

Wasserstein GANs in Generating Cat Faces

Edward Tan Yuan Chong

DAAA/FT/2B/04

P2214407

Singapore Polytechnic

ABSTRACT

This paper delves into the evolving field of Generative Adversarial Networks (GANs), with a specific focus on the Wasserstein Generative Adversarial Networks (WGANs). The study primarily contrasts the two variants of WGANs – those with weight clipping and those with gradient penalty (WGAN-GP). The architectures are implemented and experimented on a dataset of cat faces in an attempt to generate realistic images. The results reveal a substantial difference in qualitative and quantitative evaluation between the two approaches, where we attained the best FID score of 46.5 and IS of 3.59 using the WGAN-GP, which significantly outperforms the WGAN with weight clipping, and delves into the theory of why so.

I. INTRODUCTION

In recent years, Generative Adversarial Networks [1] (GANs) have emerged as a powerful tool in the realm of machine learning, enabling the creation of realistic data across various domains. As GANs continue to evolve with research, understanding and comparing different GAN architectures that have been researched and developed already, such as Deep Convolutional Generative Adversarial Networks [2] (DCGAN), Conditional Generative Adversarial Nets [3] (cGAN) and Auxiliary Classifier Generative Adversarial Network [4] (ACGAN) are crucial for advancing and improving the capabilities of generative models. In this paper, we are focusing on a specific architecture that was researched, called Wasserstein Generative Adversarial Networks (WGANs), proposed by Arjovsky, M. [7], and the issues around the performance of WGANs in generating images, methods implemented and researched on to improve WGANs performance and apply these into an experiment in generating cat faces.

A. BACKGROUND OF GANs

Generative Adversarial Networks (GANs), introduced by Ian Goodfellow and his colleagues in 2014, have revolutionized the field of generative modelling.

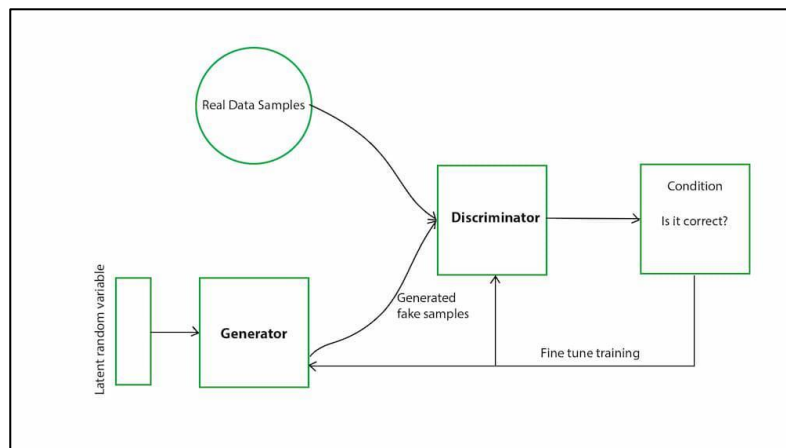


Figure 1 GAN Architecture [8]

GANs consist of two networks, the generator and the discriminator as shown in Figure 1, where they essentially compete against each other in a zero-sum game, with the generator trained to generate synthetic data, such as images, from a latent vector, that are intended to be indistinguishable from real data, while the discriminator attempts to discern whether the generated samples are real (drawn from the training dataset) or fake (created by the generator). [1]

The competition is mathematically represented through the Minimax function shown below in Figure 2 [1], however, we will not elaborate on the function as our focus is not on traditional GANs.

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

Figure 2 GAN Minimax function [1]

The generator’s goal would then be to increase the error rate of the discriminator by generating realistic images to “fool” the discriminator. Over time, the generator improves its ability to create realistic images, and the discriminator becomes better at discerning real images from fake ones. The end goal would be for the generator to create images so convincing that the discriminator is unable to tell them apart from real images.

Balancing the training of GANs is crucial as if the discriminator is too effective in discerning real from fake images, the generator will not be able to improve significantly as the discriminator will only accept generated images that are identical to the images from the training dataset [5]. On the other hand, if the generator generates very realistic synthetic data that is indistinguishable from real data, the discriminator may be unable to differentiate between the two. The goal is to achieve an equilibrium where the generator produces images so convincing that the discriminator cannot tell the difference from real images.

B. BACKGROUND OF WGANs

The Wasserstein GAN, proposed by Martin Arjovsky in 2017, represented a significant advancement in the field of Generative Adversarial Networks. The key innovation of WGANs lies in their use of the Wasserstein Distance, also known as the Earth Mover’s Distance (EMD), as a loss function, which addresses fundamental challenges encountered in traditional GANs, such as mode collapse, where the generator learns to produce a limited variety of outputs or the lack of a meaningful loss metric during training, and more.

In terms of architecture, there are many differences between WGANs and traditional GANs. Firstly, WGANs use the Wasserstein Distance as their loss function, which measures how much effort it takes to transform the distribution of the generated data into the distribution of the real data. This provides a smoother gradient to the generator, helping to address the mode collapse issue traditional GANs face, leading to better convergence, compared to traditional GANs that use binary-cross entropy as their loss function which can lead to issues like mode collapse. Secondly, WGANs enforce the Lipschitz constraint in the critic through weight clipping. The Lipschitz constraint is a fundamental concept of WGANs as they ensure that the critic is a 1-Lipschitz function, and by enforcing a 1-Lipschitz constraint in WGANs, it was found that it helps to stabilize the training process by ensuring that the gradients are bounded within a range, and prevents the vanishing or exploding gradient problems that are commonly found in traditional GANs, alongside many more benefits in training. For WGANs, they utilized weight clipping, where the weights of the critic are clipped between a specified range, for example, 0.01, and if the weights exceed this range, it is clipped back to the boundary. This helps to ensure that the weights of the critic do not grow too much and helps to keep the gradients under control. Thirdly, WGANs also introduce critic iterations in their architecture, unlike traditional GANs where the critic (discriminator) is trained to classify input as real or fake, in WGANs, the critic is instead trained to evaluate the quality of generated samples. Hence, the critics are trained and updated more frequently per generator update, allowing the critic to provide more accurate and stable feedback to the generator, enhancing the training process.

However, it was found that WGANs that enforce the Lipschitz constraint using weight clipping were quoted “a clearly terrible way to enforce a Lipschitz constraint” [7], where if the clipping parameter is too large, it takes a long time for the weights to reach the limit enforced, and if the clipping parameter is too small, it can easily lead to vanishing gradients where the number of layers is big or batch normalization is not utilized, as mentioned in [7]. Hence, this paper will also explore WGANs using Gradient Penalty (WGAN-GP), which was introduced in the research paper titled “Improved Training of Wasserstein GANs” [10], where they explored the difficulties with weight constraints from weight clipping, and how it led to inability to converge and optimization difficulties. The paper then proposes a better alternative to enforce the Lipschitz constraint, through Gradient Penalty, with a new objective function shown in Figure 3 below.

$$L = \underbrace{\mathbb{E}_{\tilde{x} \sim \mathbb{P}_g} [D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_r} [D(x)]}_{\text{Original critic loss}} + \underbrace{\lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} [(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Our gradient penalty}}.$$

Figure 3 Objective function of WGAN-GP

The way gradient penalty works is through directly constraining the gradient norm of the critic’s output with respect to its input [10], enforcing a soft version of the constraint with a penalty on the gradient norm for random samples as well, compared to weight clipping which is a hard version of the constraint. This paper will conduct an experiment to compare the two architectures of WGAN – with weight clipping, and with gradient penalty.

II. RELATED WORKS

There have been multitudes of works on WGANs and WGAN-GPs. One such example of a WGAN-GP application was done by the user “laurahanu” on GitHub, where they utilized WGAN-GP on an MRI Dataset to produce MRI images, as shown in Figure 3 below [13].

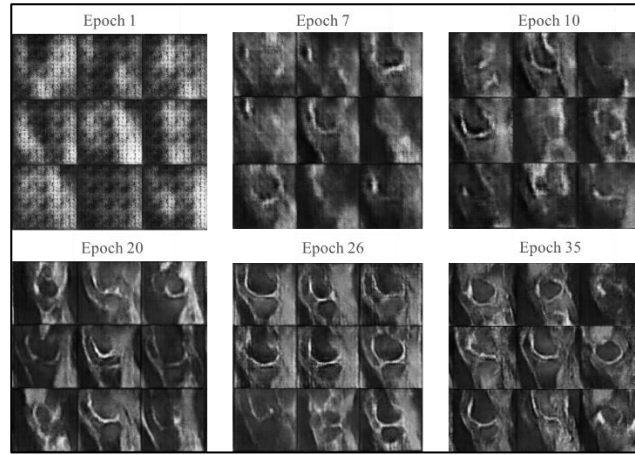


Figure 4 WGAN-GP on MRI Dataset [13]

We can clearly see that the MRI images generated by the WGAN-GP are high quality and clear, highlighting the performance of WGAN-GP

The original WGAN paper also experimented on the LSUN-Bedrooms dataset, showing great performance of the WGAN in generating realistic images, having similar performance to a standard GAN algorithm, as shown in Figure 5 below, where the left is the WGAN algorithm and the right is the standard GAN algorithm, both trained with DCGAN generator. [7]

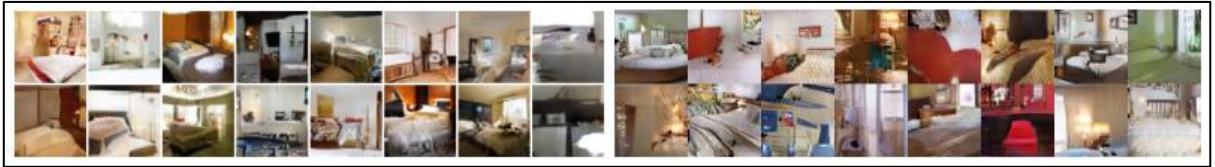


Figure 5 WGAN on LSUN-Bedrooms Dataset [7]

This shows how effective WGANs, and its variants are in generating realistic images in a small number of epochs and shines light on their image generation performance, having more stable training and optimization.

Similarly, in the WGAN-GP paper titled “Improved Training of Wasserstein GANs”, they experimented with the WGAN-GP architecture they proposed against a few other GAN architectures, including WGANs, and their results from the LSUN-Bedrooms Dataset are shown below in Figure 6.






DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline (G : DCGAN, D : DCGAN)			
			
G : No BN and a constant number of filters, D : DCGAN			
			
G : 4-layer 512-dim ReLU MLP, D : DCGAN			
			
No normalization in either G or D			
			
Gated multiplicative nonlinearities everywhere in G and D			
			
tanh nonlinearities everywhere in G and D			
			
101-layer ResNet G and D			
			

Figure 6 WGAN-GP Results from WGAN-GP paper [10]

We can clearly see that the WGAN-GP architecture outperforms all other architectures tested, especially the WGAN architecture, proving how effective gradient penalty is in enforcing the Lipschitz constraint and leading to better image generation performance. Hence, in this paper, we will experiment the difference in performance in generating realistic images between WGAN and WGAN-GP in a dataset containing cat faces.

III. METHODOLOGY

A. DATASET

In this paper, we will be utilizing the 64x64 cat faces dataset from Kaggle [9]. An example of the images is shown below in Figure 7. The initial dataset contained 15,700 images of cat faces, however, due to lack of computational resources, we will only utilize 2,000 images.

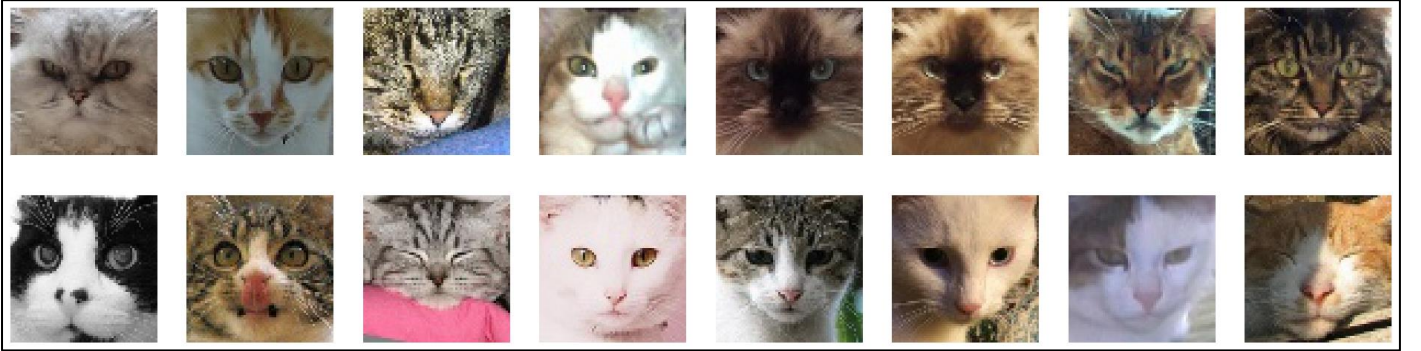


Figure 7 Images of Cat Faces Dataset

B. ARCHITECTURES

The WGAN model used in this paper follows the algorithm introduced in WGAN’s research paper, titled “Wasserstein GAN” [7]. The algorithm is shown below in Figure 8.

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: α , the learning rate. c , the clipping parameter. m , the batch size. n_{critic} , the number of iterations of the critic per generator iteration.

Require: w_0 , initial critic parameters. θ_0 , initial generator's parameters.

```

1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  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:      $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))]$ 
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 8 WGAN Algorithm [7]

The algorithm for the WGAN gradient penalty follows the algorithm shown in Figure 9, similar to the WGAN but includes modifications, such as applying gradient penalty to enforce the Lipschitz constraint instead of weight clipping and using instance normalization layers instead of batch normalization in the critic and uses the Adam optimizer instead of the RMSProp optimizer. The architectures of the generators for both models were kept the same.

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ , the number of critic iterations per generator iteration n_{critic} , the batch size m , Adam hyperparameters α, β_1, β_2 .

Require: initial critic parameters w_0 , initial generator parameters θ_0 .

```

1: while  $\theta$  has not converged do
2:   for  $t = 1, \dots, n_{\text{critic}}$  do
3:     for  $i = 1, \dots, m$  do
4:       Sample real data  $x \sim \mathbb{P}_r$ , latent variable  $z \sim p(z)$ , a random number  $\epsilon \sim U[0, 1]$ .
5:        $\tilde{x} \leftarrow G_\theta(z)$ 
6:        $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\tilde{x}$ 
7:        $L^{(i)} \leftarrow D_w(\hat{x}) - D_w(x) + \lambda(\|\nabla_{\hat{x}} D_w(\hat{x})\|_2 - 1)^2$ 
8:     end for
9:      $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$ 
10:   end for
11:   Sample a batch of latent variables  $\{z^{(i)}\}_{i=1}^m \sim p(z)$ .
12:    $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^m -D_w(G_\theta(z)), \theta, \alpha, \beta_1, \beta_2)$ 
13: end while

```

Figure 9 WGAN-GP Algorithm [10]

The WGAN generator and critic implemented in PyTorch are shown below in Figure 10.

```

WGANGenerator(
  (model): Sequential(
    (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
  )
  (1): Sequential(
    (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
  )
  (2): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
  )
  (3): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
  )
  (4): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (5): Tanh()
)

WGANCritic(
  (model): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.9, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (2): Sequential(
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.9, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (3): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (4): Sigmoid()
  )
)

```

Figure 10 WGAN Implementation

The WGAN-GP generator and critic implemented in PyTorch are shown below in Figure 11.

```

WGANGPGenerator(
  (model): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (1): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (2): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (3): Sequential(
      (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (4): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (5): Tanh()
  )
)

WGANGPCritic(
  (model): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(256, eps=1e-05, momentum=0.9, affine=False, track_running_stats=False)
      (2): ReLU(inplace=True)
    )
    (2): Sequential(
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(512, eps=1e-05, momentum=0.9, affine=False, track_running_stats=False)
      (2): ReLU(inplace=True)
    )
    (3): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (4): Sigmoid()
  )
)

```

Figure 11 WGAN-GP Implementation

The hyperparameters tested in this paper were obtained through empirical testing. For the WGAN, we found that the n_{critic} value of 3 and α , the learning rate, of 0.0002, and the alpha of 0.9 for the RMSProp optimizer yielded the best results in generating realistic images. For the WGAN-GP, we found that the n_{critic} value of 3 and α , the learning rate, of 0.0002, similar to the WGAN, with beta values β_1 and β_2 of 0.0 and 0.9 respectively for both the generator and critic Adam optimizers, and a λ , lambda for gradient penalty, of 10 worked best in producing realistic images.

C. EVALUATION METHODS

For evaluation methods, we will mainly use qualitative means of evaluating, along with quantitative metrics such as the Fréchet Inception Distance (FID) score, and Inception Score (IS), which are widely used quantitative metrics for evaluating the performance of GANs. They provide a numerical indication of the quality and diversity of images generated by GANs. The FID score measures the similarity between the distribution of generated images and real images by comparing the distance between the feature vectors of generated images to real images. A lower FID score indicates that the quality of images generated by the generator is higher and similar to real ones, indicating better performance and our goal would be to minimize it as much as possible [11]. The Inception Score (IS) measures how good the generated image is in terms of quality, and how diverse the generated image is. A higher IS indicates good image quality and diversity, and our goal is to maximise it as much as possible too [12]. Lastly, we use human evaluation as an evaluation method, as quantitative metrics are not the best basis for comparison and evaluation of GANs due to the lack of holistic evaluation, and are unable to capture human perception, hence the best means to determine if the images generated are realistic is for a human to evaluate them.

IV. DISCUSSION

A. TRAINING

In this paper, we trained both architectures for 200 epochs, then saved the best weights whenever there was an improvement in the FID score, such that we will have the best model weights for model evaluation. The results of each architecture are shown below, with Figure 12 showing the results of the WGAN architecture utilizing weight clipping to enforce the Lipschitz constraint.

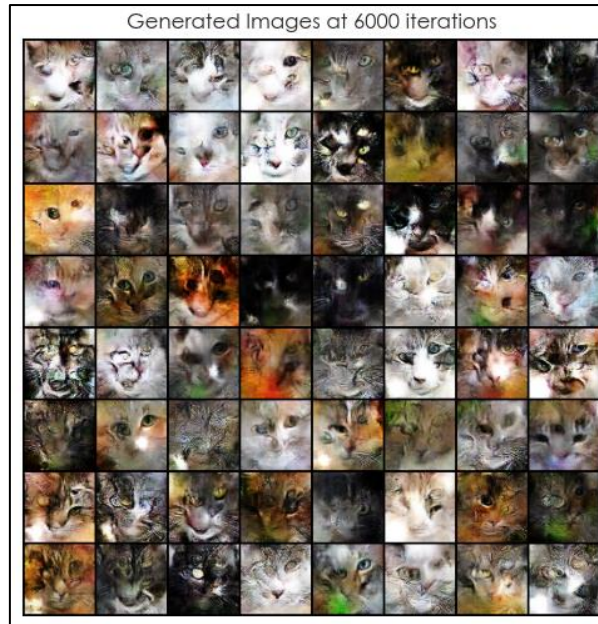


Figure 12 Best WGAN Results from Human Evaluation

From the images generated by the WGAN, generally, most images are of poor quality as some of the cat faces are rather indistinguishable, where there are just colours and some texture. While some images are rather distinguishable, where we can see the features of the cat that the generator was trying to mimic, but not very clearly and with random colour splotches here and there and the model was unable to capture the shape of the cat faces. Furthermore, some of the images seem to have just produced a black image, which is likely an attempt of the generator in producing images of black cats but failing to do so. In terms of quantitative metrics, the WGAN also only produced the lowest FID score of 160.6 and highest IS of 2.46.

However, for WGANs with gradient penalty applied, we notice a significant improvement in the results, both in terms of quantitative and qualitative metrics. The images generated by WGAN with gradient penalty are shown below in Figure 13.

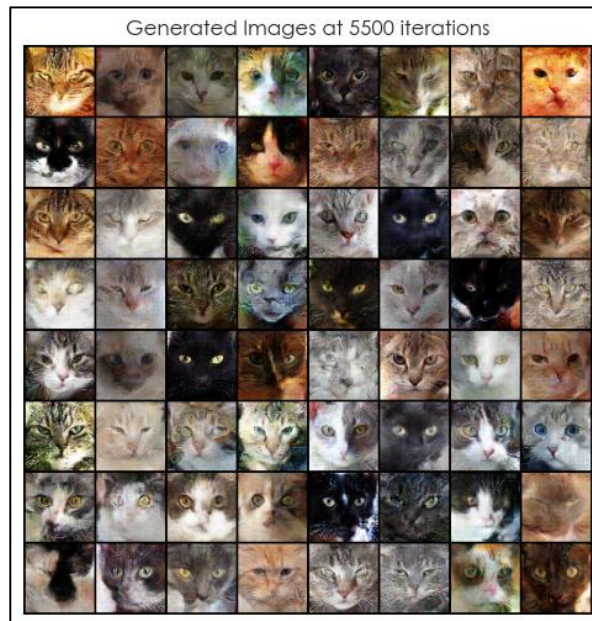


Figure 13 Best WGAN-GP Results from Human Evaluation

We can clearly see a huge difference in the quality of images generated compared to WGANs using weight clipping. The majority of images generated here are rather distinguishable, as we can clearly identify the features of the cat's face such as its eyes, nose, and mouth correctly, telling us that the model was able to capture these features well. Furthermore, in terms of quantitative results, WGAN-GP achieved the lowest FID score of 74.4, and the highest IS of 2.55.

One thing to note here was that the images selected as the best results were based on human evaluation, as even though the FID score and IS were lower for other images, it seemed that the quality of images generated at that iteration were poorer compared to the one selected from human evaluation. This shows the importance of not being overly reliant on quantitative metrics when evaluating GANs, as they may be unreliable at times.

B. MODEL EVALUATION

After training, we load back the best weights of each generator in the WGAN and WGAN-GP, evaluate them on FID score and IS, and also use the best generators to generate 64 images to plot an 8x8 image plot to compare their performance. Figure 14 below shows the evaluation of the WGAN architecture. The WGAN was able to obtain the best FID score of 97.6 and IS of 3.65.

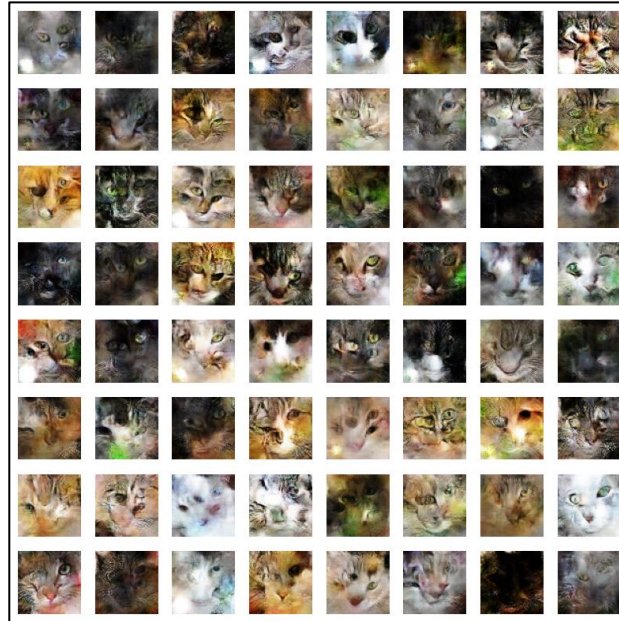


Figure 14 WGAN Model Evaluation

From the WGAN model evaluation results, we can see that the WGAN does not perform that well, similar to the evaluation from training, where only some of the images generated are distinguishable. Despite that, we can still see a decrease in the FID score and an increase in the IS score, likely due to the larger sample size used here, providing a more accurate representation of the data distributions.

Figure 15 below shows the model evaluation of the WGAN-GP architecture. The WGAN-GP was able to obtain the best FID score of 46.5 and IS of 3.59.

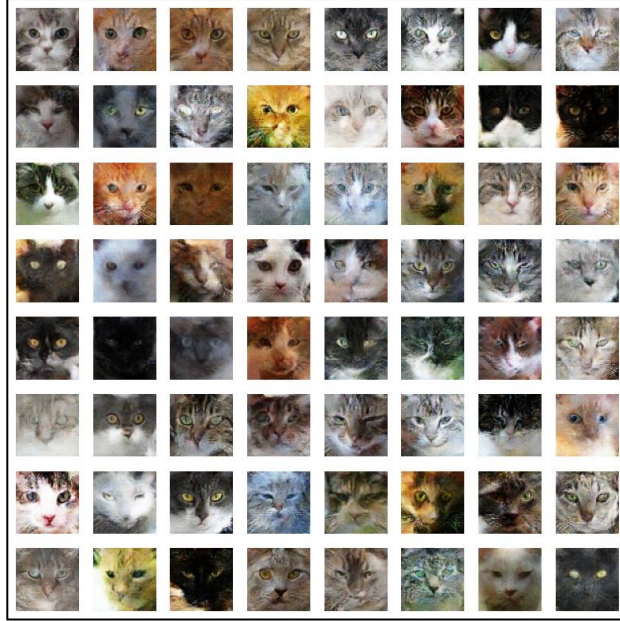


Figure 15 WGAN-GP Model Evaluation

From the WGAN-GP model evaluation results, we can see that the WGAN-GP performs very well in producing realistic images of cat faces. Most of the images of the cat faces that were generated are distinguishable and high quality, as we are able to identify most features of the cat that the WGAN struggled to replicate. Furthermore, there was similarly an increase in FID score and IS score as well, likely due to the increased sample size for more accurate calculations, and in general, telling us that the model performed very well in terms of generating high-quality images.

Comparing the two sets of images generated by the two Wasserstein GANs – one using weight clipping and another using gradient penalty – provided us with an insight into their difference in performance in generating realistic images. The quality of images generated by the WGAN with gradient penalty is clearly significantly higher than that of the WGAN with weight clipping, as seen through quantitative and qualitative evaluation. This can be due to several factors between the two architectures, such as WGAN with gradient penalty proving to have a more stable training process as it is able to add a soft constraint onto the critic weights that maintains the Lipschitz continuity more gently compared to weight clipping that imposes hard constraints on the critic weights, potentially leading to optimisation difficulties, or in terms of quality of gradients, where the gradient penalty penalises the model if the gradient norm deviates from 1 (1-Lipschitz condition), allowing the critic’s gradients to be more informative and reliable during training. In contrast, weight clipping does not directly enforce the smoothness of the function but rather has hard constraints that limit the weight’s values, which is a less elegant method and results in erratic gradients and thus poorer image quality generated. This goes to show that it was true that weight clipping is a rather “terrible” way to enforce the Lipschitz constraint in WGANs, and that gradient penalty would be a better alternative.

This experiment proves the difference between the performances of the two WGAN variants – one with weight clipping and another with gradient penalty, and how the different methods of enforcing the Lipschitz constraint can cause such a stark difference in image generation quality.

V. CONCLUSION

In conclusion, we explored WGANs with weight clipping and WGANs with gradient penalty, and how they are applied, and an application of them in generating cat faces, comparing the results of the two, allowing us to see the difference in the enforcement of the Lipschitz constraint and its impact on the images generated.

Furthermore, through this technical paper, we were able to generate clear and distinguishable images of cat faces with our WGAN with gradient penalty, attaining the best FID score of 46.5, and best IS of 3.59, but there is still room for improvement, such as through more in-depth hyperparameter tuning. However, due to a lack of computational resources and a time constraint, we were not able to do so.

The following are a few improvements that I wish to make if the circumstances and time constraints allow. I would like to perform more in-depth hyperparameter tuning to improve the results of both the WGAN and WGAN-

GP, tuning hyperparameters such as n_{critic} or the type of optimizers used, especially since GANs are very sensitive to hyperparameters, a set of hyperparameters may significantly boost our results. On top of that, I would like to make changes and attempt to improve or deepen the architecture in the generator and critic for the WGAN and WGAN-GP in attempts to improve their results, and potentially achieve a single-digit FID score.

This paper also sheds light on the importance of the method in enforcing the Lipschitz constraint as we can see the stark difference in image quality generation when each of the methods were tested on the dataset. With more research done in the future, more variants would be introduced to enforce the Lipschitz constraint more elegantly and spearhead the development of GANs and WGANs.

ACKNOWLEDGEMENT

I would like to thank Dr Saw Vee Liem who implemented this technical paper section into our Deep Learning assignment. I would also like to thank Aladdin Persson who created YouTube tutorials explaining the concept of WGANs with weight clipping and gradient penalty that provided me with the fundamental knowledge of WGANs, which inspired me to write about WGANs.

REFERENCES

- [1] Goodfellow, I.J. et al. (2014) Generative Adversarial Networks, arXiv.org. Available at: <https://arxiv.org/abs/1406.2661> (Accessed: 10 January 2024)
- [2] Radford, A., Metz, L. and Chintala, S. (2016) Unsupervised representation learning with deep convolutional generative Adversarial Networks, arXiv.org. Available at: <https://arxiv.org/abs/1511.06434> (Accessed: 10 January 2024)
- [3] Mirza, M. and Osindero, S. (2014) Conditional generative adversarial nets, arXiv.org. Available at: <https://arxiv.org/abs/1411.1784> (Accessed: 10 January 2024)
- [4] Odena, A., Olah, C. and Shlens, J. (2017) Conditional image synthesis with auxiliary classifier Gans, arXiv.org. Available at: <https://arxiv.org/abs/1610.09585> (Accessed: 10 January 2024).
- [5] Elbadawi, M., Li, H. and Sun, S. (2024) Artificial Intelligence generates novel 3D printing formulations, Applied Materials Today. Available at: <https://www.sciencedirect.com/science/article/pii/S2352940724000076> (Accessed: 17 January 2024)
- [6] Papers with code - CIFAR-10 dataset (no date) Dataset | Papers With Code. Available at: <https://paperswithcode.com/dataset/cifar-10> (Accessed: 14 January 2024)
- [7] Arjovsky, M., Chintala, S. and Bottou, L. (2017) *Wasserstein GAN*, arXiv.org. Available at: <https://arxiv.org/abs/1701.07875> (Accessed: 30 January 2024).
- [8] *Generative Adversarial Network (GAN)* (2023) *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/generative-adversarial-network-gan/> (Accessed: 30 January 2024).
- [9] www.kaggle.com. (n.d.). Cats faces 64x64 (For generative models). [online] Available at: <https://www.kaggle.com/datasets/spandan2/cats-faces-64x64-for-generative-models/data> [Accessed 30 Jan. 2024].
- [10] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. and Courville, A. (n.d.). *Improved Training of Wasserstein GANs*. [online] Available at: <https://arxiv.org/pdf/1704.00028.pdf>.
- [11] Ahirwar, K. (2020). A Very Short Introduction to Frechlet Inception Distance(FID). [online] Medium. Available at: <https://medium.datadriveninvestor.com/a-very-short-introduction-to-frechlet-inception-distance-fid-86c95deb0930>.
- [12] Enterprise AI. (n.d.). *What is an inception score (IS)?* [online] Available at: [https://www.techtarget.com/searchenterpriseai/definition/inception-score-IS#:~:text=The%20inception%20score%20\(IS\)%20is](https://www.techtarget.com/searchenterpriseai/definition/inception-score-IS#:~:text=The%20inception%20score%20(IS)%20is).
- [13] Hanu, L. (2023). laurahanu/Improved-Wasserstein-GAN-application-on-MRI-images. [online] GitHub. Available at: <https://github.com/laurahanu/Improved-Wasserstein-GAN-application-on-MRI-images> [Accessed 31 Jan. 2024].