Generating Deep Fake Images using GANs

Sahithi Kodali
kodali1@purdue.edu
Purdue University
West Lafayette, Indiana, U.S.A

Mayesha Monjur
monjur@purdue.edu
Purdue University
West Lafayette, Indiana, U.S.A

ABSTRACT

Deep fake images are synthetic images generated to mimic the looks of real people or objects images which can be used in entertainment, education, and other industries. Generative Adversarial Networks (GANs) have revolutionized the field of deep fake images as they are designed to learn and replicate the underlying distribution of the training data. GANs are composed of two neural networks that are simultaneously trained in a game-theoretic framework: a generator that generates synthetic images and a discriminator that distinguishes between the real and the synthetic images generated. The development of numerous GAN designs, each with specific strengths and drawbacks, has resulted from this adversarial training process. This paper analyses the performance of three prominent GAN architectures: Vanilla GAN, Deep Convolutional GAN (DCGAN), and Wasserstein GAN (WGAN) with weight clipping and gradient penalty. We evaluate the performance of these architectures by implementing them on CelebA dataset. Among them, WGAN with gradient penalty performed the best followed by DCGAN, while the Vanilla GAN performed the worst among all. Our findings highlight the differences and trade-offs among the three GAN models, offering valuable insights for researchers and practitioners in the field of image synthesis using GANs.

1 INTRODUCTION AND BACKGROUND

Generative Adversarial Networks (GANs) have been a potent tool for generating deep fake images during the past decade. Several GAN architectures have been developed, each seeking to get beyond the drawbacks of its forerunners and improve performance as a result of the field's ongoing improvements. Deep learning and computer vision researchers and practitioners now face the critical task of selecting the best GAN architecture for a given application or dataset.

The primary goal of this project is to implement and evaluate the three widely used GANs- Vanilla GAN, Deep Convolutional GAN (DCGAN), and Wasserstein GAN (WGAN) with weight clipping and gradient penalty to generate deep fake images. The goal of this study is to evaluate their strengths and weaknesses while taking into account variables including image quality, training stability, and computational effectiveness.

1.1 Importance and Motivation

GANs can have real-world applications in various fields. For instance, Isola et al. (2017) proposed the Pix2Pix model, an image-to-image translation method using conditional GANs [6]. Numerous applications, including style transfer, picture synthesis, and data augmentation, have extensively used this model. StyleGAN, introduced by Karras et al. (2019), has found potential applications in the field of entertainment and advertising [8].

Although deep fakes have many benefits, they can be used for malicious activities such as facial image manipulation, misinformation, identity theft, etc. Therefore, it is essential to understand the capabilities and limitations of GANs in order to come up with countermeasures against such malpractices [13].

The study of GANs can also provide valuable insights into their underlying mechanisms and inspire further research into the development of more advanced and efficient models. GANs have already had a considerable impact on the research community, stimulating new study areas and a surge in the number of papers published on the subject [2].

1.2 Anticipated project goals and evaluation outcomes

The anticipated goals of our project involve finding answers to the following questions.

- How to understand the process of generating fake images?
- Can we use this study to further analyze the difference between real and fake images, thus aiding malpractices?
- How do different models perform in generating undifferentiable fake images?

Our project goals are designed according to the outcomes we want to obtain. The success of these goals can be checked using the evaluation metrics and by reflecting on the following questions.

- Did specific GAN architecture performs better than the other architectures? If so, why?
- Can this research be incorporated in differentiating real from fake images?
- Were we able to answer the project questions with a certain degree of confidence?
- How can the outcomes be financially beneficial to the industry or make a value in society?

1.3 Literature Review

Generative Adversarial Networks (GANs) have garnered significant attention since their introduction by Goodfellow et al. (2014) due to their ability to generate high-quality samples from complex data distributions [3]. The early success of GANs in producing accurate samples from diverse data distributions, including text, photos, and audio, spurred a surge in interest among researchers.

In order to solve the drawbacks of the original GAN design, such as unstable training, and mode collapse, several developments have been introduced. The Deep Convolutional GAN (DCGAN), which Radford et al. (2015) proposed, included architectural changes to improve training stability and produce higher-quality images [12]. Many following GAN variations, including Conditional GANs (Mirza & Osindero, 2014), which produce samples conditioned on particular qualities or labels, were inspired by the DCGAN, which

has grown to be a popular choice for a variety of image synthesis applications [11]. Wasserstein GAN is another significant development introduced by Arjovsky et al. (2017) [1]. The Wasserstein distance, as opposed to the conventional adversarial loss, is used by WGAN as the optimization objective to handle unstable training and mode collapse. Training with WGANs is more reliable and less susceptible to the selection of hyperparameters. To further increase training stability, Gulrajani et al. (2017) presented the WGAN with gradient penalty (WGAN-GP), which imposes a Lipschitz restriction on the gradients of the discriminator [4]. Another significant development is the use of progressive GANs (Karras et al., 2017), which allow for the production of high-quality images by gradually raising the resolution while training. The quality of the generated images and the stability of the training process are both significantly improved by this method [7].

Due to the absence of a clear objective function, evaluating GANs continues to be difficult. Frechet Inception Distance (FID) and Inception Score (Heusel et al., 2016) are two often used measures to assess the quality and diversity of generated samples [5]. In a thorough analysis of GAN performance, Lucic et al. (2018) investigated how architectural decisions and training environments affected GAN performance [10].

1.4 How does a GAN work?

GAN includes a generator network that generates synthetic samples that are similar to the training data, and a discriminator network determines the authenticity of those samples. The generator aims to produce samples that are indistinguishable from real data, and the discriminator tries to differentiate between real and generated samples. It works based on the min-max objective where the generator tries to minimize the difference between the real and fake images while the discriminator tries to maximize the difference.

The general architecture of the model can be seen in Figure 1, where the generator tries to generate images from a noise vector. These images, along with the real images, are sent to the discriminator, which understands the difference between the two image distributions, and this feedback is sent to the generator allowing the learning of the generator.

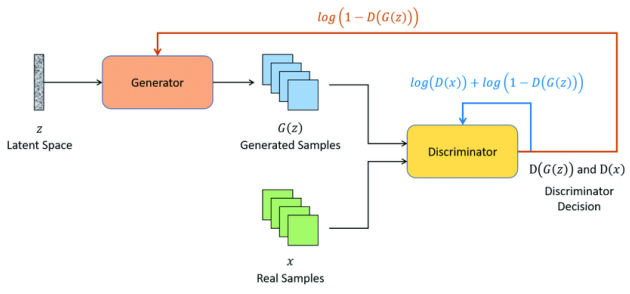

Figure 1: The structure of the basic GAN model [14]

The objective function used in this min-max optimization can be seen in Equation 1. In this equation 'x' represents samples from the real data distribution, while 'z' represents samples from the distribution of generated images by generator. While the discriminator tries to maximize the average of the log probability of real images and the log of the inverse probability for fake images, the generator tries to minimize the log of the inverse probability of fake images predicted by the discriminator as seen in Equation 1.

$$\min_G \max_D O(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))] \quad (1)$$

Using this core working of GAN, many GAN models with different metrics and optimization tricks have been developed. Three such GAN models are implemented to generate deep fake images in this paper and are discussed further.

2 DATASET AND PRE-PROCESSING

The dataset used for this project is the CelebFaces Attributes (CelebA) dataset released by the University of Hong Kong [9]. It is a large-scale face attribute dataset with more than 200k celebrity images with each having 40 attribute annotations.

For this project, 20k images are used for training the GANs, and 5k images are used for evaluating the trained Generator. These images are pre-processed to a shape suitable for implementing the three GAN models detailed further in this paper.

3 METHODOLOGY AND IMPLEMENTATION

This section discusses the three model architectures implemented to generate deep fake images. The three models explored are Vanilla GAN using only Linear layers, DCGAN which is based on convolutional layers, and WGAN based on Wasserstein distance - using weight clipping and gradient penalty. Further, these model architectures are described in detail, along with their implementation details.

3.1 Vanilla GAN

Vanilla GAN is the baseline model that incorporates only fully connected linear layers in its architecture. This architecture was first proposed by Ian Goodfellow (2014) and involves introducing generator and discriminator neural networks for generating images from a random noise that is similar to the training data provided [3].

In the Vanilla GAN implemented in this paper, the generator involves four linear layers with the first three followed by batch normalization and leakyReLU activation function, while the last layer is followed by a tanh activation to yield probabilities in the range [-1,1]. The discriminator also involves four linear layers with the first three layers followed by leakyRelu activation and dropout function, while the last layer is followed by the sigmoid function to yield probabilities in the range [0,1]. The generator and discriminator architectures can be seen in Figure 2, where the output channels at each layer are shown below the type of layer. The generator takes in a 64-dimension noise vector as input and increases it to the image size channels of 3X64X64, while the discriminator takes the 3X64X64 dimension input and produces a one-dimensional output in the form of probabilities.

For training the model, a learning rate of 0.0002, an Adam optimizer, and a Binary Cross-entropy loss function are used. The model is trained for 50 epochs, and the fake images generated after

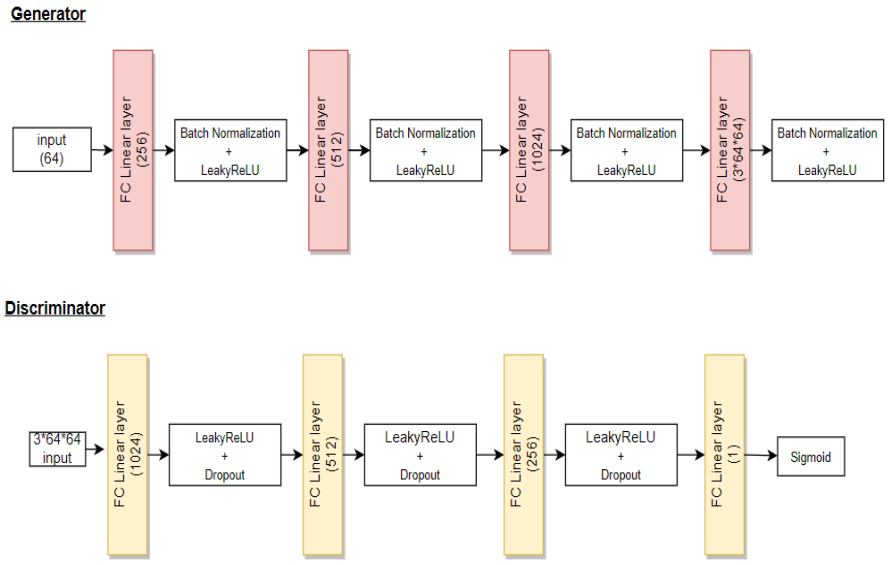

Figure 2: Generator and Discriminator architecture of the Vanilla GAN model

each epoch are stored to see the improvement and check for mode collapse. Mode collapse occurs when the generator is not optimized for each training iteration of the discriminator, and hence the learning is stagnated leading to no convergence. As discussed in 1.4, the discriminator is trained on real images and the fake images generated by the generator, to calculate the discriminator loss and optimize it. The generator is trained to produce the fake images by learning from the feedback received through the discriminator until it produces outputs similar to the real images, all of which is achieved by the min-max objective function discussed previously.

The plot of loss to iterations in training the model can be seen in Figure 3, where we can observe the generator loss to decrease and discriminator loss to increase with both reaching an equilibrium point gradually known as Nash equilibrium. However, the loss is still high and can be significantly decreased through some modifications in the architecture as discussed further.

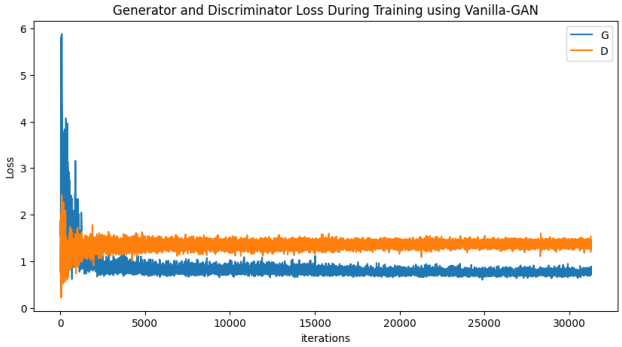

Figure 3: Plot of loss to iterations in training Vanilla GAN

3.2 Deep Convolutional GAN (DCGAN)

DCGAN is another variant of GANs that uses convolutional layers without max-pooling or fully connected layers. It was proposed by Radford et al. (2015) to address the limitations of Vanilla GAN by incorporating convolutional layers both in the generator and discriminator [12].

In the architecture implemented for this project, the discriminator comprises convolution layers with strides followed by batch normalization and leakyReLU activation functions, which are followed by a flatten, dropout layer, and a sigmoid function in the end. The generator is composed of several transpose-convolutional layers, batch normalization, and leakyReLU followed by a tanh function in the final layer to produce output in the range of [-1,1]. The architecture of the generator and discriminator can be seen in Figure 4.

The generator's task is to produce a realistic image, which it performs by starting with the dense layer that reshapes the 100-dimension input noise into a low-resolution image with 128 channels. The low-resolution image is then upsampled and refined by the transpose-convolution layers into a higher-resolution image of shape 3X80X80, which resembles the target image size. The discriminator, on the other hand, tries to classify whether the image is real or fake. This is achieved by reducing the spatial dimensions of the input image of shape 3X80X80, by employing a series of convolution layers and then feeding the flattened output feature maps into a dense layer consisting of a sigmoid function, which produces a single value in the range of [0,1], representing the probability of the input image being real. In the discriminator, batch normalization has been used after every Conv2D layer except the first one. The channels for the images have been kept the same throughout the layers, fine-tuned for this specific dataset in order to produce the best results possible.

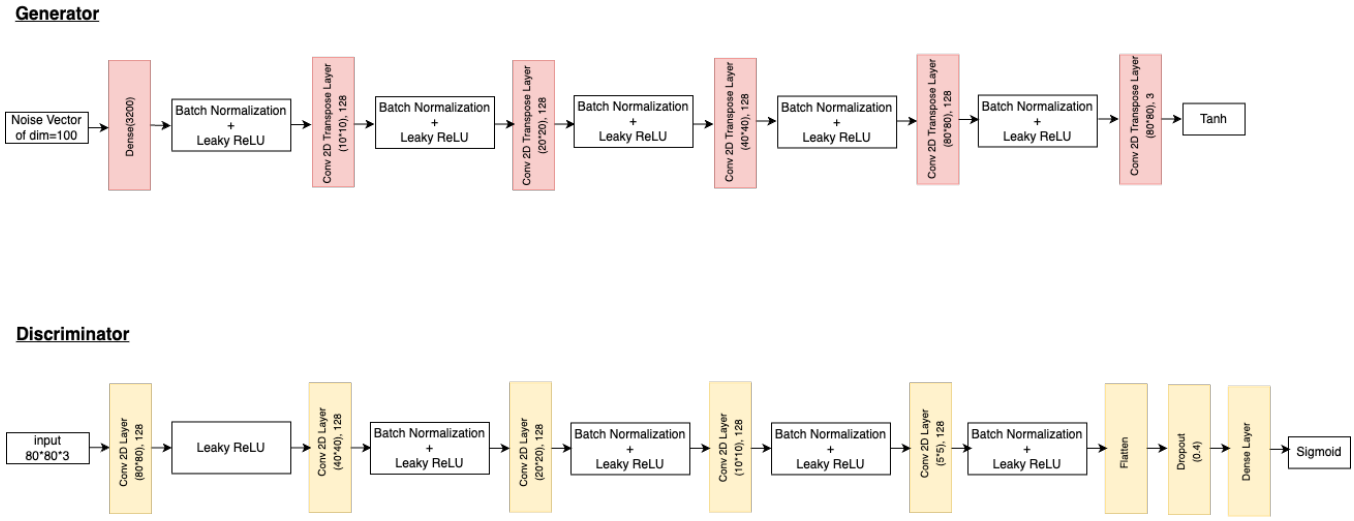

Figure 4: Generator and Discriminator architecture of the Deep Convolutional GAN

In order to train the model, a learning rate of 0.0002 and an Adam optimizer with a Binary Cross-entropy as a loss function have been used. The model has been trained for 100 epochs, with images being generated every 10 epochs in order to examine the progress. It has been observed that the quality of the images kept increasing with the epochs.

The plot of loss to epochs can be seen in Figure 5. The plot shows that the generator loss is decreasing over time which indicated that the produced images are getting more realistic while the discriminator is slightly increasing. The sudden peaks in the discriminator loss occurred when the discriminator was having a hard time distinguishing images generated by the generator. It can be observed that convergence was not the best and the model can still be improved.

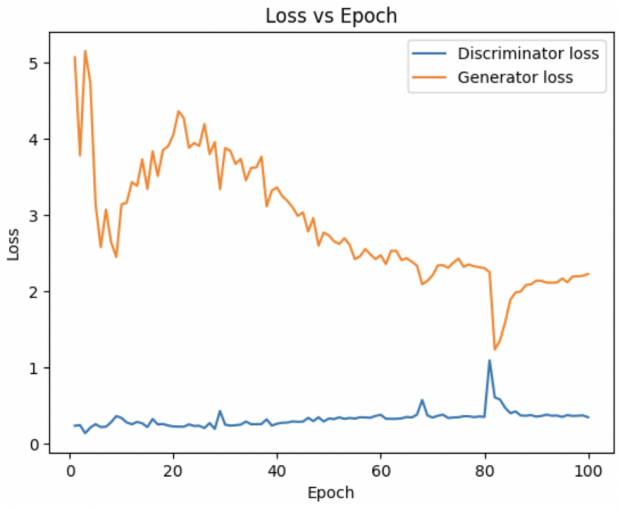

Figure 5: Plot of loss to epochs in training Deep Convolutional GAN

3.3 Wasserstein GAN (WGAN)

WGAN works based on computing the Wasserstein distance between the distributions of training images and the fake images generated. This was first proposed in the paper by Arjovsky et al. (2017) [1]. The Wasserstein distance between the distribution 'P' and 'Q' is computed as shown in Equation 2 where $\Gamma(P,Q)$ represents the set of all possible joint distributions $\gamma(X,Y)$, and 'P' is the marginal distribution with respect to 'x' while 'Q' is the marginal distribution with respect to 'y'. The equation indicates that the joint distributions are such that the mean of the norm difference between 'x' and 'y' is minimized for every sample (x,y) drawn from the joint distribution.

$$d_W(P,Q) = \inf_{\gamma(X,Y)\in\Gamma(P,Q)} E_{(X,Y)\sim\gamma}\left[||x-y||\right] \tag{2}$$

However, since this not computationally feasible to solve, the equation is rewritten in terms of supremum a shown in Equation 3 to make it tractable. In this equation, the function 'f' indicates the 1-Lipschitz function, which is the core factor to be satisfied in implementing a discriminator (known as a critic) in WGANs.

$$d_W(P,Q) = \sup_{||f||_L\leq} \left[E_{x\sim P}\{f(x)\} - E_{y\sim Q}\{f(y)\}\right] \tag{3}$$

The critic role is different from the discriminator in the GANs implemented previously in this paper. A critic instead of just predicting the real/fake images is now trained to estimate the Wasserstein distance between the real and the generated fake images. This distance is backpropagated to update the weight of the networks. The 'y' in Equation 3 are the images generated by the generator. To backpropagate the distance, it is to be ensured that the function 'f' follows the 1-Lipschitz constraint and this can be achieved by the two methods discussed further in this section - using weight clipping and gradient penalty.

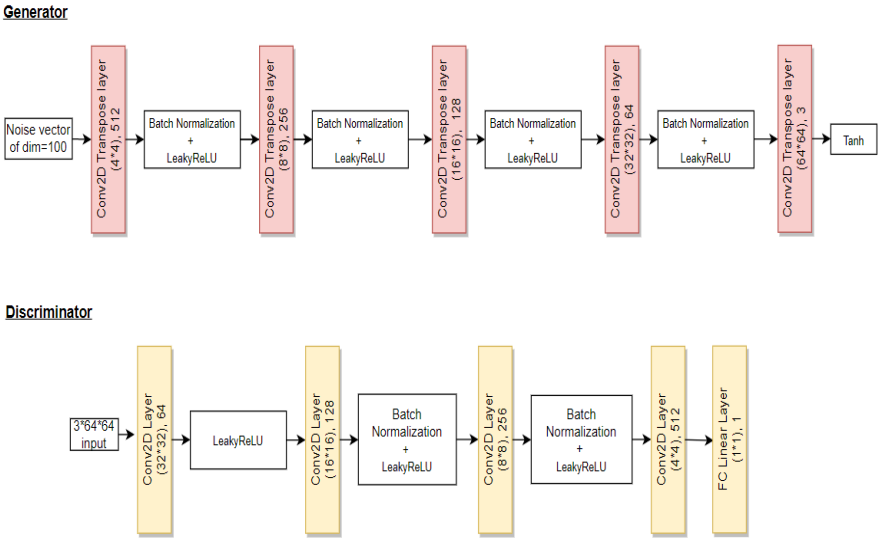

Figure 6: Generator and Discriminator architecture of the WGAN model using weights clipping

3.3.1 *Using weight clipping.*

To satisfy the Lipschitz constraint, one method used is to clip weights to a given threshold. This, in turn, is a min-max optimization since the generator tries to minimize the Wasserstein distance while the discriminator tries to maximize the Wasserstein distance. The gradient targets used to train the Critic are +1 and -1 so that the critic has a unit gradient norm in its optimized condition for any 'P' and 'Q' distributions.

The generator architecture used involves five transpose convolutional layers where the first four layers are followed by batch normalization and ReLU activation function, and the last convolutional layer is followed by a tanh activation function. The first layer uses a kernel size of 4x4, with a stride of 1 and padding of 0, while the remaining four layers use a stride of 2 and padding of 1. The discriminator architecture used involves four convolutional layers with a kernel size of 4x4, stride of 2, and padding of 1, along with a final linear layer. The first three convolutional layers are followed by leakyRelu activation and batch normalization while the last convolutional layer output is sent through the linear layer to attain a scalar output required for this model. The generator and discriminator architectures used in this model can be seen in Figure 6, where the increasing/decreasing image size and output channels at each layer are shown below the type of layer, respectively.

To train this model, a learning rate of 0.0002, an Adam optimizer, and a binary cross-entropy loss function are used. The model is trained for 50 epochs, and the fake images generated after each epoch are stored similarly to the Vanilla GAN training. To avoid mode collapse for every update of generator parameters, the critic parameters are updated five times to achieve generalization on the images and to allow the generator to learn better to generate images. The training plot of loss to iterations can be seen in Figure 7.

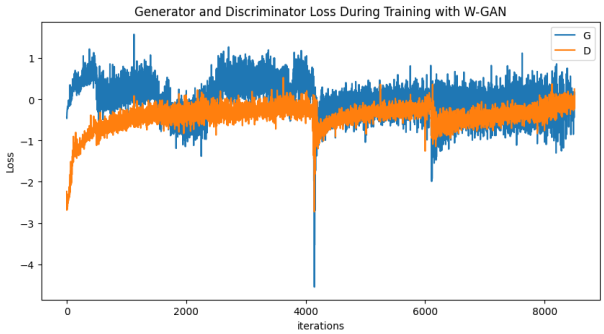

Figure 7: Plot of loss to iterations in training WGAN model using weights clipping

3.3.2 *Using gradient penalty.*

The other approach to achieve learning the Lipschitz function is using the gradient penalty. It was first introduced in the paper by Gulrajani et al. (2017) [4], which states that the training can be improved by incorporating the property that the optimal critic has a unit gradient norm at every point under the 'P' and 'Q' distributions. The addition of gradient penalty to the loss function of the critic can be seen in Equation 4, and the details of this equation can be further read in the paper by Gulrajani et al.

$$O(C,G) = E_{z\sim p_z(z)}[C(G(z))] - E_{x\sim p_{data}(x)}[C(x)] + \lambda[||\nabla_{x'}C(x')||^2 - 1]^2 \quad (4)$$

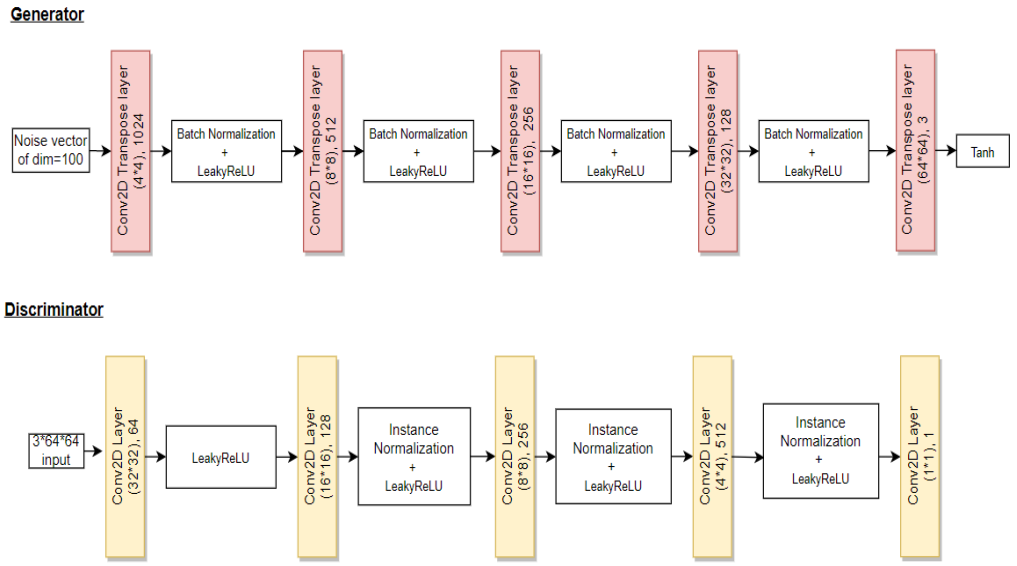

Figure 8: Generator and Discriminator architecture of the WGAN model using gradient penalty

This model training also incorporates training the critic multiple times for every update of generator parameters with the target gradient for the backpropagation of the critic as -1.

The generator architecture used involves five transpose convolutional layers where the first four layers are followed by batch normalization and ReLU activation, while the last convolutional layer is followed by a tanh activation function. The first layer uses a kernel size of 4x4, with a stride of 1 and padding of 0, while the remaining four layers use a stride of 2 and padding of 1, similar to the WGAN model using weight clipping. The discriminator architecture used involves five convolutional layers, with the first four convolutional layers using a kernel size of 4x4, stride of 2, and padding of 1, while the last layer uses a kernel of 4x4, stride of 2, and a padding of 0. The first convolutional layer is followed by the leakyRelu activation function, while the next three are followed by the instance normalization and leakyRelu activation function. The instance normalization performed here is due to the fact that the gradients of the outputs are computed with respect to each input, and hence batch normalization can not be applied. The generator and discriminator architectures can be seen in Figure 8, where the increasing/decreasing image size and output channels at each layer are shown below each layer, respectively.

To train this model, a learning rate of 0.0002, an Adam optimizer, and a binary cross-entropy loss function are used. The model is trained for 50 epochs, and the fake images generated after each epoch are stored. Again for every update of generator parameters, the critic parameters are updated five times to achieve generalization on the images and to allow the generator to learn better to generate images. The training plot of loss to iterations can be seen in Figure 9.

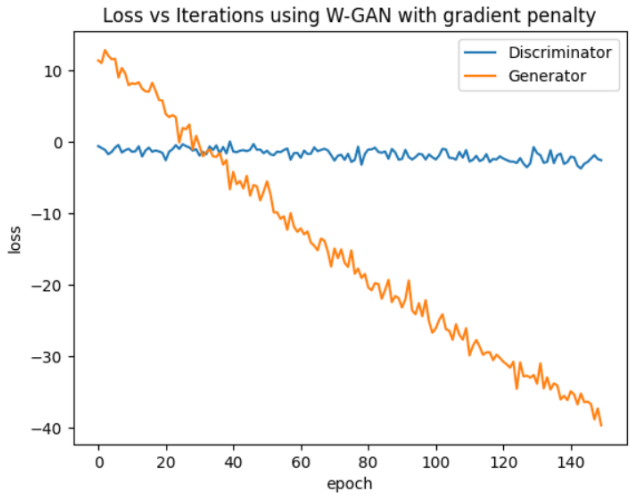

Figure 9: Plot of loss to iterations in training WGAN model using gradient penalty

4 RESULTS AND ANALYSIS

To analyze the performance of the three models implemented, 5000 fake images are generated using the trained GAN models for qualitative evaluation. These 5000 fake images are compared with 5000 unseen celebrity images to compute the Frechet Inception Distance (FID) score for quantitative evaluation. The lower the score, the higher the similarity between the real and fake images.

4.1 Vanilla GAN

For evaluating the trained Vanilla GAN model, a random noise of 64 dimension is given as input to generate 5000 fake images. These images are compared to real images using the FID metric to compare the closeness of images, and the FID score of 347.72 attained for this model can be seen in Figure 10 and a sample of fake images produced can be seen in Figure 11.

```
Downloading: "https://github.com/mseitzer/pytorch-fid/releases/download/fid_weights
100%|██████████| 91.2M/91.2M [00:00<00:00, 206MB/s]
100%|██████████| 101/101 [01:29<00:00,  1.13it/s]
Warning: batch size is bigger than the data size. Setting batch size to data size
100%|██████████| 1/1 [00:00<00:00,  3.22it/s]
FID using Vanilla-GAN:  347.72
```

Figure 10: FID score attained by Vanilla GAN

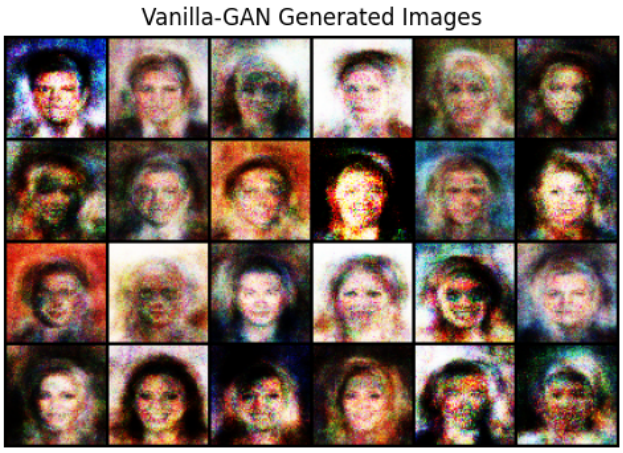

Figure 11: Sample of fake images generated by Vanilla GAN

4.2 DCGAN

In order to evaluate the trained DCGAN, 5000 images were generate by inputing a 100 dimensional random noise. Comparing the fake images to the real ones, an FID of 123.494 was achieved for this model which is significantly better than the FID score of Vanilla GAN. The FID for this model can be seen in Figure 12 and a sample of fake images generated can be seen in Figure 13. It can be observed that the images generated by DCGAN qualitatively surpasses those generated by Vanilla GAN, as they are more clear and less pixelated.

```
100%|██████████| 63/63 [06:43<00:00,  6.40s/it]
100%|██████████| 63/63 [06:25<00:00,  6.13s/it]
FID score: 123.49465351432326
```

Figure 12: FID score attained by DCGAN

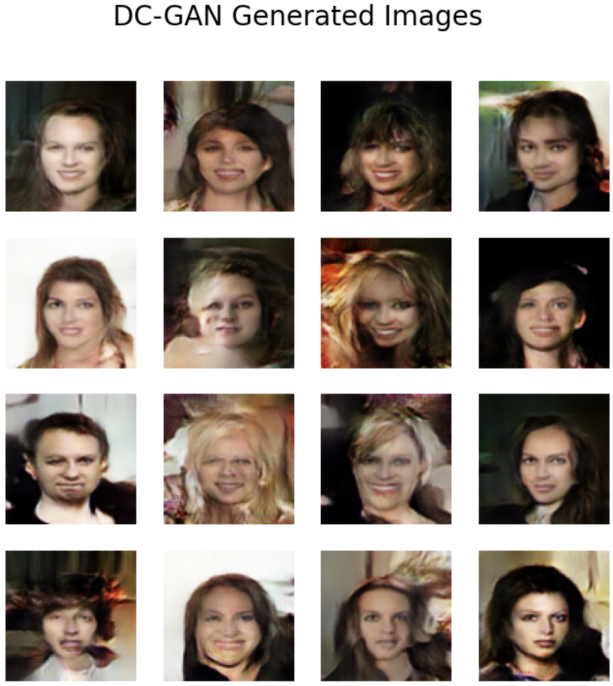

Figure 13: Sample of fake images generated by DCGAN

4.3 WGAN

The train WGAN model is evaluated by giving an input of random noise of 100 dimension vector input to generate fake images, and these images are compared to real images using FID metric to compare the closeness of images.

4.3.1 Using weights clipping.

For the method using weight clipping, the trained WGAN model attained a FID score of 218.39 as seen in Figure 14, and a sample of fake images produced can be seen in Figure 15.

```
Downloading: "https://github.com/mseitzer/pytorch-fid/releases
100%|██████████| 91.2M/91.2M [00:00<00:00, 181MB/s]
100%|██████████| 101/101 [02:23<00:00,  1.42s/it]
100%|██████████| 101/101 [00:20<00:00,  4.89it/s]
FID using W-GAN:  218.39
```

Figure 14: FID score attained by WGAN using weights clipping

4.3.2 Using gradient penalty.

For the method using gradient penalty, the trained WGAN model attained the highest FID score of 113.28, as seen in Figure 16, and a sample of fake images produced can be seen in Figure 17.

Analysis: By observing the fake images generated by the three models, we can observe that the Vanilla GAN produced very noisy images in spite of capturing the structure of the faces. This is due

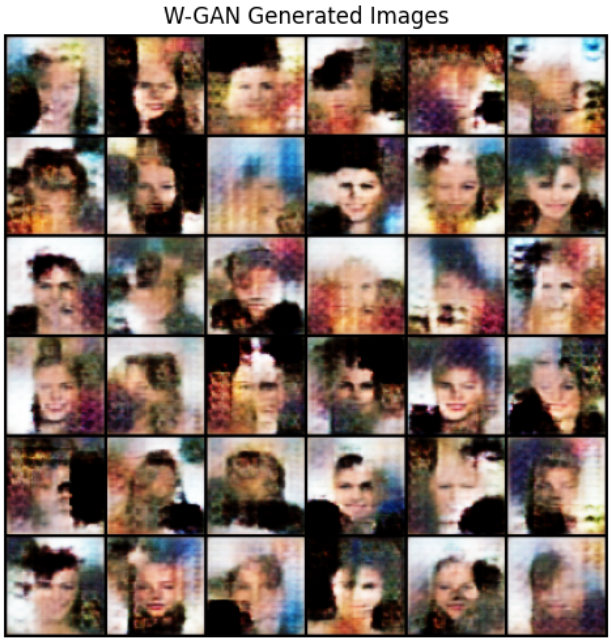

Figure 15: Sample of images generated by WGAN using weights clipping

```
100%|██████████| 101/101 [00:22<00:00,  4.43it/s]
100%|██████████| 101/101 [00:20<00:00,  4.92it/s]
FID using W-GAN with gradient penalty:  113.28
```

Figure 16: FID score attained by WGAN using gradient penalty

to the fact that Vanilla CAN uses only Linear layers which cannot capture the details of images in the best possible way. Though WGAN with weights clipping produced some visible images, they are not complete and still have lots of noise, this is due to the fact that clipping weights can sometimes lead to no convergence due to gradients being thresholded in different gradient directions. Finally, WGAN with gradient penalty produced the best images and can still be improved by increasing the data and training time. It is to be observed that DCGAN produced almost close fake images to WGAN with a gradient penalty but took 100 epochs, while WGAN took 50 epochs to achieve the same or better results.

This qualitative analysis can also be observed to be reflected in the quantitative evaluation using the FID scores attained. The WGAN with gradient penalty achieved the best score of 113.28 and the second-best DCGAN achieved a score of 123.49, while the worst score of 347.72 was attained by Vanilla GAN, supporting our qualitative analysis.

5 CONTRIBUTION OF TEAM MEMBERS

- Sahithi - Study on the working of GANS and their background, Developing and implementing the Vanilla GAN and WGANs.

Figure 17: Sample of fake images generated by WGAN using gradient penalty

- Mayesha - Developing and Implementing DCGAN , Screening through past research for literature review, Background study on GANs.

6 CONCLUSION

In this paper, we have explored different types of GANs and compared their performance on the CelebA dataset. We have seen that the Vanilla GAN performs the worst. DCGAN with the same number of channels throughout all the layers performed almost as well as WGAN using gradient penalty, but took more training time as compared to WGANs.

The results achieved from these models can have several implications. The comparisons provide insights into the performance of the GAN models, which can help researchers and practitioners make informed decisions while selecting a model for deepfake image generation. It can also inspire new techniques and modifications to improve the performance of GANs. It can also shed light on the reasons behind the performance differences among the GAN models, potentially leading to better model interpretability and explainability.

For our future studies, we intend to increase the training dataset size and train the models for more epochs in order to achieve better results. We also plan to incorporate deep fake detection techniques along with generation techniques. Finally, we want to explore recent and more advanced architectures such as StyleGAN, BigGAN, and Progressive GAN and compare their performances.

REFERENCES

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[2] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, jan 2018.

[3] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2017.

[4] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[5] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[6] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[7] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.

[8] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[9] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[10] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study, 2018.

[11] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[12] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[13] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images, 2019.

[14] David Vint, Matthew Anderson, Yuhao Yang, Christos Ilioudis, Gaetano Di Caterina, and Carmine Clemente. Automatic target recognition for low resolution foliage penetrating sar images using cnns and gans. *Remote Sensing*, 13:596, 02 2021.