Racing-GAN Multi-Generator Framework within Wasserstein And Deep Convolution Generative Adversarial Networks

Authors: Andrew Paul, Tanujay Saha

Abstract – Generative adversarial networks(GAN) are machine learning systems featuring a generative neural network competing against a discriminatory classification network to produce new data, authentic enough to imitate the probability distribution of the input. The data being referred to in this study are images since they are spatially simple. GANs have recently been adapted to involve multiple generators in parallel competing against the discriminator in an effort to reduce their training time. One approach, coined Racing-GAN, was decidedly successful in lessening the training time for the typical generator and discriminator architecture [3]. This study involves using the competitive multi-generator framework of Racing-GAN to not just compare timing efficiency, but also the effectiveness of the model in producing quality images relative to the standard GAN, as well as applying it to Wasserstein GANs and Deep Convolutional GANs (DCGAN). The novelty lies in the multi-generator framework, the Racing-GAN model. While other GAN models have experimented with parallelization, the Racing-GAN differs as it makes use of a modified loss function [3].

1 Introduction

Generative adversarial networks are machine learning systems involving a generative neural network competing against a discriminatory classification network to produce new data, authentic enough to be imitate the probability distribution of the input. The competition ideally drives both networks to improve until the new data produced is indistinguishable from the input actuality [1]. The standard GAN architecture features several issues however, related to instability in training, inefficient training, and modal collapse of the data output. The development of Wasserstein GAN and Deep Convolutional GANs (DCGAN) served to overcome the issues of modal collapse and training instability through the inclusion of differences in the divergence functions and differences in the neural architecture respectively [2][6].

In previous studies, it has been suggested that the inclusion of multiple generators can influence the efficiency and effectiveness, as well as provide a solution to the modal collapse problem.

Racing-GAN is a multi-generator structure which utilizes a modified loss function and has been decidedly successful in increasing the training performance [3]. In an attempt to assess the abilities of this multi-generator architecture, this research is aimed at comparing the different GAN architecture with the Racing-GAN through comparison of efficiency and quality of data generated, as well as determining its effect on the problem of modal collapse.

2 Background

2.1 Standard GAN

In the standard GAN, the competing models are both multilayer perceptrons, where the generator's data space, defined as $G(z; \theta_g)$, takes in an input latent sample of random numbers P_z and outputs processed data P_g to the Discriminator $D(x; \theta_d)$, which tries to distinguish between the probability distributions of the generated data and the actual [1]. G is trained to produce indistinguishable counterfeit data, by attempting to minimize the difference in the probability distributions. On the other hand, the D is simultaneously trained to distinguish between the counterfeit and the actual, by maximizing the probability of correctly labeling the input [1][3]. The result is a min-max game of the following mathematical form:

$$min_G \, max_D \, E_{x \leftarrow P_{data}} [\log D(x)] + E_{z \leftarrow Pz} [1 - \log D(G(z)]$$

A prominent problem with the standard GAN is modal collapse. The generator is sometimes incapable of representing all of the different modes of the actual data in its output. This results in the output lacking the diversity of the actual data [1][2][3].

2.2 Wasserstein GAN

In an effort to counter the modal collapse problem the Wasserstein GAN was developed. Wasserstein differs from the standard GAN due to its use of a different method to find the divergence between the probability distributions of the input data and the new data generated. GANs normally feature the Jenson-Shannon (JS) divergence, the Kullback-Leibler divergence, or the Total Variation divergence functions however, Wasserstein employs the Earth-Movers (EM) divergence function [2].

The EM function is continuous and differentiable, meaning it is less likely to be hindered by vanishing gradients and further, allows the discriminator to be trained to optimality. The ability to train to optimality thus allows for increased representation of the different modes of the actual data in the output – hence providing a solution to the modal collapse problem [2].

The EM function is preferred over the standard GANs function given it improves the stability of learning and prevents modal collapse – a prominent issue with many networks [2].

2.3 Deep Convolutional GAN

DCGAN, referring to Deep Convolutional GAN, increases stability of the network via changes to the standard architecture. It utilizes convolutional neural networks replacing pooling layers in both the generator and discriminator with convolutions and adding batch normalizations [6][8]. The generator uses ReLU activation in its layers except the output where tanh is employed. The discriminator uses the Leaky ReLU activation function in its layers [6][8]. The inclusion of the convolutional neural network infrastructure aids in feature extraction, allowing for better analysis of the generated images with the real [6].

The DCGAN does not effectively counter the modal collapse problem although it does provide increased stability in training according to a previous study [6].

2.4 Racing GAN

Racing GAN is a multi-generator framework which utilizes a modified loss function [3].

$$\max_D \, E_{X \leftarrow P_{data}} [\log D(x)] + E_{z \leftarrow P_z} [\sum_{n=1}^{k} \{1 - \log D(G_n(z)\}]$$

Taking a generators G_i and generators G_j, we can state the function for the Racing-GAN generator model:

$$\min_{G_i} E_{z \leftarrow P_z} [1 - \log(D(G_i(z)) + \sum_{j \in \{1,..k\}, j \neq i} \{\max(0, D(G_i(z)) - D(G_j(z)))\}]$$

Conceptually, the modified loss function allows the different generators to produce dissimilar images of the input and learn from each other's performances as well as the discriminator's classification of their own [3]. This not only should increase performance but should also allow the model to cater to different modes of the actual data, given it can capture the range of probability distributions in each image generated with the multiple generators. This theoretically should provide a solution to modal collapse – we hope to determine this during the experiment [3].

2.5 Fréchet Inception distance (FID) scores

FID is a score which, similar to the aforementioned divergence functions, quantifies the distance between the real and generated image [9].

FID provides a reliable means of which to assess the precision and accuracy of the different aforementioned GAN models [11].

FID is sensitive to mode collapse which provides a benefit in comparing across models, especially for this study [11].

The Fréchet distance can be formulated as:

$$FID(x, g) = \left|\left|\mu_x - \mu_g\right|\right|_2^2 + Tr(\Sigma_x + \Sigma_y - 2(\Sigma_x\Sigma_g)^{\frac{1}{2}})$$

where (μ_x, Σ_x), and (μ_g, Σ_g) are the mean and covariance of the real images, x, and generated images, g, from the data distribution and model.

2.6 Understanding GAN

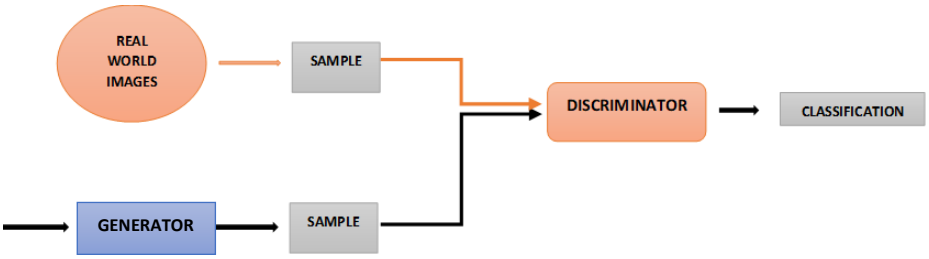


Figure 1, giving a conceptual view of the GAN structure. The generator is fed latent sample of a random numbers, generates an image, and feeds it to the discriminator. The discriminator is fed real world images during its training and takes in the generator's output and classifies it as real or fake depending on its probabilistic distribution [1].

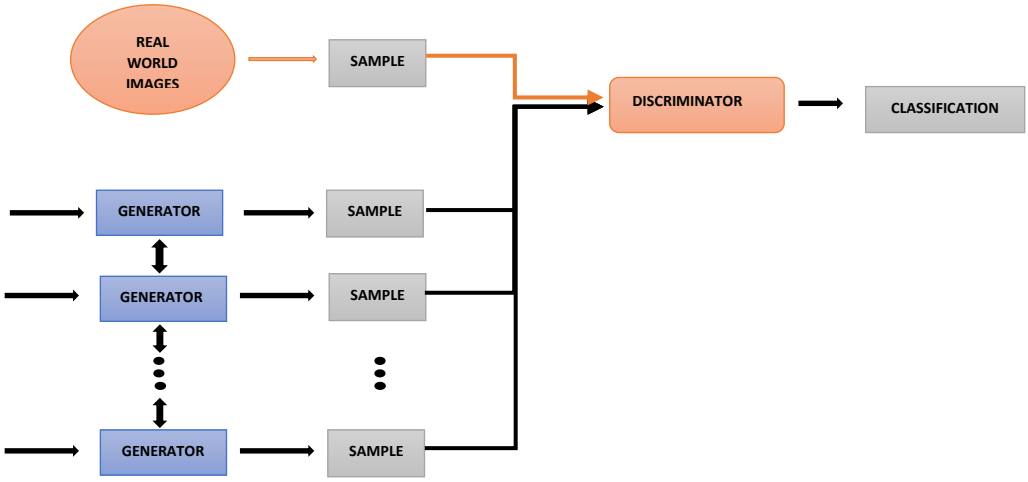


Figure 2, giving a conceptual view of the Racing-GAN structure. Multiple generators with modified loss functions take in latent samples of random numbers, generate separate dissimilar images, and pass them to the discriminator. The generators compete with each other to produce realistic images against the discriminator [3].

3. Motivation

GANs are still a relatively new field of research. While GANs continue to make great strides in image generation, optimization techniques are still being explored in terms of increasing the speed of training and simultaneously improving on the quality of fakes generated.

This work builds off of the introduction of Racing-GAN given in a previous paper, "Racing-GAN: A Competitive Multi-Generator Framework" [3]. In the previous paper, only the speed of the Racing-GAN was assessed in comparison to the standard GAN. We extend this assessment to WGAN and DCGAN, and include quality of the fakes as a factor being compared. Furthermore, we hope to determine whether Racing-GAN is actually a viable solution to the modal collapse problem.

Thus, this study serves to compare the effectiveness and efficiency, of different GAN models with Racing-GAN.

4. Approach

DCGAN and WGAN models were created using Python and its Pytorch library. Its functionality was tested in producing a small sample of images - then was run for around 7 hours to generate around 1000 images for the WGAN, 60 images for DCGAN Trial 1, 500 images for DCGAN Trial 2, and an equivalent number of images generated for their respective Racing-Gan adaptations. Note that the difference in the number of images observed for the different structures was due to the inability to not retain the gradient graphs in the backwards call of the loss function in the DCGAN algorithm, requiring much more time and resources to achieve fewer iterations – notably in the Racing-GAN adaptations as they required multiple generators. The models were then taken and modified to incorporate the loss function expressed by the Racing-GAN proposal.

The models created:

- Standard WGAN
- Standard DCGAN
- Racing-WGAN (2 Generators)
- Racing-DCGAN (2 Generators)
-

For the Racing GAN, this study will feature the use of 2 generators G_i and G_j [3]. Thus, the loss function becomes:

$$\min_{G_i} \mathrm{E}_{z \leftarrow \mathrm{P}_z} [1 - \log(\mathrm{D}(\mathrm{G}_i(\mathrm{z})) + \max(0, \mathrm{D}(\mathrm{G}_i(\mathrm{z})) - \mathrm{D}(\mathrm{G}_j(\mathrm{z})))]$$

The FID algorithm was also created using python and data visualization library matplotlib to read in a batch of images and compare their distance from another batch of real images. Each model would produce a set of images and they would be fed into this algorithm where a graph of the Fid score per iteration was processed.

Note that the term iteration is used here to mean the points in which the images were sampled from the generator output and do not necessarily coincide with the epochs of training. This was done due to timing constraints in the model training and also to allow for more continuous data points. For WGAN and DCGAN the images were sampled at a rate of 37 per epoch.

Their FID scores recorded throughout for each iteration.

$$FID(x, g) = \left|\left|\mu_x - \mu_g\right|\right|_2^2 + Tr(\Sigma_x + \Sigma_y - 2(\Sigma_x \Sigma_g)^{\frac{1}{2}})$$

5. Results

Results of the Experimentation with the WGAN

Figure showing the type of noise fed into the generators and the types of real images fed into the discriminator from left to right respectively.

Figure showing a compilation of outputs from the Wasserstein Racing-GAN over several iterations.

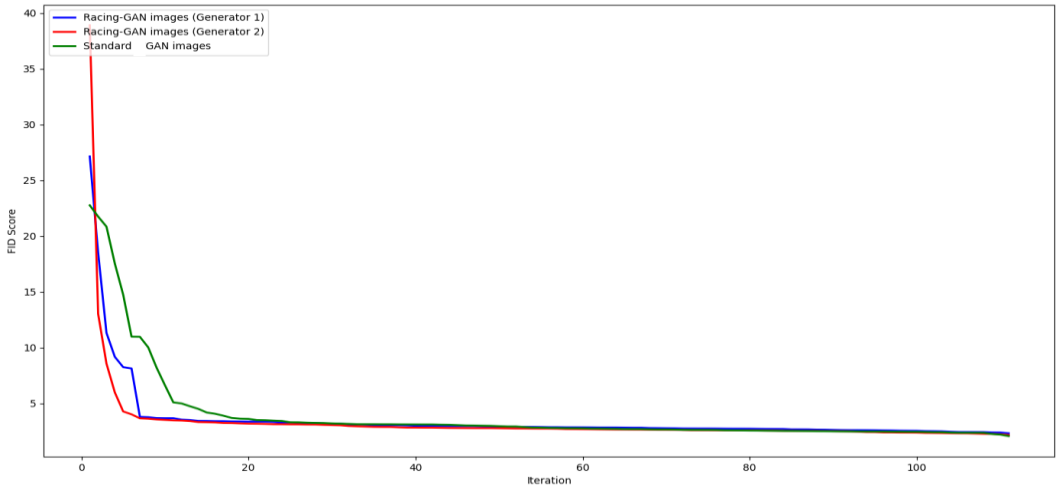


Figure showing the FID scores of the images generated in the Racing-GAN and the standard models relative to the real images.

Loss Function translated to python code and adapted to work in with the PyTorch machine learning library:

```
> loss_D = -(torch.mean(discriminator(real_imgs)) - ((torch.mean(discriminator(fake_imgs1)) + torch.mean(discriminator(fake_imgs2)))/2))
> loss_G1 = -torch.mean(prob1) - max(0.0, float(torch.mean(prob2 - prob1)))
> loss_G2 = -torch.mean(prob2) - max(0.0, float(torch.mean(prob1 - prob2))
```

Results of the Experimentation with the DCGAN

- **Trial 1**

Figure showing the type of noise fed into the generators and the types of real images fed into the discriminator from left to right respectively.

Figure showing the images produced by the Generator 1 of the DC Racing-GAN implementation over several iterations

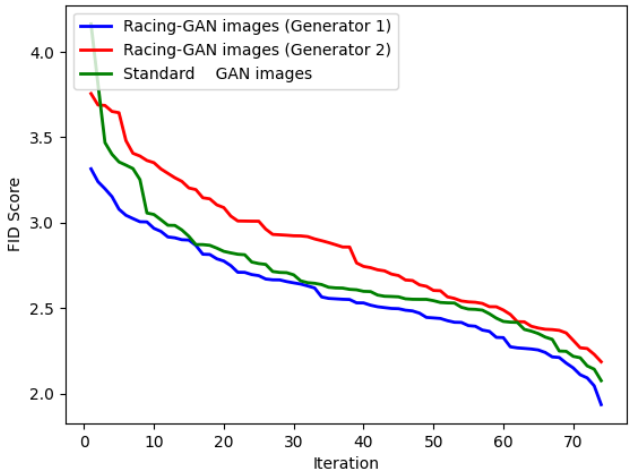


Figure showing the FID scores of the images generated in the Racing-GAN and the standard models relative to the real images.

Loss Functions translated to python code and adapted to work in with the PyTorch machine learning library:

```
> prob1 = discriminator(gen_imgs1)
> prob2 = discriminator(gen_imgs2)
> g_loss1 = adversarial_loss(prob1, valid) + adversarial_loss(torch.max(torch.zeros(1), (prob1 - prob2)), valid)
> g_loss2 = adversarial_loss(prob2, valid) + adversarial_loss(torch.max(torch.zeros(1), (prob2 - prob1)), valid)
> # Measure discriminator's ability to classify real from generated samples
> real_loss = adversarial_loss(discriminator(real_imgs), valid)
> fake_loss = adversarial_loss(((discriminator(gen_imgs1.detach()) + discriminator(gen_imgs2.detach()))), fake)/2
> d_loss = (real_loss + fake_loss)/2
```

Trial 2

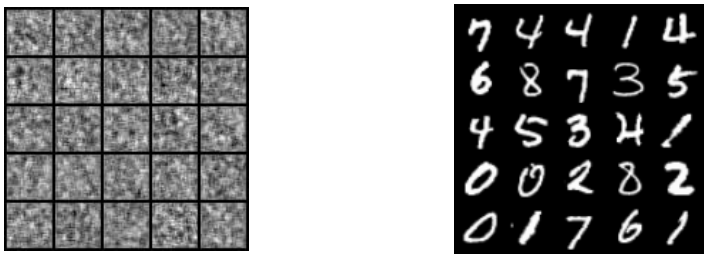


Figure showing the type of noise fed into the generators and the types of real images fed into the discriminator from left to right respectively.

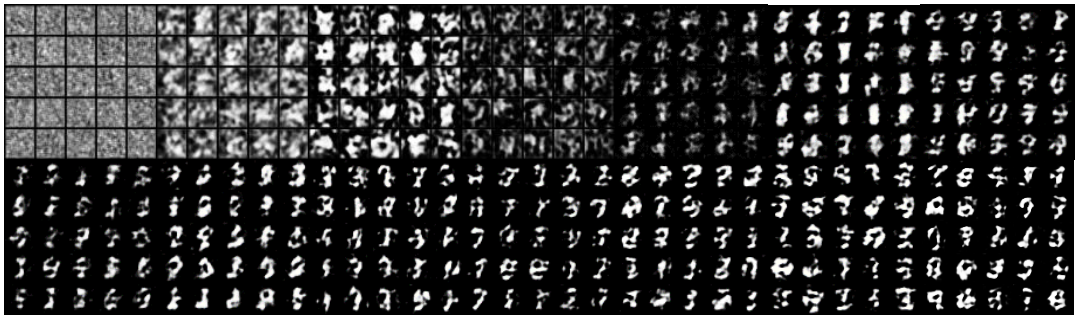
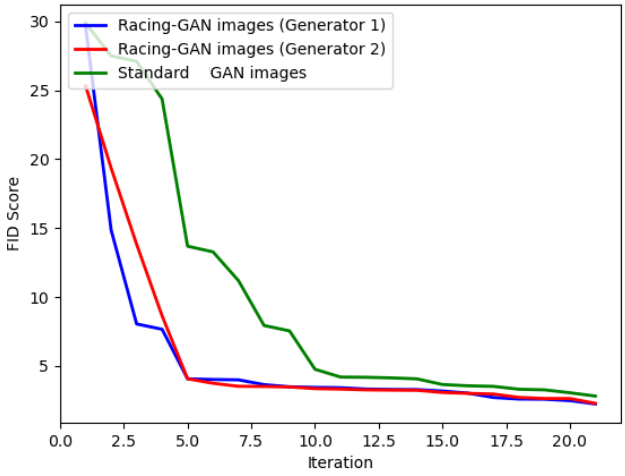
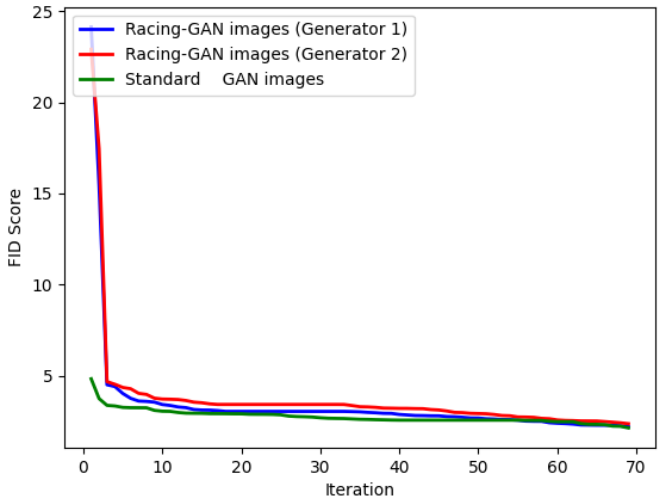


Figure showing the images produced by the Generator 1 of the DC Racing-GAN implementation over several iterations

20 iterations of DCGAN imaging



70 iterations of DCGAN imaging



Figures showing the FID scores of the images generated in the Racing-GAN and the standard models relative to the real images.

Loss Functions translated to python code and adapted to work in with the PyTorch machine learning library:

```
> prob1 = discriminator(gen_imgs1)
> prob2 = discriminator(gen_imgs2)
> g_loss1 = torch.mean(torch.log(1-prob1)) - torch.max(torch.zeros(1), torch.mean(prob1 - prob2))
> g_loss2 = torch.mean(torch.log(1-prob2)) - torch.max(torch.zeros(1), torch.mean(prob2 - prob1))
> d_loss = -torch.mean(torch.log(discriminator(real_imgs)) + (torch.log(1 - discriminator(gen_imgs1.detach()))+ torch.log(1 -
  discriminator(gen_imgs2.detach())))/2)
```

Convergence FID scores:

- WGAN – 2.3
- DCGAN
  - Trial 1 – No convergence in given sample period
  - Trial 2 (20 iterations) – 3.7
  - Trial 2 (70 iterations) – 2.1

6. Discussion

The Racing-GAN implementation of the WGAN evidently showed that it improves the speed at which convergence of counterfeit to the real images occurs. This is suggested through the FID scores which reached stability much earlier than the standard WGAN. While the standard WGAN already provided a solution to the modal collapse problem, this improvement in efficiency is significant and cause for further exploration. It should be also be noted however, that although the Racing-GAN took less iterations to converge, the time in which it took to complete an iteration was longer than that of the standard single generator framework, given the increased number of operations performed due to the inclusion of a second generator.

For trial 1, the plots did not converge suggesting that the generators were not learning from each other but rather learning independently. This can likely be accounted for in the loss function which was perhaps not an accurate representation of the Racing-GAN loss function in python. Trial 2 was conducted in order to attempt to test another loss function which would allow the generators to converge. While convergence did occur, the results produced were somewhat contradictory to the hypothesis. In calculating the FID scores after 20 iterations, the Racing-GAN performed as expected with the standard GAN converging later than the multigeneration framework. However, when the FID scores were calculated after 70 iterations the convergence was somewhat in favor of the standard WGAN model. Again, this can likely be accounted for in the loss function which perhaps was not an accurate representation of the Racing-Gan's.

7. Limitations

Due the large amount of time it takes to train the GAN models it was difficult to perform multiple tests of the reliability of the models - with adaptations to the loss function and other variables.

The code for the WGAN and DCGAN models was adapted from an online resource and there was some difficulty in translating the code and employing the racing GAN structure. Specifically, the loss function of the DCGAN was difficult to translate given it was intrinsically a Pytorch function call and could not be manipulated without removing the wrapper.

The tigergpu clusters which were being utilized to train the models were down periodically during the extent of this research due to cooling issues.

8. Conclusion and Future Work

The Racing-GAN loss function being an effective improvement in efficiency over the standard models appears to be inconclusive. Thus, the hypothesis that Racing-GAN is effective at reducing the convergence time of images generated to real sample can neither be rejected nor accepted.

Further exploration and trials should be done in order to determine how to translate the min-max loss function of the Racing-GAN model accurately in the DCGAN and WGAN approaches. Research should also be continued in order to determine the reliability of the findings - altering the real images sampled, the number of generators, the loss function, and other independent variables.

Moreover, research can be aimed at applying the Racing-GAN implementation to other types of GANs in an attempt to generalize the effectiveness of the adaptation.

5. References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio - Departement d'informatique et de recherche op ´ erationnelle ´ Universite de Montr ´ eal ´ Montreal, QC H3C 3J7, "Generative Adversarial Nets"

[2] Martin Arjovsky, Soumith Chintala, and L´eon Bottou , "Wasserstein GAN", 2017. [Online]. Available: https://arxiv.org/pdf/1701.07875.pdf

[3] Tanujay Saha, "Racing-GAN: A Competitive Multi-Generator Framework"

[4] Naoki Shibuya, "Understanding Generative Adversarial Networks". [Online].Available: https://medium.com/activating-robotic-minds/understanding-generative-adversarial-networks-4dafc963f2ef

[5] Samarth Sinha1, Han Zhang, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, And Augustus Odena, "Small-Gan: Speeding Up Gan Training Using Core-Sets". [Online].Available: https://arxiv.org/pdf/1910.13540.pdf

[6] Wei Fang, Feihong Zhang, Victor S. Sheng, and Yewen Ding, "A Method for Improving CNN-Based Image Recognition Using DCGAN". [Online].Available: https://pdfs.semanticscholar.org/5969/e1da59170a88f19d1135f315c90e56250079.pdf

[7] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li, "Mode Regularized Generative Adversarial Networks". [Online].Available: https://arxiv.org/pdf/1612.02136.pdf

[8] Alec Radford and Luke Metz, "Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks". [Online].Available: https://arxiv.org/pdf/1511.06434.pdf

[9] Jason Brownlee, "How to Implement the Frechet Inception Distance (FID) for Evaluating GANs". [Online].Available: https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/

[10] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari Inria, "How good is my GAN?" [Online].Available: http://openaccess.thecvf.com/content_ECCV_2018/papers/Konstantin_Shmelkov_How_good_is_ECCV_2018_paper.pdf

[11] Jonathon Hui, "GAN — How to measure GAN performance?". [Online].Available: https://medium.com/@jonathan_hui/gan-how-to-measure-gan-performance-64b988c47732

[12] Mario Lucic, Karol Kurach, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly, "Are GANs Created Equal? A Large-Scale Study". [Online].Available: https://arxiv.org/pdf/1711.10337.pdf

[13] Code Resources:

WGAN, Adapted from Martin Arjovsky, Soumith Chintala, Léon Bottou, [Online].Available: https://github.com/eriklindernoren/PyTorch-GAN

DCGAN, Adapted from Alec Radford, Luke Metz, Soumith Chintala, [Online].Available: https://github.com/eriklindernoren/PyTorch-GAN

FID Scores, [Online].Available: https://github.com/bioinf-jku/TTUR