Department of Computer Science

Submitted in part fulfilment for the degree of BEng.

Generating Images Using Generative Deep Learning

How batch and window sizes affect image quality in Swin Transformer GANs - Revision 1.1

James Burnell

2023-May (Revised 2023-July)

Supervisor: Adrian Bors

Acknowledgements

I would like to thank my project supervisor Dr. Adrian Bors for his assistance throughout this project. I would also like to thank my family and friends for their support.

This project was undertaken on the Viking Cluster, which is a high performance compute facility provided by the University of York. We are grateful for computational support from the University of York High Performance Computing service, Viking and the Research Computing team.

Contents

List of Figures

List of Tables

Executive Summary

There are many methods available to generate images, and new ones are being developed constantly. Because of this, the need to investigate which models perform the best arises. A machine learning module called a Shifted-Window Transformer was developed in 2021 for use in computer vision. It uses shifted attention windows to limit the focus to sections of an image rather than the whole image at once. By doing so, it is able to greatly reduce computation and memory complexity compared to a normal Transformer. This project aims to compare the performance of nine Shifted-Window Transformer-based Generative Adversarial Networks (SwinTransGANs) that have varying window and batch sizes. Three window sizes and three batch sizes were used to generate a total of nine, separate models to evaluate the effect they have on training time, memory usage, and quality of the generated images. These results were then compared to a conventional Wasserstein GAN. Further investigation was carried out into the effects of training with a reduced learning rate.

Image generation can be deceptive as it can fool people into believing things that may not be accurate. However, this project focusses on generating novel faces and does not try to replicate faces of existing people. The CelebA[1] dataset was used, which is a collection of over $200,000$ publicly available images of faces taken from the internet that categorizes each image by 40 different attributes.

A standard DCGAN discriminator was used to classify the generator output. The generator was comprised of three blocks, each containing positional embedding, two Swin transformer layers, and an upscaler. The generator accepts a latent vector of size 100 then upscales it to $(8 \times 8) \times 512$ so it can be passed into in the transformers. Binary Cross-Entropy was used as the criterion function to determine the generator and discriminator loss. This function punishes the model the further away from the true classification it is. All models were trained for 40 epochs on the University of York's G-Viking server using their Nvidia A40 GPUs.

It was found that a window size of 8 and batch size of 100 (model 8-100) was able to generate the best results after 40 epochs (356 minutes) with an FID score of 88.4. The worst model had a window size of 8 and a batch size of 200 (model 8-200) and scored 180.6 FID after 40 epochs (287 minutes).

The number of parameters were the same across all models ($10,196,472$ for the generator and $1,788,609$ for the discriminator), and the allocated memory was consistent between models. The allocated memory increased as batch size increased due to the use of a minibatch discriminator. Sample images for all models can be found in Appendix B. It was observed that increasing the window size increased the instability in the FID score for the different batch sizes. The range between the highest and lowest FID score for window size 8 was found to be 92.2, whereas for window size 2, this was only 10.8. Additional testing observed that there is a correlation between batch size and training instability (measured by mean absolute percentage error of FID scores over multiple training sessions). Models with a window size of 4 trained faster than those with a size of 2 or 8.

It was observed that the models failed to generate faces that contained attributes that were a small proportion of the dataset at the same level of quality as attributes that had a higher presence in the dataset. Female faces were produced at a higher quality more often than male faces, but the most notable observation was the models' failure to generate faces with darker skin. This is a wider issue in machine learning as many datasets have unbalanced attributes which lead to models with implicit biases. However, these faults are outside the scope of this investigation.

Future investigation could be undertaken to study what effect learning rate, attention head count, and transformer layer count have on the results. Using a Wasserstein discriminator would also be a good area of investigation as they may be able to resolve the mode collapse issue and generate better results than the currently-used DCGAN discriminator. Further investigation into balancing the dataset attributes would also be beneficial to combat the bias in results and to generate higher-quality images.

1 Introduction

1.1 Problem Overview & Motivation

The synthesis of artificial images has been a topic of great interest over the last few years. Machine learning models such as Variational Auto-Encoders (2013)[2], Generative Adversarial Networks (2014)[3], and Diffusion (2020)[4] have been used to generate novel images based on data from training sets. Deep learning image generation can be used to create new textures, create new artworks, transfer style from one image to another, create new images of objects and people, and much more. These tools allow the general public to enhance their creative abilities. Commercial settings such as game development, marketing, product design, etc can also utilize these tools to help prototype or design final assets for games, marketing campaigns, consumer products, and more.

New machine learning modules are constantly being introduced, leading to new ways to generate images all the time. Because of this constant research and development, the need to investigate their functionalities to find better solutions is continuous.

Although diffusion is a newer technique and has been shown to produce high-quality results, GANs are still faster at generating images. To better understand how to improve upon GANs, this project will investigate how batch size and window size affects the IS and FID scores.

1.2 Objectives

Shifted-Window Transformers are a recent development, only being introduced in 2021. This project aims to explore the use of Shifted-Window Transformers in combination with Generative Adversarial Networks to generate images of human faces. The project will compare the effects of varying the transformer window size and varying the training batch size. Memory usage, training time, and quality scores will be collected and computed for each model. Image quality will be evaluated using Inception Score and the Fréchet Inception Distance. The results from these models will be compared against a Wasserstein GAN which is an established architecture type. From this, the project will look for trends and patterns within the data. Since Transformers have been shown to have a strong modelling power for other tasks, a good outcome will be the models outperforming the older method of Wasserstein GANs.

1.3 Report Structure

- Chapter 2 (Literature Review) contains an analysis of existing literature surrounding the topic of GANs in image generation. It also includes a discussion comparing these methods.
- Chapter 3 (Method) explains the model this project explores and what was experimented with.
- Chapter 4 (Results) evaluates the performance of the models generated by varying window size and batch size. It also compares the models to a Wasserstein GAN and explores the effect of lowering the learning rate on models with a window size of 8.
- Chapter 5 (Conclusion) concludes the report by reflecting on the original objective, how the results demonstrate the performance of Swin Transformers in image generation, and suggests future research.

2 Literature Review

Generative Adversarial Networks (GANs) primarily consists of two networks, a generator and a discriminator. They fabricate data and classify data as either real or generated[3] respectively. The generator takes an input, passes it through its internal layers, and outputs data similar to the data used for training. The input and output data can be any form of digital information, but for image generation, it is common to have the input be a latent vector consisting of noise, and have the output be an image of a specific size in one or more colour channels[3], [5]. The discriminator takes the image as an input and outputs the probability of it being real. To train a GAN, each epoch can be split into three steps: 1. Train the discriminator on a batch of real images, 2. Train the discriminator on a batch of generated images, 3. Train the generator to maximize the discriminator loss. By performing these steps, both networks try to outcompete each other, causing the generator to improve its ability to generate realistic data in an attempt to fool the discriminator. This is formally defined as [3, eq (1)]:

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

Where $D(x)$ is the probability that x is real data, and $1-D(G(z))$ is the probability z is generated data. Because the generator learns from the discriminator's ability to classify images, it is less likely to clone the training data. This is unlike a Variational Auto-Encoder (VAE) which takes an image, passes it through an encoder to get a latent vector, then decodes it to get the original image[2]. Because of this, a VAE learns to compress/decompress existing images rather than to generate unique data. Despite this, VAE's can be used to generate new images via manipulation of the latent vector, though their quality is generally inferior to a GAN[6, Table 1].

Deep Convolutional Generative Adversarial Networks (DCGANs)[5] make use of transpose convolutional layers in the generator to convert feature maps into images using convolutional kernels. Transpose convolutions were introduced by Zeiler et al.[7] as a method to visualize the internal workings of Convolutional Neural Networks in order to better understand how to improve them. Transpose Convolutional layers take a feature map, apply a learnable kernel to perform a reverse convolution operation, and output a feature map that has a larger spatial resolution than the input. During training, the convolutional filters learn features from the training data so it

can convert a latent vector into a large, high quality image. A basic DCGAN generator consists of transpose convolutional layers, normalization layers, and activation layers. The output is passed through a $\tanh$ activation layer so that each pixel value is constrained to $[-1, 1]$. Batch normalization is employed to prevent gradient vanishing and gradient explosion[8]. Gradient vanishing is where gradients backpropagating through a network approach zero, making it hard for lower layers to update their weights significantly, preventing the model from converging. Exploding gradients, on the other hand, occur when gradients approach a very large number as they back propagate, which can lead to unstable weight changes, oscillations, and divergence during training. A DCGAN discriminator uses a CNN to generate a probability output that a given input is real or generated.

Progressive GANs (PGANs) are a method of training GANs introduced by Karras et al.[9] which starts with a smaller network and gradually adds layers during training to increase the resolution and detail. By doing this, the network is able to learn larger features early on and progressively learn smaller, more complex or abstract details as new layers are added. This occurs because larger features require less data but finer or complex features require more data. Because the network doesn't have to learn fine detail at the same time as it learns coarse detail, it is able to train more stably. And since the model starts with only a portion of the full amount of parameters, it is able to train faster as it doesn't have as many calculations to perform. To prevent sudden shock to the training when adding a new layer, the output of the new layer is gradually faded in by interpolating between the new resolution and the old resolution upscaled with nearest neighbour, as seen in Figure 2.1.

StyleGANs take the idea of style transfer and apply it to image synthesis[10]. Unlike traditional GANs that take a latent vector and expand it into an image, StyleGANs start with a pre-learned tensor and apply style transfer and scaling to generate images. As seen in Figure 2.2, the model starts with a pre-trained tensor, noise is applied, then style is applied using adaptive instance normalization. The style is generated by an 8-layer MLP which takes the latent vector as input. The result is passed through a convolutional layer, more noise is applied, and style is applied again. Each subsequent block upsamples the image and repeats this process but replaces the constant tensor with a convolutional layer. According to Karras et al., introducing noise allows for stochastic detail (such as, in the case of face synthesis, the placement of hairs, stubble, freckles, etc) to be added without the convolutional layers needing to learn to "generate spatially-varying pseudorandom numbers"[10, p. 4405]. I.E. the noise serves as the pseudorandom number, and the convolutional layer doesn't need to learn this functionality in addition to learning features.

Transformers were introduced in 2017 by Vaswani et al.[11] to replace

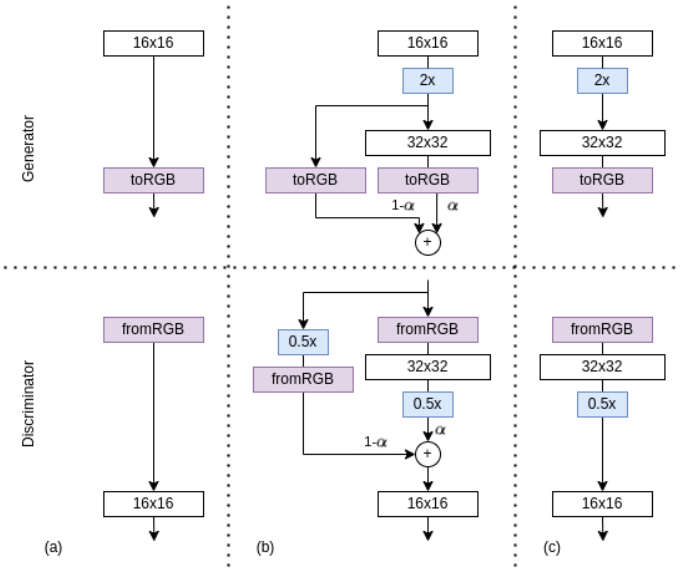

Figure 2.1: Process of introducing a new layer to the GAN. (a) Starting point. (b) Introduce new layer with interpolation. (c) The new layer is now trained enough, so interpolation is removed. As adapted from [9, Fig. 2]

Recurrent Neural Networks (RNNs) in handling sequenced data, specifically for language analysis. RNNs process sequenced data by processing the first token, passing the output to an input of the next cycle of the network, which handles the next token, and so forth. Because of this dependency on the previous token being calculated, RNNs are very slow to train because they cannot be parallelized easily. Transformers have replaced such networks as they can be calculated in parallel[11], [12]. They take a set of tokens, apply attention to calculate the relationship between them, then pass the the result along to the decoder section to generate a probability map defining which output(s) is most likely (Figure 2.3). The first step of the Encoder is embedding, where the input data gets translated into vectors using the embedding space. Closely related data have closer vectors, and conversely, data more unrelated have vectors that are more dissimilar. This expresses the semantic differences between tokens. These vectors are then combined with positional encoding to add information about the sequence of the data. This input is then passed to the Multi-Head Attention block (Figure 2.4) which generates attention vectors, which store how related each input is to all other inputs (including itself). These vectors are concatenated and passed through a linear layer to reduce the number of nodes. Multi-head attention is used to prevent the network from too strongly correlating the input token with itself instead of learning how it relates to the whole input data. Finally, the attention vectors are passed to the Feed Forward layer which is a Multi-Layer Perceptron. For language

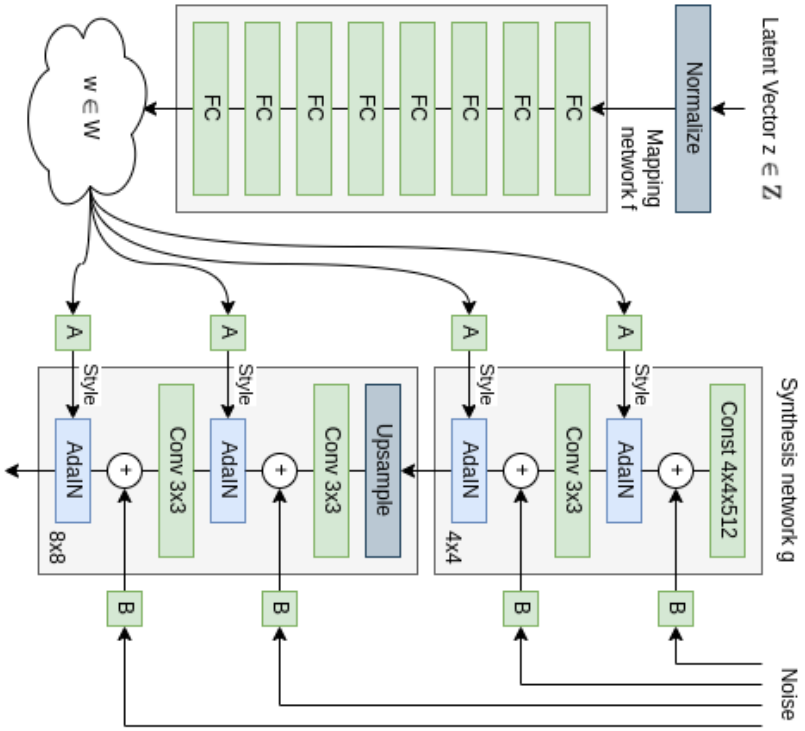

Figure 2.2: The latent vector passes through an 8-layer MLP to generate a vector in latent space W. Style is applied via the "AdaIN" layers, which stands for Adaptive Instance Normalization. "A" is an offset linear (affine) transform, and "B" is per-channel scaling for the noise input. Adapted from [10, Fig. 1]

analysis, the text is split into tokens which can be words or partial words, then these tokens get converted into vectors and processed as previously described.

Shifted-Window Transformers (Swin Transformers) were introduced by Liu et al.[13] in 2021 to improve transformers for computer vision tasks. As previously discussed, transformers were designed for use in language processing. For use in computer vision, an image is split into tokens where each pixel (or chunk of pixels) is a separate token. A relatively small image of 64x64 would result in a token count of 4096. For a size comparison, if we say each word is a token in a language model, 4096 words would be over eight A4 pages of 12pt Arial font[14]. Since transformers have a $O(n^2)$ time and space complexity where n is the token count, processing larger images quickly becomes an issue; the complexity would be $O(m^4)$ where m is the image side length (assuming a square image). To address the large memory requirement, Swin transformers operate by splitting images into small windows to apply self-attention on instead of all tokens at once like with a regular transformer[11]. Swin transformers have at least two

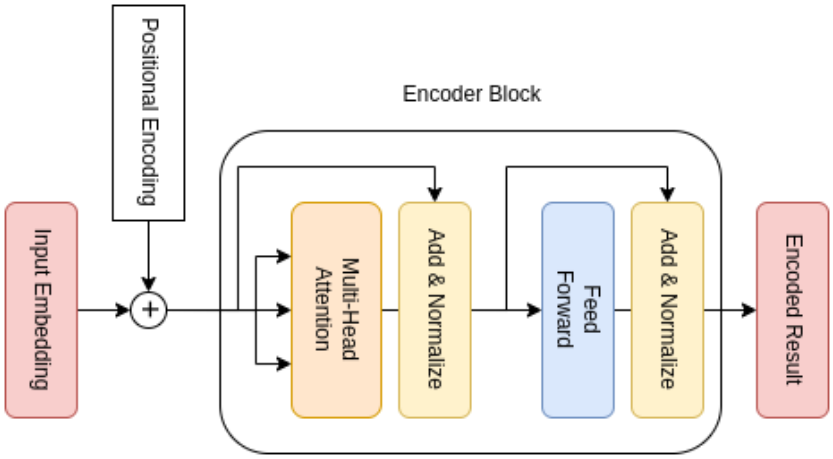

Figure 2.3: The Transformer Encoder Block - as adapted from [11, Fig. 1]

stages. The first is where local attention is applied to the windows. The second stage is where the "shifted" part of a shifted-window transformer comes from; the windows are displaced by half the window size in the x and y axes, then attention is applied again. By doing this, attention is calculated for the windows and inter-window tokens. By applying attention a second time with shifted windows, it "significantly enhance[s] modeling power"[13, p. 9993].

Transformer GANs (TransGANs) utilize Transformer Encoder blocks to generate images[15]. To save on memory and training/execution time, instead of feeding every pixel from every channel into the transformer as their own vectors, pixels are grouped by their channel and become a single input token. For example, an 8x8 192-channel image would only have 64 input tokens, each of length 192. The generator works by passing the latent vector through a multi-layer perceptron and adding it to positional encoding to generate input vectors. These are then passed through a number of transformer encoder blocks before being upscaled. The upscaling is achieved via the Pixel Shuffle algorithm[16], as shown in A.1. The image is now twice as large in width and height, and the number of channels has been quartered. This process of transformer encoding then upscaling is repeated until the desired image size has been achieved. The final step is to pass the data into a linear unflatten layer which takes the transformer outputs and converts them to 3-channel RGB image data.

StyleSwin Transformer GANs[17] combine the concepts of StyleGANs, Shifted-Window (Swin) Transformers, and TransGANs to create a style-based generator that utilizes swin transformers instead of transpose convolutional layers commonly used in StyleGANs. They were introduced to solve the issue of pure TransGANs not being on-par with other types of network. They address the limitations in computational efficiency of

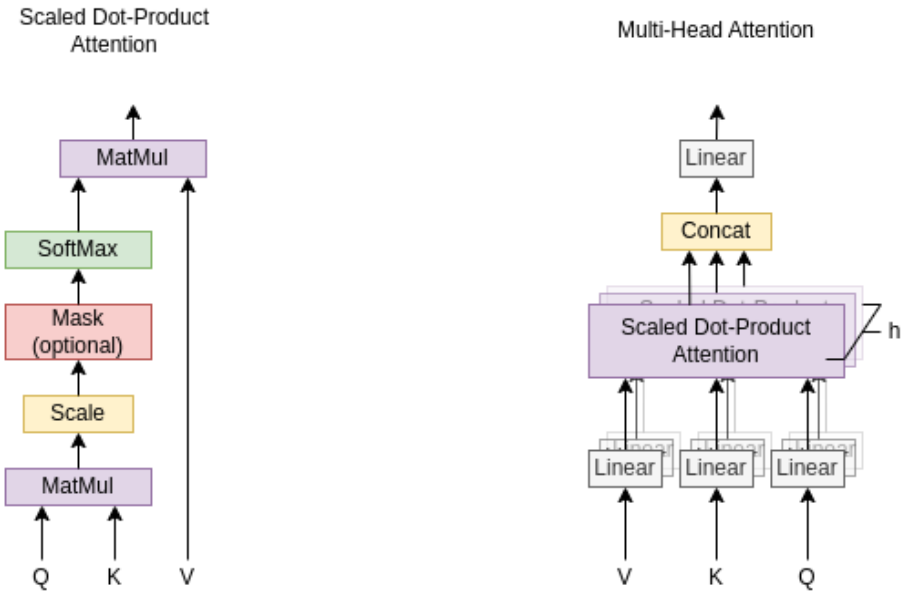

Figure 2.4: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention blocks running in parallel. Adapted from [11, Fig. 2]

TransGANs by employing Swin transformers, and address quality by employing Style-based image generation. StyleSwin GANs achieve superrior performance over prior TransGANs, particularly at high resolutions.

The generator keeps roughly the same architecture as the original StyleGAN paper[10, Fig. 1] except it doesn't inject noise, and each block consists of the following layers: AdaIN, Double-Attention Swin Transformer, AdaIN, then a MLP layer. The Double-Attention Swin Transformer captures both local and global features on an input image. It splits an image into $k \times k$ non-overlapping blocks and calculates windowed attention on them[13]. It then shifts the blocks by $k//2$ pixels and calculates the shifted window attention. The results from both attention mechanisms are concatenated for the final result[17, p. 11307]. In the paper by Zhang et al., the discriminator uses the same convolutional discriminator from Kerras et al. because they found that, despite the training being more stable, the overall result wasn't as good when they replaced the convolutional layers with transformers[17, p. 11306].

Wasserstein GANs (WGANs) address the issues of improving learning stability and reducing mode collapse in GANs. They were introduced in 2017 by Arjovsky et al.[18]. Mode collapse is where the generator converges on a limited set of results, thereby eliminating the diversity of generated images. Consequently, the discriminator is able to easily differentiate real from fake data. WGANs are less sensitive to the initial

hyperparameter settings as compared to other GAN models. They derive their name from using the Wasserstein (Earth-Mover) distance as their loss function, defined as:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[||x - y|| \right]$$

where $\mathbb{P}_r$ and $\mathbb{P}_g$ are probability distributions, and $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions. $\mathbb{E}_{(x,y)\sim\gamma} \left[||x - y|| \right]$ is the expected distance between x and y under the joint distribution γ. Therefore, this function calculates the amount of work required to move from x to y in order to transform $\mathbb{P}_r$ into $\mathbb{P}_g$. i.e. the amount of "dirt" required to be moved from one "pile" (distribution) to another "pile" (the other distribution) to transform one into the other. (Hence the name "Earth-Mover" distance)[18, eq. 1].

One key advantage of using Wasserstein distance is that it provides a more meaningful gradient during training, especially when the real and generated data are disjoint. The Wasserstein loss function leads to smoother convergence as it measures the cost of transporting "mass" to transform one probability distribution into another, as opposed to measuring the similarity of two probability distributions. Measuring the similarity as the loss function suffers from gradient vanishing when the real and generated data have little overlap, whereas the Wasserstein distance provides a continuous metric even when the two probability distributions are disjoint. In WGANs, the discriminator is replaced with a "critic" that estimates the Wasserstein distance between real and generated data. Since the critic is estimating distances rather than probabilities, an activation function such as sigmoid is not required. To ensure the critic is Lipschitz continuous, WGANs use weight clipping which constrain the weights within a certain range. This weight constraint enables the WGAN to have more stable and reliable gradients, which results in a smoother training process[18, p. 7]. However, as mentioned in the paper, clipping is not a good solution because it limits the critic's capacity. Because of this, it was improved by replacing clipping with Gradient Penalty by Gulrajani et al.[19].

Big Generative Adversarial Networks (BigGANs) were introduced by Brock et al.[20] as a scalable and stable approach for generating high-quality, high-resolution images using GANs. The primary idea behind BigGANs is to significantly increase the model size while employing other architectural and training improvements, making it possible to generate images with high fidelity and diversity. They expand upon DCGANs[5] by having deeper generators and discriminators along with more channels in each convolutional layer. This larger size allows the model to capture smaller, more complex features from the training data which allows it to generate higher quality results.

Hierarchical Latent Spaces are used to inject latent vectors at multiple

layers of the generator. To do this, the latent vector is split into equal-sized chunks and injected into the Residual Blocks within the generator at the different stages. See Figure 2.5. The size of the initial latent vector is dependent on the size of the image to be generated. Each doubling in image size requires an additional 20 dimensions in the latent vector, e.g. 120 for 128x128, 140 for 256x256[20, p. 17]. Using latent vectors at multiple stages allows the model to learn different levels of abstraction at the different stages. BigGANs also support class-conditional information which promotes better control over the generated images' characteristics.

Spectral Normalization[21] is applied to both the generator and discriminator to stabilize training and prevent the escalation of model weights. Spectral normalization enforces Lipschitz continuity on the discriminator, which ensures a more meaningful gradient flow and helps mitigate the risk of mode collapse. It also improves stability in the generator leading to needing fewer steps in the discriminator per iteration[20, p. 2].

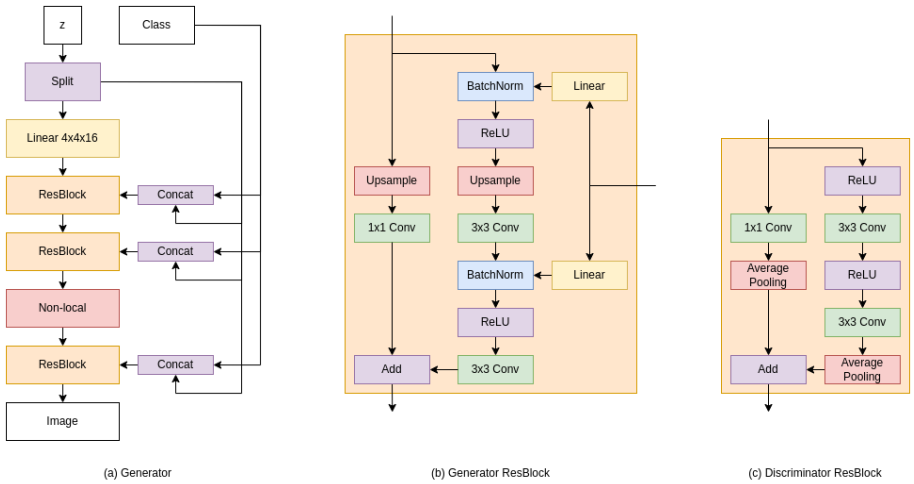

Figure 2.5: (a) The architecture of a BigGAN's generator (b) A Residual Block from a BigGAN's generator. (c) A Residual Block from a BigGAN's discriminator. Adapted from [20, Fig. 15]

Inception Score (IS) was created by Salimans et al.[22] in their paper "Improved Techniques for Training GANs" to solve the issue of judging visual quality of generated images. Prior to this, images were shown to humans who were asked to distinguish generated from real data. However, as they mention in the paper, the quality of this method "varies depending on the setup of the task and the motivation of the annotators"[22, p. 4]. I.E. If participants are asked to look for certain features, they are more likely to pick them out, resulting in a lower quality score. The Inception Score automates this process, removing the human element from judging. It uses the probability output of a pre-trained classifier model to evaluate the image quality and diversity of the generated images.

$$IS(G) = \exp\left(\mathbb{E}_x KL[p(y|x)||p(y)]\right)$$

Where KL is the Kullback-Leibler divergence[23] between the conditional distribution of the predicted class label and the marginal distribution of the class labels in the dataset. The closer the distributions, the better the quality and the diversity of the generated images. $p(y|x)$ is the predicted class probability of generated image x, and $p(y)$ is the marginal distribution of the class labels in the dataset.

Fréchet Inception Distance (FID) was introduced by Heusel et al. in 2017[24] to improve upon the existing Inception Score method. FID calculates the Fréchet distance between Gaussian distributions for the probability of observing a real image $p(y)$, and for the probability of generating model data $p(y)$. The formula for which is[25]:

$$d^2 = |\mu_X - \mu_Y|^2 + tr\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}\right)$$

where (μ_X, Σ_X) is the mean and covariance obtained from $p(x)$ and (μ_Y, Σ_Y) is the mean and covariance obtained from $p(y)$.

2.1 Discussion

In this review, I examined several approaches to image generation using different types of GANs. DCGANs provide a solid foundation for image generation but they can suffer from stability issues during training. PGANs help alleviate this issue by gradually increasing the resolution during training, making the training more stable and efficient. StyleGANs, on the other hand, focus on style transfer which results in higher-quality with more diverse samples. WGANs further helped with training stability by introducing a new loss function which can better indicate modelling ability and convergence. Despite BigGANs' ability to generate high-fidelity images, their computational cost makes them difficult to train and run at scale. However, their idea of injecting parts of the latent vector at multiple stages seems to work well for image quality as it allows the layers to focus on specific details. This is similar to the StyleGAN where it injects noise at multiple points each layer. TransGANs require a lot of memory to operate, but transformers have shown very promising results in other tasks such as language modelling. SwinTransGANs greatly improve the performance of TransGANs by greatly reducing the memory and time complexity, and have been shown to produce higher-quality results. Adapting the power of transformers to generate images is an interesting step forward and was shown to generate highly detailed results in StyleSwin GANs which combines Shifted-Window Transformers with StyleGANs.

3 Method

3.1 Architecture & Design

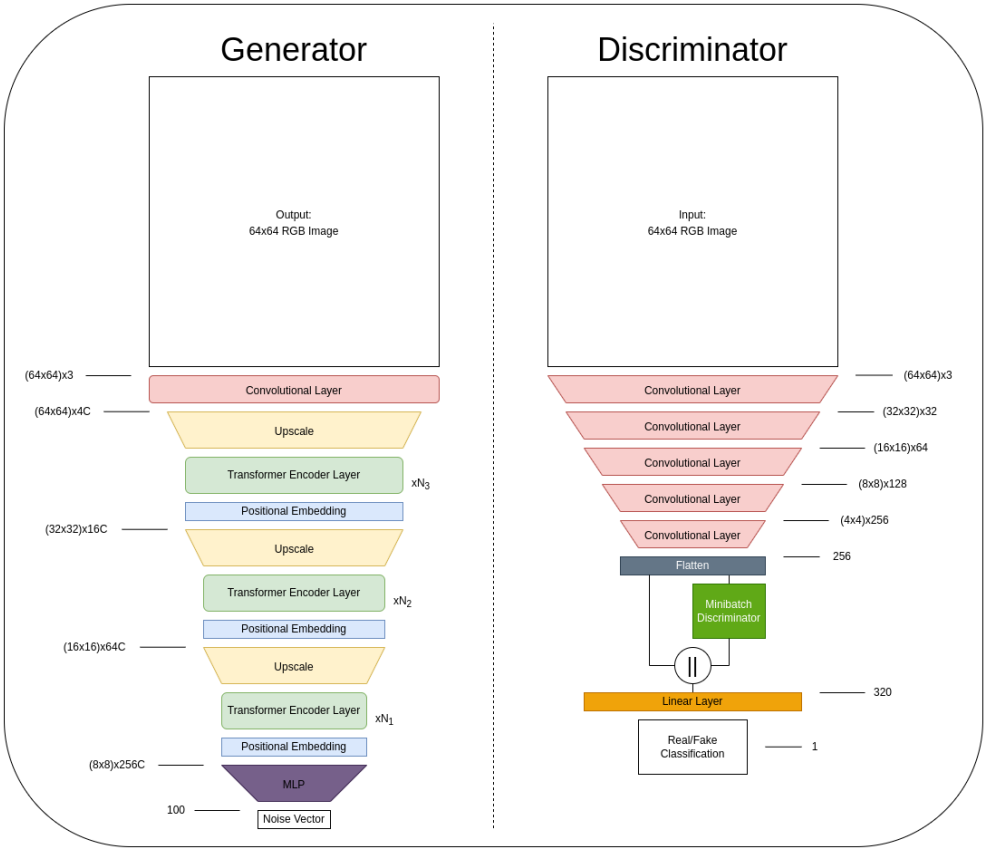

Figure 3.1: A diagram of the transformer generator (left), and the convolutional discriminator (right)

Transformers have the same output shape as the input[11]. At first glance this would suggest that in order to generate an $n \times n$ image, we would need to pass in an $n \times n$ noise vector. However, a major problem with this approach is that transformer layers require a lot of memory. To reduce memory usage, Shifted-Window (Swin) Transformers are employed. Swin Transformers reduce the focus of the transformer to a limited number of vector inputs at a time which can significantly reduce the memory requirement to process them[13]. The image is split into chunks of a specified

width and height before attention is applied. A second pass is used to shift the windows by half their size to apply attention to the inter-chunk areas (See Figure 3.2).

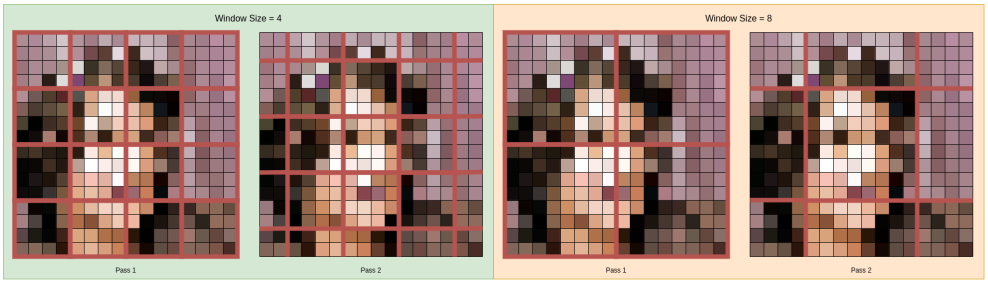

Figure 3.2: Comparison of shifted-window attention with window size of 4 (left) and 8 (right) on a 16x16 image

Positional encoding is added to every transformer input vector. By doing this, the position of each pixel is encoded into the data so that the transformer is able to better understand the relation between neighbouring pixels. Instead of using a fixed value, a learnable positional vector is used. As discussed by Vaswani et al.[11, p. 6], using a learned positional embedding "produce nearly identical results". This model uses a fixed sequence length so there would be little benefit in using the sinusoidal embedding as outlined in their paper.

Pixel Shuffle Upscaling is used to upscale the image in between transformer layers. Unlike algorithms such as bilinear scaling which copy and smooth data, pixel shuffle takes r^2 channels and converts them into blocks of $r \times r$ pixels by assigning each channel a specific location in the block[16]. By using pixel shuffle, we take smaller images with more channels and convert them into larger images with four times fewer channels (Figure A.1).

The discriminator from a DCGAN[5] was used. The discriminator uses a series of convolutional layers to reduce the image size and extract key features. The output is a single value passed through a sigmoid function which represents the probability of the image being real. A Minibatch Discrimination is used to help prevent Mode Collapse[22, p. 3]. Instead of analysing each image by itself, the Minibatch Discriminator layer inspects multiple images at once and calculates their similarity. This approach encourages the generator to produce a diverse output set and disincentivizes converging on a limited set of results. The method for minibatch discrimination outlined by Salimans et al.[22, p. 3] can be calculated as follows: Let $f(x_i)$ be a feature vector of length A produced by previous discriminator layers; multiply by a tensor T of shape $A \times B \times C$, resulting in a matrix M_i of shape $B \times C$. Where A is the number of input features, B is the number of output features, and C is the number of kernel dimensions. The

L1-distance between the rows of matrix M_i is calculated before applying a negative exponential. The output of a Minibatch Discriminator layer is defined as follows[22, p. 3]:

$$o(x_i)_b = \sum_{j=1}^{n} exp(-||M_{i,b} - M_{j,b}||) \in \mathbb{R}$$
$$o(x_i) = [o(x_i)_1, o(x_i)_2, ..., o(x_i)_B)] \in \mathbb{R}^B$$
$$o(X) \in \mathbb{R}^{n \times B}$$

Where $\mathbb{R}$ is the set of real numbers and a power of $\mathbb{R}$ represents the shape of the vector/matrix/tensor. n is the batch size. The result is then concatenated with the input tensor. If a batch of images are all very similar, it is easy to know that it is generated data. I used an input of 256 and an output of 64 for the minibatch discriminator.

Between the final transformer layer and generator output, a convolutional layer is used to convert the encoded vectors into RGB data. The convolution uses three 1x1 kernels which act to convert each vector into a pixel with three colour channels without introducing any pattern filtering.

3.2 Parameters

To keep the number of parameters at a reasonable level, each transformer layer contains only two transformer blocks. That is, $N_1 = N_2 = N_3 = 2$ as seen in 3.1. The channel multiplier C is applied to every layer and can cause the parameter count to explode even with small values. Consequently, to strike a balance between model complexity and parameter count, the channel multiplier was also set to 2. This means the initial linear layer accepts a vector of size 100 and outputs a vector of size $32,768$, which is the input and output size for each layer in the generator until the final 1x1 convolutional layer which converts the result into a $64 \times 64 \times 3$ RGB image. The number of transformer heads was kept constant at 4 heads for every layer. Other parameters are discussed in the results section, Chapter 4.

3.3 Dataset

This project used the CelebA dataset[1] which contains over 200,000 images of faces taken from publicly available sources online. Preprocessing was applied to the dataset to convert the images into a usable form for training. The following transformations were applied: 1. Resize the images

to 64px then, 2. center crop the images to 64x64 pixels to ensure the image is square and the face is in the center. Normalization was not applied as it would change the colour of the generated images.

Figure 3.3: 24 pre-processed samples from the CelebA dataset[1]

3.4 Implementation

Binary Cross-Entropy loss was used as the training criterion because training a GAN is a binary classification task. It calculates the difference between the predicted probabilities and the true labels. It is defined as: $BCELoss(y, p) = -(y * log(p) + (1 - y) * log(1 - p))$ where y is the true label (either 0 or 1), and p is the predicted probability (between 0 and 1). It maximises when the predicted probability is the inverse of the true label, and minimizes the loss when they are the same.

Pytorch 2.0 was used to define the network, training, and evaluation. A python script was created with multiple arguments to allow the user to change the parameters without modifying the code.

The dataset was loaded using the Pytorch DataLoader class. It utilized 3 workers and 4 prefetched batches per worker to keep the GPU fed during training.

Shifted-window transformers are not currently included with Pytorch so

the official code from the transformer paper[26], [27] was used. Pytorch also does not include a minibatch discriminator, so code was adapted from Salimans et al.'s paper "Improved Techniques for Training GANs"[22], [28]. Their code was written with Lasagne/Theano[29] which had to be adapted to Pytorch for my project. Evaluation was undertaken by using existing libraries for Inception Score (inception-score-pytorch)[30] and Fréchet Inception Distance (Clean-FID)[31].

The Clean-FID library was used for the Fréchet Inception Distance calculation as it includes steps to clean the data to standardize testing. The CelebA dataset statistics are cached on the first run through, then compared to each model. Caching the statistics reduces the computation time as it doesn't have to recompute the result for each model. Clean-FID applies standardized data pre-processing steps of resizing, compression, quantization, and filtering. By doing so, the resulting values are easier to compare as they all use the same pre-processing steps.

A pre-existing Wasserstein GAN model was used as a comparison[32]. The hyper-parameters were kept as close to the SwinTransGAN hyper-parameters as possible.

To generate results, the model was executed on the University of York's G-Viking cluster using a single Nvidia A40 GPU for each model. Three CPU cores were used per model to load and preprocess images from the dataset in order to keep the GPU fed with data. The models are trained for 40 epochs to create comparable results and to limit the server usage. A time limit was not used as the models may train at different speeds.

3.5 Testing Method

For the experiments, the window size and the batch size have been modified to generate nine different models. The window sizes 2, 4, and 8 were used in combination with batch sizes of 100, 200, and 400. Because the image being generated is a power of 2, the decision to double the window size for each set of batches was made so that the image would remain evenly divisible by it. The batch size was also doubled for every set of window size. Though this does not have an affect the number of parameters, the minibatch discriminator will have more data to process as it has more samples to take into account. By doubling the batch size instead of picking arbitrary values or adding a constant amount, it allows the results to be interpreted easier. All other parameters were kept the same as to not contaminate or confuse the results.

Window size is an important parameter in Swin Transformers as it controls how the input data is grouped and how many pixels the groups contain as seen in Figure 3.2. A smaller window size means a smaller group to apply attention on, conversely, a larger window size means a larger group for attention to be applied. However, because attention requires a lot of memory, a larger window size would require more memory. We investigated the effect the window size has on training the model and the results the model produces.

Batch size was selected as a parameter to investigate because it affects the accuracy of the loss function. A larger batch exposes the discriminator to more images from the dataset which gives a better average loss value. This should lead to more stable training and a reduce the variance in gradient updates. By changing the batch size between models, we investigated the effect it has on the Swin Transformer model and whether the result is improved as batch size increases.

Training time was measured by taking the difference between starting time and end time. This was possible because each model was trained in a single session. The training time does not include the time taken to load the data initially or load the models. Allocated memory and cached memory were measured throughout the training process and the highest value was selected. Once they had been trained, 50,000 sample images were generated using random noise vectors for each model. Inception Score and Fréchet Inception Distance were then calculated using these images. As suggested by Parmar et al.[31], the images were saved as PNG files to avoid compression as this would be applied later by the Clean-FID library. Progress was recorded in a log file automatically thanks to the G-Viking servers. However, for live-updates, a webhook was used to send key information such as current epoch and batch number, generated image samples, current loss values, and loss graphs.

A Wasserstein GAN was also trained to compare results with the Swin Transformer GAN models. Using an established method to compare results allows for a greater understanding of how well the models performed because the previous methods are already well-documented. WGANs were chosen as they are a more recent development than DCGANs, have been shown to be more stable during training, and are better able to avoid mode collapse[18]. However, the WGAN generator still uses a standard DCGAN architecture.

4 Experiments & Results

Parameter	Value
Learning Rate	0.0001*
Attention Head Count	4
Transformer Layer Count	2
Data Loader Workers	3
Worker Prefetch Count	4
Latent Vector Size	100
Discriminator Feature Multiplier	32
Generator Channel Multiplier	2
Adam Beta Values	(0.5, 0.999)
Training Seed	Time at script execution
FID & IS Sample Size	50,000

Table 4.1: A table showing the values of the parameters used to train the models. (*) the learning rate was halved for Table 4.4.

Because the generator and discriminator use different architectures, the discriminator feature multiplier had to be tuned via experimentation. During development of the code, a value of 32 for the feature count multiplier was found to produce stable results.

Every SwinTransGAN model had $10,196,472$ parameters for the generator, and $1,788,609$ for the discriminator. The Wasserstein GAN models had $3,576,707$ parameters for the generator, and $2,765,569$ for the discriminator. Seven out of the nine SwinTransGANs were able to achieve a better FID score than the best WGAN, however, they took significantly longer to train and required much more memory. Because of the differences in architecture, the parameter count isn't necessarily an indication of modelling power. Transformers require more parameters to function than transpose convolutional layers. Because of this, a like-for-like parameter count comparison cannot be made. However, efficiency can still be measured. The Wasserstein GANs managed to achieve results not too dissimilar from the the SwinTransGANs despite having a higher FID score. They also have a higher Inception Score than any of the other models. The WGANs were able to train at least 2.3x faster than the fastest SwinTransGAN, with less memory usage, and resulting with similar generation quality. However, please note that the training time and cached memory for the WGANs may

Window Size	Batch Size	Allocated Memory (MiB)↓	Cached Memory (MiB)↓	Training Time (mins)↓	Inception Score↑	FID Score↓
2	100	207	**2528**	368	1.38	89.66
	200	210	4776	300	1.38	97.23
	400	220	9198	255	1.39	100.49
4	100	**205**	3172	325	1.31	159.37
	200	211	6166	241	1.37	94.93
	400	221	11598	**210**	**1.41**	98.86
8	100	208	5876	356	1.36	**88.43**
	200	214	10792	287	1.39	180.58
	400	222	23010	250	1.40	103.18
N/A WGAN	*100*	*278*	*488**	*89**	*2.63*	*103.20*
	200	*278*	*732**	*82**	*2.50*	*104.14*
	400	*278*	*1316**	*78**	*2.50*	*116.82*

Table 4.2: Comparison after 40 epochs of memory usage, training time, inception score, and FID score on varying window sizes and batch sizes for SwinTransGAN models and a WGAN model. The best of each category for the are in bold, excluding results for WGAN. The superscript (*) indicates the values cannot be directly compared between SwinTransGAN and WGAN because the dataset was loaded and cached differently.

be incomparable as the library used loaded the data with a different prefetch count and took advantage of memory pinning. Despite the SwinTransGAN models outperforming the WGAN models in FID score, they had more parameters so it may not be a fair comparison.

As batch size increased, the training time decreased. This was consistent across all window sizes and architectures. The models with window sizes 2 and 8 had similar training times, but models with window size 4 trained at least 30 minutes faster in each batch size. This can be best seen in Figure 4.1 which graphs out the training times. This result was not expected because a larger window size mean there are more calculations that need to be computed. However, what could be happening is that the window size 2 models were under-saturating the GPU's processors, window size 4 was saturating them, and window size 8 was over-saturating them. Consequently leading to slower training times for window size 2 and 8, and a faster training time for window size 4. However, further investigation would be required to verify this and it is outside the scope of this paper.

Counter-intuitively, increasing the batch size did not consistently increase or decrease the quality of the image. For a window size of 2, the FID scores got worse as batch size increased, but for window sizes of 4 and 8, the

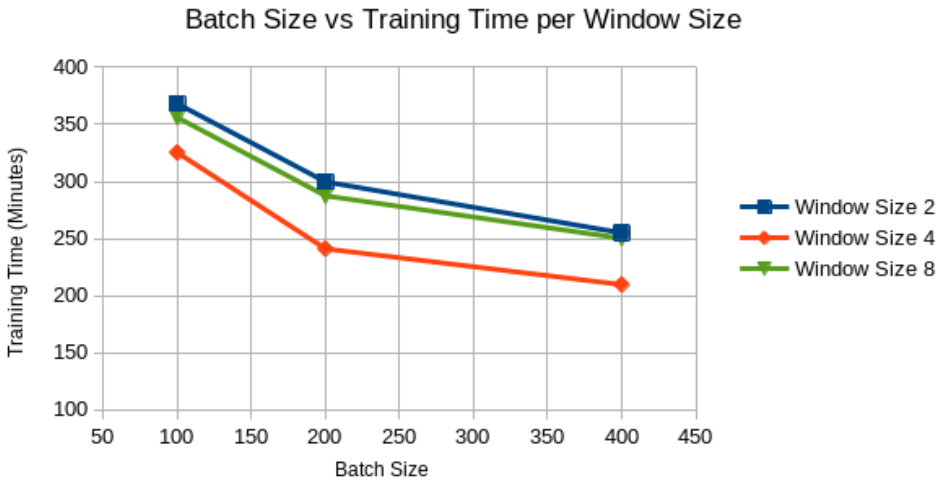

Figure 4.1: Graph of batch size vs training time separated by window size.

Window Size	FID Score Range↓
2	**10.83**
4	64.44
8	92.15

Table 4.3: Comparison of FID score range and Swin Transformer window size as derived from Table 4.2

scores didn't correlate with batch size. However, changing the window size did affect the range of FID scores generated. As window size increased, the FID score range increased, as seen in Table 4.3. This suggests, given the current architecture and hyper-parameters, increasing the window size introduces more instability to training. Because a larger window size has more vectors for attention to be applied, the model may be learning at a different rate to models with smaller window sizes. Figure 4.5 could indicate the learning rate is too high for window size 8 because the faces change wildly between epoch steps instead of converging. However, Figure 4.3 demonstrates the opposite, though this could be because both window size and batch size affect what the optimal learning rate should be. Table 4.4 shows the results of investigating using a smaller learning rate for models with window size of 8.

For the main results, a learning rate of 0.0001 was used. However, after collecting these results, to further test models with window size 8, a lower learning rate of 0.00005 was used. This was to see if the large discrepancy between batch size results is due to the window size or due to learning instability due to the learning rate being too high. The results for this second round of experiments can be seen in Table 4.4.

Figure 4.2: Manually-selected images from a batch of 5000 generated by the best performing model (model 8-100). The left 3 columns are of male faces, the right 3 columns are of female faces.

As the FID scores show, the models failed to produce good-quality images most of the time. The closer the FID score is to zero, the more accurate the generated images are to the original dataset. The best performing model (model 8-100) only achieved a score of 88.4 which indicates the average result is poor. Figure B.7 shows a sample of model 8-100's output, in which a high level of artifacting and distortion can be seen. However, the model is capable of generating higher-quality results as seen in Figure 4.2 which shows a manually-selected collection of the best images from a pool of 5000 samples. Though no accurate numerical score can be derived from just 24 images, a visual inspection indicates the model was able to learn the facial features correctly, and that with more training, it may be able to generate higher-quality images more consistently.

Since the model only trained for 40 epochs and it appeared to be training stably as shown by the loss graph Figure 4.4, it is likely that the model would be able to converge on a higher-quality result. Figure 4.3 shows samples of faces generated with the same latent vectors for epochs 1, 5, 10, and 40. This figure demonstrates the model is converging on a set of parameters that generate good quality faces. Because of this, the model likely would be able to generate better result with more training. In order for convergence to occur, the generator and discriminator need to be balanced in capabilities so that one doesn't outperform the other. If the discriminator

Figure 4.3: Progressive outputs of the best performing generator (model 8-100) at epoch 1, 5, 10, and 40 using the same latent vectors for each set of images. The generator used a window size of 8 and batch size of 100. As training continues, the faces converge on a higher quality output.

is better at classifying than the generator is at generating, the result will collapse. If the generator is more powerful than the discriminator, the result will also collapse. Both architectures need to be balanced for the result to converge. For the convergence to be of good quality, both models need to be complex enough to be able to model the desired images.

All the SwinTransGAN models have an inception score within 0.1 of each other. Due to the small size of the generated images, the inception score is not a good measurement for performance. Like with FID, the Inception Score uses an input image size of 299x299. However, unlike FID, it doesn't compare the generated samples to the original samples. Consequently, the result is only dependent on the model's ability to classify the images and not representative of how good they are compared to the training data. The WGAN achieved the highest inception score with almost double that of the highest SwinTransGAN, but it also failed to outperform most of the other models in FID score. A visual inspection of the best WGAN's outputs (Figure B.10) show that the images are less blurry but suffer from strong distortions and artefacts.

To collect the images for Figure 4.2, 5000 images were generated and manually inspected to select the best 12 male faces and the best 12 female faces. Out of the 5000 images, only around 100 of them had minimal distortions or artifacting to regard them as natural-looking. Within the 100 good samples, there were twice as many good-quality female faces as male ones, which could be explained by the dataset being comprised of only 40% male faces (according to the attributes). Another thing to note is that the model failed to generate faces with darker skin tones at an acceptable visual quality. Again, the dataset only contains a small percentage of faces with darker skin tones compared to lighter skin tones due to demographics. This is a wider concern in machine learning as it generates implicit bias towards or against certain groups[33]. This issue could not be corrected

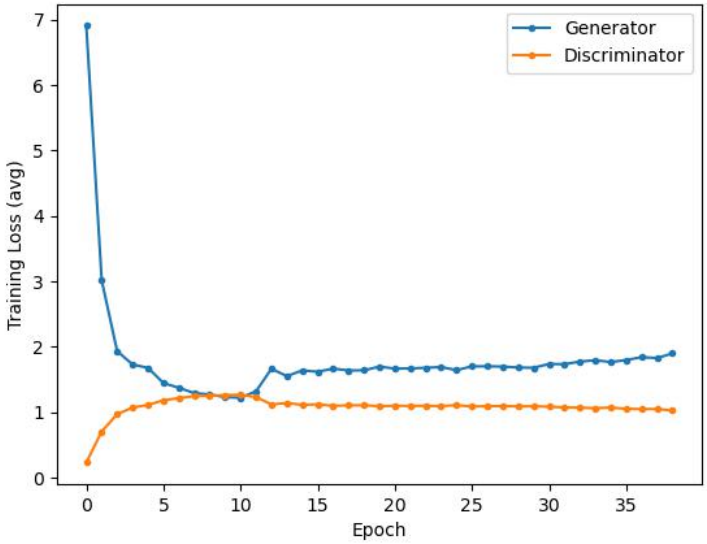

Figure 4.4: Generator and Discriminator loss graph for best performing model (model 8-100)

as the CelebA dataset does not have an attribute for skin tone, meaning a weighted sampler for the training data could not be used.

The allocated memory was consistent between window sizes and only varied with batch size. This is due to the use of a minibatch discriminator that has to process more data when a higher batch size is used. As expected, the memory increase is double between 200 and 400 batch size compared to 100 and 200 (Table 4.2) because there are twice as many images to process. Despite the use of a minibatch discriminator, multiple models suffered from mode collapse. The models most affected were models 4-100 and 8-200 as seen in Figures B.4 and B.8 respectively.

While the 8-100 model was able to produce the best results, model 8-200 produced the worst results. The model instantly fell into mode collapse. As seen in Figures 4.6 and 4.5, the model was producing mostly noise with a slight face-like impression until around epoch 18. This noise image varied very little, indicating the extent of the mode collapse. When it started developing faces, they were heavily distorted and suffered from mode collapse. It failed to converge on a solution and drastically varied outputs for the same latent vectors between epochs (see Figure 4.5). The top-center images from epochs 20, 25, 30, 35, and 40 are a great example of this and they share basically no traits with each other. The shape of the face, the skin tone, the sex, and the background all change from one snapshot to the next despite them all being generated from the same latent

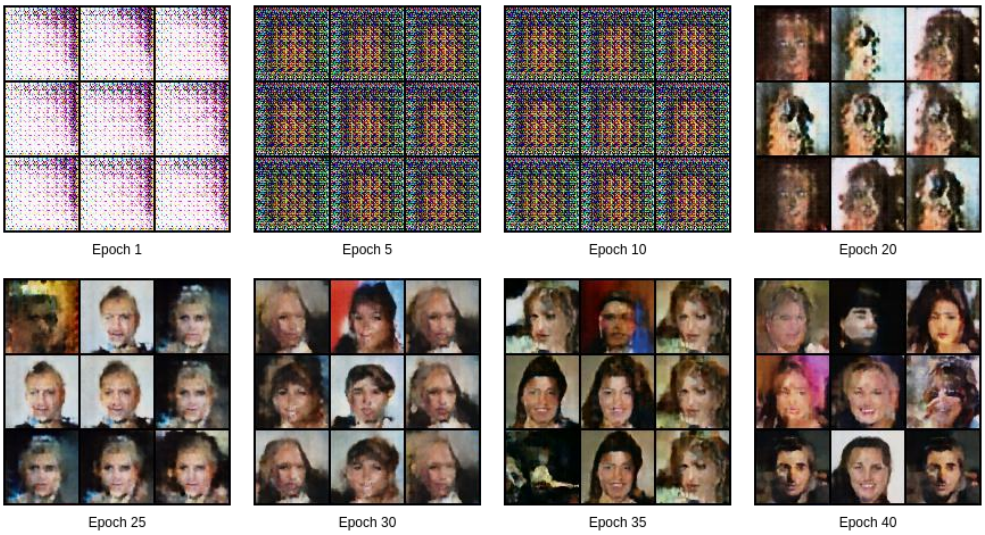

Figure 4.5: Progressive outputs of the worst performing generator (model 8-200) at epoch 1, 5, 10, 20, 25, 30, 35, and 40 using the same latent vectors for each set of images. The generator used a window size of 8 and batch size of 200. The model failed to start generating faces until after epoch 18. It also failed to converge on a solution and continued to wildly vary the generated images given the same latent vector.

vectors. Contrast this with Figure 4.3 which shows that the output from model 8-100 was converging on a result for each latent vector, and that the faces changed in a gradually.

Batch Size	Inception Score↑	FID Score↓
100	1.35 ± 0.01	94.16 ± 2.79 (3%)
200	1.37 ± 0.01	107.93 ± 6.45 (6%)
400	1.45 ± 0.03	144.72 ± 18.1 (12.5%)

Table 4.4: Average results of training 3 models for each batch size with a window size of 8 and a learning rate of 0.00005 (half that of Table 4.2)

As expected, training with a lower learning rate was unable to produce a better result in the same amount of time as a higher learning rate. The lower learning rate had a best score of 94.2, whereas the higher learning rate achieved 88.4. In contrast to the results found in Table 4.2, the data collected from training with a learning rate of 0.00005 (see Table 4.4) indicate a positive correlation between batch size and FID score, at least for models with a window size of 8. That is to say, an increased batch size worsens the resulting generated images. Training with this lower learning rate was able to produce results that follow a trend rather than the seemingly random FID

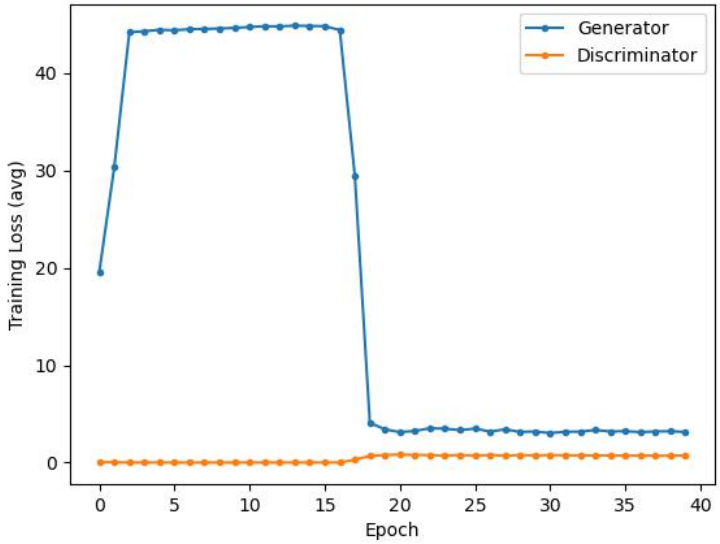

Figure 4.6: Generator and Discriminator loss graph for worst performing model (model 8-200)

values a learning rate of 0.0001 was generating. As expected, the smaller learning rate was able to lead to more stable training and less chance of collapse. Similar to Table 4.3 suggesting window size increases training instability, Table 4.4 suggests that training instability increases with batch size. The mean absolute error for FID goes from 3.0%, to 6.0%, to 12.5%. This is roughly a 2x increase in error for a 2x increase in batch size. Due to limitations of time, these experiments couldn't be executed for window sizes 2 and 4. However, if the data can be extrapolated to these window sizes, we should see FID score positively correlating with batch size from them too. This trend cannot continue forever because the percentage error cannot increase indefinitely. Additionally, once the batch size is large enough, the calculated loss should stabilize, therefore stabilizing the training. However, the FID score is unlikely to improve with a larger batch size as these results show. If these tests were extended, we should see the FID score plateau.

5 Conclusion

In this project I explored how changing window size, batch size, and to a lesser extent, learning rate affects Shifted-Window Transformers. The results demonstrate using a window size of 4 yielded the fastest training times within reasonable memory requirements. It was also found that a window size of 8 produced the best result with a batch size of 100. Further testing supported the idea that a smaller batch size leads to better results (see Table 4.4). This further testing implied that training instability increased with batch size as measured by mean absolute percentage error. Table 4.3 indicates there is a wider discrepancy between model results as window size increases.

The models trained in this project were able to generate human faces, some of which were of good quality (see Figures B.7 & 4.2). However, the results collected show that further work is required to refine the generated images. But they also demonstrate that Shifted-Window Transformers are a viable option for generating novel images of human faces. As previously discussed in the introduction (1), being able to generate realistic faces would aid further machine learning research by creating datasets that have reduced legal and ethical concerns. The models produced by this project failed to generate realistic images most of the time and are therefore not ready to generate production-quality datasets. However, as discussed in the results section, with further training, these models likely would converge on a higher-quality output. Further investigation is required before realistic images can be generated consistently from the architecture designed in this project.

The models failed to produce results for faces with attributes that were a smaller proportion of the dataset. As discussed previously, this is a wider issue as biased data can cause implicit bias towards or against certain groups. The dataset used lacked the data required to create a weighted sampler to balance the attributes.

5.1 Future Work

A window size of 4 was found to yield the lowest training times instead of training time scaling with the window size (see Figure 4.1). Further investigation into this would be warranted to better understand the root cause.

There are many parameters can could be investigated with this architecture besides window size. The channel multiplier, the attention head count, and the transformer layer count parameters are also important variables. The channel multiplier would provide the model with extra data channels for each pixel of the image, which could lead to higher-quality images. The attention head count controls the number of parallel attention mechanisms being applied to the data. Each head learns to focus on different aspects of the data, so increasing the count could also lead to higher quality results. Then the transformer layer count controls for how many transformers are used in each block of the generator. More transformers could allow the the model to capture more detail from the training set.

As discussed in the literature review, Wasserstein GANs are a better way to discriminate generator outputs. They have been shown to improve convergence and help prevent mode collapse. Given the models in this project suffered from mode collapse, investigating the combination of Wasserstein GANs with Transformer GANs could lead to more stable training and potentially higher-quality results.

While undertaking this project, a paper was published in April that implements a new type of vision-based transformer that uses spectral and multi-headed attention[34]. Patro et al. demonstrate this new model has superior accuracy in image classification tasks. A future project could be to investigate the use of "SpectFormers" in image generation.

A Pixel Shuffle

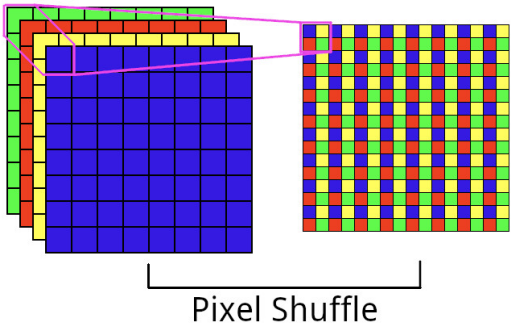

Figure A.1: Pixel shuffle takes r^2 channels and converts them into $r \times r$ grids of pixels. As adapted from [16, Fig. 1]

B Random Samples From Models

Figure B.1: Random samples from model window size 2, batch size 100

Figure B.2: Random samples from model window size 2, batch size 200

Figure B.3: Random samples from model window size 2, batch size 400

Figure B.4: Random samples from model window size 4, batch size 100

Figure B.5: Random samples from model window size 4, batch size 200

Figure B.6: Random samples from model window size 4, batch size 400

Figure B.7: Random samples from model window size 8, batch size 100. (Best model)

Figure B.8: Random samples from model window size 8, batch size 200. (Worst model)

Figure B.9: Random samples from model window size 8, batch size 400

Figure B.10: Random samples from the WGAN batch size 100 model

Bibliography

[1] Z. Liu, P. Luo, X. Wang and X. Tang, 'Deep learning face attributes in the wild,' in *Proceedings of International Conference on Computer Vision (ICCV)*, Dec. 2015.

[2] D. P. Kingma and M. Welling, 'Auto-encoding variational bayes,' *arXiv preprint arXiv:1312.6114*, 2013.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza *et al.*, 'Generative adversarial networks (2014),' *arXiv preprint arXiv:1406.2661*, vol. 1406, 2014.

[4] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon and B. Poole, 'Score-based generative modeling through stochastic differential equations,' *arXiv preprint arXiv:2011.13456*, 2020.

[5] A. Radford, L. Metz and S. Chintala, 'Unsupervised representation learning with deep convolutional generative adversarial networks,' *arXiv preprint arXiv:1511.06434*, 2015.

[6] S. Bond-Taylor, A. Leach, Y. Long and C. G. Willcocks, 'Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models,' *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[7] M. D. Zeiler and R. Fergus, 'Visualizing and understanding convolutional networks,' in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, Springer, 2014, pp. 818–833.

[8] S. Ioffe and C. Szegedy, 'Batch normalization: Accelerating deep network training by reducing internal covariate shift,' in *International conference on machine learning*, pmlr, 2015, pp. 448–456.

[9] T. Karras, T. Aila, S. Laine and J. Lehtinen, 'Progressive growing of gans for improved quality, stability, and variation,' *arXiv preprint arXiv:1710.10196*, 2017.

[10] T. Karras, S. Laine and T. Aila, 'A style-based generator architecture for generative adversarial networks,' in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[11] A. Vaswani, N. Shazeer, N. Parmar *et al.*, 'Attention is all you need,' *Advances in neural information processing systems*, vol. 30, 2017.

[12] Y. Kim, C. Denton, L. Hoang and A. M. Rush, 'Structured attention networks,' *arXiv preprint arXiv:1702.00887*, 2017.

[13] Z. Liu, Y. Lin, Y. Cao *et al.*, 'Swin transformer: Hierarchical vision transformer using shifted windows,' in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[14] *Words per page: Convert words to pages calculator*, https://wordcounter.net/words-per-page, Accessed on May 10, 2023.

[15] Y. Jiang, S. Chang and Z. Wang, 'Transgan: Two pure transformers can make one strong gan, and that can scale up,' *Advances in Neural Information Processing Systems*, vol. 34, pp. 14 745–14 758, 2021.

[16] W. Shi, J. Caballero, F. Huszár *et al.*, 'Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1874–1883.

[17] B. Zhang, S. Gu, B. Zhang *et al.*, 'Styleswin: Transformer-based gan for high-resolution image generation,' in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 304–11 314.

[18] M. Arjovsky, S. Chintala and L. Bottou, 'Wasserstein generative adversarial networks,' in *International conference on machine learning*, PMLR, 2017, pp. 214–223.

[19] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin and A. C. Courville, 'Improved training of wasserstein gans,' *Advances in neural information processing systems*, vol. 30, 2017.

[20] A. Brock, J. Donahue and K. Simonyan, 'Large scale gan training for high fidelity natural image synthesis,' *arXiv preprint arXiv:1809.11096*, 2018.

[21] T. Miyato, T. Kataoka, M. Koyama and Y. Yoshida, 'Spectral normalization for generative adversarial networks,' *arXiv preprint arXiv:1802.05957*, 2018.

[22] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, 'Improved techniques for training gans,' *Advances in neural information processing systems*, vol. 29, 2016.

[23] S. Kullback and R. A. Leibler, 'On information and sufficiency,' *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[24] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, 'Gans trained by a two time-scale update rule converge to a local nash equilibrium,' *Advances in neural information processing systems*, vol. 30, 2017.

[25] D. Dowson and B. Landau, 'The fréchet distance between multivariate normal distributions,' *Journal of multivariate analysis*, vol. 12, no. 3, pp. 450–455, 1982.

[26] Z. Liu, H. Hu, Y. Lin *et al.*, 'Swin transformer v2: Scaling up capacity and resolution,' in *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[27] Microsoft, *Microsoft/swin-transformer: This is an official implementation for "swin transformer: Hierarchical vision transformer using shifted windows".* Dec. 2022. [Online]. Available: https://github.com/microsoft/Swin-Transformer.

[28] OpenAI, *Openai/improved-gan: This is an official implementation for the paper "improved techniques for training gans".* Nov. 2018. [Online]. Available: https://github.com/openai/improved-gan.

[29] *Lasagne*, Aug. 2015. [Online]. Available: https://pypi.org/project/Lasagne/.

[30] S. Barratt, *inception-score-pytorch: Inception Score for PyTorch*, https://github.com/sbarratt/inception-score-pytorch, Mar. 2020.

[31] G. Parmar, R. Zhang and J.-Y. Zhu, *On aliased resizing and surprising subtleties in gan evaluation*, 2022. arXiv: 2104.11222 [cs.CV].

[32] L. Changyu, *WassersteinGAN-PyTorch: an op-for-op PyTorch reimplementation of Wasserstein GAN*, https://github.com/Lornatang/WassersteinGAN-PyTorch, Jan. 2021.

[33] J. Buolamwini and T. Gebru, 'Gender shades: Intersectional accuracy disparities in commercial gender classification,' in *Conference on fairness, accountability and transparency*, PMLR, 2018, pp. 77–91.

[34] B. N. Patro, V. P. Namboodiri and V. S. Agneeswaran, 'Spectformer: Frequency and attention is what you need in a vision transformer,' *arXiv preprint arXiv:2304.06446*, 2023.