Generation of synthesis Anime faces using deep learning architecture

submitted by

Asis Rai

for the degree of MSc in Data Science

of the

University of Bath

October 2020

Abstract

Animation Industry is facing a huge surge in demand because of its rapid growth over the years. This has been a huge problem in the creative process where animators of anime industry in particular, are finding it difficult to create new characters in a short amount of time, specially with female characters because their faces show more emotions than their male counterparts. The problem is that Anime is almost entirely drawn by hand and one scene can take 5-15 days to make, with the huge demand surge, this has been a growing problem for the animators in anime industry. The problem domain appears well-suited to Generative Adversarial Networks. GANs are deep learning algorithms that utilises Artificial Neural Networks for image generation. Recent advancements in Generative Adversarial Networks have allowed to generate state of the art fake synthesis human faces.

However, Generative Adversarial Networks have only been explored in animation domain. Few attempts have been made with older Generative Adversarial Networks algorithms to generate anime faces but those attempts have been futile because of reasons such as limited dataset, lazy training methods, and lazy implementations etc. that revealed huge gap in the related literature. Therefore, most improved and stable Generative Adversarial Networks algorithm was implemented, A Style-Based Generator Architecture for Generative Adversarial Networks (StyleGAN) for anime faces generation.

Extensive work was done to create a unique custom dataset of 217,800 anime face images from Danbooru2019 dataset, using two open source licensed applications. lbpcascadeanime face is used to crop anime faces from the dataset and waifu2x to upscale all cropped faces to 512^2 resolution images.

StyleGAN was trained using the unique dataset for 47 days, experimenting though a series of hyper-parameters. The results from final trained model were indistinguishable fake synthesis anime faces. A hypothesis was created to evaluate the images generated. Rating and Preference Judgement evaluation method was found through literature and mimicked through an online questionnaire quiz to test the hypothesis and evaluate the trained model. The results of the questionnaire was very surprising where the average score was 49%, indicating failure to distinguish between real anime faces drawn by animators and anime faces generated by the trained model with StyleGAN. The results answered the research question and validated the trained model and our research.

Contents

List of Figures

List of Tables

Acknowledgements

- Many thanks to my supervisor, Ken Cameron, for providing me with short but effective feedback and creating an account for me to get access to the GPU cloud.
- Many thanks to the University of Bath, for letting me use their GPU cloud server.
- Many thanks to Tom Haines, for providing help and support during problems faced with the GPU cloud.
- Many thanks to my director of studies, Dr Marina De Vos, for understanding my personal issues because of COVID-19 and providing me with extra time to finish the project.
- Last but not least, I want to thank my family for their sacrifices that they have made for me. Specially, my mother and sister, who provided me with financial support, emotional support and always being there when i needed them. I am so grateful to have them in my life and will forever be in their debt.

Chapter 1

Introduction

1.1 Background

The Animation Industry has changed drastically because of technological advancements. Global animation industry is now one of the fastest growing industries, growing 5% annually and was reported worth $244 billion in 2015 [Mar17], estimated to rise by $26.8 billion by the end of 2022 [Wat]. This research is focused on the Anime animators of the animation domain. It is reported that Anime industry itself contributes more than $19 billion every year to the growth of the global animation industry [Mar19] and expected to reach $36.26 billion by 2025.

Anime refers to any animated show or movie that uses signature aspects of Japanese-style animation, like vibrant colors, dramatic panning, and characteristic facial expressions. Anime is classified as a niche form of entertainment which contains multitudes of genres contents produced such as From drama, action, and romance to historical fiction, horror, comedy, and more, for a wide range of viewers, to match any taste [Bon].

Figure 1-1: Some of the anime characters from different genres

Anime is almost entirely drawn by hand. One cut/one scene would have three to four animators working on it and it takes a lot of time to make [Mar19]. Each scene will need to be reviewed vigorously in order to ensure the scenes run smoothly, which can take between 5-15 days. To create one novel episode from scratch with novel characters, can takes six months to make [Nou]. The rapid growth of the industry has proved to be more intrusive in the creative process for animators when delivering novel episodes and are facing mental blocks when creating new characters for novel episodes [Gri18]. Lino DiSalvo, a renowned animation director, reported that female faces are more challenging for animators to animate, as they show more emotions than male counterparts [AMI].

In addition, due to the ongoing global spread of COVID-19, major global animation events such as Comic-con, games-con, and Cannes film festival are cancelled and moved to be digitally available for consumers [Mag20]. This has seen rapid consumer growth in digital demand in animation [Sta20], causing a new paradigm shift for the animators to work remotely from home, forcing them to adapt into a new environment and get up-to-speed quicker than before, into their inherit nature [Glo20]. It is estimated that nearly 83% of the animation workforce alone suffer from work-related stress, causing businesses to lose $300 billion yearly, 43% of absenteeism, 26% billion treatment costs and 120,00 deaths [Mil19] [Eva18]. Therefore, technologies and methods must be research and implemented, to assist the animators, who are the real driving force behind every successful animation and behind the growth of industry.

Modern deep learning algorithms such as Generative Adversarial Network(GAN) has proven to achieve to generate state of the art AI-generated synthesis fakes. Proposed by I. Goodfellow et. al [GPAM*] in 2014, Generative Adversarial Networks (GANs) are a deep learning framework which makes use of Artificial Neural Networks (ANNs) to learn the underlying distribution of a given dataset. Since then, GAN has gained a lot of popularity in various deep learning domains, due to their successes in generating state of the art synthesis images of human faces. Synthesis image generation is a task of generating new images from an existing dataset [RL].

1.2 Scope

This project religiously investigates Generative Adversarial Networks, deep learning algorithms that utilises Artificial Neural Networks for image generation and develops a model for image generation.

Literature have shown that previous works applying Generative Adversarial Networks to animation have not been able to have any success in generating images of animation domain, particularly in anime. Related literature have found a gap in previous attempts of generating anime images, have been futile because of lazy training methods, limited dataset and lazy implementation. New attempts have also not yet been made on the most improved Style based Generative Adversarial Network architecture, StyleGAN. This project achieves generation of indistinguishable fake synthesis of anime faces by using Style based Generative Adversarial Network architecture, StyleGAN.

Significant attention is devoted to the investigation into StyleGAN, to learn more into the code structure base, training requirements and dataset requirements. Useful information is extracted from the investigation, such as hyper-parameters tuning, dataset requirements, and more.

Significant time was devoted to preparing a custom anime dataset because a suitable dataset could not be found on the web. Therefore, Danbooru2019 dataset was downloaded. It large-scale anime characters database with 3.69m. Open-source software lbpcascadeanime face is used to crop anime faces from the dataset and waifu2x was used to upscale all cropped faces to 512^2 resolution images. The uniquely prepared dataset contained total of images 217,800 images and 12GB in size.

Following the completion of custom anime dataset, StyleGAN scripts are modified to match the dataset style and hyper-parameters. Various experiments with hyper-parameters tuning were conducted through trianing StyleGAN, trianing for a total of 47 days, 20 hours, and 14 minutes and 61040 iterations. The results of the final model trained showed StyleGAN is capable of generating indistinguishable fake synthesis of anime faces, its success mainly because of trianing time and uniquely made dataset.

Literature showed that the best evaluation method for Generative Adversarial Networks is Rating and Preference Judgement. Therefore, a hypothesis is created to evaluate the fake synthesis generated by StyleGAN.The Hypothesis is that users will not be able to tell the difference between artist created characters and images generated by StyleGAN. To test this hypothesis, an experiment is conducted, where user study was created to mimic the evaluation method of Rating and Preference Judgement. The user study involved an online questionnaire quiz and 26 participants that were familiar with anime domain. The questionnaire quiz tested their ability to distinguish between real images drawn by animators and images created by the StyleGAN model. The results of the user study was very surprising, where the average score of the 26 participants is 49%. The result of the survey dictated in proving the hypothesis correct and answering the research question.

All parts such as the limitations, rationale for each action, and modifications made etc. are all documented in detail in their subsequent chapters and sections.

1.3 Research question:

Can Generative Adversarial Networks, deep learning algorithms that utilises Artificial Neural Networks for image generation, be used to assist animators in creating anime characters?

Chapter 2

Literature Review

A review is conducted of relevant theory and previous works in GANs and synthesis image generation. More emphasis are provided to the deep generative adversarial network algorithms and approaches capable of generating state of the art synthesis images.

2.1 Deep leaning

The augmentation of deep learning with Artificial Neural Networks (ANNs) has led to the successful application of deep learning to generate state of the art synthesis images. The workings of ANNs will be described, before detailing several sate-of-the-art deep learning algorithms of generative adversarial networks (GANs), augmented with ANNs for synthesis image generation.

2.1.1 Deep Learning

Machine learning and deep learning are both subsets artificial intelligence. Machine learning is associated with algorithms that can change themselves without human intervention to get the desired result by learning on structured data. In a structure data, the patterns within the data are more explicitly define. For example, a structured dataset tends to have, a target value for every input and a function that best fits the dataset so that the this function can be used to test the dataset to predict the output[Fag20].

Deep learning is a subset of machine learning where algorithms function similarly to machine learning, but can learn on unstructured data, where data are convened in many levels of artificial neural networks, each providing a different interpretation of the data it conveys. In a unstructured dataset, no functions are pre-defined to predict the output and deep learning algorithms tries to resembles the data in the same way that the neural connections that exist in the human brain and makes sense/actions of the output as it happens in real time[HAR19].

2.1.2 Artificial Neural Networks (ANNs)

Artificial Neural Networks, also known as neural networks, form the base of deep learning and it is inspired by the structure of the human brain. Neural networks takes in unstructured data, train themselves to recognise patterns in the data and predict the output for a new set of similar data.

In neural networks, there are three layers. Input layers, hidden layers and output layers, they are all connected by neurons. The hidden layers are where most of the computation happens. For example, let us consider an image of a circle, which has pixel size of 28 X 28, which in total is 784 pixels.

1. Each Pixel is fed as input to each neuron in the input layer and each channels is assigned a numerical value known as weight.
2. The inputs are multiplied by their corresponding weights and the sum is sent as input to the neurons in the next layer called hidden layer.
3. Each of the neurons are associated with a numerical value called bias, which is added to the input sum and this value is passed to threshold function called the Activation Function.
4. The result of the activation function determines that if a particular neuron will be activated or not, and activated neurons transmits data to the neurons of the next layers of the hidden layers, through channels. In this manner, the data is propagated through the network and it is called Forward Propagation.
5. In the output layer, the neuron with the highest value fires and determines the output. The values are probabilities. For example, the highest value is predicted as an image of a square, however the prediction is wrong as the output desired is an image of a circle.
6. The networks knows that the prediction is wrong through training. During training, along with the input, the network also has the inputs fed through it. The predicted

output is compared against the actual output to realise the error in prediction. The magnitude of the error indicates, how wrong the prediction was, the output values suggests the network how higher or lower the predicted values were, giving indication as directions and magnitude of change, to reduce the error. This information is transferred backward through the network layers, known as Backpropagation.

7. Based on the information received, the weights are adjusted and the cycle of Forward Propagation and Backpropagation keeps iterating with multiple inputs. This process continues until the weights are assigned, such as the network can predict the shape of the image correctly to the desired circle shape.

The entire training process can take a day to months, depending on the complexity of the datasets. However, time is a reasonable trade off, compared to its capabilities and adopted by machine learning algorithms, to work with un-structured data with more capabilities than other machine learning algorithms, making the adopted algorithms, deep learning algorithms.

2.2 Deep learning algorithms with Artificial Neural Networks

2.2.1 Generative Adversarial Networks

Proposed by I. Goodfellow et. al [GPAM*], Generative Adversarial Networks(GANs) are a deep learning framework which makes use of Artificial Neural Networks (ANNs) to learn the underlying distribution of a given dataset. In retrospect, the innovative concept of generative adversarial networks is often used to solve tasks within the field of generative modelling [TB18]. In general, such tasks can be subdivided into the following two sub-tasks:

1. Probability density function (pdf) estimation: given a dataset containing a large collection of data samples, determine the probability density function that is associated with this dataset.
2. Sample generation: given the probability density function of the dataset, generate new data samples by sampling from this distribution.

Generative Adversarial Networks are able to perform generative modelling tasks by making two agents with artificial neural networks - compete in what is called a zero-sum game: a term used within economics to represent a process in which the losses and gains of all participants are equal to each other [Rag09].

The objectives of these two agents can be considered as being opposite to each other. The first artificial neural network - usually referred to as the Generative Model G - aims at learning the underlying data distribution of the given dataset, and - from this distribution - generate new samples which are indistinguishable from those of the real dataset. On the other hand, the second artificial neural network - often referred to as the Discriminative model D - aims at detecting perturbations in these generated samples, with the goal of being able to classify samples as either being from the real dataset or been generated by the generative model G [GPAM*]. This process is visualized schematically in Figure 2-1.

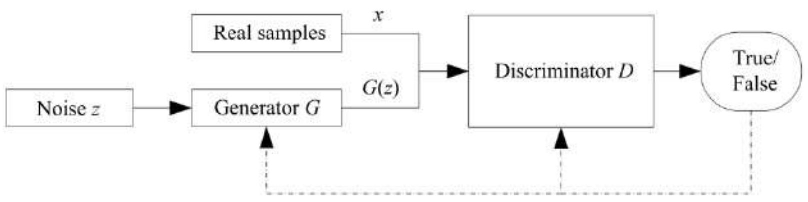


Figure 2-1: Schematic overview of the computational procedure used in GANs [TB17]

In order to train a Generative Adversarial Network, the data distributions p_z and p_g are defined. Here, p_z represents the the data distribution of the noise input, and p_g represents the generator's data distribution over the real data samples x. In essence, the Generator G acts as a mapping function $G(z; \theta_g)$, which transforms random noise z to the data space. Since the Generator G is represented by an artificial neural network, θ_g represents the network's optimizable parameters. The Discriminator D can be considered as a mapping function $D(x; \theta_d)$ which transforms a sample from the data space to a scalar value. Here, the scalar value represents the probability that the input sample x originates from the real dataset, rather than from the Generative network G.

During a GANs training process, the Discriminator aims at maximizing the probability of correctly assigning the output label (i.e., originating from the real dataset x or originating from the Generator distribution p_g) to the inputted samples. On the other hand, the Generator aims at fooling the Discriminator by creating increasingly better samples, which can be done by minimizing $log(1 - D(G(z))$. Mathematically, this allows to represent the training procedure of a generative adversarial network as a mini-max game between the Generator G and the Discriminator D as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.1}$$

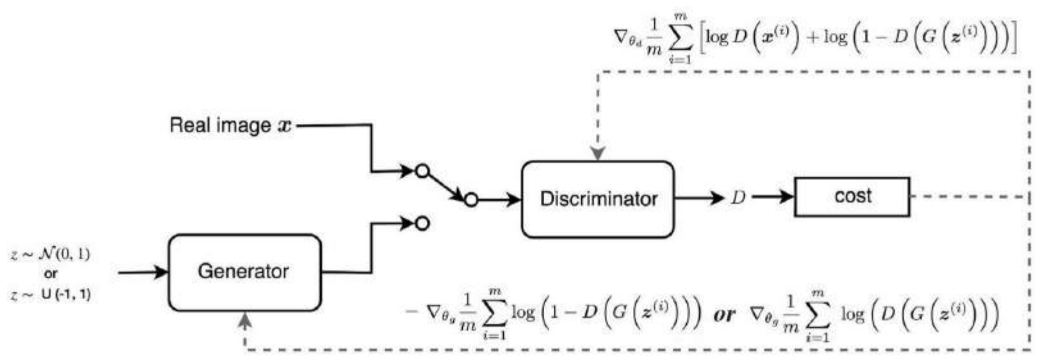

$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$
Real image x
Discriminator
D
cost
$z \sim \mathcal{N}(0, 1)$ or $z \sim U(-1, 1)$
Generator
$-\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$ or $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$

Figure 2-2: Schematic overview of how the discriminator and the generator is trained, using the corresponding gradient.[Hui18b]

In essence, this results in the loss function of a minimax GAN (also referred to as a Vanilla GAN) to be proportional to the Jensen-Shannon Divergence (JS-divergence), which is a commonly used measure for quantifying the similarity between two probability density functions. By minimizing the JS-divergence, one minimizes the difference between the probability density function of the real dataset x and the Generator distribution p_g.

2.2.2 Traditional GAN training problems

Whereas regular mini-max Generative Adversarial Networks have proven to be a powerful tool for multiple applications, their training procedure is widely considered as being unstable and highly unreliable [Bar18]. This training difficulty can be attributed to the following two problems:

1. **Vanishing Gradients Problem**: In general, optimizing the discriminator part of a GAN is considered to be relatively easy, as this comes down to training a simple binary classifier artificial neural network. This leaves one to optimize the generative part of the GAN, equaling the minimization of the JS-divergence between p_x and p_g. This can be solved as a simple gradient-descent problem whenever the distributions p_x and p_g are partly overlapping. However, when the images created by the generator part exhibit a distribution that is far from that of the real data distribution, the JS-divergence between the two distributions will increase, causing

the gradient to become zero. This results in the generator part to become untrainable, preventing the GAN from further improving its generated images [Bar18].

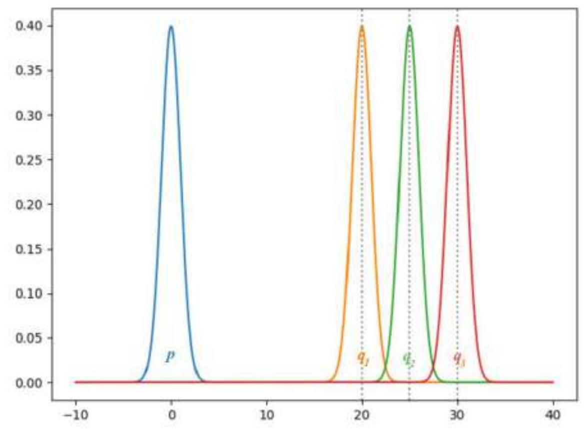


Figure 2-3: Plot example of JS-divergence JS(p,q), where data distribution q of the generator's matches with the ground truth of real images p, ranging from 0-30.[Hui18b]

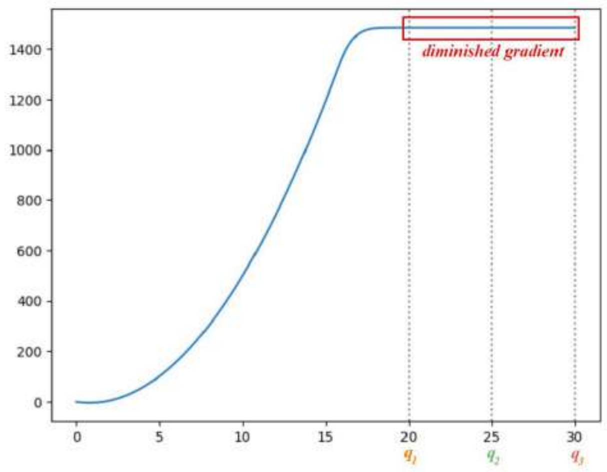


Figure 2-4: Plot example, where gradient of JS-divergence vanishes from q_1 to q_3, leading the gradient to become 0 and causing the generator to become extremely slow to untrainable.[Hui18b]

2. **Mode Collapse**: Another common problem that occurs during the training of GANs is mode collapse. Usually, the prime objective of training a GAN is to enable

it to generate a wide variety of different samples which are similar to those of the original dataset. As discussed in the previous section, it does this by training the generator network of the GAN to fool the discriminator network. However, it may happen that - in cases where the discriminator has not been trained properly yet - the generator is able generate an optimal sample $x*$ which is able to fool the discriminator, causing it to generate the same sample all the time. In this case, the best strategy of the discriminator is to learn to reject this sample (also referred to as 'mode', hence the term 'modal collapse'), causing it to over-fit. In turn, this will cause the discriminator to find the next best sample (i.e., mode) which can be used to fool the discriminator, and the process repeats itself. Each iteration, the generator and the discriminator over-fit to to the temporal weakness of the opponent, preventing the generator from learning the true underlying distribution from the data set [Bar18].

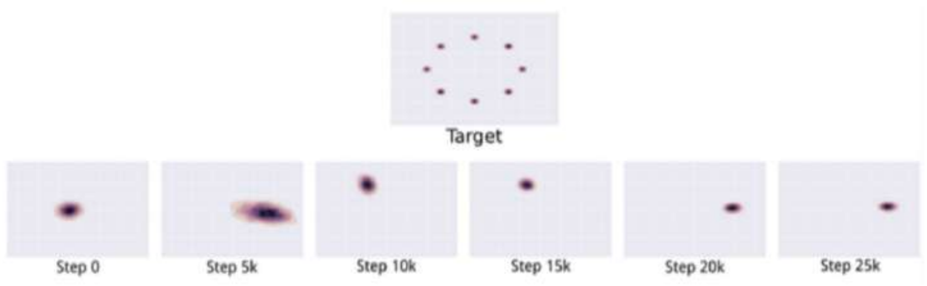


Figure 2-5: An illustration of model collapse on a toy dataset, bottom row showing how GAN is trained overtime and the generator producing only a single mode at a time cycling between different modes, as the discriminator is learning to reject each one. [MPPD17] [Bar18]

In 2016, Samuel A. Barnett proposed to alleviate regular mini-max GANs from the vanishing gradient problem was to add a continuous stream of random noise to the generator's input [Bar18]. This causes the distribution p_g to spread out, therefore increasing the probability of creating an overlap between p_g and p_x and therefore reducing the vanishing gradient problem. However, a downside of this technique is that the added noise is often visible in the newly generated samples, therefore reducing the quality of the GAN's output.

2.2.3 WGAN

In 2017, M. Arjovsky, S. Chintala, and L. Bottou proposed an innovative way in training a generative adversarial network by making use of the Wasserstein distance [ACB17]. Their proposed solution - referred to as a Wasserstein Generative Adversarial Network or WGAN - rejects the use of the JS-divergence due to the commonly occurring vanishing gradient problem. Instead, M. Arjovsky et al. propose the use of the Wasserstein distance, also referred to as the earth mover's (EM) distance [ACB17]:

$$W\left(p_x, p_g\right) = \inf_{\gamma \sim \Pi\left(p_x, p_g\right)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|] \tag{2.2}$$

The Wasserstein distance W between two probability density functions represents the cost that is associated with transforming one of the probability density functions to the shape of the other in an optimal way (i.e., optimal transport), and can therefore be interpreted as a metric for calculating the difference between two probability density functions.

To computer (Infimum) inf, all possible joint distributions in $\Pi\left(p_r, p_g\right)$ needs to be exhausted and therefore, it is not intractable. M. Arjovsky, S. Chintala, and L. Bottou proposed a new transformed formula to tackle this problem, where to measure the maximum value, inf is replaced by (Supremum) sup:

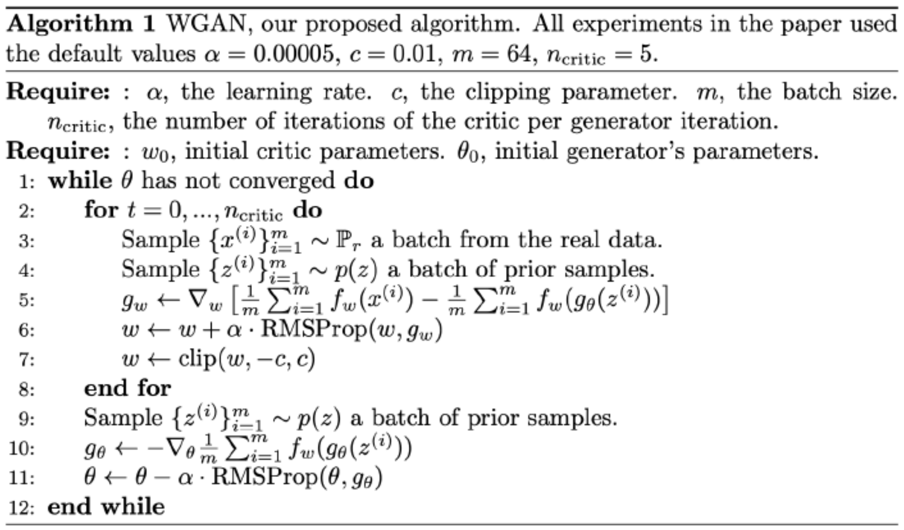
$$W\left(p_r, p_g\right) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x\sim p_r}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)] \tag{2.3}$$

To satisfy $\|f\|_L \leq K$, new function Wasserstein metric f is added, which comes from K-Lipschitz family $\{f_w\}_{w\in W}$, parameterized by w. To find optimal f_w, w is used by the discriminator model. The Wasserstein Distance w between p_r and p_g is measured by the the loss function L, now the formula can be written as:

$$L\left(p_r, p_g\right) = W\left(p_r, p_g\right) = \max_{w\in W} \mathbb{E}_{x\sim p_r}\left[f_w(x)\right] - \mathbb{E}_{z\sim p_r(z)}\left[f_w\left(g_\theta(z)\right)\right] \tag{2.4}$$

The discriminator now behaves differently, compared to the original GAN algorithm. The discriminator now trains on K function to compute the W Wasserstein distance in the training process and the loss function L eventually, begins to get smaller and the output of the generator moves closer to the real data distribution [Lil19].

Using the Wasserstein distance as a basis for the GAN's cost function instead of the JS-divergence has multiple benefits. First of all, compared to the JS-divergence, the Wasserstein distance has a smooth gradient. This reduces the vanishing gradient problem, allowing the generator to keep learning regardless of its current performance. In



Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

```
Require: : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size.
    $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Require: : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
 1: while $\theta$ has not converged do
 2:     for $t = 0, ..., n_{\text{critic}}$ do
 3:         Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
 4:         Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
 5:         $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
 6:         $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
 7:         $w \leftarrow \text{clip}(w, -c, c)$
 8:     end for
 9:     Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
10:     $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
11:     $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: end while
```

Figure 2-6: Wasserstein Generative Adversarial Network's Algorithm [ACB17].

addition, due to the smooth gradient of the Wasserstein distance, training a Wasserstein GAN does not require one to add additional noise to the generator's input. This results in newly generated images that are free from noise or perturbations, which is not the case for the mini-max GAN.

However, Wasserstein GAN still has it's bottlenecks. M. Arjovsky, S. Chintala, and L. Bottou highlighted "Weight clipping is a clearly terrible way to enforce a K-Lipschitz constraint" [ACB17]. WGAN suffers in unstable training when clipping window becomes large(weight clipping), resulting in slow convergence and also suffers same symptom as minimax GAN of vanishing gradients, when the clipping window becomes too small [Lil19].

2.2.4 DCGAN

Alec Radford, Luke Metz and Soumith Chintala proposed a new class of convolutional neural network (CNNs) called Convolutional generative adversarial networks (DCGAN) on top of GANs topology architecture to improve the training stability in most settings. In 2015, supervised learning with convolutional neural networks (CNNs) were seeing many adoptions, this was seen as major motivation to the authors. To bridge the gap between CNNs and GANs, the authors adopted and modified three family architectures of CNNs into DCGAN. [RMC15].

2.2.5 Convolutional Neural Network (CNNs)

Convolutional neural networks (CNNs) consist of different layers that are connected to each other. These layers do not consist of fully connected neurons and are made up of different filters that are applied to the output from the previous layer [Iva17]. In essence, these filters are a set of cube-shaped weights which are applied to the image with the purpose of learning the features within it. The different layers within a convolutional neural network are constituted out of different filters, each with their own specific purpose. The most common layers that are used within convolutional neural networks are the convolutional layer (hence the name convolutional neural network), the max-pooling layer and the fully connected layer and are usually considered as being the building blocks of CNNs [DLKS18].

1. **Convolutional Layers**

   The convolutional layer consists of a set of cube-shaped filters which are convolved with the input data (or the output data from the previous layer when the convolutional layer is situated deeper in the network). Each of these filters has a small width and height (i.e., common heights are 3, 5 or 7 pixels) and a depth that equals the depth of the input image [Wu17]. This means that the depth of the convolutional filter is equal to 1 when dealing with black-and-white images and equal to 3 when dealing with RBG (color) images. The different convolutional filters within the layer are slid over the input data and compute the dot product of the layer's input and the filter, resulting in a so-called convolved feature map. This convolutional step can be expressed by the following equation [Wu17]:

   $$y_{i^{l+1}j^{l+1}d} = \sum_{i=0}^{H} \sum_{j=0}^{W} \sum_{d^l=0}^{D} f_{i,j,d,d^l} \times x^l_{i^{l+1}+i,j^{j+1}+j,d^l} \tag{2.5}$$

   Where H, W and D represent the height, width and depth of the filter, respectively. Note that, usually, the width and height of this feature map is lower than that of the input data and depends on the filter size and the number of pixels that the filter is moved every step. However, the depth of the feature map is always equal to that of the convolutional layer.

   The objective of a convolutional layer is to extract features from the input data. CNNs are not limited to one convolutional layer and, usually, multiple convolutional layers are used within the same network [AMSS17]. This allows the network to learn increasingly more complex feature as the data propagates through the network. For example, the first convolutional layer is responsible for detecting low-level features (edges and color), whereas subsequent layer capture higher-level features such as general shapes or objects. This allows the network to gain an increasingly better understanding of the images in the data set.

2. **Pooling Layers**

   Pooling layers are periodically inserted in convolutional networks and are responsible for reducing the size of the convolved features obtained in previous layers. The main reason for doing this is to decrease the computational power required to process the data by means dimensionality reduction. In general, two types of pooling layers are used today, being max pooling layers and average pooling layers [Wu17]. Again, these layers consist of filters that are slid over the image, resulting in an output with reduced width and height and with a depth that is equal to that of the input data. The difference between max pooling layers and average pooling layers is that the output of max pooling layers depends on the maximum pixel values found during the sliding process, whereas average pooling layers make use of the average pixel values found during the sliding process [LGT16].

3. **Fully Connected Layers**

   Normally, a convolutional neural network consists of a series of convolutional layers and pooling layers which are able to detect increasingly more complex features that are hidden within the input image. However, the prime purpose of a convolutional neural network is to make it practically applicable, meaning that the detected features need to be interpreted, enabling he network to draw useful conclusions. This goal can be achieved by equipping the tail of the network with a few additional, fully connected layers [Wu17].Fully connected layers are added to the end of the network and take as input the flattened vector-representation of the convolutional and pooling layers. In essence, the fully connected layers represent a regular fully connected neural network which is trained with the purpose of classifying the object in the input image. Thus, instead of being trained on the input image directly (as would be the case in classical Multilayer Perceptrons), the network is trained on the features resulting from the different layers of the convolutional neural network, allowing it to achieve a much higher performance level [DSKA17]. The output from the fully connected layer is a one-dimensional vector which represents the probabilities of the input image belonging to a certain class. Usually, the prediction (i.e., the network's output) is the class which is associated with the highest probability within this one-dimensional probability vector. An example of a possible CNN architecture with two convolutional layers, two max-pooling layers and a fully connected layers is represented in the image below.

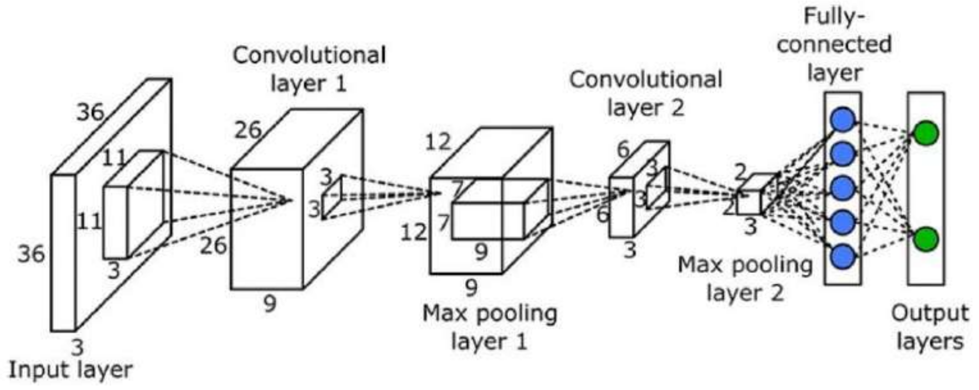


Figure 2-7: Convolutional neural network design which consists of two convolutional layers, two max-pooling layers and a fully connected layer [RCA*18]

In summary, CNNs is a major drawbacks, when it comes to detecting faces and therefore, are mainly used in image classification, localisation, semantic segmentation, and action recognition tasks. For example, when there are face oval, two eyes, a nose and a mouth, it is a normal indicator that it is a face. However, for a CNN, orientational and relative spatial relationship between the components are not important to a CNN. The main component of CNN is the convolutional layer and it learns by detecting feature components in image pixels such as edges and colour gradients. When a CNN is detecting a face, it makes prediction based on if the components of face features are present in the image pixels. If all features are displayed in the image pixels, no matter the position, then CNN classifies the image as a face [Sum18].

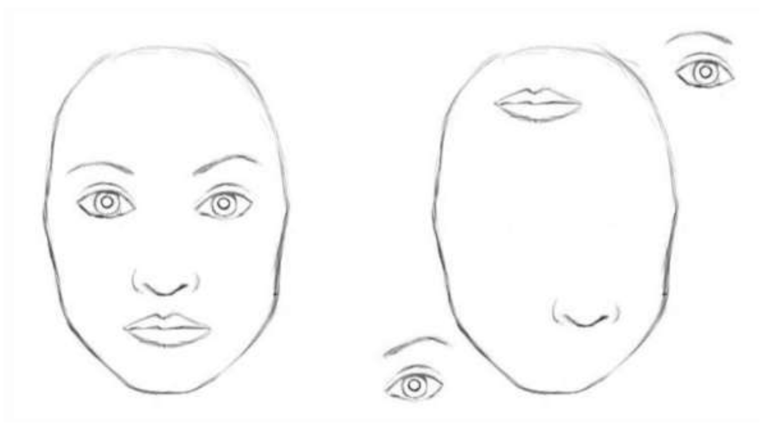
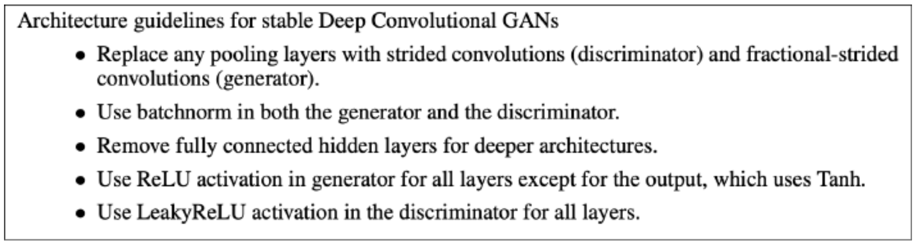


Figure 2-8: Drawback: CNN detecting both pictures as similar, since they both contain similar elements [Sum18]

2.2.6 Proposed and Modified Architecture changes in DCGAN



Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Figure 2-9: Proposed architecture guidelines for (DCGAN) Deep Convolutional GANs [RMC15].

The first proposed change is all convolutional network [SDBR15]. Instead having deterministic spatial pooling functions (max pooling and average pooling), they replaced it with strides convolutions functions of all convolutional network, to allow the architecture network to learn its own spatial down-sampling. This approach also allows to the network to learn up-sampling in the generator and the discriminator.

Second proposed changes was eliminating all fully connected layers on top of convolutional features. Instead of 2 max-pooling layers, it is replaced by global average pooling. It is the middle ground which connects maximum feature convolutional features to the generator and discriminator's input and output. This results in improvement of model stability but has a downside in convergence speed. In the first layer of a normal GAN, a uniform noise distribution Z is taken as input. This is reshaped and placed at the start of the convolutional stack and reshaped into 4-dimensional tensor. Finally, a single sigmoid output is fed into the last convolutional layer by flattening it, for discriminator to receive it's output. This is visualised below:

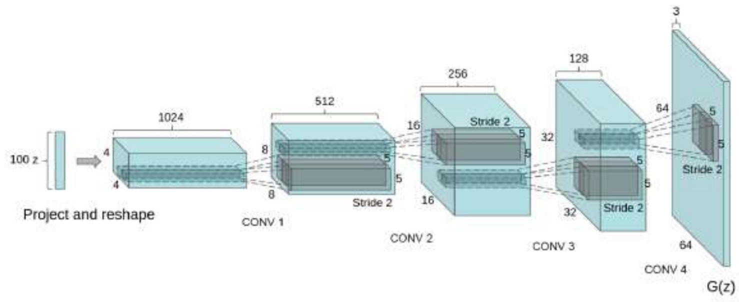


Figure 2-10: Modified generator of DCGAN with CNN's architecture [RMC15].

Third proposed idea is called batch normalization. This helps with learning stabilization by taking vector features or matrix and normalizing them, meaning that they can have the same parameter (mean) and another parameter for the standard deviation. This is similar to a standard multi-variate Gaussian and it is also how the features in the layers of the neural networks are distributed. This helps the generator to keep learning as it prevents all samples collapsing to a single point of failure and helping the gradients keep flowing when exposed to heavier models, contrary to the original GAN. The batch normalization is only applied to generator's input layer and the discriminator's output layer as applying to all layers results in sample oscillation and model instability.

The last proposed idea was to take way the input layers which are multi-layer perceptrons layer and instead use ReLU activation in generator and LeakyReLU activation in discriminator, with the exception of the output layer which has a negative slope, Tahn [-.1, 1]. The two activations allows a DCGAN model to learn quicker to saturate and cover all colour space of training distribution. Compared to the original GAN layer which uses max-out activation, the LeakyRelU activation can rectify and work better when dealing with higher resolution image modelling.

In summary, the proposed model provides stable set of architectures, reducing training problems associated with GANs and is a good representation for training images for supervised learning and generative modeling. The architecture makes use of ReLU activation function takes maximum between input value and 0. When the network produces nothing but zeros, the LeakyReLU allows the network to pass negative values to pass through, stopping the network from reaching the dying state, allowing the generator to keep receiving gradient values from the discriminator.

However, there are still some form of model instability, as if models are trained for longer then there is a chance of a subset of filters collapsing, to a single oscillating mode [RMC15]. Furthermore, the feature of one pixel is also limited to one region and this affects the representation capacity of the network as it is restrained by filter size. Therefore, the features have to dilute through layers of convolutional operations in order to connect the regions far apart and the dependencies are not guaranteed to be maintained [Wen18].

2.2.7 SAGAN

In 2018, Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena proposed a Self-Attention Generative Adversarial Network (SAGAN), which allows attention-driven, long-range dependency modelling for image generation tasks [ZGMO18]. This enables the generator and the discriminator to model relationships between spatial regions better [Wen18], compared the original GAN.

Compared to DCGAN, SAGAN is able to capture global dependencies easier because it has access to regions of different sizes and can learn the relationship of one pixel and all other positions in regions that are far apart, visualised in figure below:

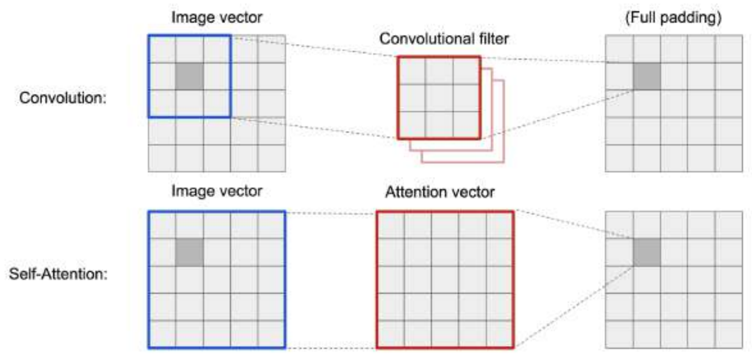


Figure 2-11: Convolution Vs Self-attention access to regions [Wen18].

SAGAN adopts the non local model of Non-local Neural Networks [WGGH17], to introduce self-attention to the GAN framework. The image features are transformed into three feature spaces f, g and h where:

$$Key \quad is \quad f(x) = W_f x, \quad Value \quad is \quad g(x) = W_g x, \quad and \quad Query \quad is \quad h(x) = W_h x \tag{2.6}$$

After the transformation, dot-product attention is applied to get the self-attention feature maps.

$$\alpha_{i,j} = softmax\left(f\left(x_i\right)^\top g\left(x_j\right)\right) \tag{2.7}$$

$$\mathbf{o}_j = \sum_{i=1}^{N} \alpha_{i,j} h\left(x_i\right) \tag{2.8}$$

W_f, W_f and W_f are all 1x1 convolutional vectors. $\otimes$ is matrix multiplication and the softmax operation is performed in each row [ZGMO18]. $\alpha_{i,j}$ is an entry in the attention map. It indicates the amount of attention the model should give when the location j is being synthesized by the i position. $\mathbf{o}_j$ is the final output the column vector of:

$$o = (o_1, o_2, \ldots, o_j, \ldots, o_N) \tag{2.9}$$

In addition, output $\mathbf{o}_j$ is further multiplied by a scale parameter and added back to the original input feature map and the final output becomes:

$$y_i = \gamma o_i + x_i \tag{2.10}$$

Where γ is learning scalar and initialized as 0 and gradually increased while training. γ is configured and allows the network to rely on the cues in the local neighbourhood because it is easier, then gradually learned to assigned more weight to the regions further away (non-local evidence). This enables to learn the easier task and then gradually increase the complexity of the task. The attention model below was applied to both generator and the discriminator, which trains in an alternating fashion by minimizing the hinge version of the adversarial loss [ZGMO18]. However, the paper fails to explain specific use of the adversarial loss function.

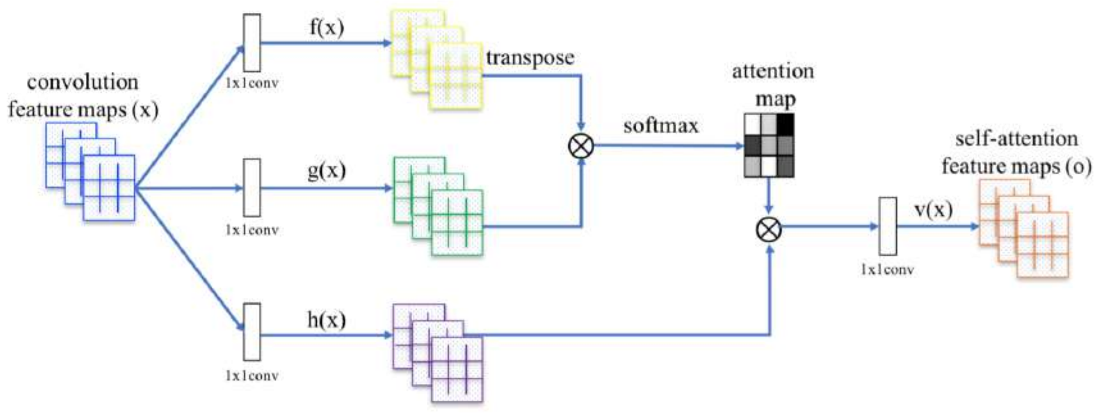


Figure 2-12: Self-attention module for the SAGAN [ZGMO18].

In essence, the paper proposed self-attention layers by applying spectral normalization to the weights of both generator and the discriminator, contrary to regular GAN which only normalizes weights. The spectral normalization value is always set to 1 to constraint the weights of the K-Lipschitz to control the gradient flow. The self-attention module is effective in modelling long-range dependencies and much more effective than DCGAN and generator training is much more stable than GANs with the use of spectral normalization and also speeds up training time for regularized discriminator.

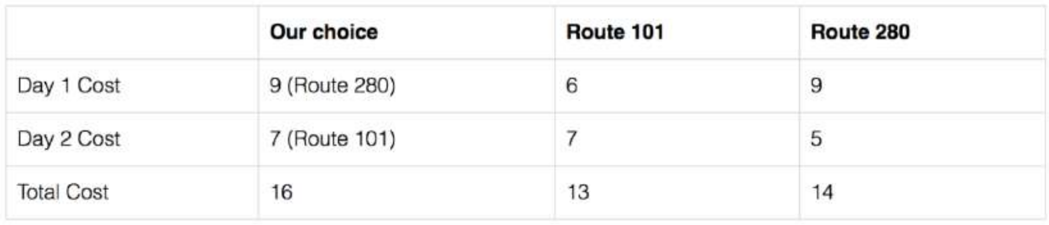
2.2.8 DRAGAN

Y. Jin et al. proposed the use of Deep Regret Analytic Generative Adversarial Networks (DRAGAN) [KAHK17] in 2017.

DRAGAN applies no regret $= R(T)$ algorithm to traditional GAN architecture:

$$R(T) := \sum_{t=1}^{T} L_t\left(k_t\right) - \min_{k \in K} \sum_{t=1}^{T} L_t(k) \tag{2.11}$$

An example would be, suppose we want to travel to a University from town center. Each day $= T$, we have different routes $= k_t$ that we can take. Loss function $= L_t$ is used to calculate the cost of taking the route on the day. After T days, we calculate the total cost and then repeat the calculation for each day when we take the same route. Eventually we have a total computed cost for all different routes and we select the one with the lowest cost then we are left with a difference and that is our regret $= R(T)$.



	Our choice	Route 101	Route 280
Day 1 Cost	9 (Route 280)	6	9
Day 2 Cost	7 (Route 101)	7	5
Total Cost	16	13	14

Figure 2-13: Example of choices [Hui18a].

$$R(T) \quad = \quad (9+7) - \min(6+7, 9+5) \quad = \quad 3 \tag{2.12}$$

By applying the no-regret algorithm to find the equilibrium (where the discriminator maximizes and generator minimizes), it reduces the chances of model collapse when training due to the game converging to bad local equilibria [Hui18a]. However, their model still suffer some issues. Class labels in training data are not evenly distributed which leads to measure bias and images are of low resolution and fuzzy.

2.2.9 Progressive Growing of GANs (ProGAN)

In 2017, T. Karras et al. proposed a progressive growing strategy in order to train a Generative Adversarial Network to generate increasingly more realistic images [KALL17]. In such a training strategy, the GAN is first trained to generate low resolution images, after which the complexity of the generative model is increased progressively in order to to generate increasingly more detailed faces. This allows to drastically speed up the training procedure, whilst still ensuring model stability.

When training starts, both the generator (G) and discriminator (D) having a low spatial resolution of 44 pixels. As the training advances, layers are added incrementally to G and D, thus increasing the spatial resolution of the generated images. NxN refers to convolutional layers operating on NxN spatial resolution. This allows stable synthesis in high resolutions and speeds up training time[KALL17].

After training, when the generator maps random number of an image, it is possible to interpolate between images by interpolating the random number. However, training time is recorded to be 1-2 months [KALL18], when desired output image resolution is 512 pixels and higher, as ProGAN utilises progressive growing from lower resolution iterations and growing all the way to the max 1024 pixels [KALL17]. Furthermore, ProGAN lacks the ability to control specific features of the generated images, any changes in inputs lead to affecting multiple features at the same time and causing feature entanglement [Hor18].

2.2.10 Style-GAN

Proposed in 2018 by Tero Karras, Samuli Laine, and Timo Aila, StyleGAN is the most recent and stable releases of GANs. Style-Based Generator Architecture Networks (Style-GAN) replaces traditional GAN generator architecture with an alternative generator architecture, which automatically learns un-supervised separation of high-level attributes (pose and identity) and stochastic variation in the generated images (freckles, hair). It also allows intuitive, scale-specific control of the synthesis, improving distribution of quality metrics for better interpolation properties, better latent factors of variation, compared to the traditional GAN architecture [KLA19a].

Unlike other approaches of GANs, which aim to stabilize training procedure by introducing a new architecture, Style-GAN's generator architecture technique focuses on loss functions, regularization, and hyper-parameters. To do this, a new Mapping Network is added:

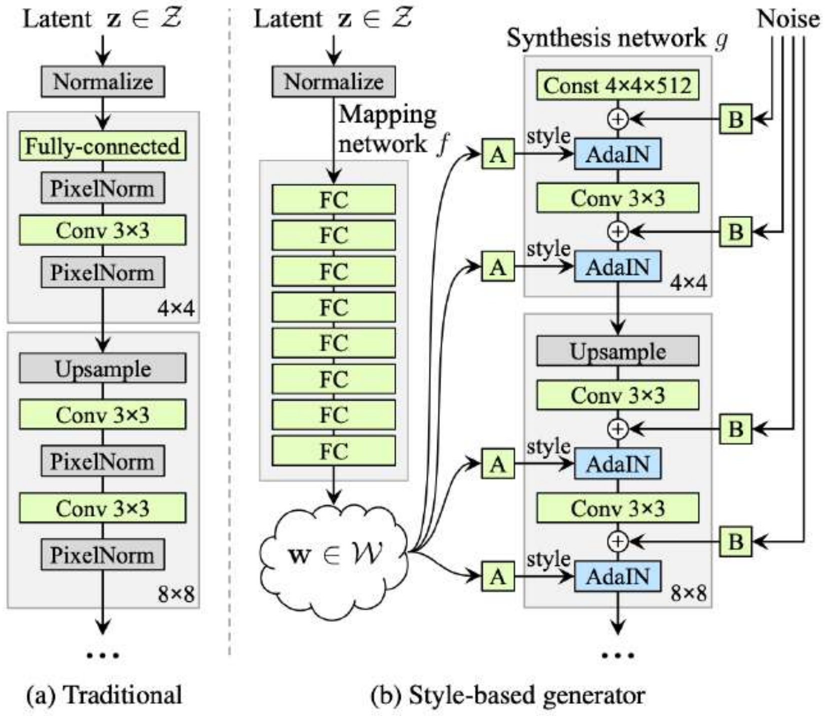


Figure 2-14: Traditional GAN Generator Vs Style-based generator [KLA19a].

In a traditional GAN, the generator takes a random Latent $\mathcal{Z}$ input vector and uses convolutional layers to normalize into a realistic output image. The Latent vector contains high level image specification features for the generator. However, the image specification cannot not contain features, for example hair colour, distance between eyes and facial hair etc, and only able to understand CUDATensors and FP16s. Therefore, a new Latent $\mathcal{Z}$ is selected when generating a new image and it is not suitable when fine control image style is desired as there is also no control over model distribution for the generator, over $\mathcal{Z}$ vectors. This problem was called feature entanglement, when there was a desire to change the features of the - for example hair colour, $\mathcal{Z}$ vector number could be altered for this but the output could be anything from a different gender, skin tone or generated face could have glasses.

StyleGAN aims to reduce this problem of feature entanglement by including 8 neural Mapping networks, which maps $\mathcal{Z}$ to a second $\mathcal{Z}$. This allows StyleGAN to have a neater $\mathcal{Z}$ space representation and does not make the output image drastically different [Ara19].

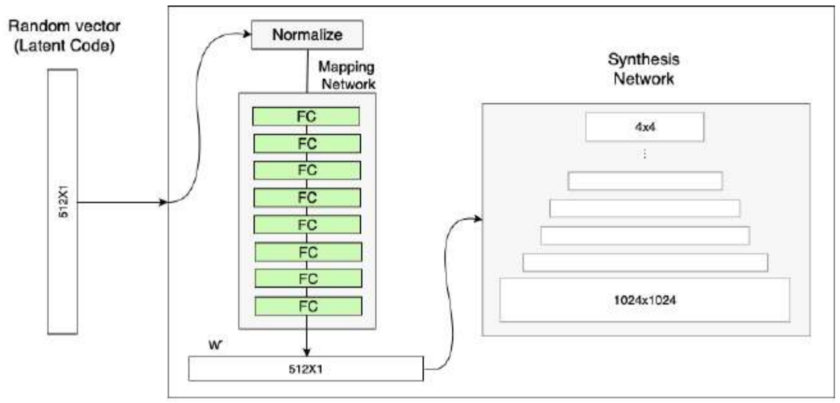


Figure 2-15: Systematic view of the Mapping Network [Lyr19].

In addition, 512-dimension $\mathcal{Z}$ vector as the input and the output were added. This allowed StyleGAN's generator to figure out how to use the numbers of the $\mathcal{Z}$ vectors with the help of the mapping network and reduce feature entanglement. The numbers used by the authors were arbitrary choices and hyper-parameters were able to be tuned, depending on dataset using and output desired.

The second function added was Adaptive Instance Normalization (AdaIN), Schematic overview displayed in figure 2-14.

AdaIN operation:

$$AdaIN\left(\mathbf{x}_i, \mathbf{y}\right) = \mathbf{y}_{s,i}\frac{\mathbf{x}_i - \mu\left(\mathbf{x}_i\right)}{\sigma\left(\mathbf{x}_i\right)} + \mathbf{y}_{b,i} \tag{2.13}$$

Where:

$\mathbf{x}_i$ is feature map
$\mathbf{y}$ is style inputs
$\sigma\left(\mathbf{x}_i\right)$ is variance of feature map input
$\mu\left(\mathbf{x}_i\right)$ is the mean of the feature map input

In traditional GANs, once the $\mathcal{Z}$ vector is given to the generator as the input, it could not be used again when it passes through one convolutional layer to another because the generator could not refer back to the same $\mathcal{Z}$ vector. AdaIN fixes this problem by feeding the $\mathcal{Z}$ vector into every layer and allowing the generator to keep referring back to the $\mathcal{Z}$ vector values. AdaIN uses a linear model called "learned affine transformation" [KLA19a], which maps the $\mathcal{Z}$ to two scalars and control the features of the output image:

$$Scale = y_s \quad and \quad Bias = y_b$$

The two scalars help to perform AdaIN:

$$y = (y_s, y_b) = f(w) \tag{2.14}$$

$f(w)$ is learned affine transformation. x_i is an instance that is applied to AdaIN. In traditional GANs, the mean and variance are performed for an entire mini batch. However, with AdaIN, it is computed per-channel and per-sample. The two scalars are infused into hidden layers activitions of the generator, becoming the gain = y_s and bias = y_b parameters. This helps to improve the quality of the style transfer images. With batch normalization, AdaIN is able to inject style information into the generator and the generator is able to know what kind of image it needs to output, rather than just $\mathcal{Z}$ input used by traditional GANs.

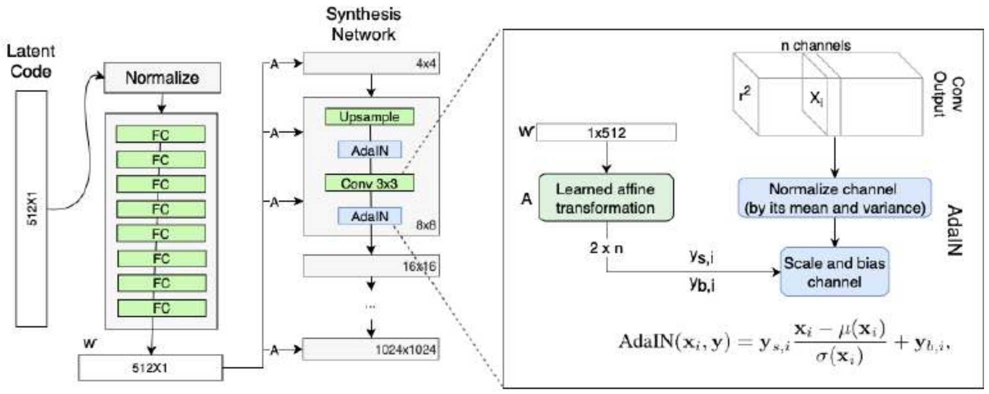

Latent Code
512X1
Normalize
FC
w
512X1
Synthesis Network
A
4x4
Upsample
AdaIN
Conv 3x3
8x8
16x16
1024x1024
n channels
Conv Output
r²
Xᵢ
w
1x512
A
Learned affine transformation
2 x n
Normalize channel (by its mean and variance)
ys,i
yb,i
Scale and bias channel
AdaIN
$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i}\frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$

Figure 2-16: Schematic overview of Adaptive Instance Normalization (AdaIN) function [Lyr18].

Another function added was Stochastic variation. Features of human faces such as freckles, placement of hairs, wrinkles and any feature that makes an image look realistic are classed as stochastic. Traditional GANs suffer from features entanglement problem, when stochastic features are added as noise into GAN images, which affects the output generated where there are features placement problems. The noise is added in a similar way in StyleGAN but to the AdaIN module which improves the resolution level of visual expression of features and does not encounter features entanglement problem.

Another technique added was Style mixing. StyleGAN's generator makes use of intermediate vector W but this could mean that the network learns levels that are highly correlated. To avoid this, two input vectors are selected randomly to generate intermediate vector W, by training some levels with first input vector and randomly switching with second input vector to train with the remaining levels. This ensures that levels learned by the network remains un-correlated. This allows to combine multiple images features (Style mixing) in a coherent way, something that traditional GAN's generator lacked [Lyr18].

To conclude, Style-GAN paper proposes the most ground breaking techniques to date. Their proposed techniques, especially the Mapping Network and the Adaptive Normalization (AdaIN) allows superior control over input choice and were to produced not only images of high quality but also realistic looking images. However, the model training requires extensive hardware requirements and time, for a chance of convergence.

2.3 Generative Adversarial Networks in image generation

2.3.1 State of the art human faces

Tero Karras, Samuli Laine, and Timo Aila, implemented Style-Based Generator Architecture Networks (Style-GAN) to generate state-of-the art results trained on dataset of human faces. The dataset included variations such as age, ethnicity, image background and accessories such as eyeglasses, sun- glasses, hats, etc and benefits evidently because of improved architecture and heavy usage of the 8 layers in Mapping network. Their proposed techniques, especially the Mapping Network and the Adaptive Normalization (AdaIN) allowed superior control over input choice and were able to produce pictures of bedroom images, cat images and car images and the results are better than expected. However, the results of state of the art fake synthesis human faces were the highlight of their work.

However, training requires extensive hardware requirements.

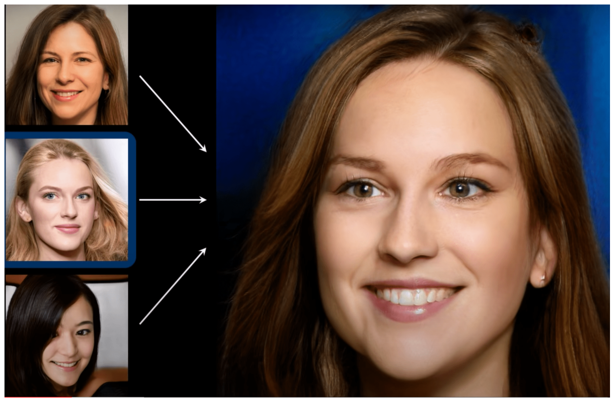


Figure 2-17: Style features inherited from pictures on the left to generate a state of the art fake human face [KLA19b].

Figure 2-18: Uncurated set of images produced by StyleGAN generator with car, car and bedroom dataset, respectively [KLA19b].

2.3.2 Photos to Emoji

Y. Taigmna, A. Polyak, and L. Wolf have used the concepts of Generative Adversarial Networks to create a framework that is capable of transforming images from one domain S to another domain T [TW16]. Their result - which is referred to as a Domain Transfer Network (DTN) - is capable of creating a generative function G which maps images from domain S to domain T, and simultaneously preserves the identity of the image in both domains. They used this framework to automate the mapping between facial images (domain S) and emojis (domain T), resulting in a model that is capable of creating a look-a-like emoji based on a newly given facial image. However, their discriminator generator is only able to handle 152 X 152 RGB images and other image sizes have to be scaled first to match the image dimension and type, to be able to begin any form of training.

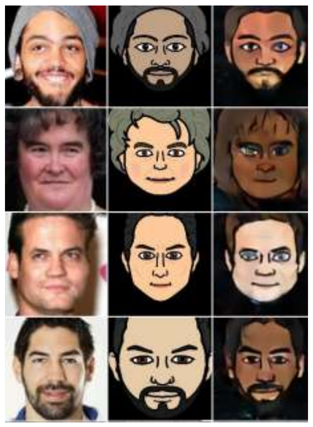


Figure 2-19: FACES: FROM [LEFT]PHOTOS TO [RIGHT]EMOJI [TW16].

2.3.3 PokeGAN

Similar applications have been developed by open source projects. Code-sharing platform, Github, show many projects related to the generation of Pokémon characters with the use of GANs [mg17]. Github user, Moxie Gushi, attempted generating new kind of fake Pokemons using architecture based on WGAN algorithm.

The model was trained on a dataset of 502 Pokemon images with 5000 iterations. The success of this project was limited because of two reasons, the fact only 502 Pokemon images were given to the model to train and only 5000 iterations were made and is a major factor on why this project did not succeed in generating new images of Pokemon.

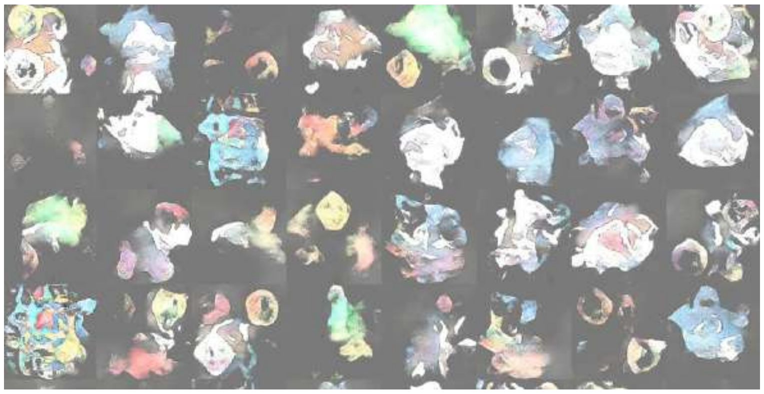


Figure 2-20: Model collapse, result of limited dataset and small iterations [mg17].

2.3.4 DCGAN to Simpsons

Greg Surma's attempted to generate Simpsons faces with DCGAN [Sur19b]. Simpsons is a popular cartoon television show [Sur19a]. The implementation makes use of loss functions to allow back propagation and uses Adam optimiser to stable the model and bring a balance between the generator and discriminator. The dataset consisted of 9877 face images of Simpson's character faces.

The model got better at learning and was successful in generating fake Simpson faces of low resolution. In addition, there were few malformed faces. The author hypothesized that with a cleaner and bigger dataset, combined with more excrement of hyper-parameter tuning and training time, the output could have been much better.

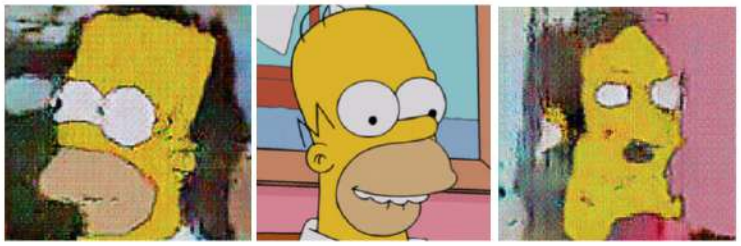


Figure 2-21: [Left] Artificial Vs [Middle] Real Vs [Right] Malformed Artificial [Tok19].

2.3.5 DCGAN to IllustrationGAN

Inspired by the work of Y. Jinetal's DCGAN [KAHK17] , Russell T.d implemented a model called IllustrationGAN [T.d]. The generator up-scales features by 2, followed by stride-1 layer for the generator and the discriminator. There are also more fully connected layers, as opposed to just one fully connected layer in DCGAN. Batch discrimination is replaced with Minibatch discrimination which allows the discriminator to look at multiple data examples in combination to avoid model collapse [Imp].

Lastly, a novel regularization applied to the work of the generator, where a small z-predictor network is added in the generator as input, to predict value z. Once the fully connected layer of the generator learns a function, the z-predictor attempts to learn the complete opposite. This helps the generator from collapsing as it only learns preserved information of z value [T.d].

The model was trained on a custom dataset of 20,000 anime faces and automatically cropped using a face detector. However, the output pictures are 128*128 pixels and suffers badly from over-fitting.

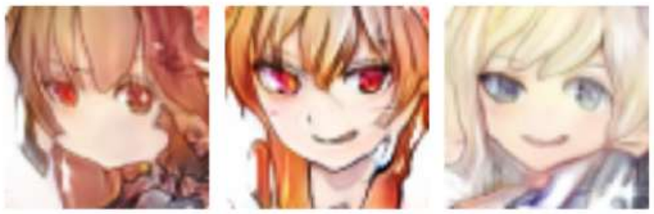


Figure 2-22: Unsuccessful anime faces generated by IllustrationGAN, badly over-fitted [T.d].

2.3.6 SAGAN to Anime faces

Another approach to generating anime faces was attempted by Git-hub user, ewrfcas [Ewr], called Anime-GAN-tensorflow, which is built on top of the SAGAN architecture.

Their model was trained to to 50,000 steps/iterations, whilst keeping close hyper-parameter size between the discriminator and the generator. However, after 70k steps the quality of the generated images began to degrade and the final output of the images worsened, can be seen in 2-23. Overall, the final quality of the generates images are no no different than IllustrationGAN. They look badly over-fitted and appears to have suffered from model collapse as they look very unnatural to be considered anime faces.

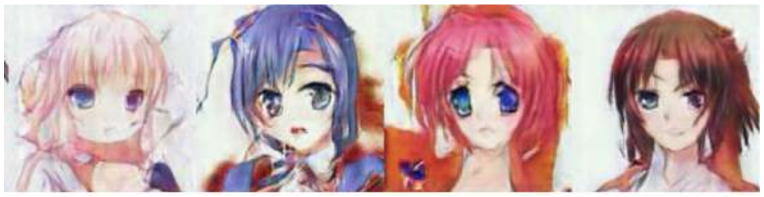


Figure 2-23: Unsuccessful Anime faces generated by SAGAN [T.d].

2.4 Summary and Outline

The literature literature review revealed that StyleGAN is the most state of the art deep learning algorithm, utilising artificial neural networks, constantly improved with new concepts and methods to evolve into becoming a new separate algorithm on it own. However, This literature review of related works reveals a gap in the application of Generative Adversarial Network, where it has been well trained and tested in-order to produce state of the art synthesis images human faces through StyleGAN, only few attempts have been made and they have ended highly unsuccessful in animation domain, specially with attempts to anime of SAGAN and DCGAN. The attempts have been unsuccessful because of mainly lack of and poor dataset, hardware requirement constraint, lack of hyper-parameters and learning rates tuning. This suggests that anime may have been a difficult application for traditional and older Generative Adversarial Network algorithms and suggests that it could be more suited to the more then the newer and better style transfer architecture of StyleGAN, as it has the most improved architecture to date with most stable and faster training time, and also another gap discovered, that attempts to generate anime faces in StyleGAN has not been yet attempted.

After careful consideration, it was decided that StyleGAN, will be used as the deep learning tool, to train and generate anime images. A review of previous works with GANs into animation found no existing attempted solution of StyleGAN to anime, therefore it was decided to implement one as part of this project.

At the heart of a training a successful StyleGAN is the time and the quality of the dataset. Based on previous work of StyleGAN on human faces, we expect the generated anime images to be high quality and indistinguishable from real anime images that are drawn by real animators. The authors of StyleGAN have released the source-code on open-source platform called GitHub [NP], under Creative Commons Public Licenses. The license effectively states that their code cannot be edited for producing derivative works, which could be re-training their generated models. However, the source-code may be used to train a model from scratch and copyright is not applied to the model or the source-code as the source-code was merely used for simply training another model [Com].

Therefore, after careful consideration, it was decided that the source-code will be used and modified to fit the training of anime images. Using the source-code provides the following benefits:

- Open source and free
- Clean and tested code, written by professionals in Python3 programming language. This gives less chances of running into code bugs, which could take a lot of time and support to fix, and not affordable for this project, as the project has a limited time frame deadline.
- Training GANs in general requires substantial training time with hyper-parameters experiment and because of the limited time frame of the project, the time can be spent on finding ways to optimally train StyleGAN.
- The time can be better spent on gathering a high quality dataset, as found through related works that most related works failed because of low quality and limited dataset. This will minimise the risk of running into training problems occurred when training GANs.
- More time can be spent with conducting different hyper-parameters experiments to ensure successful training and project success.

Based on the application of StyleGAN on human faces, expectation is that final generated anime images are to be high quality and indistinguishable from real anime images that are drawn by real animators. Therefore, we hypothesise that generated images will be high quality and users will not be able to tell the difference. Hence, evaluation metrics will be looked at to test the hypothesis and to determine if the hypothesis is proven and project is a success.

2.4.1 Literature review: Evaluating Generative Adversarial Networks

The training procedure of a Generative Adversarial Network is different from that of other machine learning training procedures in that both the generative network G and the discriminative network D are trained in a concurrent way. Therefore, there is no traditional objective function, as is normally the case during regular machine learning model training. Based upon an extensive analysis, H. Alqahtani, M. Kavakli-Thorne, and G. Kumar, concluded that the search for an appropriate metric for objectively evaluating a GAN's performance is still an open question within the field of generative modeling [AKTK19].

However, due to the increased interest in Generative Adversarial Networks during the past couple of years, several techniques - both qualitative and quantitative - have been proposed for evaluating GAN's performance. For Example, A. Borji has performed an extensive study regarding 24 quantitative and 5 qualitative measures which have been used for determining a GAN's performance [Bor19]. There are three commonly used qualitative methods being rating and preference judgement, nearest neighbors, and rapid scene categorization - which are often used for visually evaluating the a GAN's output:

1. **Rating and Preference Judgement**: In this setting, a group of subjects is asked to rate different Generative Adversarial Networks based on the output images they produce. For example, Snell et al. used this framework to determine whether participants preferred pixel-wise-loss optimized networks or perceptually-optimized networks by showing them three images: one for each network in addition to an image from the original dataset [SRL*17]. Participants are then asked which output they prefer, taking into consideration the example from the original dataset. In an evaluation setting like this, usually the first few examples are used as practise-rounds in order to allow the subject to become acquainted with the rating methodology.

2. **Nearest Neighbors**: Nearest neighbors evaluation is a technique that is used to visually evaluate the level of over-fitting in a Generative Adverserial Network. This is done by showing a series of generated images by the GAN, along with their corresponding nearest neighbors in the training set. If the resemblance between the generated images and the their corresponding nearest neighbor is too high, the network can be considered as over-fitted.

   However, a common problem with using this methodology is that is requires one to define a distance metric in order to determine the nearest neighbor [TOB15]. Typically, the Euclidean distance is used to determine the distance between different samples. However, this distance metric shows a high sensitivity to small perturbations within the data, resulting in situations where highly similar images may have large Euclidean distances and causing an image's nearest neighbor to be unrepresentative.

3. **Rapid Scene Categorization**: Multiple studies have shown that human beings are capable of categorizing the scenes and certain characteristics of images after only being exposed to the image for a very short duration of time [Oli05]. The Rapid Scene Categorization evaluation technique utilizes this knowledge by exposing subjects to images for a very short duration of time, and asking them to classify the image as being either real or fake (i.e., generated by a Generative Adversarial Network). Denton et al. successfully implemented this evaluation methodology by exposing a group of subjects to a set of real images along with images generated by three different GAN models, for periods between 50ms and 2000ms [DCR15]. Based upon the choices made by the subjects, they concluded that their GAN architecture was better at generating new CIFAR10 images compared to the GAN architecture proposed by Goodfellow et al. [GPAM*].

After careful consideration, Rating and Preference Judgement method is chosen. The hypothesis will be tested with an experiment where a questionnaire will be created and given to an audience to compare between real images drawn by animators and images generated by the chosen StyleGAN model.

The reason for this approach is because a quantitative method such as nearest neighbour, compares distribution distances between two data points, which could be misleading and un-representative, as some images may have high-distances because of high diversity in images. In addition, there are no qualitative methods by previous works of anime that are good enough to be compared with. In addition, comparing with other works other than anime or animation, such as human faces metric scores, would be misleading because the features in image types have different variations, e.g. rich colours and facial features difference. The qualitative score for human faces will be far different than qualitative scores of anime. Therefore, it is better to evaluate the generated pictures by testing ability of an audience to distinguish images generated by StyleGAN, against the real pictures drawn by animators. In this way, it will provide more meaningful evaluation of results, to determine if the hypothesis is proven successful and validate the research.

Chapter 3

Methodology

In order to successfully develop and train StyleGAN, more research is conducted. This chapter will explain the methodology undertaken to evaluate the reliability and validity of the research.

3.1 StyleGAN

In this section, more investigation is conducted into StyleGAN's implementation, to get more insight into the implementation that the authors used to generate state of the art fake human faces. This would give us more ideas about the details that went wrong, avoid potential problems, and could be crucial in saving time in training.

In summary, from section 2.2.10, it is known that StyleGAN makes a number of additional improvements, compared to the older GANs, discussed in chapter 2:

1. StyleGAN includes 8 neural Mapping networks, which maps $\mathcal{Z}$ to a second $\mathcal{Z}$, to allow a neater $\mathcal{Z}$ space representation.
2. 512-dimension $\mathcal{Z}$ (neurons) vector added as input and output to reduce feature entanglement.
3. Adaptive Instance Normalization (AdaIN) function added to allow feeding of $\mathcal{Z}$ vector into every layer, allowing the generator to keep referring back to the $\mathcal{Z}$ vector values to better control the features of the output image. AdaIN uses learned affine transformation technique where, AdaIN is able to inject style information into the generator and the generator is able to know what kind of image it needs to output.

4. Stochastic variation function added, where stochastic features are added to AdaIN module which improves the resolution level of visual expression.

5. Style mixing technique added to the generator, to ensure levels learned by the network remains un-correlated, to allow combining multiple images features (Style mixing) in a coherent way.

3.1.1 Truncation trick ψ in $\mathcal{W}$

Furthermore, StyleGAN's source-code makes use of another technique called Truncation trick ψ. It is only used in sample generation time and not during training time.

If the distribution of training data is considered, it is clear that areas of low density are poorly represented, making it difficult for the generator to learn on those areas.It is known that drawing latent vectors from a truncated or otherwise shrunk sampling space tends to improve average image quality, although some amount of variation is lost [KLA19a].

It follows the similar strategy used of drawing latent vectors $\mathcal{Z}$, where they compute centre of the mass of $\mathcal{W}$ as:

$$\overline{\mathbf{w}} = \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[f(\mathbf{z})] \tag{3.1}$$

Then scale the deviation of a given$\mathbf{W}$ from the center as:

$$\mathbf{w}' = \overline{\mathbf{w}} + \psi(\mathbf{w} - \overline{\mathbf{w}}), where \psi < 1 \tag{3.2}$$

In short, the idea is to draw latent vector $\mathcal{Z} = \mathcal{N}(0, 1)$, then removing variables that are greater than certain sizes such as 1, 0.7, 0.5, and 0 etc., and re-sample those. This strategy helps to avoid the latent variables $\mathcal{Z}$ or combination of latent values $\mathcal{Z}$ low and high density distribution of training data, where the generator has generated lowest data points or the highest. e.g -1 and 1.5. The authors consider this to be the most optimal method of Truncation, as latent variables $\mathcal{Z}$ can be generated from full $\mathcal{N}(0, 1)$ distribution where the datapoints are considered to be most accurately modelled [KLA19a].

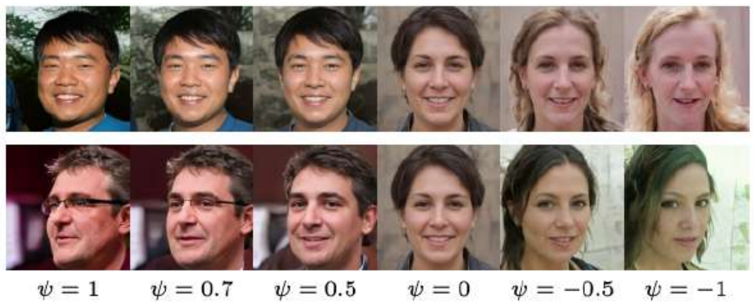


Figure 3-1: Effect of different truncation scaling to styles [KLA19a].

In Figure 3-1, when truncation $\psi \to 0$, all faces converged to the mean face, which means the faces are all similar with no diversity (viewpoint, glasses, age, coloring, hair length, and often gender). If -ψ is applied like -0.5 or -1, corresponding opposite or "anti-face" are generated.

3.1.2 Training setup

- StyleGAN inherits most training details from ProGAN's source [KLA19a]. The inherited details are same discriminator architecture, resolution dependent minibatch sizes, Adam hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ [KB17], and exponential moving average of the generator [KLA19a]. However, mirror (data) augmentation is enabled. **Data augmentation** is a strategy to help increase the diversity of data available for training models, without actually collecting new data, by cropping, padding, and horizontal flipping [HLL19]. However, it consumes more memory when enabled.
- ProGAN's equalised learning rate method is followed, where trivial $\mathcal{N}(0, 1)$ is used at initialization and then explicitly scale the weights at run time [KALL17]. However, it is noted that setting the learning rate to 0.002 instead of 0.003 for 512^2 and 1024^2 leads to better stability.

3.1.3 Datasets

Human face dataset

Their paper provided state-of-the art results trained on dataset of human faces called Flickr-Faces-HQ (FFHQ). Dataset consisted of 70,000 high quality images at 1024^2 resolution which were crawled from Flickr. Dataset included variations such as age, ethnicity, image background and accessories such as eyeglasses, sun- glasses, hats, etc.

LSUN Datasets

Other datasets were downloaded from the LSUN domain [YSZ*20]. **LSUN dataset:** Dataset constructed of a large scale coloured images based on human's effort in conjunction with deep learning and image classification models [FYX15].

- LSUN Bedroom dataset: 50,000 images at 256^2 resolution. Total training was 70 million images. Truncation trick $\psi = 0.7$. Dataset trained using same setup as FFHQ Dataset. Important note: "We suspect that the results for BEDROOM are starting to approach the limits of the training data, as in many images the most **objectionable issues** are the **severe compression artifacts** that have been inherited from the **low-quality training data**" [KLA19a].
- LSUN Car dataset: 50,000 images at 512 X 384 resolution. Truncation trick $\psi = 0.7$. Total training was 46 million images. Dataset trained using same setup as FFHQ Dataset. Important note: "CARS has much **higher quality training data** that also allows higher spatial resolution (512×384 instead of 256^2)" [KLA19a].
- LSUN Cat images: 256^2 resolution. Total training was 70 million images. Truncation trick $\psi = 0.7$. Dataset trained using same setup as FFHQ Dataset. Important note: "CATS continues to be a **difficult dataset** due to the **high intrinsic variation in poses, zoom levels, and backgrounds**" [KLA19a].

3.1.4 Source Code

Python is the main programming language used to StyleGAN. File types .py are program file or script written in Python.

- `networks_stylegan.py:` contains code to all improved network architectures of StyleGAN.
- `training_loop.py:` contains code to default key arguments such as hyper-parameters and learning rates that affects the learning and training dynamics.
- `train.py` Main training sequence is initialised with train.py file, where different hyper-parameters and learning rates can given to replace the default one which is set to 0.003. The other parameters are also set in this file, such as number of GPU's available, minibatch, max images/iterations/steps for training.
- `dataset_tool.py:` contains scripts to operate on datasets. StyleGAN does not take normal image dataset for training and they must be converted to StyleGAN's

data set format (.tfrecord) [NP].This is done with dataset_tool.py, which converts normal images and stores as multi-resolution TFRecords.

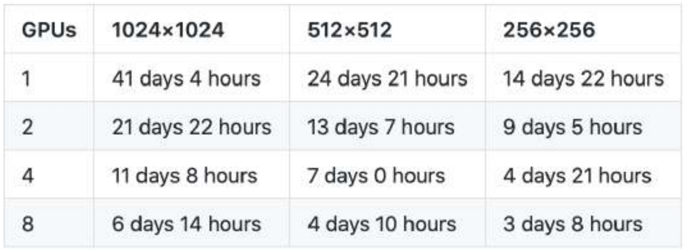
Upon close inspection of the script, there are three conditions that must be met for dataset conversion:

1. 'Input images must have the same width and height' - This means that all images in dataset must have the same width and height ratio. For example, 512 X 512.
2. 'Input image resolution must be a power-of-two'- this means that all images in dataset have same resolution. For example, images cannot be different resolution such as 256, 512 or 1024. They must all be the same.
3. 'Input images must be stored as RGB or grayscale' - all images must be the same colour space. Gray-scale is a range of shades of gray without apparent color and in RGB colour space, there are 256 shades of red, green and blue, which are be mixed to get other colour variations.
4. `pretrained_example.py` contains script for generating samples after training and uses truncation trick ψ, discussed in section 3.1.1.

3.1.5 Technology requirements:

Minimum requirements listed by authors on Github [NP].

- In-terms of operating system, both Linux and Windows are supported, Linux is strongly recommend for performance and compatibility reason.
- Python 3.6 installation. Anaconda3 with numpy 1.14.3 or newer are strongly recommended.
- TensorFlow 1.10.0 or newer with GPU support.
- One or more high-end NVIDIA GPUs with at least 11GB of DRAM. NVIDIA DGX-1 with 8 Tesla V100 GPUs is recommended.
- NVIDIA driver 391.35 or newer, CUDA toolkit 9.0 or newer, cuDNN 7.3.1 or newer.
- Training times (based on their training on human face dataset):



GPUs	1024×1024	512×512	256×256
1	41 days 4 hours	24 days 21 hours	14 days 22 hours
2	21 days 22 hours	13 days 7 hours	9 days 5 hours
4	11 days 8 hours	7 days 0 hours	4 days 21 hours
8	6 days 14 hours	4 days 10 hours	3 days 8 hours

Figure 3-2: Estimated training times for different resolution images, depended on the number of GPUs. [KLA19a].

3.1.6 Summary

A thorough investigation is conducted to identify problems and requirements for training StyleGAN.

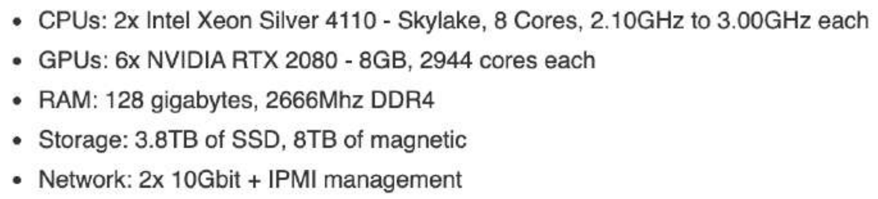
Following are identified as important in dataset collection and training:

- Dataset must be of high quality in order to generate high spatial resolution images in return. Their main dataset used 70,000 high quality images at 1024^2 resolution. Low resolution datasets could lead to objectionable issues like severe compression artifacts. Dataset cannot be of difficult dataset of high intrinsic variation in poses, zoom levels and backgrounds as noted with Cats images.
- Dataset images must have the same width and height, resolution must be a power-of-two, and must be stored as RGB or grayscale.
- Truncation trick $\psi = 0.7$ produces balanced featured, across all 3 generated datasets but can experiment with others - 1, 0.7, 0.5, 0, -0.5 and -1, displayed in figure 3-1.
- It is noted that setting the learning rate to 0.002 instead of 0.003, leads to better stability at training stage.
- Data augmentation is enabled to increase diversity in images generated and can only be trained on GPU's.
- Training times must be considered as the project time is limited, when deciding output image.
- Their generator also has a total of 26.2 million trainable parameters, therefore dataset size is not limited.

3.2 Hardware setup

The requirements in section 3.1.5 states that at least one or more high-end NVIDIA GPUs with at least 11GB of DRAM is required for training. It also requires a lot of training time, unless there are more than one GPU's available.

The University of Bath have three High Performance GPU clouds and one of them is available to use, which 6 GPU clusters [Bat]. The hardware specification matches with minimum requirements highlighted in section 3.1.5, apart from the DRAM of GPU being 3GB lower. However, this shouldn't cause any issue for training and the only trade-off is minor performance loss and could take longer than the expected training times highlighted in section 3.1.5.



- CPUs: 2x Intel Xeon Silver 4110 - Skylake, 8 Cores, 2.10GHz to 3.00GHz each
- GPUs: 6x NVIDIA RTX 2080 - 8GB, 2944 cores each
- RAM: 128 gigabytes, 2666Mhz DDR4
- Storage: 3.8TB of SSD, 8TB of magnetic
- Network: 2x 10Gbit + IPMI management

Figure 3-3: Bath University's GPU cloud specification [Bat].

The GPU cloud is free for students to use but it is also a shared resource and other students would be using it at the same time. Therefore, a reasonable workflow will be created at the end of the chapter to plan ahead and adhere to fair usage policy.

3.3 Dataset preparation

To get state-of-the art results trained on dataset of human faces called Flickr-Faces-HQ (FFHQ). Dataset consisted of 70,000 high quality images at 1024^2 resolution which were crawled from Flickr, discussed in section 3.1.3. Therefore, this is an indication of what might be needed, in order to achieve similar results in anime images generation. However, anime dataset with similar specification are really hard to obtain, generally because of lack of studies of anime in generative model domain. Therefore, in order to achieve as close results to state of the art human faces, it was decided that dataset would be collected and prepared manually. This way, quality of the images can be insured and the requirements of the dataset tool, discussed in section 3.1.4, can be adhered to.

3.3.1 Danbooru2019 Dataset

After countless hours of searching the web, Danbooru2019 dataset was found [AcB20]. The dataset contains a total of 3.69 million raw images of different variants of anime characters and 108 million tags that are associated with these images. The dataset can be freely downloaded using torrent client software or via a anonymous rsync server. The quality of the images vary from 512^2 to 1024^2 in resolution and the overall dataset is messy, unstructured, and with number of different file types, such as PNG and JPEG. The primary reason for choosing this dataset is that it contains 3.69 million anime images that are minimum 512^2 resolution.

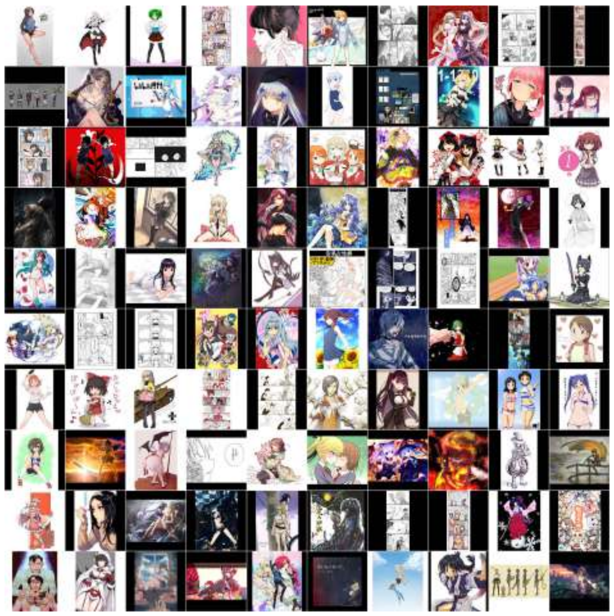


Figure 3-4: Representation of the images in the Danbooru2019 dataset [AcB20]

3.3.2 Dataset extraction Tools

Based on the investigation conducted in section 3.1.3, it was noted that LSUN cat images did not perform well in training because the dataset contained difficult high intrinsic variation in poses, zoom levels and backgrounds. Therefore, the collected dataset cannot have high intrinsic variation in poses, zoom levels and backgrounds. On the other-hand, human face dataset contained variations such as age, ethnicity, image background and accessories such as eyeglasses, sun- glasses, hats, etc but the images were limited from head to shoulders, and StyleGAN managed to get state of the art results.

Therefore, similar method will be followed to prepare the dataset gathered. Anime faces from will be cropped head to shoulders, from every image, restricting high intrinsic backgrounds but allowing accessories and poses. This way, it can ensure that the dataset is not difficult for StyleGAN to learn from. In addition, this will help us solve the problem at hand by generating anime faces, where animators find it particularly difficult to animate.

Furthermore, requirements of the dataset tool must be met. The desired output is 1024^2. However, due to the time limit constraint of the project, 512^2 is the resolution chosen to be trained and output desired, it should be more then sufficient in practice. Therefore, all images in the data needs to be cropped, resized to 512^2 resolution, converted to one specific file format (either JPEG or PNG) and all of them must have the same colour space (grey scale or RGB).

There are traditional face detection algorithms that is used for face detection such as Paul Viola and Michael Jones's Haar Cascade Classifier. However they do not work on Anime faces because in Haar Cascade Classifier, where different filters - usually referred to as HAAR features - are applied to images or regions of images as a convolutional kernel. Due to the fact that different facial regions are often characterised by certain characteristics (e.g., the region around the eyes is often dark, whereas the top-nose region is often more bright), the results from these convolutional operations allow to detect human faces within images quite consistently but it does not work on anime faces as they do not have the same characteristics as human faces.

Upon searching on Github, a face detection script for anime called lbpcascadeanimeface was found. It is a unique cascade developed and trained on anime faces [nag18], that works effectively in extracting anime character faces. It can extract image resolutions of 64^2, 128^2, 256^2, 512^2 and 128^2. This script will be used to crop anime faces. Extracting higher resolution requires more computational resources as the script runs on GPUs only. For example, the process of extracting 512^2 images could take 8-10 seconds for just one image. Therefore, time needs to be managed effectively and focused on securing a reasonable amount of data from 3.69 million images.

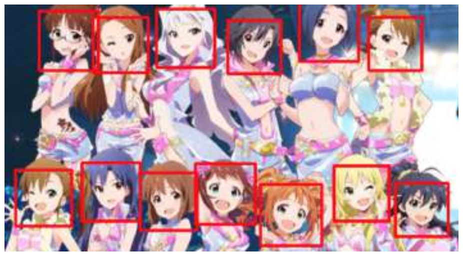


Figure 3-5: An example of *lbpcascadeanimeface* detection running on an anime image [nag18].

Some of the anime faces extracted could be low resolution, even-though they are instructed to be the same resolution, because of image/background/character position. In this case, they will need to be either removed or up-scaled. Therefore, as a contingency plan, another script called waifu2x [Nag] was discovered on Github. The scripts is image Super-Resolution, specially for Anime-Style Art. For example, we can edit the script to take a low resolution image to upscale to 512^2. Waifu2x makes use of the CNN's architecture, where it compresses the image and applies selective blur from GIMP, which is a high quality image manipulation package which allows from retouching to restoring to creative composites, then it keeps repeating the process until the noise is reduced to 0 and the final image is scaled to 512^2.

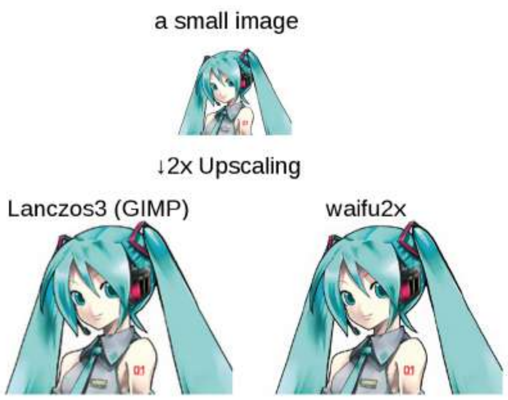


Figure 3-6: workings of waifu2x [Nag].

The overall quantity of the dataset will be depended on time it takes. However, the aim is to match Flicker-HQ dataset of 70,00, used to generate human faces.

3.4 Technology review

All technologies and terminologies that the project plans to use are explained in this section.

- **Python** is the primary programming language of choice because it extensive selection of open-source machine learning libraries with high-level APIs, its simplicity and comprehensibility, and its large community.
- **OpenCV** is an open-source computer vision library written in C++ and Python, that was initially developed by Intel [ope]. It offers a wide range of built-in applications that are commonly used within the field of computer vision, including face recognition, gesture recognition, and motion tracking.
- **Torch** is a scientific computing framework with wide support for machine learning algorithms [RCKS]. This framework is needed by waifu2x to work with a GPU.
- **NumPy** is a scientific Python package that is extensively used for the computation and handling of N-Dimensional data arrays.
- **Anaconda** is a free and open-source distribution and the distribution includes data-science packages.
- **TensorFlow** is an open-source software library that is primary focused on machine learning applications like artificial neural networks and deep learning. [ten].
- **Github** is an open-source repository hosting service which allows versioning control system which means each time a file is pushed, GitHub keeps a history, making it easy to explore the changes that occurred to it at different time points and this way, can reverse changes/mistakes [Ger20].
- **GPU** is a processor that is specially-designed to handle intensive graphics rendering tasks.
- **Linux** is an operating system but manages the communication between software and hardware without the operating system (OS).
- **CUDA** is a parallel computing platform and programming model that makes using a GPU for general purpose computing simple and elegant.
- **Pip** is the standard package manager for Python.
- **Virtual environment** allows to create own customised version of Python, into which any extra libraries can be installed without the permission of the main administrator.

- **Tensorboard** provides the visualization and tooling needed for machine learning experimentation, such as Tracking and visualizing metrics such as loss and accuracy for training.
- **Screen** is a terminal program in Linux which allows us to use a virtual terminal as full-screen window manager, which are typically, interactive shells. In an event of network disconnection, a program can keep running.
- **MobaXterm**provides important remote network tools for connecting to the GPU cloud, such as SSH, Unix commands and more.
- **SSH**, also referred to as Secure Shell, is a method for secure remote login from one computer to another.
- **Git** is a distributed version-control system for tracking changes in source code during software development.

3.5 Project Workflow

A workflow is created so that it can be followed for better time management and keep track of important stages later on.

1. Setup Linux and virtual environment.
2. Install all software dependencies (libraries).
3. Git pull StyleGAN code and other scripts scripts.
4. Download Danbooru2019 dataset.
5. Crop anime faces from Danbooru2019 dataset, using lbpcascade script.
6. Manually delete duplicate and empty images, if there are any.
7. Use waifu2x script to convert all images to 512^2, image type JPEG or PNG and all colour space to RGB.
8. Manually check all images, deleting duplicate and empty images.
9. Convert the prepared dataset to StyleGAN's format (.tfRecords) using dataset_tool.py
10. Make changes to Style-GAN scripts to match anime faces dataset
11. Do a small training to make sure program runs smoothly.
12. Do bigger training with different hyper-parameters desired result is achieved and satisfied
13. Create user survey and gather results
14. Evaluate the results and findings

Chapter 4

Development

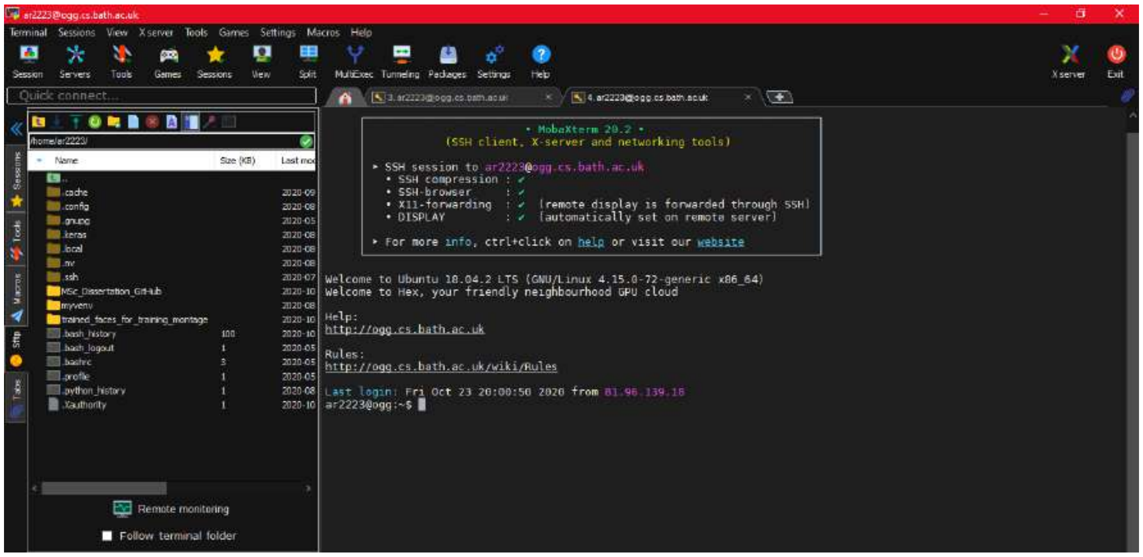
4.1 Linux setup

As discussed in section 3.1.5, The university of Bath provided a powerful GPU cloud to use. However, to access the GPU and run any code, SSH needs to be setup. Secure shell (SSH) communicates with another computer (GPU cloud) and gets Command Line access, to run remote code on the GPU cloud computer. For that, MobaXterm was installed and SSH setup to get access to the GPU cloud.

A graphical user interface (GUI) is for doing simple things, e.g. browsing the web and moving files around. For some tasks it might even be preferable. But to interact with a computer properly and make it do anything even moderately advanced, command line is used to do everything on GPU cloud. The command line is more simpler - it makes everything clear, to help avoid mistakes. Therefore, commands can be given to the command line to communicate to the GPU cloud computer.

For instance, command used in command line to login:

```
ssh ar2223@ogg.cs.bath.uk
```


ar2223@ogg.cs.bath.ac.uk
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick connect...
3. ar2223@ogg.cs.bath.ac.uk
4. ar2223@ogg.cs.bath.ac.uk
Sessions Tools Macros Sftp Tabs
/home/ar2223/
Remote monitoring
Follow terminal folder

Name	Size (KB)	Last mod
..		
.cache		2020-09
.config		2020-08
.gnupg		2020-05
.keras		2020-08
.local		2020-08
.nv		2020-08
.ssh		2020-07
MSc_Dissertation_GitHub		2020-10
myvenv		2020-08
trained_faces_for_training_montage		2020-10
.bash_history	100	2020-10
.bash_logout	1	2020-05
.bashrc	3	2020-05
.profile	1	2020-05
.python_history	1	2020-08
.Xauthority	1	2020-10

```
                 • MobaXterm 20.2 •
     (SSH client, X-server and networking tools)

 ▸ SSH session to ar2223@ogg.cs.bath.ac.uk
   • SSH compression : ✓
   • SSH-browser     : ✓
   • X11-forwarding  : ✓  (remote display is forwarded through SSH)
   • DISPLAY         : ✓  (automatically set on remote server)

 ▸ For more info, ctrl+click on help or visit our website

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-72-generic x86_64)
Welcome to Hex, your friendly neighbourhood GPU cloud

Help:
http://ogg.cs.bath.ac.uk

Rules:
http://ogg.cs.bath.ac.uk/wiki/Rules

Last login: Fri Oct 23 20:00:50 2020 from 81.96.139.16
ar2223@ogg:~$
```

Figure 4-1: Example of SSH, through MobaXterm, into the GPU cloud

Figure 4-1 is a screenshot of what the terminal shell interface looks like, after a successful login has been made. On the left hand, it is the main user home directory and its contents. On the right hand ride, is the terminal shell to give commands and communicating to the Cloud operating system.

4.2 Installing dependencies

To make sure that code can run without any errors, their dependencies needs to be installed, such as packages and libraries they need to run.

1. **Virtual environment:** To create a virtual environment:

```
    \emph{python3 -m venv ~/myvenv}

```

2. **Copy global libraries into virtual environment:**

```
    python3 -m venv --system-site-packages ~/myvenv}

```

   Note: Global libraries needed for StyleGAN code to run (discussed in section 3.1.5) were were already installed and available, by administrator of the GPU cloud. Therefore, it was only needed to be copied over to the virtual environment.

3. **Activate virtual environment:**

```
source ~/myvenv/bin/activate

```

4.3 Data collection and cleaning

After GPU cloud access was setup and libraries installed

1. **Downloading the dataset**

```
rsync --verbose --recursive rsync://78.46.86.149:873/
danbooru2019/ ./danbooru2019/}

```

Note: rsync is designed to synchronise data between servers and is much faster other download methods. verbose means that information will be displayed on the screen to let the user know what is going on, behind the screen. recursive cop pies directories recursively, between GPU cloud and the server from where the dataset is being downloaded.

The overall size of the database was 1.6TB and couple of hours to download into location: /mnt/slow2/

2. **Preparing lbpcascade animeface script**

```
cd /home/ar2223/MSc_Dissertation_GitHub/Code_for_Data_Set
git pull https://github.com/nagadomi/lbpcascade_animeface

```

Note: Git pull fetches and downloads face cropping script into home directory, from GitHub remote repository.

Limitation of the script: The script is for only face extraction. The script works by cropping on the center of an anime face and it will cut out all the other details such as hair, shoulders, and backgrounds etc because the code of the bounding box is set to that.

Changes made: The changes made to the code was by playing around with the bounding box code around the face, such as changing the values of x and y variables multipliers in the code to crop more details from an image, such as different accessories on the head, such as gadgets, hats, and shoulders etc.

Changes in code: detect.py

```
i=0
for (x, y, w, h) in faces:
    image[int(y*0.25): y + h, int(x*0.90): x + int(w*1.25)]
    cv2.imwrite(outputname+str(i)+".png", cropped)
    i=i+1

```

3. **Cropping faces**

   The script was ran from shell terminal with command:

```
detectandcrop() {
BUCKET=$(printf "%04d" $(( $@ % 1000 )) )
ID="$@"
CUDA_VISIBLE_DEVICES="0" python ~/home/ar2223/MSc_Dissertation_GitHub/Code_for_Data_Set/lbpcascade_animeface/examples/detect.py  \ ~/home/ar2223/MSc_Dissertation_GitHub/Code_for_Data_Set/lbpcascade_animeface/lbpcascade_animeface.xml  \ ./mnt/slow2/danbooru2019/original/$BUCKET/$ID.* "/mnt/slow2/Anime_Dataset_prepared/$ID"
}
export -f detectandcrop

mkdir /mnt/slow2/Anime_Dataset_prepared/
find ./mnt/slow2/danbooru2019/original/ -type f -name "*.jpg" | parallel detectandcrop

```

   Explanation: Bucket print out what file/folder the script is working on to crop images from. ID is the image id. CUDA_VISIBLE_DEVICES is to restrict the code using only one GPU and the number reflects the first GPU in the cloud.

   The script is running in parallel, scanning, cropping and saving faces at the same time. It scans and crops images in (/mnt/slow2/danbooru2019/original/) folder, it is where the danbooru2019 dataset was downloaded in, and saves the cropped images into (/mnt/slow2/Anime_Dataset_prepared), this is the directory the cropped images are saved in, with the command "mkdir", and saves with the same unique ID of the picture that was extracted from.

4. **Up-scaling and cleaning dataset**

   – **Cleaning empty files**

```
find /mnt/slow2/Anime_Dataset_prepared/ -size 0 -type f -delete

```

Terminal shell command to delete any images that is empty or failed by the script.

- **Deleting low quality files**

```
find /mnt/slow2/Anime_Dataset_prepared/ -size -100k -type
f -delete

```

Terminal shell command to delete any images that is less than 100kb in size, a good quality image would be at least 100kb in size.

- **Deleting duplicate files**

```
fdupes --delete --omitfirst --noprompt /mnt/slow2/
Anime_Dataset_prepared/

```

Terminal shell command to delete any duplicate images that maybe in the folder, it identifies it by looking at the unique id of images given by the cropping script, in section 3.

- **Up-scaling images Getting the upscaling script**

```
cd /home/ar2223/MSc_Dissertation_GitHub/Code_for_Data_Set
git pull https://github.com/nagadomi/waifu2x

```

Note: Git pull fetches and downloads face cropping script into home directory, from GitHub remote repository.

Up-scaling to 512^2 Rationale: The upscaling script runs on GPUs only and one image would take up to 9-12 seconds to upscale to 512^2, higher resolution would take twice the time. Therefore, due to the limitation of time of the project, 512^2 was chosen to be the fixed size for all images in the dataset.

```
upscaleimages() {
SIZE1=$(identify -format "%h" "$@")
SIZE2=$(identify -format "%w" "$@");

if (( $SIZE1 < 512 && $SIZE2 < 512  )); then
    echo "$@" $SIZE
    TMP=$(mktemp "/tmp/XXXXXX.png")
    CUDA_VISIBLE_DEVICES="0" /home/ar2223/
MSc_Dissertation_GitHub/Code_for_Data_Set/waifu2x/waifu2x.
lua -model_dir \ ~/home/ar2223/MSc_Dissertation_GitHub/
Code_for_Data_Set waifu2x/models/upconv_7/art -tta 1 -m
scale -scale 2 \
    -i "$@" -o "$TMP"
    convert "$TMP" "$@"
    rm "$TMP"
    fi;  }

export -f upscaleimages
find /mnt/slow2/Anime_Dataset_prepared/ -type f | parallel
 upscaleimages

```

Explanation: The command compares two formats and sizes, then if two format and sizes are different, then a temporary directory is created and those images are saved in the temporary directory. The upscaling script then up-scales the images saved in temporary directory, into 512^2 and saves it back into the dataset directory (/mnt/slow2/Anime_Dataset_prepared/). This is done in a loop, therefore the script keeps doing this until all images are up-scaled into 512^2 resolution then the temporary directory and temporary images are deleted.

– **Conversion to JPG**
Rationale: JPG image can contain up to 16.8 million colors and are extremely good at handling rich colour images. JPEG has a very efficient compression technique that combines high compression degree with minimal quality loss, compared to PNG [Lar].

To do this, ImageMagick was used. ImageMagick is a free open source software, which can create, edit, compose, or convert bitmap images. It can read and write images in a variety of formats (over 200) including PNG and JPEG [LLC].

```
find /mnt/slow2/Anime_Dataset_prepared/ -type f \
mogrify -resize 512x512\

```

Converting everything to RGB to meet StyleGAN's dataset tool format, with ImageMagick:

```
find /mnt/slow2/Anime_Dataset_prepared/ -type f \
    fgrep -v " JPEG 512x512 512x512+0+0 8-bit sRGB"

```

- **Final quality checks**
  Finally after all the followed process were done, it was time to manually inspect the dataset and take out any images that had been oddly cropped and that did not look good enough.

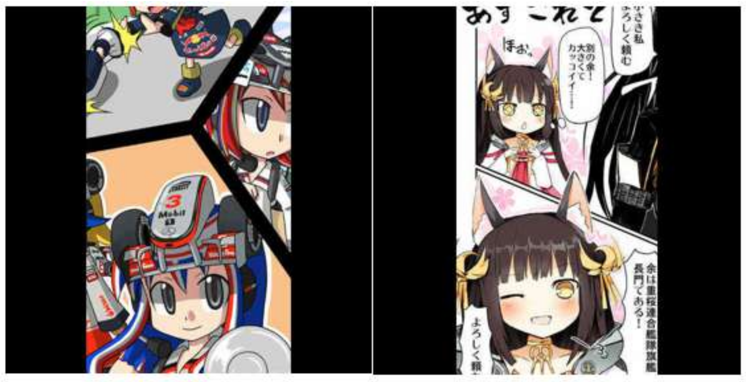


Figure 4-2: Example of two images that thought to be too high intrinsic and deleted

4.4 Summary

The total time it took to create the dataset was 28 days. It was an extremely long process but now a high quality dataset was gathered for training. **The final dataset contained a total of 217,800 images and 12GB in size.**

4.5 Training and Experimenting

1. Getting StyleGAN code Code is pulled from open source website called GitHub.

```
    git clone https://github.com/NVlabs/stylegan

```

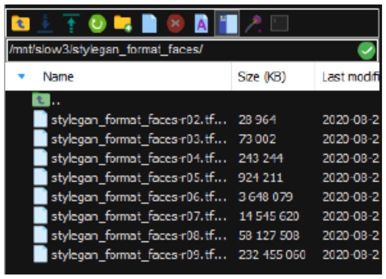
2. **Enabling Virtual environment and enabling screen session:** Virtual environment is enabled so that any extra libraries/packages can be installed without administrator's permission, if needed. Screen allows a code to keep running, if there is a network disconnection between personal computer and GPU cloud, and saves time and progress in an that event.

```
cd stylegan #to go inside stylegan directory
source ~/myvenv/bin/activate #activates virtual environment
screen #enables screen session

```

3. **Dataset conversion to StyleGAN format:** The dataset gathered needs to be converted to StyleGAN's own format of .TFRecords, as discovered in Section 3.1.4. The dataset was converted through command:

```
python3 dataset_tool.py create_from_images /mnt/slow2/
converted/stylegan_format_faces /mnt/slow2/
Anime_Dataset_prepared/

```

Limitation: StyleGAN only reads .tfRecords files (a TensorFlow supported data format which stores each image as arrays at every relevant resolution) as input for training model. Therefore, all images were converted to a readable format by dataset_tool.py. The conversion also takes a long time and 25x additional disk space. **The dataset prepared was 12GB and StyleGAN converted .tfRecords was 296GB**.



```
/mnt/slow3/stylegan_format_faces/

Name                                 Size (KB)     Last modifi
..
stylegan_format_faces-r02.tf...      28 964        2020-08-2
stylegan_format_faces-r03.tf...      73 002        2020-08-2
stylegan_format_faces-r04.tf...      243 244       2020-08-2
stylegan_format_faces-r05.tf...      924 211       2020-08-2
stylegan_format_faces-r06.tf...      3 648 079     2020-08-2
stylegan_format_faces-r07.tf...      14 545 620    2020-08-2
stylegan_format_faces-r08.tf...      58 127 508    2020-08-2
stylegan_format_faces-r09.tf...      232 455 060   2020-08-2
```

Figure 4-3: Screenshot of converted .tfRecords

4. **Modifying codes for training** After successful conversion of the dataset, train.py needed to be modified.

train.py

```
# Dataset.
desc += '-stylegan_generated_faces';     dataset = EasyDict(
tfrecord_dir='/mnt/slow2/converted/stylegan_format_faces',
resolution=512);

desc += '-4gpu'; submit_config.num_gpus = 4; sched.
minibatch_base = 16; sched.minibatch_dict = {4: 512, 8: 256,
16: 128, 32: 64, 64: 32, 128: 16}

```

Rationale: Setting the converted StyleGAN format dataset by editing the directory path in dataset variable. Secondly, selecting the desired output resolution to images that is desired, which is 512^2. Thirdly, selecting a pre-configured number of GPUs setting, as mentioned in section 3.1.5, there are 6 GPUs available. However, to comply with fair usage, the number of available GPUs that were not used was checked first by command:

```
    nvidia-smi -l

```

It was revealed that there were all GPUs available at the time but it was not known when others would want to use GPUs as-well, therefore only 2 GPUs were used for trial training. training_loop.py was left as default to spectate how it would react to first training and see if there were any errors that required to modify it.

5. **Trail training** A first trail training was conducted through command:

```
    CUDA VISIBLE DEVICES=0,1 python3 train.py

```

Note: CUDA VISIBLE DEVICES allocates how many GPU's to use for training, in this case it was GPU number 0 and 1.

StyleGAN crashed after 1d 23h 45m of training. It was suspected that the minibatch variable and learning rates because in training_loop.py was behind it because the operating system killed the process with error "Out of memory: kill process 30712, score 148 or sacrifice child". Therefore, following was changed in training_loop.py:

```
    minibatch_repeats       = 1 # Number of minibatches to run
    before adjusting training parameters.

```

Rationale: It was changed from 5 to 1 because of spaciousness of gradient accumulation, as the implementation is set for a GPU with 11GB VRAM minimum and the ones we have used for 8GB each. Therefore, changing this should make the training more stable:

Learning rates changed in train.py:

```
    sched.G_lrate_dict = {128: 0.0015, 256: 0.002, 512: 0.002,
    1024: 0.002}

```

Rationale: GPU memory is not the same as system memory, the process because the process reaper of the GPU doesn't care about GPU memory as if asked for too much then the program gets told it has and gets to decide how to handle that. Therefore, it was suspected it was also because it was loading too many images into system memory at a give time, causing a memory leak. Therefore, learning rate was reduced to 0.002 from 0.003.

Furthermore, it was noted that, when training was restarted after a crash, the progress was started from scratch and completely ignored the training process and the trained model, which were trained for 1d 23h 45m. Overall 2 days of training was wasted. Therefore, to crash-proof future training from a crash, following changes to training_loop.py was made:

training_loop.py

```
if resume_run_id is not None:
        if resume_run_id == 'latest':
            network_pkl , resume_kimg = misc.locate_latest_pkl()
        else:
            network_pkl = misc.locate_network_pkl(resume_run_id , resume_snapshot)
        print('Loading networks from "%s"...' % network_pkl)
        G, D, Gs = misc.load_pkl(network_pkl)

```

Rationale: StyleGAN saves a pickle snapshot every 300 ticks/steps, which contains all the progress made with image generation for 300 steps. When a new training was ran after a crash or after changing hyper-parameters, it would start from scratch and not from the last saved pickle snapshot. Therefore, the following code checks if there is a 'latest' snapshot that was saved from last training in results folder, if there is then it finds that pickle snapshot and gets the network_pkl (id) and resume_kimg - which is the total number of steps that was taken in last pickle snapshot, higher resume_kimg means that starting training from a higher resolution.

6. **Further Training and Experiments**

Tensorboard was used to monitor the losses of generator and discriminator. Later in training, if the discriminator proved to be too strong then the generator would not produce improving images and if the generator wins ultimately, then the generator would stop producing images at all. Therefore, it is important to monitor the losses to make sure the loses were balanced, therefore the training and image quality would improve steadily.

Note: Tensorboard is a tool that provides the visualization for tracking and visualizing metrics such as loss and accuracy.

There was a firewall behind personal computer and the GPU cloud and did not allow any parts through other than SHH. Therefore, SSH port tunnelling was set up to connect between local computer to the port of the GPU cloud, through command:

```
ssh -L localhost:8898:localhost:6000 ar2223@ogg.cs.bath.ac.uk
# First command

tensorboard --logdir=/mnt/slow1/stylegan/results/0002-sgan-
stylegan_generated_faces-4gpu --port=6000 # Second command

```

Rationale: The first command is to essentially attaches port 8898 on GPU cloud to port 6000 of the personal local machine. Then all the information sent by tensorboard is now forwarded to local machine using port 6000, without the firewall getting on the way. Tensorbaord is initialised into local browser by typing http://localhost:8080.

logdir is the location variable, where the training results are stored in, when a new training is started it is given a unique number and a new directory such as 0001, 0002, and 002 and so on. The location directory would need to be changed aswell, every time a new training was started.

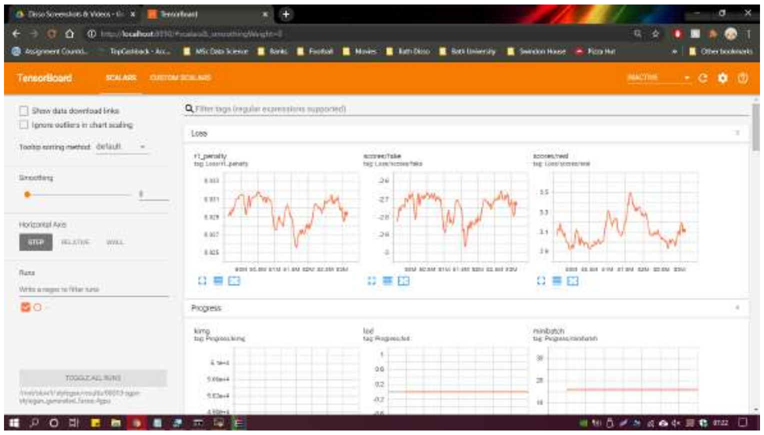


Figure 4-4: Example of Tensorboard running, showing loses of Discriminator and Generator.

Learning rates Hyper-parameters Learning rates was found crucial in the training stages later on. Same objectionable compression artifacts started to appear in later stages of training, as discussed in section 3.1.3. This proved to because of learning rate of the Discriminator proving to be too high for the generators, therefore learning rates were individually edited, for generator and the discriminator in train.py.

```
    sched.G_lrate_dict = {128: 0.0015, 256: 0.002, 512: 0.002,
    1024: 0.002}
    sched.D_lrate_dict = {128: 0.0015, 256: 0.002, 512: 0.001,
    1024: 0.001}

```

Experimented learning rates and Iterations of G/D

Training Time	Generator	Discriminator	Iterations
1d 23h 45m	0.003	0.003	6630
3d 19h 29m	0.002	0.002	13300
1h 51m 39s	0.002	0.001	13420
8d 13h 34m	0.002	0.001	20380
2d 06h 37m	0.001	0.001	23890
6h 31m 11s	0.001	0.0003	24040
17d 02h 21m	0.0003	0.0001	49580
7d 01h 38m	0.0003	0.0003	61040

Table 4.1: Complete training details of experiments performed with hyper-parameters tuning.

Summary: The final model was trained for 47 days, 20 hours, and 14 minutes, for 61040 iterations.

4.6 Sample generation

As discussed in section 3.1.1, the sample generation uses Truncation trick ψ and it is generated with pretrained_example.py, discussed in section 3.1.4. However, the script only generates one face and the seed is always set as one number so that there is no reproducibility. Therefore, the code was modified to generate 1000 random sample of images that were trained with following truncation trick ψ hyper-parameters.

truncation trick ψ	Sample range
1.0	1000
0.7	1000
0.5	1000
0.0	1000
-0.5	1000
-1.0	1000

Table 4.2: truncation trick ψ used and generated samples respectively.

Modified code: Following changes were made to pretrained_example.py.

```
def main():
    tflib.init_tf()
    _G, _D, Gs = pickle.load(open("/mnt/slow1/stylegan/results/00013-
    sgan-stylegan_generated_faces-4gpu/network-snapshot-060380.pkl", "
    rb"))

    # Print network details.
    Gs.print_layers()

    for i in range(0,50):
        #Pick latent vector.
        rnd = np.random.RandomState(None)
        latents = rnd.randn(1, Gs.input_shape[1])

        # Generate image.
        fmt = dict(func=tflib.convert_images_to_uint8, nchw_to_nhwc=
    True)
        images = Gs.run(latents, None, truncation_psi=-1.0,
    randomize_noise=True, output_transform=fmt)

        # Save image.
        path= "/mnt/slow1/stylegan/generated_pictures/"
        os.makedirs(path, exist_ok=True)
        png_filename = os.path.join(path, 'truncation_psi=-1.0**'+str(
    i)+'.png')
        PIL.Image.fromarray(images[0], 'RGB').save(png_filename)
```

Command to run the code:

```
cd /mnt/slow1/stylegan #go to code directory

CUDA VISIBLE DEVICES=0 python3 pretrained_example.py #run code
```

4.7 Generate Figures

generate_figures.py contained script that was used to generated random samples figures in 2-18, to demonstrate different aspects of style noises in random samples in StyleGAN's paper [KLA19a]. The script was modified in-order to show same style of anime figures generated. Following changes were made to generate_figures.py:

– variables bedroom, cat and car were disabled as it will not be used anywhere in the code.

```
#url_bedrooms    = 'https://drive.google.com/uc?id=1
MOSKeGF0FJcivpBI7s63V9YHloUTORiF' # karras2019stylegan-
bedrooms-256x256.pkl
#url_cars        = 'https://drive.google.com/uc?id=1
MJ6iCfNtMIRicihwRorsM3b7mmtmK9c3' # karras2019stylegan-cars
-512x384.pkl
#url_cats        = 'https://drive.google.com/uc?id=1
MQywl0FNt6lHu8E_EUqnRbviagS7fbiJ' # karras2019stylegan-cats
-256x256.pkl

```

– Truncation trick ψ was changed to fit anime face model.

– Model name changed to anime model.

– Image dimension reduced to 512^2 resolution to fit the dimension anime model was trained on.

– Latent space was expanded to show more insightful samples.

```
    def main():
    tflib.init_tf()
    path= "/mnt/slow1/stylegan/generated_figures/"
    os.makedirs(path, exist_ok=True)
    draw_uncurated_result_figure(os.path.join(path, 'figure02-uncurated-ffhq.png'), load_Gs(url_ffhq), cx=0, cy=0, cw=512, ch=512, rows=3, lods=[0,1,2,2,3,3], seed=5)
    draw_style_mixing_figure(os.path.join(path, 'figure03-style-mixing.png'), load_Gs(url_ffhq), w=512, h=512, src_seeds=[639,701,687,615,2268], dst_seeds=[888,829,1898,1733,1614,845], style_ranges=[range(0,4)]*3+[range(4,8)]*2+[range(8,16)])
    draw_noise_detail_figure(os.path.join(path, 'figure04-noise-detail.png'), load_Gs(url_ffhq), w=512, h=512, num_samples=100, seeds=[1157,1012])
    draw_noise_components_figure(os.path.join(path, 'figure05-noise-components.png'), load_Gs(url_ffhq), w=512, h=512, seeds=[1967,1555], noise_ranges=[range(0, 18), range(0, 0), range(8, 18), range(0, 8)], flips=[1])
    draw_truncation_trick_figure(os.path.join(path, 'figure08-truncation-trick.png'), load_Gs(url_ffhq), w=512, h=512, seeds=[91,388, 389, 390, 391, 392, 393, 394, 395, 396], psis=[1, 0.7, 0.5, 0.25, 0, -0.25, -0.5, -1])

```

– **Training Montage** At every tick/steps snapshots are generated to show training progress and saved into results directory. If all snapshots are combined, then it can be compressed in a time lapse to show the training progress/montage overtime. To do this, FFmpeg was used. FFmpeg is a free and open-source software project consisting of a large suite of libraries and programs for handling video, audio, and other multimedia files and streams [tea].

First, all snapshots images from all results directory, were copied to a new directory:

```
cp /mnt/slow1/stylegan/results/00013-sgan-stylegan_generated_faces-4gpu/fakes*.png /home/ar2223/trained_faces_for_training_montage/

#other directories 0001 - 00013
```

Secondly, all images are sorted and FFmpeg command given to combine all pictures and make the training montage video:

Sort:

```
ls (/home/ar2223/trained_faces_for_training_montage/fakes*.png  |
    sort --numeric-sort)
```

Combine and create:

```
ffmpeg -framerate 15 -i - -r 30 -c:v libx264 -pix_fmt -crf 40 -vf
    "scale=iw/2:ih/2" -preset veryslow -tune animation /home/
    ar2223/facestraining_10k_iterations.mp4

# 30 is frame rate per second
# framerate is to show 15 images per second in a frame
# internal library for compatibility issues with Linux
# 40 is video quality
# scale=iw/2:ih/2 will compress the image size by factor of 2
    without losing quality and reduces the video size
# preset+tune converts into a slow animation movie like settings
```

Chapter 5

Results

5.1 Dataset result

Dataset development is discussed in section 4.3. The total time it took to create entire the dataset was 28 days. The final dataset contained a total of 217,800 images and 12GB in size.

Original dataset images example (without performing dataset cleaning):

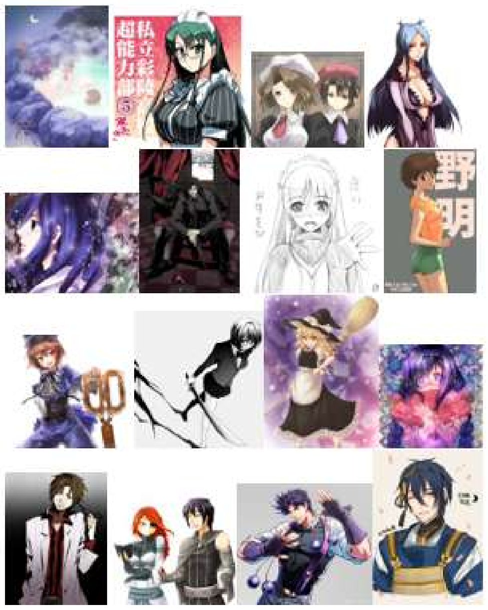


Figure 5-1: original dataset images example, arranged in 4X4 Grid.

Final dataset example after development:

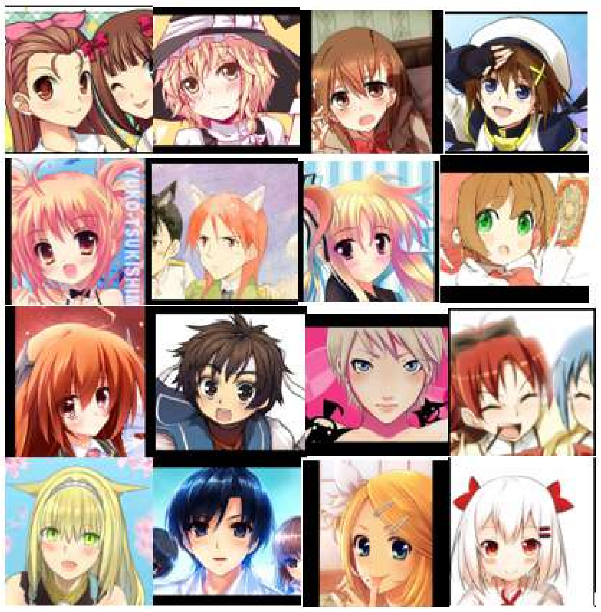


Figure 5-2: Cleaned dataset samples example, arranged in 4X4 Grid.

5.2 Training results

5.2.1 Trial training

First training started with some random noise:

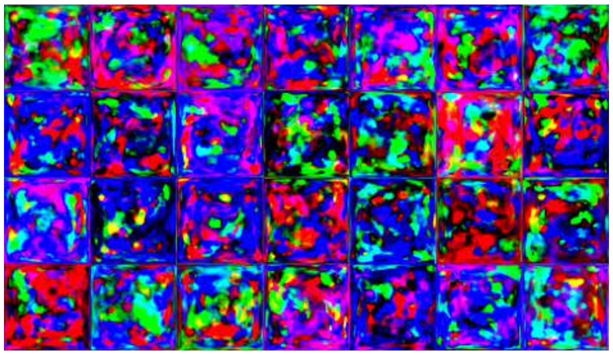


Figure 5-3: First snapshot of training, generated by the generator.

Over the course of training, the training progress is given as ticks and iterations. The training gets better as more iterations are performed and generator starts to adapt to the latent spaces.

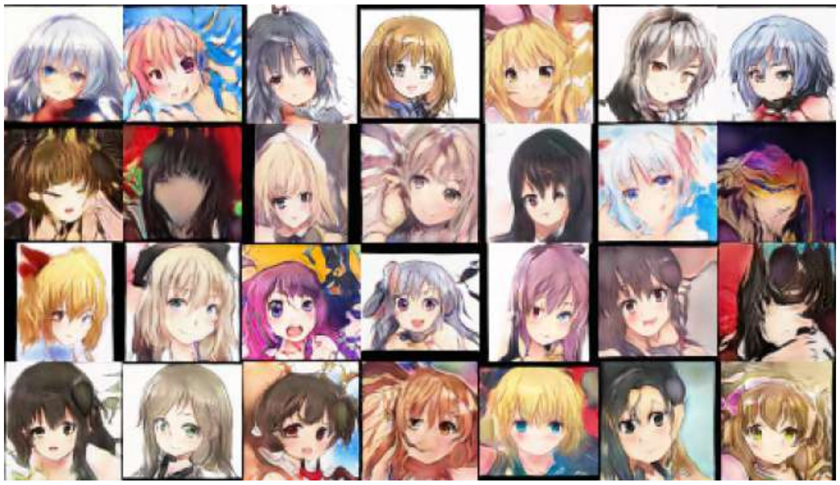


Figure 5-4: Training snapshot at 4281 iterations

With more ticks, the minibatch sizes began to decrease, as the variations of images learned by the generator began to decreased aswell. The level of detail began decrease at the same time, meaning that the training was reaching high resolution images and was in the main stages of training and learning. However, the training crashed at 6630 iterations.

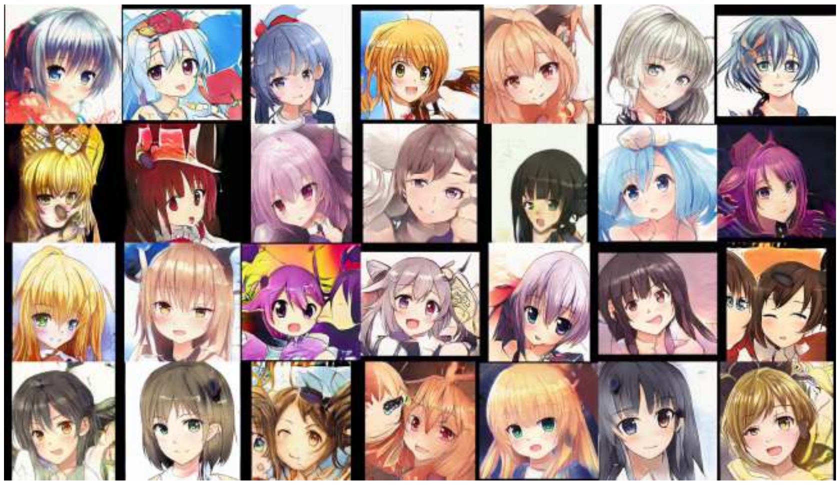


Figure 5-5: Training snapshot at 6630 iterations

Training Time	Generator L/R	Discriminator L/R	Iterations
1d 23h 45m	0.003	0.003	6630

Table 5.1: Trail training details, with learning rates of G/D

5.3 Training improvements with hyper-parameters

It was suspected that the crash was due to gradient accumulation, somewhere in GPU memory because there was a memory leak and the GPU reaper killed the process. The learning rates controls how quicker the generator and discriminators learns and if the learning rate is too high, when the minibatches starts to decrease after reaching high level of detail, means that the training stability is compromised and memory starts to leak, this was evident because the code was written with 11GB GPU memory in mind and compared to 8GB used for training. It was evident that learn rates and minibatch repetition needed to be lowered. After making changes described in section 4.5, the training was more stable and the variations in learning began to improve, as the losses in Tensorboard, started to reduce aswell, indicating that the training was stable and improving.

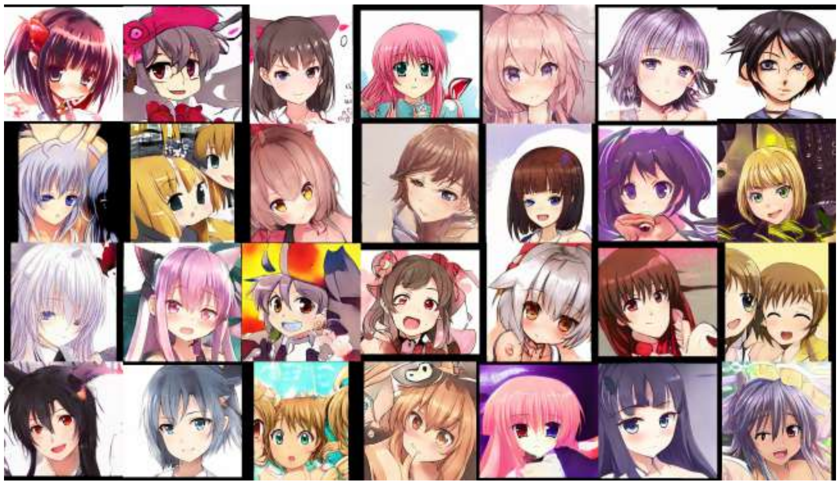


Figure 5-6: Training snapshot at 13300 iterations

Training Time	Generator L/R	Discriminator L/R	Iterations
3d 19h 29m	0.002	0.002	13300

Table 5.2: Second training details, with learning rates of G/D

Second training was started from 6630 iterations, growing progressively until 13300 iterations because

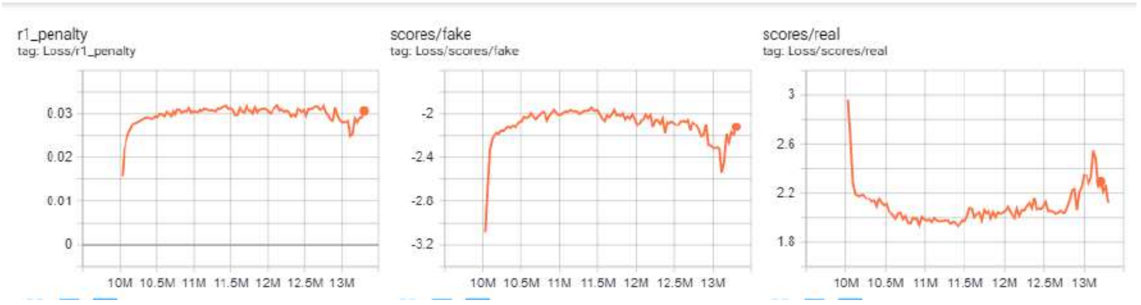


Figure 5-7: Real(discriminator) loss lower than fake(generator), generated images decreasing in quality

To fix the issue of discriminator winning against the generator and to balance the learning rates, the discriminator's learning rates was reduced to 0.001, keeping Generator's learning rate to 0.002. It was trained with the setting until 20380 iterations.

Training Time	Generator L/R	Discriminator L/R	Iterations
1h 51m 39s	0.002	0.001	13420
8d 13h 34m	0.002	0.001	20380

Table 5.3: Further training details, with lower learning rate of discriminator

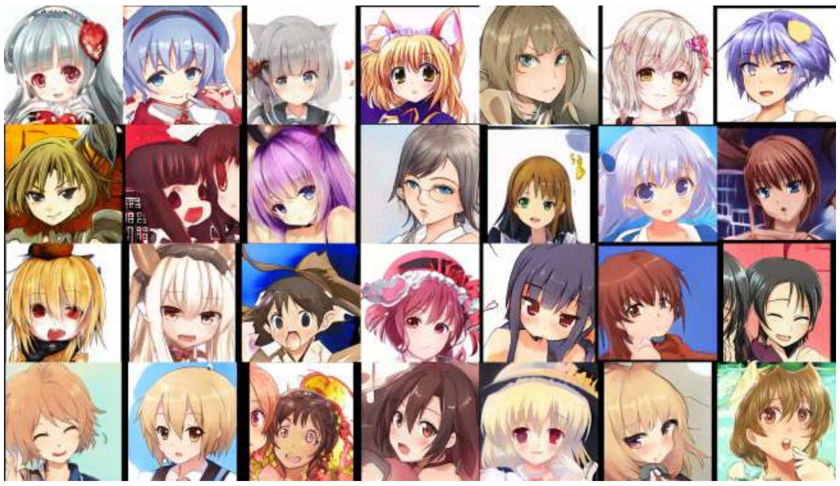


Figure 5-8: Real(discriminator) loss lower than fake(generator), generated images decreasing in quality

However, the training was stopped at 20380 iterations because compression artifacts mentioned in 3.1.3.

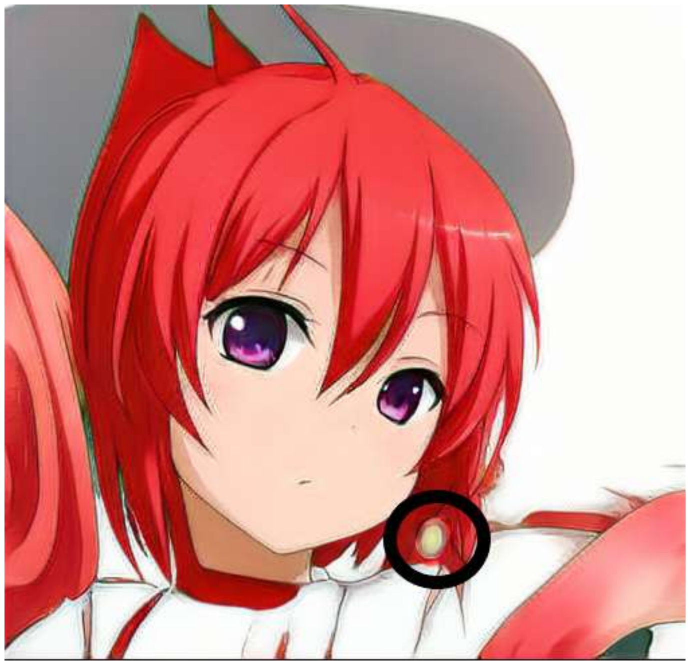


Figure 5-9: Example of an artifact forming in training image, highlighted by the black circle.

These artifacts only started appearing in late stages of training, in some images and especially when the images progressively grew to a high resolution. The source of the artifacts was unclear as the dataset was up-scaled with 512^2 resolution. Therefore, it could not have been from a low quality dataset. By looking at the training images, it was clear that the artifacts morphed into new features in the images and thought it was part of StyleGAN's generator architecture. It was suspected that it might have been because of overfitting. Overfitting happens when function is too closely fit to a limited set of data points. It was suspected that the generator might started to memorise noise of the real dataset images and that it might be a sign of overfitting or training heading towards model collapse. Therefore, the solution thought was to reduce learning rates and train for longer times to bring the losses of generator and discriminator to a stable value where the error range will not change, this could stabilise training better and remove the artifacts. More experiments were conducted with hyper parameters tuning and are listed in the following table 5.4.

The solution worked and the images were free of artifacts from 40,000 iterations. However, there was no convergence between the the generator and the discriminator. An algorithm is said to converge when, as the iterations proceed, the output gets closer and

Training Time	Generator	Discriminator	Iterations
2d 06h 37m	0.001	0.001	23890
6h 31m 11s	0.001	0.0003	24040
17d 02h 21m	0.0003	0.0001	49580

Table 5.4: Training details and hyper-parameter values to get rid of artifacts in images

closer to some specific value [Gri]. However, the loses of the generator were fluctuating and never settling into one specific value. This is considered to be a hill-climbing algorithm, where it may reach a local maximum but not to a global maximum, where the algorithm could converge but not to the optimal value [Gri]. Not reaching convergence could mean that the algorithm could still improve and at least to a local maximum, improving the generators quality even more but time was very limited at this point. Therefore, one last experiment was conducted for a chance of local maximum and better generation of anime faces.

Training Time	Generator	Discriminator	Iterations
7d 01h 38m	0.0003	0.0003	61040

Table 5.5: Training details of last experiment for convergence

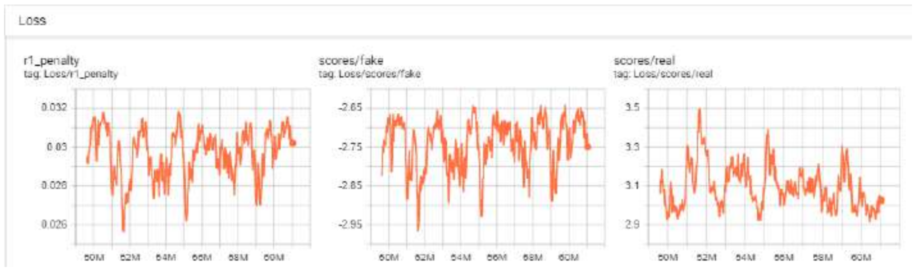


Figure 5-10: Tensorboard: Convergence unsuccessful

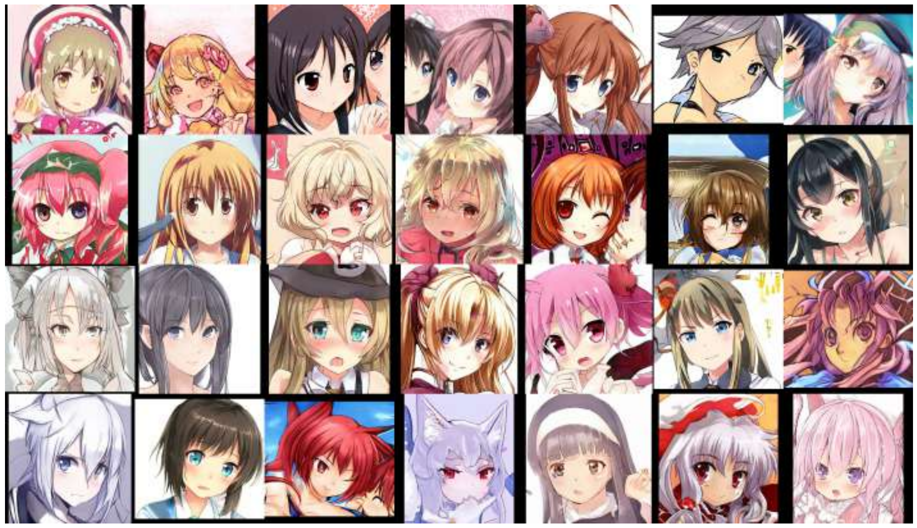


Figure 5-11: Final training snapshots

The final experiment was trained for 7 more days and 61040 iterations. Convergence was unsuccessful, displayed in figure 5-10. However, the image quality improve drastically in figure 5-11, with more variations and diversity added to the anime faces. With no time to spare because of limited project time to meet deadline, it was time to generate final trained samples.

5.4 Final Samples

For this section, the final samples will be presented in the same format as the state of the art human faces in figure 3-1. It's the best possible method to show the generated samples and diversity (viewpoint, glasses, age, coloring, hair length, and often gender), of the images learned by the generator. The code and method use to generate the final samples are discussed in section 4.6. More samples can be found in Appendix section 7.1.7.

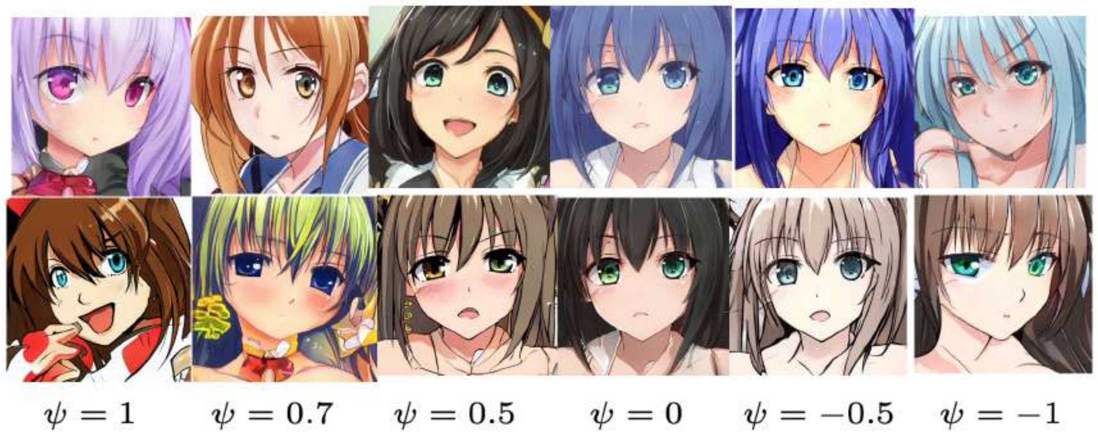


Figure 5-12: Truncation scaling to styles, in final anime trained model

Similar to figure 5-13, when truncation ψ is faded $\rightarrow 0$, all faces are converged to mean faces. Applying -ψ gives corresponding opposite faces. However, applying $+\psi$ gives more diversity in faces, emotions, accessories worn by different characters. The best balance between the quality of the images and diversity seems to be between $\psi = 0.5$ and $\psi = 0.7$. ψ 1.2 seems to generate even more diverse images, which has not been explored.

5.4.1 Truncated $\psi = 1.2$ Samples

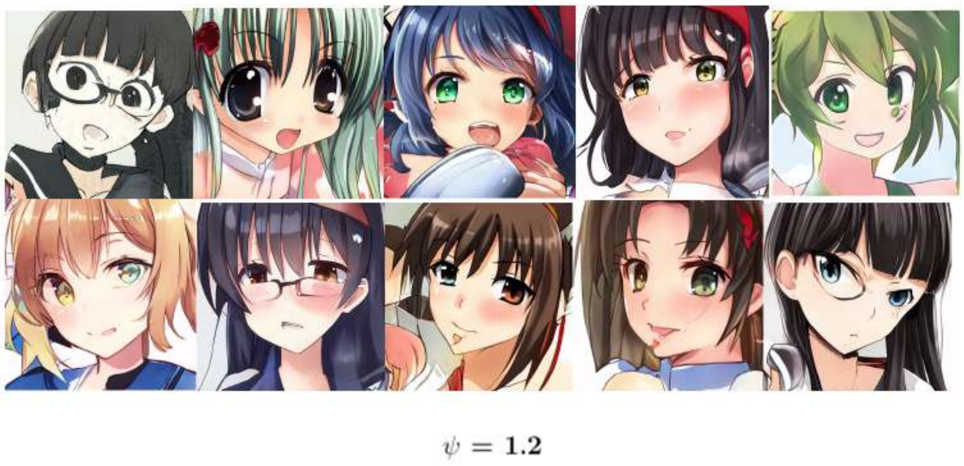


Figure 5-13: $\psi = 1.2$ Samples

With Truncation $\psi = 1.2$, there are far more better diversity, however the image quality are not as good as compared to $\psi = 0.7$. The best balance between quality and diversity seems to be $\psi = 0.7$. However, samples from $\psi = 0.7$ seems more visually interesting to look at and motivates more inspiration.

5.4.2 Random Sample figures

Discussed in section 4.7, generate_figures.py script was modified to generated random samples figures, to demonstrate different aspects of style noises in random samples.

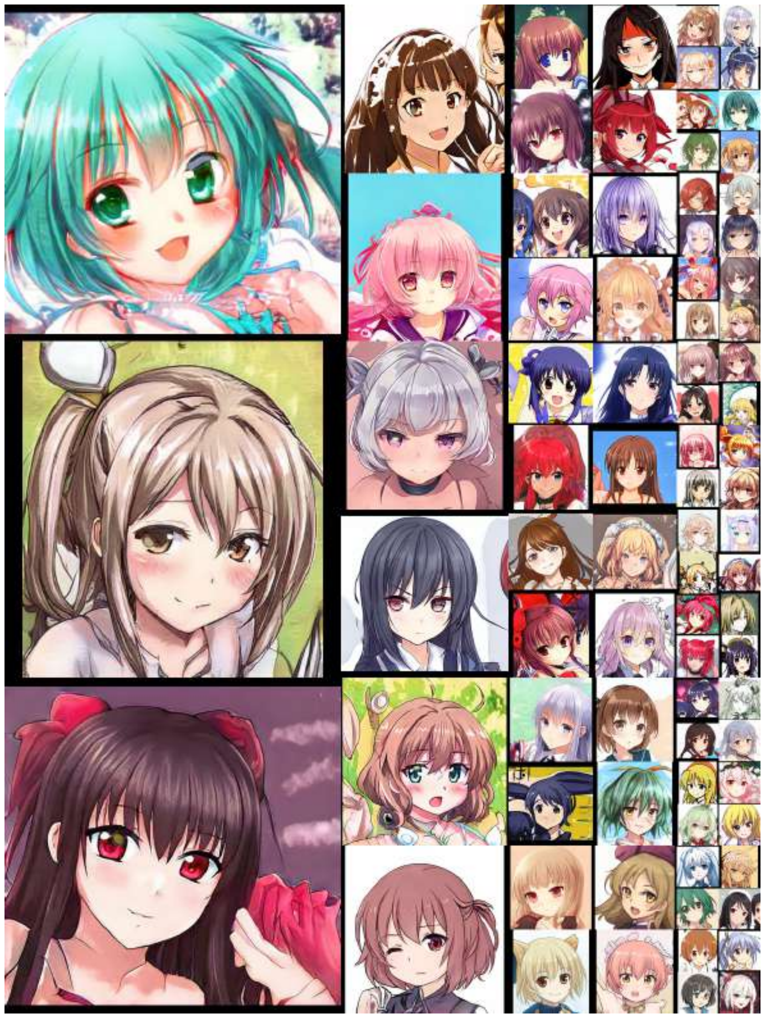


Figure 5-14: Uncurated set of random images produced by StyleGAN generator with the anime dataset. Truncation trick used $\psi = 0.7$

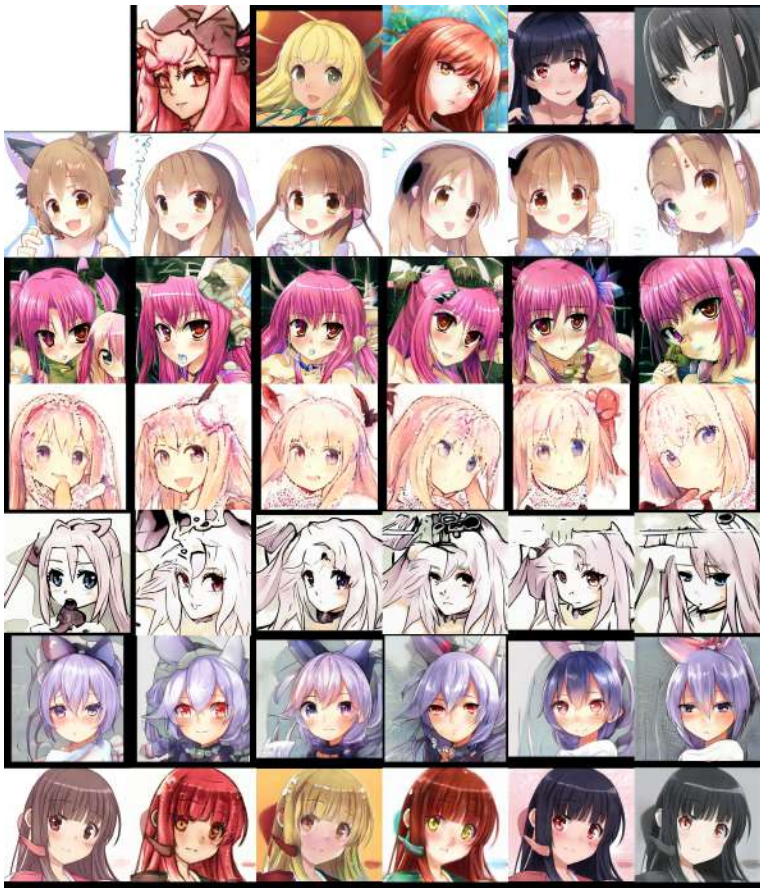


Figure 5-15: Two sets of images generated from their respective latent codes (sources A and B).

In figure 5-15, **Source A = left row, Source B = top column**. The rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. **Column = 2, 3, and 4 from the top, are Coarse styles from source B**. **Column 5 and 6 are Middle styles from source B**. **The last column = 7, is Fine from B**. Copying the styles corresponding to coarse spatial resolutions $\left(4^2 - 8^2\right)$ brings high-level aspects such as pose, general hair style and face shape from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A. If copied from the styles of middle resolutions $\left(16^2 - 32^2\right)$ from B, it inherits smaller scale facial features, hair style, eyes from B, while the pose, general face shape, and from A are preserved. Finally, copying the fine styles $\left(64^2 - 512^2\right)$ from B brings mainly the color scheme and microstructure [KLA19a].

5.4.3 Summary

In summary, the trained model is used to generate high quality images with truncation ψ = 1.2, 1, 0.7, 0.5, 0, -0.5, and -1. ψ = 0.5 and 0.7 offers the best balance between quality and diversity. As expectation, final generated anime images are to be high quality, however, to test the hypothesis created in section , a user study was created to find out if an audience could be able to tell the difference between artist created anime characters and anime images generated by a machine.

5.5 Hypothesis testing

Hypothesis: Users will not be able to tell the difference between artist created anime faces and anime faces generated by StyleGAN.

Design Rationale: Due to the limited time, the best way to ability of an audience to distinguish images generated by StyleGAN, against the real pictures drawn by animators was through online quiz. After searching the web, Survey Monkey was chosen as the platform to perform the test on. Survey Monkey is an online survey development cloud-based software, which allows to create questionnaires that can be modified and designed to a user's choice [Lur]. Furthermore, each survey can be sent to participants though a website link, where they can access the survey through a phone or computer.

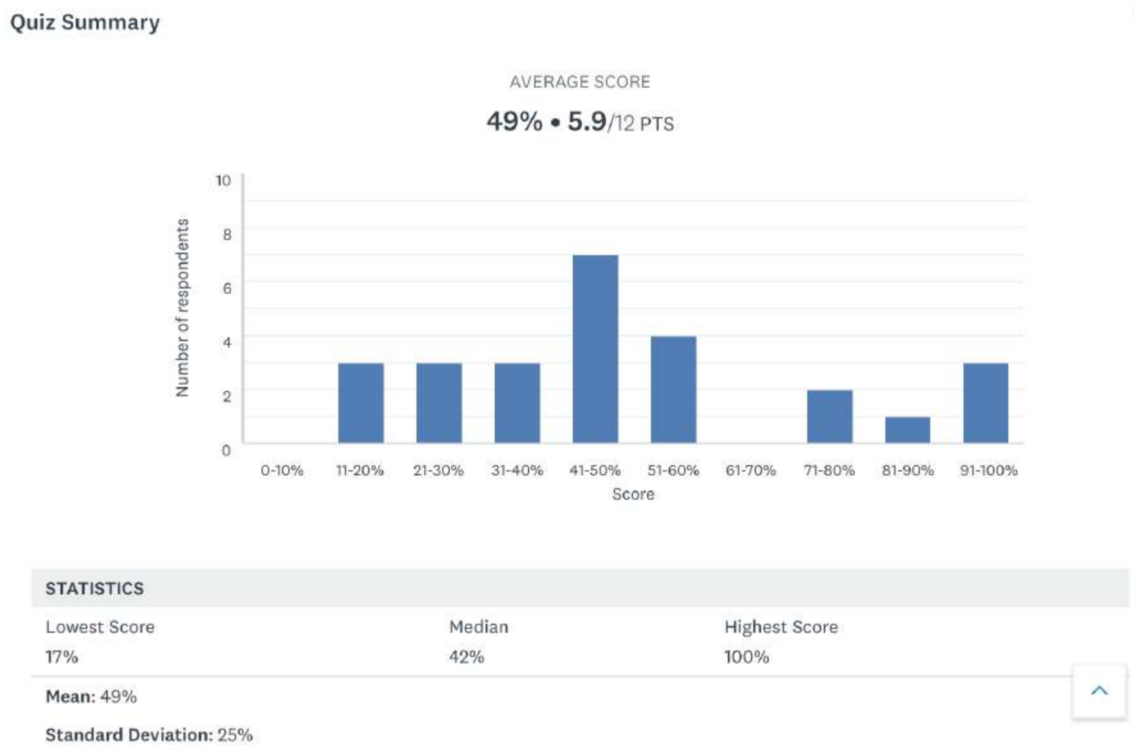
The survey would be created as a quiz, where 12 questions would be given, each question containing two pictures, one real and one generated by our model. The participant has to identify the real picture in each question, out of 12. After they complete all questions, an average score is given to participants. If they failed to identify below 50% of real pictures then the user has failed the quiz, meaning that they were unable to identify the real pictures and the generated pictures are as good as the pictures that are drawn by real animators. Average scores would be taken from all participants after each quiz test and then an average score will be collected from quiz results from all participants. If the average score is below 50% then the hypothesis was successful.

Note: The questions would be randomized for every participants and only 12 questions are given, to reduce the time to 2 minutes on average to complete one quiz, this way the quiz should get 100% completion rates from all participants and provide a meaningful result.

5.5.1 User Study

The online quiz can be accessed through Appendix section 7.1.6.

Online quiz was created and it was shared to random participants, majority of them were students, who claimed to have watched anime at-least once. No personal identification was collected.



STATISTICS		
Lowest Score	Median	Highest Score
17%	42%	100%

Mean: 49%

Standard Deviation: 25%

Figure 5-16: Quiz result, 49% average score from 26 participants.

Figure displays the average score was 49% out of 26 participants. Only 6 out of 26 participants managed to get more than 70%, 2. The generated samples given to participants were generated from truncation ψ = 1.2, 1, 0.7, 0.5, 0, -0.5, and -1, best of 12 out of them were manually selected. For the real samples, 12 best images were selected as they were drawn by real animators and up-scaled. The quiz was also sent to random individuals and they passed it on to their friends and families, this was to remove any biases getting into the overall result.

Overall, the result was surprising. The samples generated by StyleGAN were of really quality and contained a range of diversity in them. However, it was not expected that it would manage to fool participants who were familiar with anime and they couldn't tell that those samples were fake and generated by the trained model. This result of this user study proved that hypothesis was correct and adds more validity to the research project.

5.6 Discussion

Overall, the development and results demonstrated that with a high quality dataset, StyleGAN can successfully trained on anime images and managed to generate state of the art anime faces. The result of user study confirms that the generated samples are good enough for to be used in the anime industry as a new character or as an inspiration to create a new character by animators and validates our hypothesis, as anime audiences were not able to identify between real images drawn by animators and anime images generated by our trained model. This also validates the research work undertaken in this project from development to training a novel StyleGAN anime model, which can generate undisguisable anime faces.

There are still room for improvements. Firstly, the dataset contained more female faces than male faces. Although that was desirable and fitting for the problem, it is suspected that more or at-least even male and females faces would have added even more diversity to the samples generated by the trained model. There were only about 5% of male faces in the dataset and it was suspected that it is the reason why the generator morphed all male faces into female faces. Overall, this did not cause any issues to the trianing but it would be interesting to see the outcome, had there been a dataset of even gender faces.

The limitations were tackled as they were discovered during the development process and are discussed in development section 4. However, there were few unpredictable circumstances. For example, the StyleGAN dataset type (.tfRecords) took longer to convert than expected. It caused more issues as the overall size was going over 200GB and caused it to crash the first time. It was because it was being converted to home directory, where the maximum storage allowed was only 210GB. The first conversion took 6 hours until it crashed and had to be restarted again from scratch, directing the conversion tool to saved the converted datatype into another directory with 3TB storage space. The whole process was tedious and the overall conversion time was more than 20hours.

Even with the hyper-parameters tuning in section 5.5, the model was not able to converge. It may be just question of more training time required with even more lower learning rates, but because of time constraint, the training had to be stopped. It would be interesting to see what the results would look like if the model had actually converged because even without convergence, the images generated by the model are indistinguishable from the real ones.

StyleGAN is computationally expensive to train. Significant time and effort was during training and experimenting with hyper-parameters. Learning rates 0.003 proved to be too powerful, causing memory fault and forcing the GPU reaper to kill the trianing process. Furthermore, after adjusting the learning rates, one would need to wait for several GPU hours, 6-8 hours at least, to see any real changes to the quality of the images. When the artifacts started to appear during later stages in the training, 8 days were wasted, trying to find the right learning rates to find the right balance of Generator and Discriminator. Finally, the right balance was after 15 days of observation into losses of G/D, in Tensorboard. It maybe because of StyleGAN uses progressing growing technique, growing from lower resolution to reaching highest and performing then starting the main phase of the training. During the development of the project, StyleGAN 2 was released. StyleGAN 2 removed progressive growing completely, which could have faster response to learning rates, at more performance/computation cost [KLA19a].

Chapter 6

Conclusions

Rigorous investigation of Generative Adversarial Networks, deep learning algorithms, utilising Artificial Neural Networks in image generation, showed that StyleGAN was the best algorithm to use for image generation because of it's state of the art style transfer architecture. In addition, investigation of related works in application of GANs in animation revealed that there were only few solution that attempted to generate anime/animation characters and no attempts had been made with StyleGAN. Therefore, StyleGAN was chosen as the algorithm to be used to generate anime images.

Extensive investigation was conducted on StyleGAN to find out important aspects to help in training, such as the requirements, hyper-parameters and dataset etc. More extensive effort was put into developing a custom high quality dataset, which was used to train StyleGAN, after investigation found no appropriate dataset that was high quality and suitable to train on, as needed by StyleGAN requirements.

Significant time was spent on training and experimenting with StyleGAN, to generate indistinguishable anime faces. A user study was conducted after final training phase and generating samples to test the hypothesis that users will not be able to tell difference between real anime images and images generated by our StyleGAN model. The user study concluded that the audience/participants were unable to identify between real images drawn by animators and images generated by the our trained model, with an average score of 49% out of 100%. The average score was based on 26 participants. More responses from more participants could have been gathered. However, due to the time limit of the project, 26 participants was considered sufficient enough to evaluate the project, as most participants were familiar with anime genre.

To conclude, the project successfully implemented and trained StyleGAN on a custom created dataset to generate indistinguishable fake anime faces from real anime faces drawn by animators. The research, development and results have been documented in detail. This project concludes that StyleGAN can be used as a deep learning tool to generate indistinguishable fake anime faces, which could be used as inspiration in creating new characters, for the animators.

6.0.1 Future work

A variety of improvements and areas of investigation are identified:

- The custom dataset could be expanded to include more male faces to bring balance between in generated pictures. The dataset contained more female faces than male faces. If the dataset was expanded to bring balance, it could have added even more diversity to the generated pictures. Due to the time constraint, this was not explored.

  During the development of the project, StyleGAN 2 was released. StyleGAN2 removed progressive growing completely, which could have faster response to learning rates, at more performance/computation cost. It would be interesting to see how it performs against the model trained by StyleGAN.

- It would be interesting to find out why the strange artifacts appeared in later stages of the training. StyleGAN's paper suggested it could be due to low training image dataset. However, the dataset was prepared manually with extensive work spent on clean and up-scaling. Furthermore, the artifacts disappeared when trained on lower leaning rates for 2 weeks. Therefore, it is definitely not low quality dataset and it would be interesting to find out exactly why.

Bibliography

[ACB17] Arjovsky M., Chintala S., Bottou L.: Wasserstein gan. *arXiv preprint arXiv:1701.07875.s* (2017).

[AcB20] Anonymous, community D., Branwen G.: Danbooru2019: A large-scale crowdsourced and tagged anime illustration dataset. https://www.gwern.net/Danbooru2019, January 2020. Accessed: 02-06-2020. URL: https://www.gwern.net/Danbooru2019.

[AKTK19] Alqahtani H., Kavakli-Thorne M., Kumar G.: An analysis of evaluation metrics of gans. *International Conference on Information Technology and Applications (ICITA)* (2019).

[AMI] AMIDI A.: Frozen" head of animation says animating women is "really, really difficult. URL: https://www.cartoonbrew.com/disney/frozen-head-of-animation-says-animating-women-is-really-really-difficult-89467.html.

[AMSS17] Albawi S., Mohammed A., Saad T., Saad A.: Understanding of a convolutional neural network. *10.1109/ICEngTechnol.2017.8308186* (2017).

[Ara19] Arasanipalai A. U.: How to generate game of thrones characters using stylegan, 2019. URL: https://nanonets.com/blog/stylegan-got/#stylegan.

[Bar18] Barnett S. A.: Convergence problems with generative adversarial networks (gans). *arXiv:1806.11382* (2018).

[Bat] Bath U. O.: Balena hpc cluster. URL: https://www.bath.ac.uk/corporate-information/balena-hpc-cluster/.

[Bon] Bond J.-M.: Why anime is more popular now than ever. URL: https://www.dailydot.com/parsec/what-is-anime/.

[Bor19] BORJI A.: Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding 179* (2019), 41–65.

[Com] COMMONS C.: What our licenses do. URL: `https://creativecommons.org/licenses/`.

[DCR15] DENTON E. L., CHINTALA S., R. F.: Deep generative image models using a laplacian pyramid of adversarial networks. *Advances in neural information processing systems* (2015), pp. 1486–1494.

[DLKS18] DUTTA J. K., LIU J., KURUP U., SHAH M.: Effective building block design for deep convolutional neural networks using search. *arXiv preprint arXiv:1801.08577* (2018).

[DSKA17] DRISS S. B., SOUA M., KACHOURI R., AKIL M.: A comparison study between mlp and convolutional neural network models for character recognition. *In Real-Time Image and Video Processing 2017 (Vol. 10223, p. 1022306). International Society for Optics and Photonics* (2017).

[Eva18] EVANS J.: Tips for dealing with stress in the animation industry, 2018. URL: `https://animatedjobs.com/animationjobs/tips-for-dealing-with-stress-in-the-animation-industry/`.

[Ewr] EWRFCAS: Anime-gan-tensorflow. URL: `https://github.com/ANIME305/Anime-GAN-tensorflow`.

[Fag20] FAGGELLA D.: What is machine learning?, 2020. URL: `https://emerj.com/ai-glossary-terms/what-is-machine-learning/`.

[FYX15] FISHER YU ARI SEFF Y. Z.-S. S.-T. F., XIAO J.: Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop, 2015. URL: `https://www.yf.io/p/lsun`.

[Ger20] GERGANA: Intro to github for version control, 2020. URL: `https://ourcodingclub.github.io/tutorials/git/`.

[Glo20] GLOBAL L.: How amgi animation is adapting to covid-19 challenges, 2020. URL: `https://www.licenseglobal.com/streaming-and-tv/how-amgi-animation-adapting-covid-19-challenges/`.

[GPAM*] GOODFELLOW I., POUGET-ABADIE J., MIRZA M., XU B., WARDE-FARLEY D., OZAIR S., BENGIO

Y.HTTPS://WWW.CARTOONBREW.COM/DISNEY/FROZEN-HEAD-OF-ANIMATION-SAYS-ANIMATING-WOMEN-IS-REALLY-REALLY-DIFFICULT-89467.HTML T. J. V. Y. P.:.

[Gri] GRISCOM D.: What does it mean for an algorithm to converge? URL: https://softwareengineering.stackexchange.com/questions/288777/what-does-it-mean-for-an-algorithm-to-converge.

[Gri18] GRIGNON R.: These five trends are rocking the animation industry, 2018. URL: https://techcrunch.com/2018/06/26/these-five-trends-are-rocking-the-animation-industry/.

[HAR19] HARGRAVE M.: Deep learning, 2019. URL: https://www.investopedia.com/terms/d/deep-learning.asp.

[HLL19] HO D., LIANG E., LIAW R.: Why should you care about data augmentation?, 2019. URL: https://bair.berkeley.edu/blog/2019/06/07/data_aug/#:~:text=Data%20augmentation%20is%20a%20strategy,to%20train%20large%20neural%20networks.

[Hor18] HOREV R.: Explained: A style-based generator architecture for gans - generating and tuning realistic artificial faces, 2018. URL: https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431.

[Hui18a] HUI J.: Gan — dragan, 2018. URL: https://medium.com/@jonathan_hui/gan-dragan-5ba50eafcdf2.

[Hui18b] HUI J.: Gan — why it is so hard to train generative adversarial networks!, 2018. URL: https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b/.

[Imp]

[Iva17] IVARS N.: Deep convolutional neural networks: Structure, feature extraction and training. *Information Technology and Management Science. 20. 10.1515/itms-2017-0007* (2017).

[KAHK17] KODALI N., ABERNETHY J., HAYS J., KIRA Z.: On convergence and stability of gan. *arXiv preprint arXiv:1705.07215* (2017).

[KALL17] Karras T., Aila T., Laine S., Lehtinen J.: Progressive growing of gans for improved quality, stability, and variation. *arXiv:1710.10196 1* (2017). doi:https://arxiv.org/pdf/1710.10196.pdf.

[KALL18] Karras T., Aila T., Laine S., Lehtinen J.: Progressive growing of gans for improved quality, stability, and variation – official tensorflow implementation of the iclr 2018 paper, 2018. URL: https://github.com/tkarras/progressive_growing_of_gans.

[KB17] Kingma D. P., Ba J. L.: Adam: A method for stochastic optimization. *arXiv:1412.6980* (2017). doi:https://arxiv.org/abs/1412.6980.

[KLA19a] Karras T., Laine S., Aila T.: A style-based generator architecture for generative adversarial networks. *arXiv:1812.04948v3 2* (2019). doi:https://arxiv.org/pdf/1812.04948.pdf.

[KLA19b] Karras T., Laine S., Aila T.: A style-based generator architecture for generative adversarial networks, 2019. URL: https://www.youtube.com/watch?time_continue=84&v=kSLJriaOumA&feature=emb_title.

[Lar] Larramo M.: Which image format is best for email newsletters: Jpeg, gif or png? URL: https://www.samlogic.net/articles/which-image-format-best-for-email-newsletters-jpeg-gif-png.html.

[LGT16] Lee C. Y., Gallagher P. W., Tu Z.: Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. *Artificial intelligence and statistics (pp. 464-472)* (2016).

[Lil19] Lilian W.: From gan to wgan, 2019. arXiv:1904.08994.

[LLC] LLC I. S.: Imagemagick. URL: https://imagemagick.org/index.php.

[Lur] Lurie Z.: Surveymonkey® - surveymonkey.co.uk. URL: https://www.surveymonkey.co.uk/.

[Lyr18] LyrnAI: Style-based gans – generating and tuning realistic artificial faces, 2018. URL: https://www.lyrn.ai/2018/12/26/a-style-based-generator-architecture-for-generative-adversarial-networks/.

[Lyr19] LyrnAI: Stylegan architecture mapping network, 2019. URL: https://www.lyrn.ai/wp-content/uploads/2018/12/StyleGAN-generator-Mapping-network.png.

[Mag20] MAGAZINE A.: Updating: Covid-19 risk shuts down global festivals, cons, conferences, 2020. URL: `https://www.animationmagazine.net/events/covid-19-risk-shuts-down-global-festivals-cons-conferences/`.

[Mar17] MARKETS R.: Global animation industry 2017: Strategies trends opportunities report - the most authoritative global animation industry analysis - research and markets, 2017. URL: `https://www.prnewswire.com/news-releases/global-animation-industry-2017-strategies-trends--opportunities-report---the-most-authoritative-global-animation-industry-analysis---research-and-markets-300391335.html`.

[Mar19] MARGOLIS E.: The dark side of japan's anime industry, 2019. URL: `https://www.vox.com/culture/2019/7/2/20677237/anime-industry-japan-artists-pay-labor-abuse-neon-genesis-evangelion-netflix#:~:text=Anime%20brings%20in%20more%20than%20%2419%20billion%20a%20year`.

[mg17] MOXIE GUSHI: pokegan, 2017. URL: `https://github.com/moxiegushi/pokeGAN`.

[Mil19] MILENKOVIC M.: Worrying workplace stress statistics. retrieved from the american institure of stress, 2019. URL: `https://www.stress.org/42-worrying-workplace-stress-statistics`.

[MPPD17] METZ L., POOLE B., P.FAU D., DICKSTEIN J. S.: Unrolled generative adversarial networks. *arXiv:1611.02163v4* (2017).

[Nag] NAGADOMI: Image super-resolution for anime-style art.

[nag18] NAGADOMI: A face detector for anime/manga using opencv, 2018. URL: `https://github.com/nagadomi/lbpcascade_animeface`.

[Nou] NOURIE A.: If i wanted to make an animated tv show, what would i need to get started? URL: `https://www.quora.com/If-I-wanted-to-make-an-animated-TV-show-what-would-I-need-to-get-started`.

[NP] NVIDIA R., PROJECTS: Stylegan - official tensorflow implementation.

[Oli05] OLIVA A.: Gist of the scenes. *Neurobiology of attention* (2005), pp. 251–256.

[ope] Open cv. URL: `https://opencv.org/`.

[Rag09] Raghavan T.: Zero-sum two person games. 10073–10096.

[RCA*18] Rui X., Changying L., Andrew P., Yu J., Shangpeng S., Jon R.: Aerial images and convolutional neural network for cotton bloom detection. *Frontiers in Plant Science. 8. 10.3389/fpls.2017.02235* (2018).

[RCKS] Ronan, Clément, Koray, Soumith: What is torch?

[RL] Robert Ross M.-V., Ludovic: Image generation. URL: `https://paperswithcode.com/task/image-generation`.

[RMC15] Radford A., Metz L., Chintala S.: Self-attention generative adversarial networks. *arXiv:1511.06434 1* (2015). `doi:https://arxiv.org/abs/1511.06434`.

[SDBR15] Springenberg J. T., Dosovitskiy A., Brox T., Riedmiller M.: Striving for simplicity: The all convolutional net. *arXiv:1412.6806 1* (2015). `doi:https://arxiv.org/pdf/1412.6806.pdf`.

[SRL*17] Snell J., Ridgeway K., Liao R., Roads B. D., Mozer M. C., Zemel R. S.: Learning to generate images with perceptual similarity metrics. *IEEE International Conference on Image Processing* (2017), 4277–4281.

[Sta20] Standard B.: Covid-19 spells disaster for film industry, but boosts digital media: Kpmg, 2020. URL: `https://www.business-standard.com/article/companies/covid-19-spells-disaster-for-film-industry-but-boosts-digital-media-kpmg-120041400383_1.html/`.

[Sum18] Sumit S. H.: Drawbacks of convolutional neural networks, 2018. URL: `https://sakhawathsumit.github.io/sumit.log/2018/07/21/drawbacks-of-convolutional-neural-networks.html`.

[Sur19a] Surma G.: Dcgan image generator, 2019. URL: `https://github.com/gsurma/image_generator`.

[Sur19b] Surma G.: Generating simpsons with dcgans, 2019. URL: `https://towardsdatascience.com/image-generator-drawing-cartoons-with-generative-adversarial-networks-45e814ca9b6`.

[TB17] Turhan C. G., Bilge H. S.: Generative adversarial networks: introduction and outlook. *Journal of Automatica Sinica (IEEE/CAA) 4(4)* (2017), 588–598.

[TB18] Turhan C. G., Bilge H. S.: Recent trends in deep generative models: a review. *International Conference on Computer Science and Engineering (IEEE)* (2018), 574–579.

[T.d] T.d R.: Illustrationgan. URL: `https://github.com/tdrussell/IllustrationGAN`.

[tea] team F.: about ffmpeg. URL: `https://ffmpeg.org/about.html`.

[ten] Why tensorflow. URL: `https://www.tensorflow.org/about`.

[TOB15] Theis L., Oord A. V. D., Bethge M.: A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844* (2015).

[Tok19] Tokis K.: Simpsons faces, 2019. URL: `https://www.kaggle.com/kostastokis/simpsons-faces`.

[TW16] Taigman Y., A., Wolf L.: Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200* (2016).

[Wat] Watch M.: Impact of covid-19 on 3d animation market - exclusive report by transparency market research. URL: `https://www.marketwatch.com/press-release/impact-of-covid-19-on-3d-animation-market---exclusive-report-by-transparency-market-research-2020-04-23?mod=mw_quote_news`.

[Wen18] Weng L.: Attention? attention! *lilianweng.github.io/lil-log* (2018). URL: `http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html`.

[WGGH17] Wang X., Girshick R., Gupta A., He K.: Non-local neural networks. *arXiv:1711.07971 1* (2017). `doi:https://arxiv.org/abs/1711.07971`.

[Wu17] Wu J.: Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China, 5, 23* (2017).

[YSZ*20] Yu F., Seff A., Zhang Y., Song S., Funkhouser T., Xiao J.: Index of /lsun/objects/, 2020. URL: `http://dl.yf.io/lsun/objects/`.

[ZGMO18] Zhang H., Goodfellow I., Metaxas D., Odena A.: Self-attention generative adversarial networks. *rXiv:1805.08318v1 2* (2018). `doi:https://arxiv.org/abs/1805.08318v1`.

Chapter 7

Appendix

7.1 Supplementary Materials

7.1.1 Dataset

Full custom cleaned dataset was performed on the GPU cloud, then zipped and uploaded to google drive, it can be accessed through: `https://drive.google.com/file/d/1J2ioU9RYlesddG7m5ocbySyTkVNrzjQc/view?usp=sharing`

7.1.2 GitHub

The code was uploaded and version control was utilised: `https://github.com/AsisRai/MSc_Dissertation`

7.1.3 StyleGAN Files

The files includes StyleGAN codes and the training results: `https://drive.google.com/file/d/1yXhBIcvwdvGL2jy-GKM1QJxI5wr_YV5f/view?usp=sharing`

7.1.4 Training pictures

All training pictures are zipped and uploaded to google drive: `https://drive.google.com/file/d/1--kJHl42PG1nl3mbxc4aKPKXWK_VWTQY/view?usp=sharing`

7.1.5 Training Montage

There are 4 montage files prepared:

- 10k iterations: `https://drive.google.com/file/d/1EQ1B4hfyBorWeWXyILJZ1Hvdp2HnDtFU/view?usp=sharing`
- 14k iterations: `https://drive.google.com/file/d/1xSQyVdP08f79J_XJE8BdRaQLlHKsnVK2/view?usp=sharing`
- 49k iterations: `https://drive.google.com/file/d/10sRVyv--LbjhBHbsxRM2GSvQMMhvJBQF/view?usp=sharing`
- 61k iterations: `https://drive.google.com/file/d/1yoqzdy6Tz9BP6z5wbr81c7AJ7_0v0J0K/view?usp=sharing`

7.1.6 Questionare Quiz

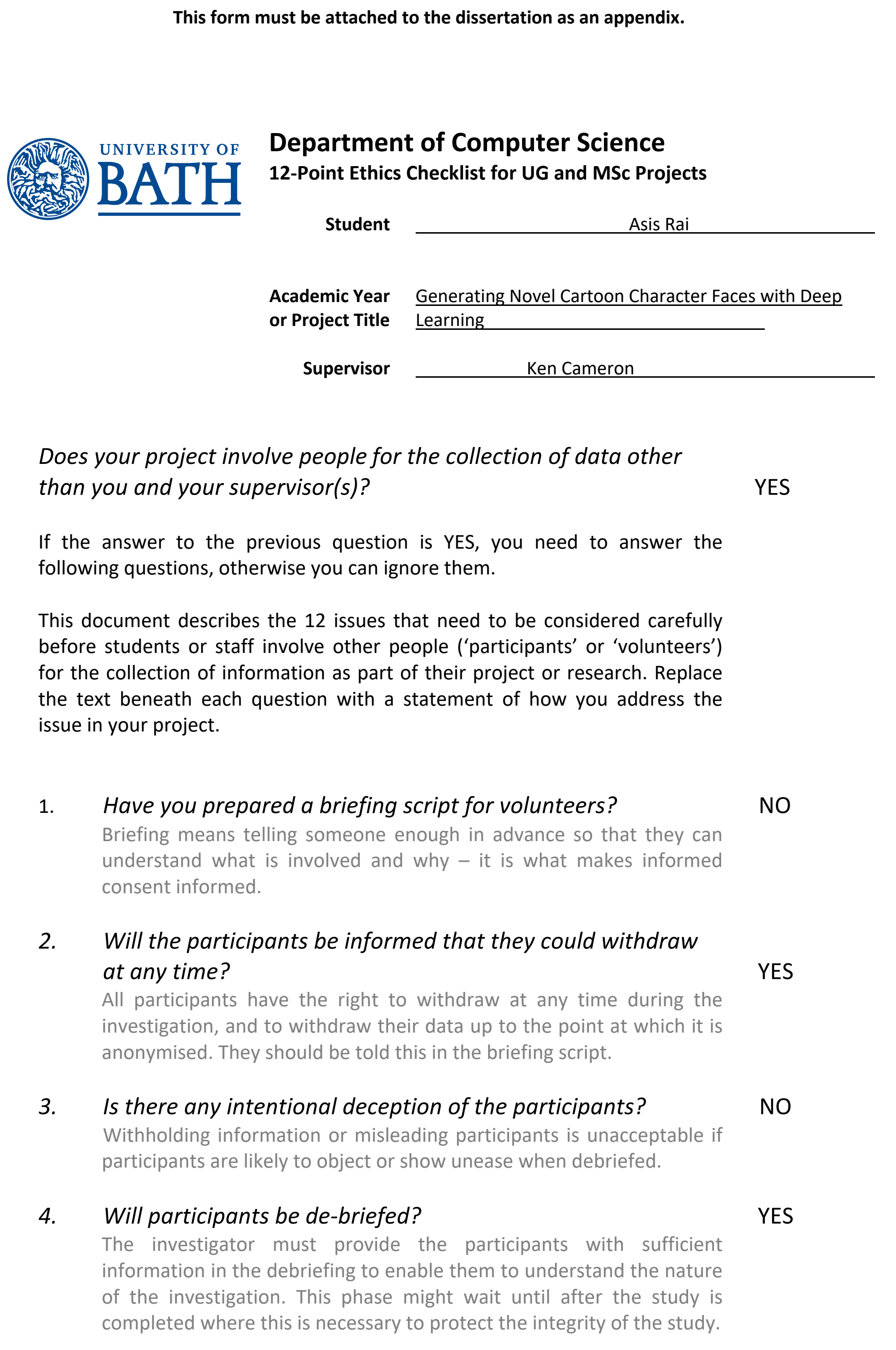
The survey can be accessed through this link: `https://www.surveymonkey.co.uk/r/3TLP96W`

7.1.7 Randomly generated samples

6000 samples generated from truncation $\psi = 1.2\psi = 1, \psi = 0.7, \psi = 0.5, \psi = 0, \psi = -0.5, \quad and \quad \psi = -1$, 1000 samples per truncation.

Samples can be access on: `https://drive.google.com/file/d/1L6wv66Q07J13S24Fs3aO3Kk6iUTG80xe/view?usp=sharing`

7.2 12-Point Ethics Checklist



This form must be attached to the dissertation as an appendix.

UNIVERSITY OF BATH

Department of Computer Science

12-Point Ethics Checklist for UG and MSc Projects

Student	Asis Rai
Academic Year or Project Title	Generating Novel Cartoon Character Faces with Deep Learning
Supervisor	Ken Cameron

Does your project involve people for the collection of data other than you and your supervisor(s)? YES

If the answer to the previous question is YES, you need to answer the following questions, otherwise you can ignore them.

This document describes the 12 issues that need to be considered carefully before students or staff involve other people ('participants' or 'volunteers') for the collection of information as part of their project or research. Replace the text beneath each question with a statement of how you address the issue in your project.

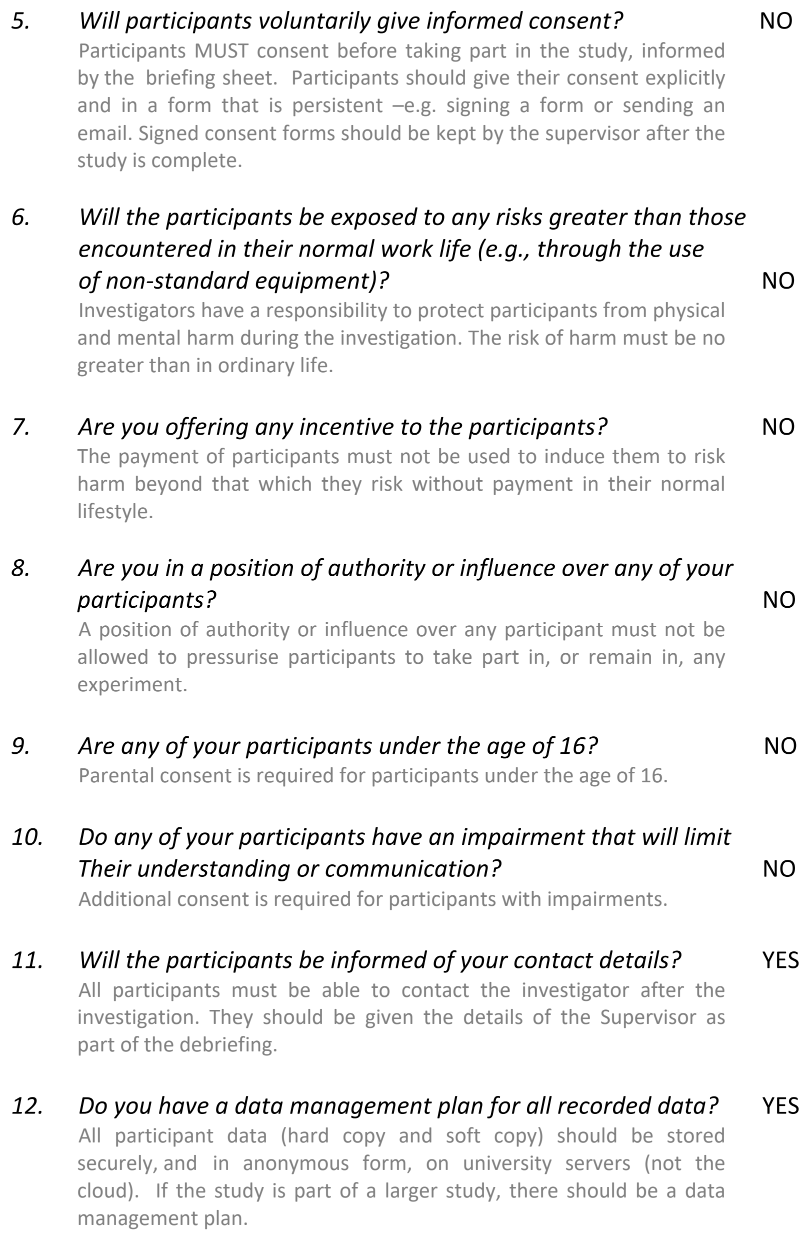
1. *Have you prepared a briefing script for volunteers?* NO
   Briefing means telling someone enough in advance so that they can understand what is involved and why – it is what makes informed consent informed.

2. *Will the participants be informed that they could withdraw at any time?* YES
   All participants have the right to withdraw at any time during the investigation, and to withdraw their data up to the point at which it is anonymised. They should be told this in the briefing script.

3. *Is there any intentional deception of the participants?* NO
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

4. *Will participants be de-briefed?* YES
   The investigator must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. This phase might wait until after the study is completed where this is necessary to protect the integrity of the study.

Figure 7-1: 12-point Ethics checklist page 1



5. *Will participants voluntarily give informed consent?* NO
Participants MUST consent before taking part in the study, informed by the briefing sheet. Participants should give their consent explicitly and in a form that is persistent –e.g. signing a form or sending an email. Signed consent forms should be kept by the supervisor after the study is complete.

6. *Will the participants be exposed to any risks greater than those encountered in their normal work life (e.g., through the use of non-standard equipment)?* NO
Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.

7. *Are you offering any incentive to the participants?* NO
The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

8. *Are you in a position of authority or influence over any of your participants?* NO
A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. *Are any of your participants under the age of 16?* NO
Parental consent is required for participants under the age of 16.

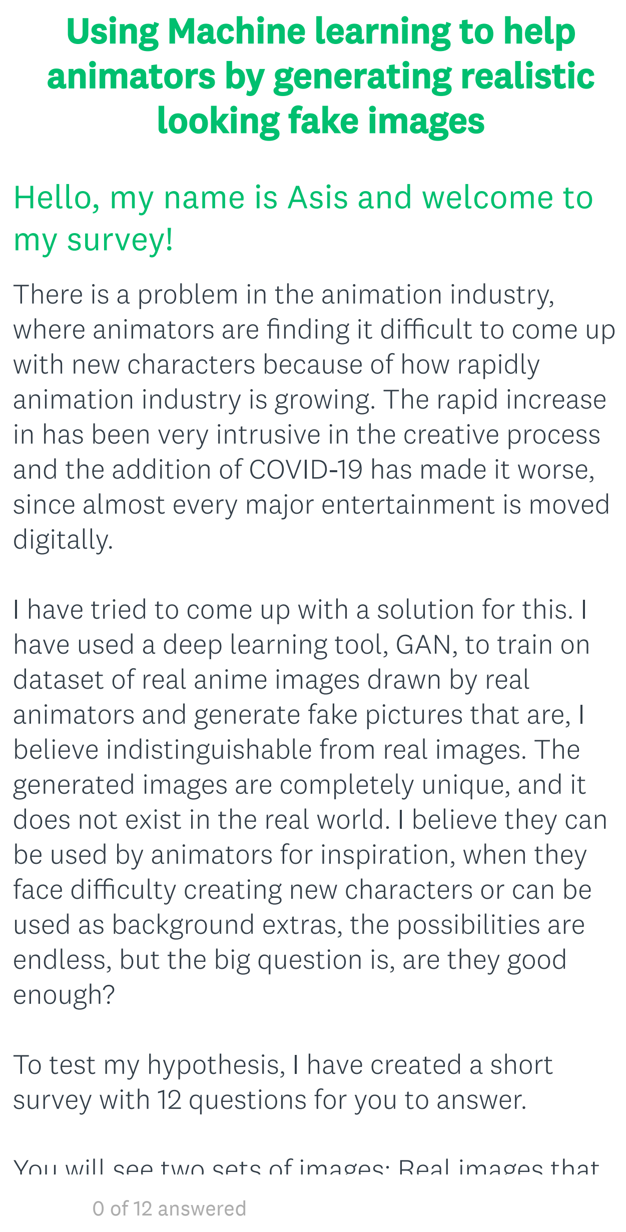
10. *Do any of your participants have an impairment that will limit Their understanding or communication?* NO
Additional consent is required for participants with impairments.

11. *Will the participants be informed of your contact details?* YES
All participants must be able to contact the investigator after the investigation. They should be given the details of the Supervisor as part of the debriefing.

12. *Do you have a data management plan for all recorded data?* YES
All participant data (hard copy and soft copy) should be stored securely, and in anonymous form, on university servers (not the cloud). If the study is part of a larger study, there should be a data management plan.

Figure 7-2: 12-point Ethics checklist page 2

7.2.1 Briefing script for Participants



Using Machine learning to help animators by generating realistic looking fake images

Hello, my name is Asis and welcome to my survey!

There is a problem in the animation industry, where animators are finding it difficult to come up with new characters because of how rapidly animation industry is growing. The rapid increase in has been very intrusive in the creative process and the addition of COVID-19 has made it worse, since almost every major entertainment is moved digitally.

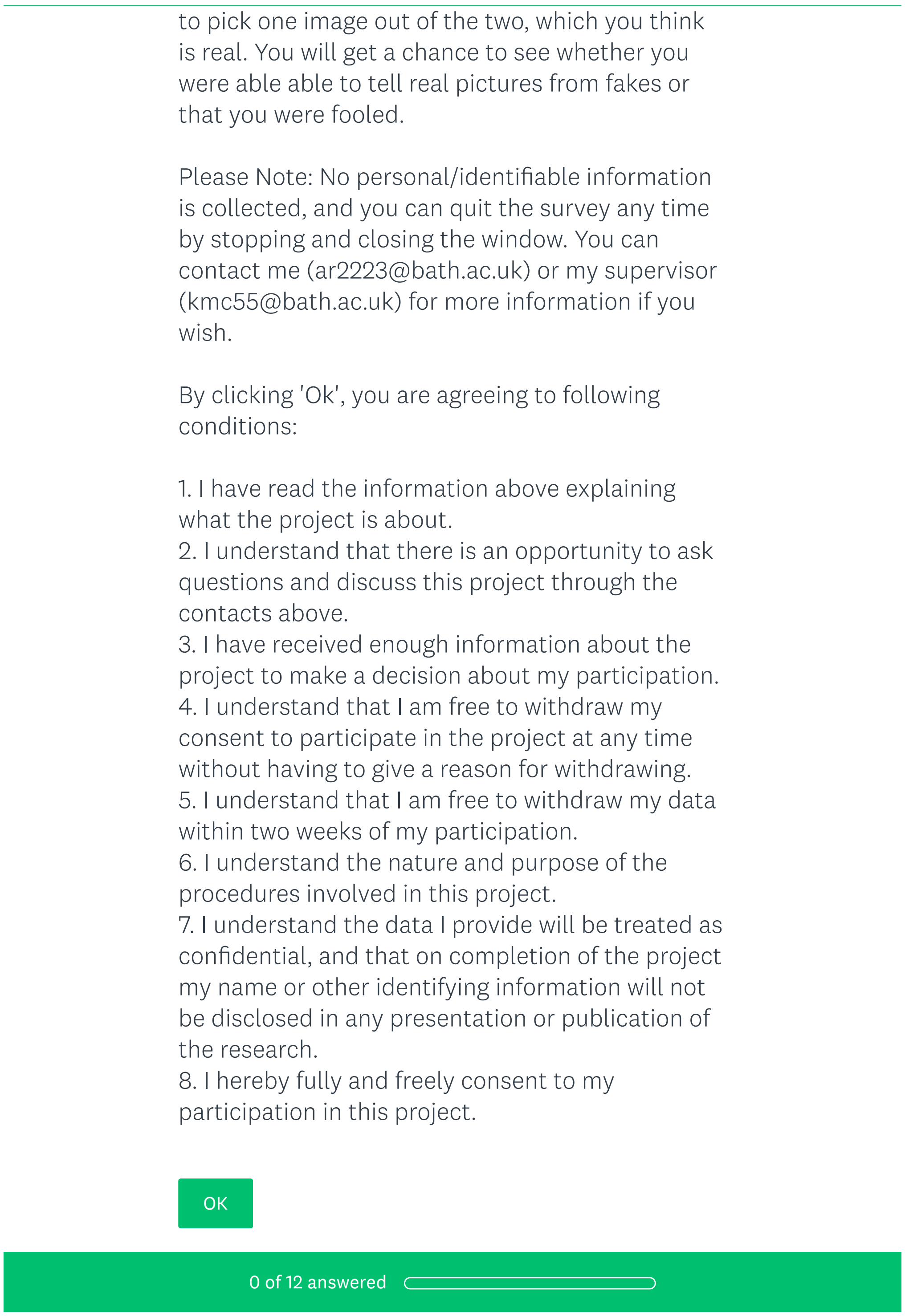
I have tried to come up with a solution for this. I have used a deep learning tool, GAN, to train on dataset of real anime images drawn by real animators and generate fake pictures that are, I believe indistinguishable from real images. The generated images are completely unique, and it does not exist in the real world. I believe they can be used by animators for inspiration, when they face difficulty creating new characters or can be used as background extras, the possibilities are endless, but the big question is, are they good enough?

To test my hypothesis, I have created a short survey with 12 questions for you to answer.

You will see two sets of images: Real images that

0 of 12 answered

Figure 7-3: Briefing script, Survey Monkey.



to pick one image out of the two, which you think is real. You will get a chance to see whether you were able able to tell real pictures from fakes or that you were fooled.

Please Note: No personal/identifiable information is collected, and you can quit the survey any time by stopping and closing the window. You can contact me (ar2223@bath.ac.uk) or my supervisor (kmc55@bath.ac.uk) for more information if you wish.

By clicking 'Ok', you are agreeing to following conditions:

1. I have read the information above explaining what the project is about.
2. I understand that there is an opportunity to ask questions and discuss this project through the contacts above.
3. I have received enough information about the project to make a decision about my participation.
4. I understand that I am free to withdraw my consent to participate in the project at any time without having to give a reason for withdrawing.
5. I understand that I am free to withdraw my data within two weeks of my participation.
6. I understand the nature and purpose of the procedures involved in this project.
7. I understand the data I provide will be treated as confidential, and that on completion of the project my name or other identifying information will not be disclosed in any presentation or publication of the research.
8. I hereby fully and freely consent to my participation in this project.

Figure 7-4: Briefing script, Survey Monkey.