Investigating Diffusion Models' Superiority in Image Synthesis over GANs

Addisu Zena, Mmesomachi, Patrick Uwigize, Okechukwu Okeke
Skotech University, Russia, Moscow
Email: Addisu.Zena@skoltech.ru, Mmesomachi@skoltech.ru, patrick.uwigize@skoltech.ru, OkechukwuOkeke@skoltech.ru

***Abstract*—This project explores the theoretical foundations, architectural differences, and empirical performance of diffusion models compared to Generative Adversarial Networks (GANs). We highlight the inherent advantages of diffusion models in terms of stability, quality of generated images, and scalability, providing a comprehensive analysis that underscores their potential superiority in the domain of image synthesis.**

I. Introduction

Generative Adversarial Networks (GANs) have been the dominant approach in image synthesis for several years, achieving impressive results in generating high-quality, realistic images. However, in recent years, a new class of generative models, known as Diffusion Models, have emerged and shown remarkable performance in image synthesis, often outperforming GANs. This report aims to investigate the superiority of Diffusion Models over GANs in the field of image synthesis, providing a comprehensive analysis of the key factors that contribute to their improved performance.

II. Problem Statement

The primary objective of this project is to understand the fundamental differences between Diffusion Models and GANs, and how these differences lead to the superior performance of Diffusion Models in image synthesis tasks. Specifically, we aim to explore the following research questions: What are the architectural and training differences between Diffusion Models and GANs, and how do these differences impact their performance in generating high-quality, diverse, and realistic images? How do the noise-handling capabilities of Diffusion Models contribute to their ability to generate sharper, more detailed, and more stable images compared to GANs? What are the scalability advantages of Diffusion Models, and how do they enable these models to generate high-fidelity images at higher resolutions and on more complex data distributions? What insights can be gained from the comparative analysis of Diffusion Models and GANs that can inform the future development of generative models and their applications in various domains?

III. Data Description

The dataset used in this study is the CIFAR-10 dataset, a widely used benchmark for image synthesis tasks. The CIFAR-10 dataset consists of 60,000 32x32 color images, divided into 10 classes, with 6,000 images per class. The dataset covers a diverse range of natural images, including airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. This dataset provides a challenging testbed for evaluating the performance of Diffusion Models and GANs in generating high-quality, diverse images.

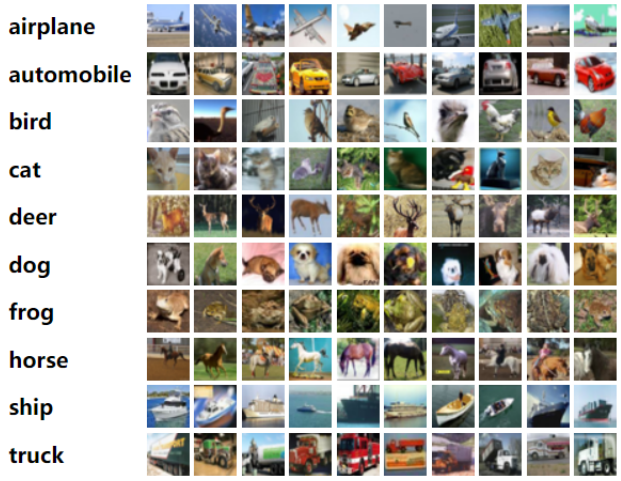

Fig. 1. cifar10 data set

IV. Background

A. Generative Adversarial Networks (GANs)

GANs is discribed in more details in [1]. In summary, Generative Adversarial Networks (GANs) are a type of machine learning framework consisting of two neural networks, a Generator and a Discriminator, that compete against each other.

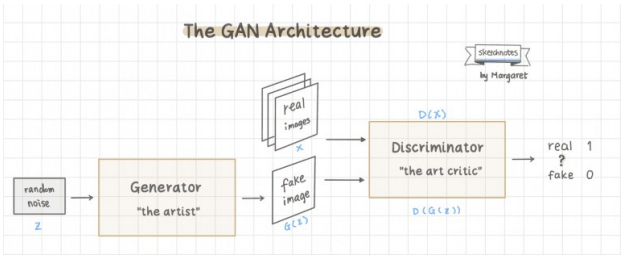

Fig. 2. GAN architecture

- Generator (G) generates fake data samples from a noise distribution while Discriminator (D) acts as a classifier for identifying real or fake data. Training process consist of 4

stages. In first stage, G generates fake data samples from a noise distribution, secondly, D evaluates real and fake data samples, thirdly, update discriminator by maximizing the probability of corecly identifying real and fake data and lastly, the generator minimises the discriminator's ability to distinguish between real and fake data.To learn the generator's distribution p_g over data x, we start with a prior on input noise variables $p_z(z)$ and map this to data space using $G(z;\theta_g)$, where G is a differentiable function with parameters θ_g. Additionally, we have another multilayer perceptron $D(x;\theta_d)$ that produces a single output representing the probability that x is from the actual data rather than p_g.

We train D to maximize its accuracy in distinguishing between real data and generated samples. At the same time, we train G to minimize $\log(1 - D(G(z)))$. This process involves D and G competing in a minimax game, which can be described by the value function $V(G, D)$:

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

It should be noted that, in practice, the equation 1 above do not adequately provide sufficient gradient for G to learn well. In case $\log(1 - D(G(z)))$ saturates, instead of training G to minimize $\log(1-D(G(z)))$, we can train G to maximize $\log D(G(z))$. This approach results in the same fixed point for the dynamics of G and D, but provides much stronger gradients early in the learning process.

The generator aims to produce realistic images, while the discriminator attempts to distinguish between real and generated images. Despite their success, GANs are notorious for training instability and mode collapse [2], [3].

B. Diffusion Models

Diffusion models, on the other hand, generate images by iteratively denoising a variable starting from pure noise. This process is inspired by non-equilibrium thermodynamics, where the generative process is modeled as a gradual transformation from a simple distribution to a complex data distribution [4]. Unlike GANs, diffusion models are typically more stable and do not suffer from mode collapse.

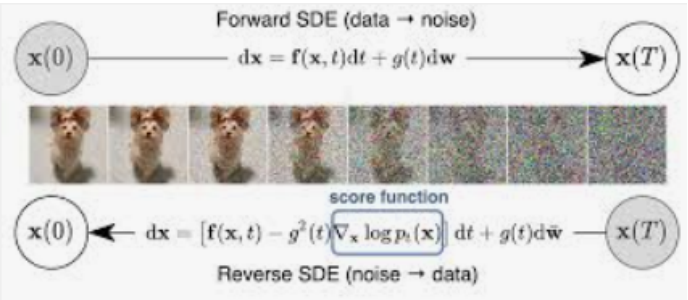

Fig. 3. Diffusion model process (source:https://theaisummer.com/diffusion-models/)

The process involves starting with a noisy input x_T and iteratively refining it to produce a cleaner output x_0 in T steps. These models aim to produce slightly cleaner versions of the original samples by learning to predict and remove the noise.

During training, the model is trained to predict the noise component of a noisy sample x_t and minimize the difference between the predicted noise $\epsilon(x_t, t)$ and the true noise using a mean-squared error loss:

$$||\epsilon(x_t, t) - \epsilon||^2$$

Recent advancements in diffusion models have introduced several improvements. For example, a hybrid training objective that combines mean-squared error L_{simple} and variational lower bound L_{vlb} has been proposed:

$$L_{\text{hybrid}} = L_{\text{simple}} + L_{\text{vlb}}$$

Another notable innovation is the development of DDIM, which offers an alternative approach to sampling by allowing for non-Markovian noise processes. The denoising distribution $p_\epsilon(x_{t+1}|x_t)$ can be represented as a Gaussian:

$$p_\epsilon(x_{t+1}|x_t) = N(x_{t+1}; \mu_\epsilon(x_t, t), \Sigma_\epsilon(x_t, t))$$

Overall, presented in [5] diffusion models represent a powerful tool for generating samples from distributions, with ongoing research aimed at further enhancing their performance and efficiency.

V. Methodology

A. Diffusion Models

Diffusion Models are a class of generative models that learn to generate new samples by gradually transforming a simple distribution (e.g., Gaussian noise) into the target data distribution through a series of diffusion steps. The key components of Diffusion Models include: Diffusion Process: The diffusion process gradually adds noise to the input data, creating a sequence of increasingly noisy versions of the original data. This process is designed to capture the complex structure of the target data distribution. Denoising Process: The denoising process, also known as the reverse diffusion process, learns to gradually remove the noise from the noisy samples, ultimately generating new samples that resemble the original data distribution. This process is modeled using powerful neural network architectures, such as U-Nets or Transformers. Training Procedure: Diffusion Models are trained using a variational inference approach, where the model learns to predict the noise added at each diffusion step. This training procedure is more stable and less prone to mode collapse compared to the adversarial training of GANs.

B. Generative Adversarial Networks (GANs)

GANs are a class of generative models that consist of two neural networks, a generator and a discriminator, that are trained in an adversarial manner. The generator learns to generate new samples that resemble the target data distribution,

while the discriminator learns to distinguish between real and generated samples. The key components of GANs include: Generator: The generator network is responsible for generating new samples that resemble the target data distribution. It learns to map a latent space to the data distribution through a series of transposed convolutions and nonlinearities. Discriminator: The discriminator network is trained to distinguish between real and generated samples. It learns to extract features from the input images and classify them as real or fake. Adversarial Training: The generator and discriminator are trained in an adversarial manner, where the generator tries to fool the discriminator, and the discriminator tries to accurately classify real and generated samples. This training procedure can be challenging and prone to instability issues, such as mode collapse.

VI. Architecture of DC GAN

The generator in a DCGAN maps a latent space vector (z) to an RGB image of the same size as the training images (e.g. 3x32x32). This is accomplished through a series of strided 2D convolutional transpose layers, each paired with a 2D batch normalization layer and a ReLU activation[1][2]. The generator output is passed through a tanh function to scale it to the input data range of [-1, 1]. Batch normalization layers after the conv-transpose layers are critical for proper gradient flow during training[1][2].

The discriminator in a DCGAN is a binary classification network that takes an image as input and outputs a scalar probability indicating if the image is real or fake[1][2]. It processes the input image (e.g. 3x32x32) through a series of convolutional, batch normalization, and LeakyReLU layers, outputting the final probability through a sigmoid activation[1][2]. Strided convolutions are used for downsampling instead of pooling, allowing the network to learn its own pooling function[1][2]. Batch normalization and LeakyReLU activations promote healthy gradient flow critical for training both the generator and discriminator[1][2].

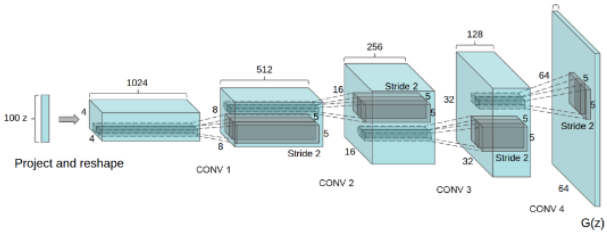

Fig. 4. dc gan:refer figure 14 and 15 under Appendix section

VII. Experimental Setup

In this study, we will train and evaluate both Diffusion Models and GANs on the CIFAR-10 dataset. We will explore various architectural and hyperparameter choices for both models, including: Diffusion Models: Diffusion process depth and noise schedule Neural network architecture (e.g., convolutional network) Optimization hyperparameters (learning rate, batch size, etc.) Sampling techniques (e.g., DDIM, Latent Diffusion) GANs: Generator and discriminator architectures (e.g., DCGAN, SAGAN) Adversarial training hyperparameters (e.g., learning rate, gradient penalty) Stabilization techniques (e.g., spectral normalization, gradient clipping) Regularization methods (e.g., self-attention, progressive growing) We will compare the performance of the trained models using both quantitative and qualitative metrics, such as Inception Score, Fréchet Inception Distance, and visual inspection of generated images. Additionally, we will analyze the generated images to gain insights into the strengths and limitations of each model.

VIII. Evaluation metric

Fréchet Inception Distance (FID): Measures similarity between generated and real image distributions Lower scores indicate better quality and similarity Uses pre-trained Inception model features to compare mean and covariance Sensitive to both quality and diversity, widely used for generative models Inception Score (IS): Evaluates diversity and fidelity of generated images Analyzes conditional and marginal probabilities of class labels Higher scores indicate more realistic and diverse generated images

Performance metrics	Dc Gan	Diffusion model
Inception score	(2.50)(3)	1.75
FID	20	30
precision	0.64	0.78
Recall	0.6	0.56

Fig. 5. Tabel:1

IX. Results and Discussion

Figure 6: The output of the DCGAN (Deep Convolutional Generative Adversarial Network) model after 30 epochs is not bad.At this level of epoch, dc gan produces good image. Figure 8: This figure shows the loss of the discriminator (marked in yellow) and the generator (marked in blue) during the training process. Figure 9: The output of the DCGAN model after 100 epochs is bad, and the generator starts producing poor-quality results. To overcome this, the hyperparameters need to be experimented with. Figure 10: The loss of the discriminator and generator for 100 epochs is shown. For the first 49 epochs, the loss of both the discriminator and generator is normal. However, after 50 epochs, the generator becomes extremely successful and starts producing garbage results. Figure 11: This figure shows the output of the diffusion model during the noising forward process. Figure 12: This figure shows the

output of the diffusion model during the denoising process, which is for 40 epochs. To get higher-quality images, the model needs to be run for at least 100 epochs, but this takes a long time. Even running the diffusion model for 20 epochs takes around two hours.

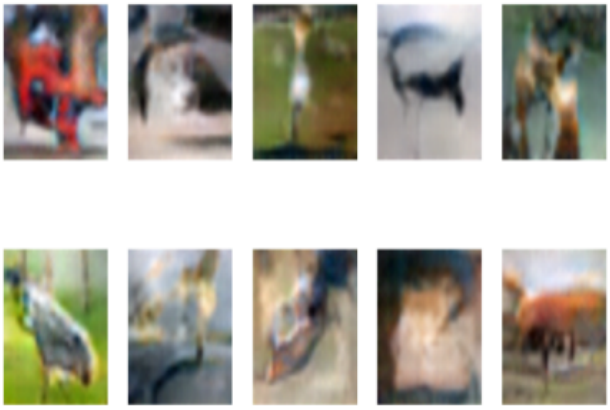

Fig. 6. Output 1

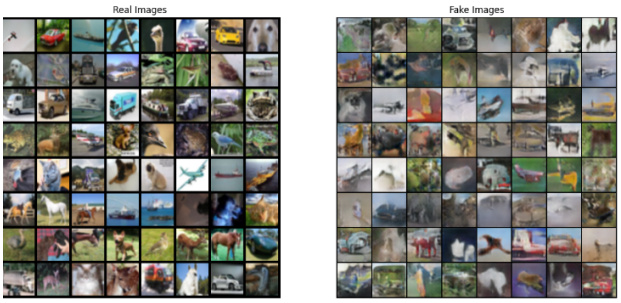

Fig. 7. Output 2

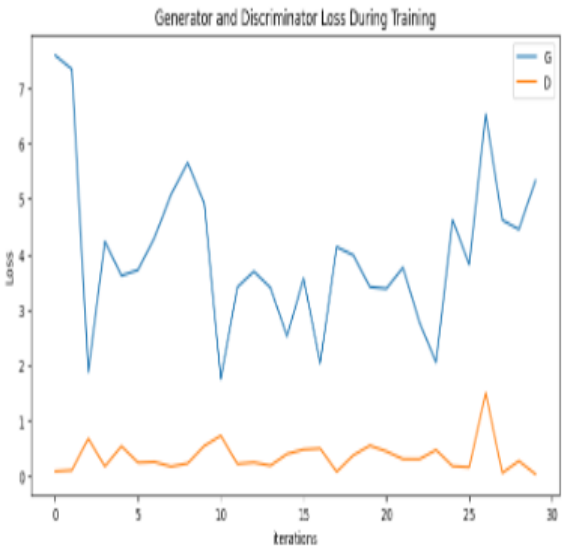

Fig. 8. Graph 2: Output 3

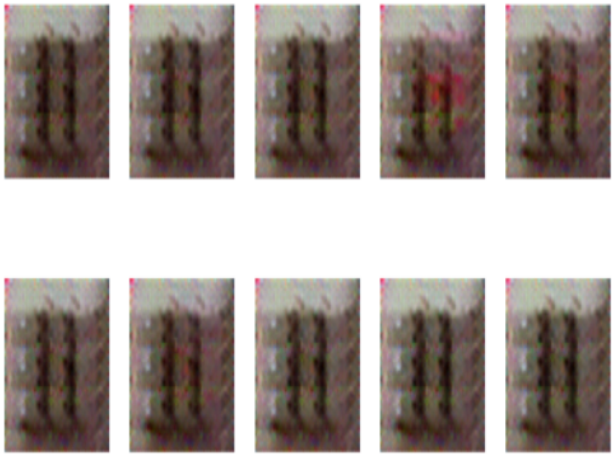

Fig. 9. Output 4

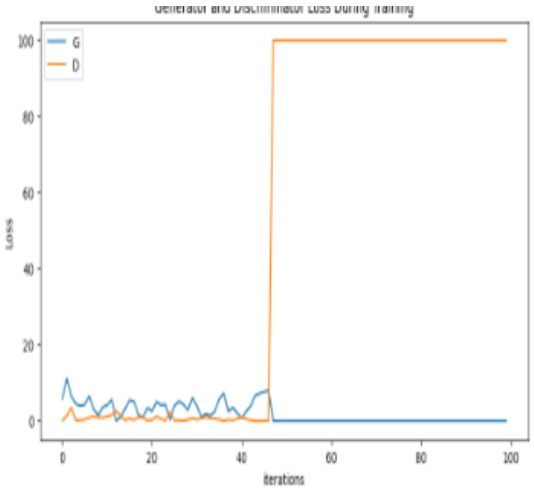

Fig. 10. Output 5

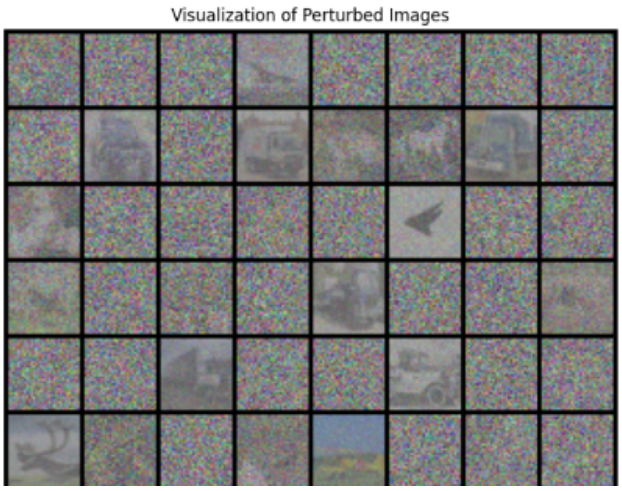

Fig. 11. Output 6

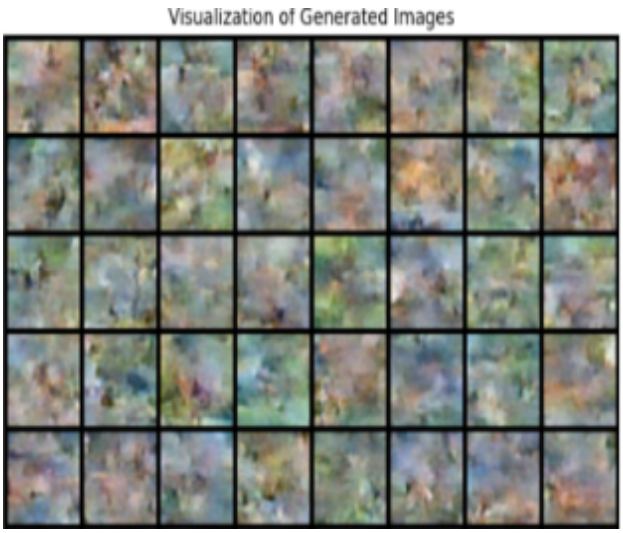

Fig. 12. Output 7

X. Conclusion

Our findings highlighted the importance of careful hyper-parameter selection for both models. For GANs, the quality of the output varied significantly with the number of training epochs. Specifically, during epochs 30-49, the GAN produced good quality images, while epochs 50-100 resulted in exceptionally successful outputs from the generator. In contrast, diffusion models required a higher number of epochs to achieve high-quality images, with the first 30-40 epochs producing subpar results. This study underscores the nuanced requirements for optimizing each model and suggests that, while diffusion models may ultimately produce superior images, they demand more extensive training to reach their full potential.

XI. contribution

1 Addisu Zena and Mmesomachi Nwachukwu: diffusion model, Gan model training and coding.

2 Patrick Uwigize and Okechukwu Okeke: diffusion model training and coding.

References

[1] I. Goodfellow *et al.*, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, pp. 2672–2680, 2014.

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

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[4] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," *arXiv preprint arXiv:1503.03585*, 2015.

[5] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *arXiv preprint arXiv:2105.05233*, 2021.

XII. Appendix

A. Github

you can scan the following QR code

Fig. 13. Github repository

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Fig. 14. Descriminator used for dc gan

```
Generator(
  (main): 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): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
```

Fig. 15. Generator used for dc gan

```
batch_size = 128
image_size = 64
nc = 3
nz = 100
ngf = 64
ndf = 64
num_epochs = 30
lrG = 0.0001
lrD = 0.0004
beta1 = 0.5
ngpu = 1
```

Fig. 16. hyper parameter of dc gan

```
timestep_embedding_dim = 256
n_layers = 8
hidden_dim = 256
n_timesteps = 1000
beta_minmax=[1e-4, 2e-2]

train_batch_size = 128
inference_batch_size = 64
lr = 5e-5
epochs = 40

seed = 1234
```

Fig. 17. Hyper parameter of diffusion model