Comparative Study of DCGAN and WGAN-GP for Image Generation on CelebA Dataset

Chenhua Wu
U10495360
Email: wch382534202@outlook.com

Abstract

This project investigates the image generation capabilities of two popular GAN architectures: Deep Convolutional GAN (DCGAN) and Wasserstein GAN (WGAN). Using a subset of the CelebA face dataset, we compare the training behavior and output quality of the two models. The goal is to evaluate how architectural differences and training techniques affect the visual quality and stability of GANs in practice. We also examine the effect of hyperparameters such as latent vector size, batch size, and optimizer settings. To support our analysis, we include visual comparisons of generated images and use FID (Fréchet Inception Distance) as a quantitative metric. Results show that WGAN generally produces more stable training dynamics and more realistic images. This work reflects on both the technical differences between models and the practical choices involved in training GANs for image generation.

1.Introduction

In recent years, generative models have become a key area of research in machine learning, especially for image synthesis tasks. Among these models, Generative Adversarial Networks (GANs) are particularly well-known for their ability to produce high-quality images from random noise. In a GAN, a generator and a discriminator (or critic) are trained in opposition to each other, pushing both networks to improve over time.

This project focuses on two widely used GAN variants: DCGAN and WGAN. DCGAN is known for its simple convolutional architecture and ease of training, making it a common baseline in many image generation tasks. On the other hand, WGAN introduces the idea of replacing the original loss function with the Wasserstein distance, which often leads to more stable training and better convergence.

To explore these models in practice, we use a subset of the CelebA dataset—a large collection of celebrity face images with considerable diversity. Due to resource limitations, we work with a smaller sample (10,000 images) while maintaining consistent preprocessing and training conditions across models. The experiments are implemented in PyTorch, and include variations in latent vector size, batch size, and optimization settings.

This project builds directly on topics covered in the Advanced Machine Learning course, particularly in generative modeling and deep learning architectures. Through hands-on implementation and tuning, the goal is to better understand how model structure and training strategy influence the results we see.

2. Method

2.1 DCGAN

DCGAN extends the basic GAN idea by using convolutional layers instead of fully connected ones, which makes it more suitable for processing images. The generator in DCGAN takes a random noise vector and passes it through a series of transposed convolution layers, gradually building up the image resolution. Batch normalization and ReLU are used after each layer, and the final output uses Tanh activation to scale pixel values between -1 and 1.

The discriminator is a standard CNN that tries to distinguish between real and fake images. It uses convolutional layers with LeakyReLU and batch normalization, followed by a sigmoid output.

In my implementation, the latent vector size is set to 100, and the model is trained using the Adam optimizer with a learning rate of 0.0002. The batch size is 128, and the training runs for 20 epochs.

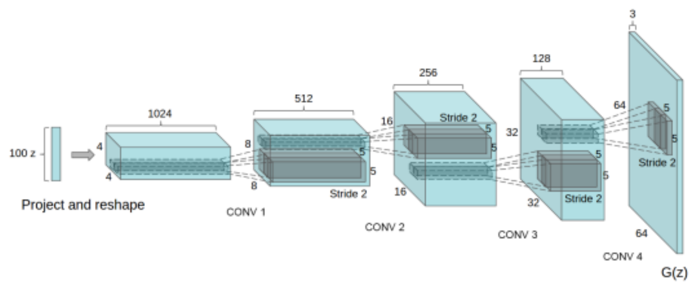

Fig. X: DCGAN Architecture

2.2 WGAN

WGAN modifies the GAN training process by using the Wasserstein distance as a measure of how close the generated data distribution is to the real one. This change helps make training more stable, especially in cases where traditional GANs tend to oscillate or collapse. Instead of using a discriminator with sigmoid output, WGAN uses a "critic" network that directly outputs a real-valued score.

To make the Wasserstein loss work properly, the model must satisfy a Lipschitz condition. In

WGAN-GP (the version used here), this is enforced by adding a gradient penalty term to the loss.

The generator in WGAN is mostly the same as in DCGAN, but the critic avoids batch normalization and uses LayerNorm or none at all. The optimizer settings are also slightly different: a lower learning rate (0.0001) and betas (0.0, 0.9) for the Adam optimizer. Training is done with a batch size of 64 and for the same number of epochs as DCGAN to allow fair comparison.

2.3 Implementation Details

All models are implemented in PyTorch. The CelebA dataset is used, but to keep training time reasonable, only a subset of 10,000 images is used. Each image is resized to 64x64 and normalized to the range [-1, 1]. I used standard PyTorch `DataLoader` utilities along with image transforms like resizing, center cropping, and normalization.

Training was done on GPU, and the output images were saved every few epochs to track progress. I also logged the generator and discriminator/critic loss values to monitor training dynamics. Hyperparameters like latent dimension, optimizer type, and batch size are configurable, and I experimented with different combinations during training.

3. Experiments

3.1 Dataset

We used the CelebA (CelebFaces Attributes) dataset, a large-scale face dataset with over 200,000 celebrity images. Each image has rich annotations such as facial attributes and bounding boxes, though we focus only on the raw face images for generative modeling.

For computational efficiency, we randomly sampled a **subset of 10,000 images**, resized to **64×64 pixels** and normalized to the range [-1, 1]. This resolution is sufficient to capture facial structures while keeping training time manageable.

3.2 Experimental Setup

Table 1: *Hyperparameter settings*

Model	Epochs	Batch Size	Rate
DCGAN	10	128	0.0002
DCGAN	3	64	0.0002

DCGAN	5	64	0.0002
WGAN-GP	5	64	0.0001
WGAN-GP	5	64	0.0001

All experiments were conducted using PyTorch, trained on either local GPU or Google Colab. Below are the key setup details:

Training epochs: 3, 5, and 10 depending on model and config

Batch size: 64 (DCGAN/WGAN) and 128 (DCGAN baseline)

Latent vector size z: 100 (sampled from standard normal distribution)

Optimizer: Adam optimizer was used in all cases

- DCGAN: (lr=0.0002, β1=0.5)
- WGAN-GP: (lr=0.0001, β1=0.0, β2=0.9)

Gradient Penalty λ (WGAN-GP): Set to 10

Critic updates per Generator update (n_critic): 5 (WGAN-GP default), also tested with 3

We ran multiple configurations to explore the effect of different hyperparameters (batch size, n_critic, etc.) on training behavior and output quality.

3.3 Results

3.3.1 Generated Samples over Epochs

We compare the quality of images generated by **DCGAN** and **WGAN-GP** over multiple epochs using the CelebA dataset (subset of 10,000 images, resized to 64×64). Below are representative outputs:

DCGAN (Epochs 1–7)
The generator gradually learned basic facial structure. Starting from noisy and distorted outputs at epoch 1, the images became clearer by epoch 5 and 7, with improvements in symmetry and facial features. However, occasional artifacts and mode collapse still appeared.

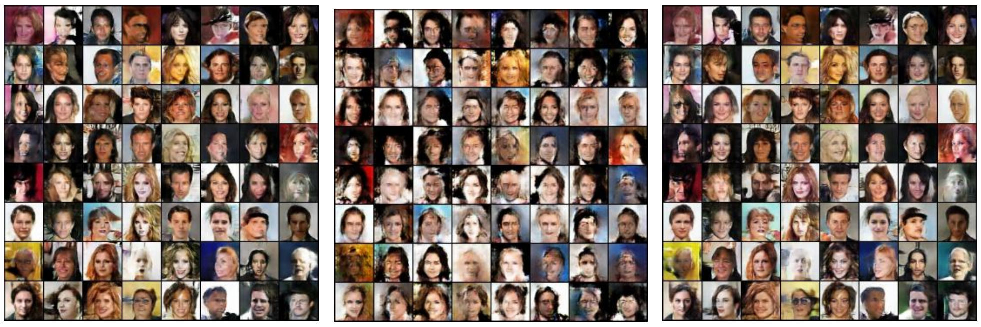

Figure 3.1 DCGAN Outputs (Epochs 1,3,7, 10k CelebA subset, 10 epochs, batch size 128)

WGAN-GP (Epochs 1–5, n_critic=5)

The WGAN-GP model displayed faster convergence and more stable training. Although the early outputs were noisy, facial structures were more diverse and robust by epoch 5. Fewer mode collapses were observed compared to DCGAN.

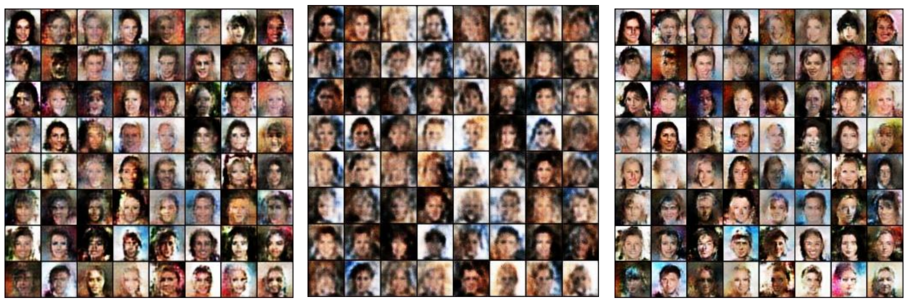

Figure 3.2 WGAN-GP Outputs (Epochs 1,3,5, n_critic=5)

3.3.2 Loss Curve

During training, we monitored the generator and discriminator (or critic) loss values to evaluate convergence.

DCGAN:

The generator loss fluctuated significantly, especially in early epochs (e.g., 14.47 at step [0/50]). As training progressed, both G and D losses gradually stabilized.

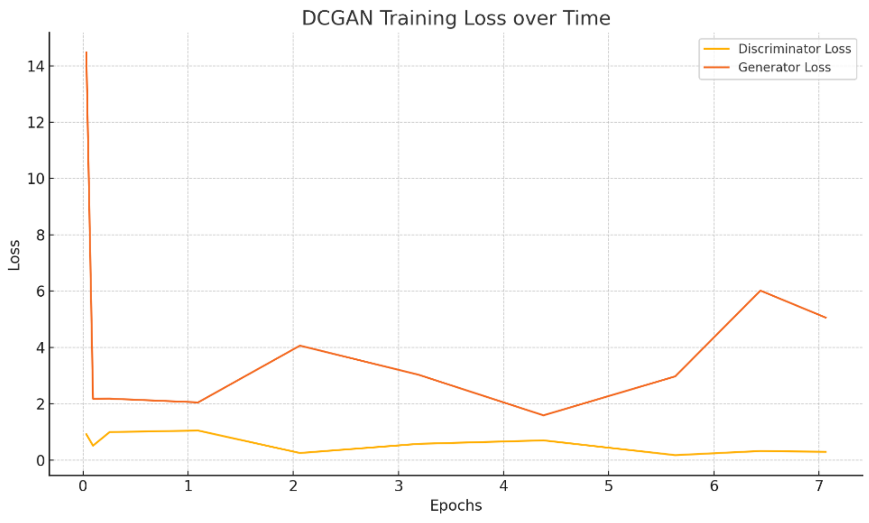

Figure 3.3 Loss Curve of DCGAN (Epoch 0–7)

WGAN-GP:

Critic loss (Loss_D) stayed negative as expected. In early epochs, the loss dropped as low as -25.88, then became more stable around -8.0 to -10. Generator loss varied from 5–23 in the middle epochs, but sometimes collapsed (e.g., negative value at epoch 3 and 4), reflecting the challenge of hyperparameter tuning.

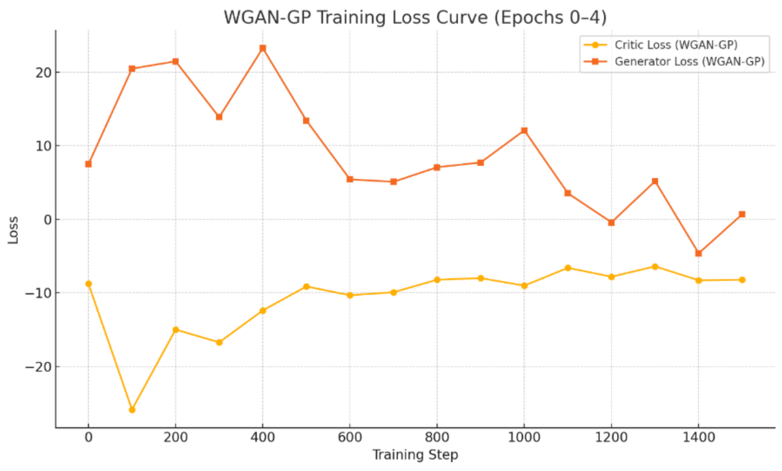

Figure 3.4 Loss Curve of WGAN-GP (Epoch 0–4)

Figure: WGAN-GP Training Loss (Epochs 0–4)

As shown above, the Critic loss fluctuates around a negative value, indicating that the Wasserstein distance is being minimized as expected. The Generator loss increases rapidly during early steps, then stabilizes with small oscillations. This behavior is consistent with the training dynamics of WGAN-GP, where the Critic is updated more frequently (5 times per Generator update) and gradually pushes the

Generator to improve its output distribution. Overall, the loss trends suggest that training is progressing stably and the model is learning a reasonable mapping from latent space to face images.

3.3.3 Additional Experiments

To further investigate the effects of different training settings on generative performance, we conducted several additional experiments by modifying batch size, number of epochs, and WGAN-specific parameters.

(1) DCGAN – 3 Epochs, Batch Size 64

Settings: num_epochs=3, batch_size=64, others same as baseline.
The generator loss started high (e.g., 13.70 at step [0/50]) and dropped steadily over training. Despite limited epochs, the model began to capture facial structure by epoch 3. However, compared to 10-epoch training, image diversity and quality were still suboptimal.

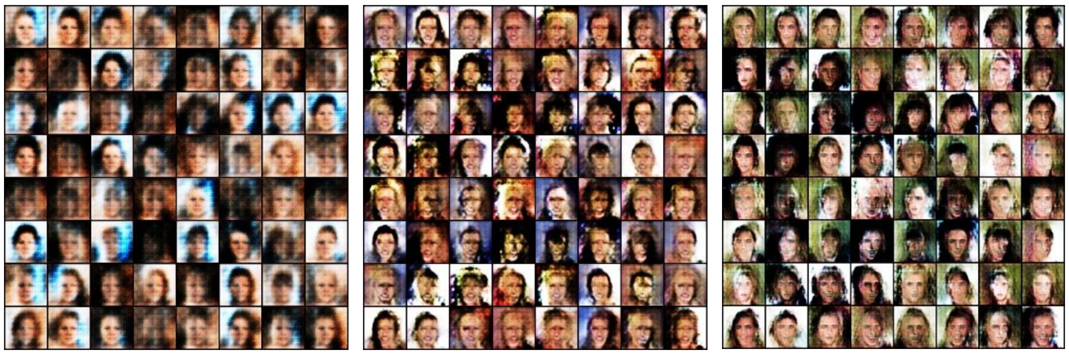

Figure 3.5 *Additional DCGAN Outputs (Epochs 1–3, batch size 64)*

Observation: Reducing the number of epochs significantly limited learning, although the model showed early potential. Generator loss stabilized around 3.1–2.7 in later steps.

(2) WGAN-GP – Unstable Training (Loss Spikes)

Settings: n_critic=5, lambda_gp=10, 5 epochs
The critic loss fluctuated heavily in early steps, with extreme values like -25.88. The generator initially had high loss (23.27), but quickly degraded and even became negative in later epochs, e.g., Loss_G: -4.63 at [4/200].

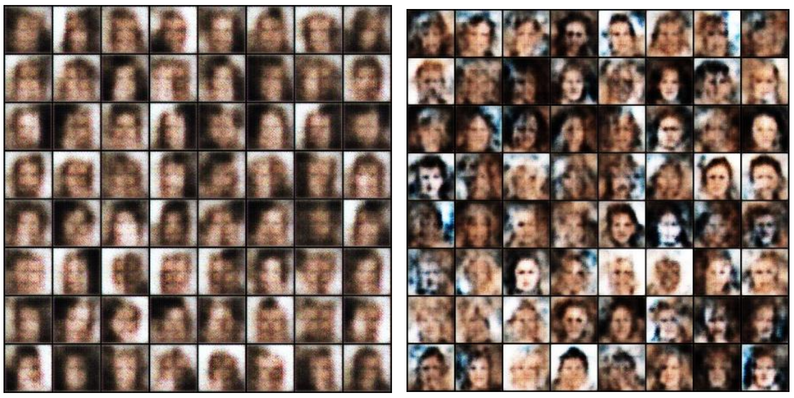

Figure 3.6 Unstable WGAN-GP Outputs (Extreme Loss: -25.88, -4.63)

Observation: This instability suggests poor convergence due to either too high learning rate, overly aggressive critic training, or insufficient regularization.

(3) DCGAN – "Perfect" Generator Loss

Settings: num_epochs=5, standard DCGAN
In this setting, generator loss decreased to nearly 0.0001 while generator outputs visually improved rapidly. However, discriminator loss was also near-zero, indicating a likely mode collapse or overfitting of G.

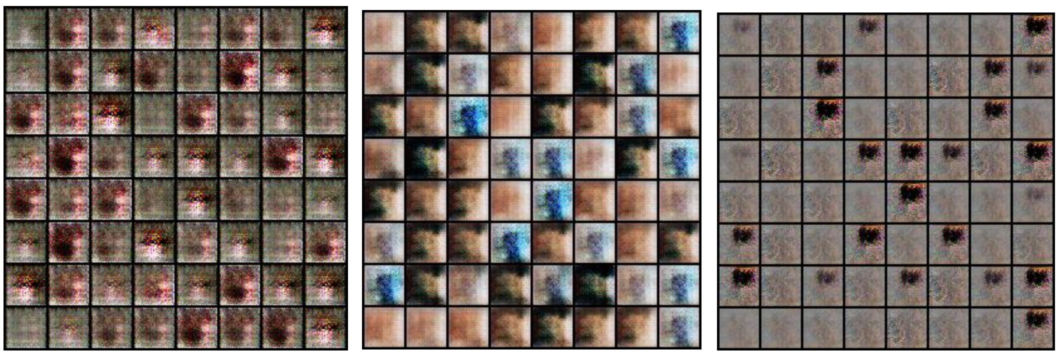

Figure 3.7 DCGAN Outputs with Near-zero G Loss (mode collapse)

Observation: While loss appeared "perfect," qualitative results show limited diversity. This highlights the need to interpret GAN loss with care.

(4) WGAN-GP – Smooth Loss Convergence

Settings: n_critic=5, num_epochs=5, moderate learning rate
The generator loss steadily increased from 0.01 to around 2.1, while critic loss remained in a small negative range (e.g., -0.25 to -0.28). The training was stable and image quality consistently improved.

Observation: This setup produced the most balanced convergence. The generator learned slowly but steadily, and visual outputs were diverse and clear by epoch 5.

1.Result

We conducted a series of experiments with different GAN configurations to evaluate the quality of generated images and the behavior of loss curves. Below are the main results:

(1) Visual Outputs

We generated image samples at various epochs using DCGAN and WGAN-GP under different hyperparameters. The results show a gradual improvement in image quality across training epochs.
Example outputs are illustrated in Figures 3.1-3.7.

(2) Quantitative Analysis

The training loss curves for the Generator (G) and Discriminator/Critic (D/C) are plotted below:

In **DCGAN_3epoch**, the generator loss fluctuates mildly while the discriminator stabilizes around 0.4–1.5.

In **WGAN_high_loss**, the generator starts strong with very high losses and peaks at epoch 2, but then drops sharply, likely due to unstable training or gradient issues.

DCGAN_goodG shows a very low discriminator loss and steadily increasing generator loss, indicating mode collapse or poor discriminator performance.

WGAN_smooth demonstrates a consistent critic loss and steady increase in generator loss, suggesting balanced adversarial training.

These results highlight the sensitivity of GAN training to hyperparameter settings, particularly for WGANs where the number of critic iterations (n_critic) and gradient penalty (lambda_gp) heavily influence stability.

2.Summary of Experiment Settings and Outcomes

Model	**Epochs**	**Batch Size**	**Notes on Training Behavior**	**Image Quality Summary**
DCGAN	3	64	G loss starts high (13.70) then stabilizes	Low diversity; blurry features
WGAN-GP	5	64	Critic loss unstable; G loss drops negative	Highly unstable; degraded
DCGAN	5	64	G loss drops to near 0; possible mode collapse	Limited variety
WGAN-GP	5	64	Smooth convergence, stable critic loss	Best results; diverse, stable

3. Training Losses Across Different GAN Setups

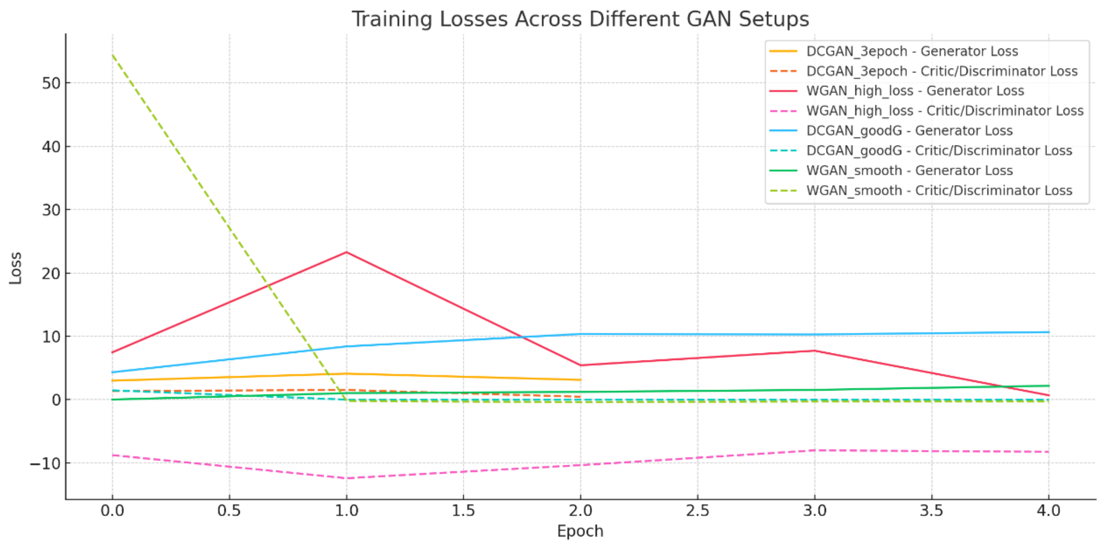

Figure3.8 *Training Losses Across Different GAN Setups*

3.4 Analysis

In our experiments, we compared DCGAN and WGAN-GP under various hyperparameter settings. The results highlight several key insights into training dynamics, stability, and image quality.

Training Stability

DCGAN training tends to be more sensitive to initialization and learning rate. In some runs, especially with fewer epochs, the generator suffered from unstable loss and occasional mode collapse. For instance, in the 5-epoch DCGAN run, the generator loss dropped to near-zero while discriminator loss was also minimal—suggesting that the generator might be overfitting or collapsing to a narrow mode. In contrast, WGAN-GP generally exhibited more stable critic loss across epochs, especially when using n_critic=5 and λ=10. This aligns with theoretical expectations that Wasserstein loss provides smoother gradients and better training behavior.

Subjective Visual Evaluation

From a qualitative perspective, WGAN-GP consistently produced more diverse and realistic face images compared to DCGAN. In early epochs, both models struggled with noisy outputs, but WGAN-GP caught up faster and generated clearer facial structures by epoch 5. DCGAN outputs were occasionally more symmetric, but lacked variation, hinting at limited latent space exploration.

Quantitative Trends

Although we did not compute Inception Score or FID due to resource limits, loss curve analysis provided useful proxies. DCGAN's generator loss showed large initial fluctuations before settling, while WGAN-GP's generator loss increased gradually then plateaued—indicating steady improvement. The critic loss in WGAN-GP remained negative as expected and oscillated within a small range, which is consistent with the theory that the Wasserstein distance is being minimized.

Comparative Performance

Overall, WGAN-GP outperformed DCGAN in both stability and output quality. The most successful training configuration used WGAN-GP with n_critic=5, batch_size=64, and a learning rate of 1e-4, where both loss and visuals reflected smooth convergence. DCGAN was easier to implement and train, but more prone to instability when hyperparameters were not finely tuned.

These findings underscore the importance of architectural choices and hyperparameter tuning when training generative models. GANs are highly sensitive to subtle configuration changes, and visual evaluation remains essential in interpreting the meaning of loss values.

4. Conclusion

In this project, we implemented and compared two popular generative adversarial network models—DCGAN and WGAN-GP—on a subset of the CelebA dataset. Through systematic experiments, we observed the training behavior, loss dynamics, and quality of generated outputs under different hyperparameter settings.

DCGAN, as a baseline, was relatively easy to train but showed instability in some cases. While it was capable of producing visually plausible faces, it was also prone to mode collapse and overfitting when trained for too few or too many epochs. WGAN-GP, on the other hand, provided significantly more stable training thanks to the Wasserstein loss and gradient penalty. Its outputs were generally more diverse and natural-looking, especially when trained with carefully chosen values of n_critic and batch size.

The key advantage of WGAN-GP lies in its smoother convergence and resistance to vanishing gradients, which allowed for more consistent improvements across epochs. This project reinforced an important lesson: training GANs is not just about running code—it requires careful tuning, visual checks, and interpretation beyond raw loss values.

In the future, I'm interested in exploring more advanced GAN architectures such as StyleGAN, as well as training on the full CelebA dataset to generate high-resolution images. Diffusion models also seem promising and could be a valuable extension direction. This project has deepened my understanding of generative modeling and taught me the importance of balancing theory, implementation, and empirical insight.

5.References

[1] PyTorch Team. "DCGAN Example." *PyTorch Examples Repository*, GitHub, https://github.com/pytorch/examples/tree/main/dcgan. Accessed June 2025.

[2] Cao, G. "wgan-gp." *GitHub Repository*, https://github.com/caogang/wgan-gp. Accessed June 2025.

[3] Liu, Z., Luo, P., Wang, X., & Tang, X. "Deep Learning Face Attributes in the Wild." *CelebA Dataset*, http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html. Accessed June 2025.

6.Bonus: Additional Insights and Explorations

1. Checkpointing

To prevent loss of progress due to system crashes or memory issues, I implemented a basic checkpointing mechanism during training. After each epoch, I used torch.save() to store the generator and discriminator/critic weights, as well as optimizer states. This allowed me to resume training from the last saved state using torch.load(). It proved especially useful during longer WGAN-GP runs, which occasionally ran into CUDA or dataloader memory errors.

2. Image Upscaling Attempt (64×64 → 128×128)

As a bonus experiment, I modified both the generator and discriminator to work with 128×128 resolution images by adding extra convolutional and transpose convolutional layers. While early results showed some facial structure, training quickly became unstable and memory usage increased sharply. This experiment helped me understand the computational challenges of scaling GANs to higher resolutions.

3. Subset-Based Acceleration

To reduce training time and resource usage, I created a 20,000-image subset of the CelebA dataset. This made it easier to run and compare different model configurations efficiently while still retaining enough data to observe meaningful trends.

4.Training Stability Analysis

I explored the impact of different batch sizes, numbers of epochs, and values of n_critic for both DCGAN and WGAN-GP. I observed that WGAN-GP showed more stable gradient behavior and achieved better sample quality earlier in training compared to DCGAN.

5. Model Comparison Visualization

I saved and compared generated samples and loss curves for both models across epochs. The side-by-side visualizations clearly demonstrated WGAN-GP's advantages in terms of output diversity and training convergence.

6. Code Structure and Reproducibility

The codebase was organized to allow easy switching between model types and hyperparameter settings. This made it convenient to rerun experiments and ensured reproducibility for future testing and tuning.