y., Kim, J., Yoo, S., & Uh, Y. (2021, June 22). Exploiting Spatial Dimensions of Latent in GAN for Real-time Image Editing. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. https://arxiv.org/abs/2104.14754

Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. ArXiv.org. https://arxiv.org/abs/1412.6980

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT*, *16*(2), 146–160. https://doi.org/10.1007/bf01931367

Mo, S., Cho, M., & Shin, J. (2020). Freeze the Discriminator: a Simple Baseline for Fine-Tuning GANs. *CoRR*. https://arxiv.org/abs/2002.10964

Nguyen, P.-T., Tran, T.-H., Dao, V.-H., & Vu, H. (2022). Improving Gastroesophageal Reflux Diseases Classification Diagnosis from Endoscopic Images Using StyleGAN2-ADA. *Artificial Intelligence in Data and Big Data Processing*, 381–393. https://doi.org/10.1007/978-3-030-97610-1_30

Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., Shyu, M.-L., Chen, S.-C., & Iyengar, S. S. (2018). A Survey on Deep Learning. *ACM Computing Surveys*, *51*(5), 1–36. https://doi.org/10.1145/3234150

Sbai, O., Elhoseiny, M., Bordes, A., LeCun, Y., & Couprie, C. (2018). DeSIGN: Design Inspiration from Generative Networks. *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*.

Shannon, C. E. (1949). Communication in the Presence of Noise. *Proceedings of the IRE*, *37*(1), 10–21. https://doi.org/10.1109/jrproc.1949.232969

Shen, Y., Yang, C., Tang, X., & Zhou, B. (2022). InterFaceGAN: Interpreting the Disentangled Face Representation Learned by GANs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(4), 2004–2018. https://doi.org/10.1109/TPAMI.2020.3034267

Situ, Z., Teng, S., Liu, H., Luo, J., & Zhou, Q. (2021). Automated Sewer Defects Detection Using Style-Based Generative Adversarial Networks and Fine-Tuned Well-Known CNN Classifier. *IEEE Access*, *9*, 59498–59507. https://doi.org/10.1109/access.2021.3073915

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. https://jmlr.org/papers/v15/srivastava14a.html

Wang, Y., Gonzalez-Garcia, A., Berga, D., Herranz, L., Khan, F. S., & Weijer, J. van de. (2020). MineGAN: Effective Knowledge Transfer From GANs to Target Domains With Few Images. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9332–9341. https://openaccess.thecvf.com/content_CVPR_2020/html/Wang_MineGAN_Effective_Knowledge_Transfer_From_GANs_to_Target_Domains_With_CVPR_2020_paper.html

Wang, Y., Wu, C., Herranz, L., van de Weijer, J., Gonzalez-Garcia, A., & Raducanu, B. (2018). Transferring GANs: generating images from limited data. *In Proceedings of the European Conference on Computer Vision (ECCV)*, 218–234.

https://openaccess.thecvf.com/content_ECCV_2018/html/yaxing_wang_Transferring_GANs_generating_ECCV_2018_paper.html

Zhang, D., & Khoreva, A. (2018, December 21). PA-GAN: Improving GAN Training by Progressive Augmentation. *Proc. NeurIPS*.

Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., & Metaxas, D. N. (2017). StackGAN: Text to Photo-Realistic Image Synthesis With Stacked Generative Adversarial Networks. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 5907–5915. https://openaccess.thecvf.com/content_iccv_2017/html/Zhang_StackGAN_Text_to_ICCV_2017_paper.html

Zhao, Z., Singh, S., Lee, H., Zhang, Z., Odena, A., & Zhang, H. (2021). Improved Consistency Regularization for GANs. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(12), 11033–11041. https://ojs.aaai.org/index.php/AAAI/article/view/17317

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. ArXiv.org. https://arxiv.org/abs/1703.10593