Image Generation with Generative Adversarial Networks

Martyna Majchrzak, Władysław Olejnik

June 13th, 2023

Abstract

This report presents a project focused on bedroom image generation using Generative Adversarial Networks (GANs) and the LSUN Bedroom dataset. Various GAN architectures were trained and evaluated to generate high-quality bedroom images. The best-performing architecture generated 64x64 images and achieved a Fréchet Inception Distance (FID) value of 229.87. These results were then compared with the state-of-the-art model for this specific problem, which is the Diffusion GAN trained on the LSUN Bedroom dataset and yielded an impressive FID value of 1.43.

Contents

1 Introduction

The aim of this raport is to present the results of a project conducted as part of the Deep Learning course at Warsaw University of Technology. The project included image generation with Generative Adversarial Networks using the LSUN bedroom dataset.

2 Theoretical Introduction

Generative Adversarial Networks (GANs) have emerged as a powerful class of deep learning models that can generate synthetic data with remarkable realism. GANs have gained significant attention and popularity due to their ability to learn complex patterns from training data and generate novel samples that resemble the original distribution.

Generative Adversarial Networks, introduced by Ian Goodfellow and his colleagues in 2014 [GPAM+14], consist of two key components: a generator and a discriminator, both usually implemented using deep neural networks. The generator aims to create synthetic data samples that mimic the real data distribution, while the discriminator is trained to distinguish between real and generated samples. The goal of GANs is to train both the generator and discriminator simultaneously, leading to an adversarial learning process.

The training of GANs can be viewed as a two-player minimax game. The generator tries to minimize the discriminator's ability to correctly classify generated samples as fake, while the discriminator aims to maximize its accuracy in distinguishing between real and fake samples. Through this adversarial interaction, the generator and discriminator learn from each other, improving their respective capabilities over time. The generator takes as input a random noise vector and transforms it into synthetic data samples. The discriminator, on the other hand, receives both real and generated samples and predicts whether each sample is real or fake.

2.1 DCGAN

A DCGAN is a direct extension of the GAN described above, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. It was first described by Radford et. al. in the paper Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks [RMC15]. The discriminator is made up of strided convolution layers, batch norm layers, and LeakyReLU activations. The input is a 3x64x64 input image and the output is a scalar probability that the input is from the real data distribution. The generator is comprised of convolutional-transpose layers, batch norm layers, and ReLU activations. The input is a latent vector z, that is drawn from a standard normal distribution and the output is a 3x64x64 RGB image. The strided conv-transpose layers allow the latent vector to be transformed into a volume with the same shape as an image. In the paper, the authors also give some tips about how to setup the optimizers, how to calculate the loss functions, and how to initialize the model weights.

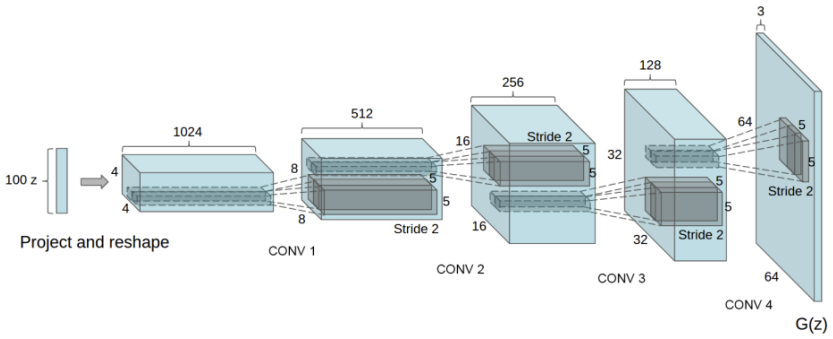

Figure 1: Generator scheme form the original DCGAN paper

2.2 Diffusion-GAN - state-of-the-art model

Diffusion-GAN [WZH+22] has emerged as a state-of-the-art model for solving the LSUN Bedroom image generation problem [ben23]. Generating realistic and diverse bedroom images poses a significant challenge due to the complexity and variability of room layouts, furniture arrangements, and lighting conditions. Diffusion-GAN addresses these challenges by combining the strengths of diffusion models and GANs.

The diffusion process in Diffusion-GAN allows for progressive refinement of noise into a base image, gradually enhancing its details and capturing the intricate characteristics specific to LSUN Bedroom scenes. This iterative diffusion technique helps the model better understand the underlying distribution of bedroom images, enabling it to generate high-quality and realistic samples. Additionally, the adversarial training of the GAN component ensures that the generated images exhibit compelling visual fidelity and coherence.

Diffusion-GAN's effectiveness in tackling the LSUN Bedroom image generation problem lies in its ability to learn and capture the complex spatial relationships between different elements within a bedroom, such as beds, wardrobes, windows, and lighting fixtures. The model can generate diverse and visually appealing bedroom scenes with a remarkable level of realism, surpassing previous approaches in terms of image quality and diversity.

3 Implementation

The code can be found in the github repository [ima]. The project structure is explained in README. We used PyTorch [PGM+19] for the DCGAN network implementation training and Weights & Biases [Bie20] to track the results of the experiments and create plots. We used Google Colab with GPU hardware accelerator to perform the experiments in the notebooks.

4 Dataset description and preprocessing

The images used to train the models come from a 10% sample of the bedroom category from the LSUN (Large-Scale Scene Understanding) dataset [YSZ+15]. The LSUN dataset that contains a vast collection of images depicting different scenes and objects, but the bedroom subset focuses specifically on bedroom scenes of various perspectives, colours, sizes etc. The archive that contains 303125 files in jpg format was downloaded from the kaggle website [kag]. All images are at least 256x256 resolution.

To prepare the images for modelling we resized them to the appropriate size (64x64 or 128x128 depending on the used network architecture), centered and normalized them. Then, a data loader with a batch size of 256 was created. The figure 2 presents examples of dataset images after the transformations.

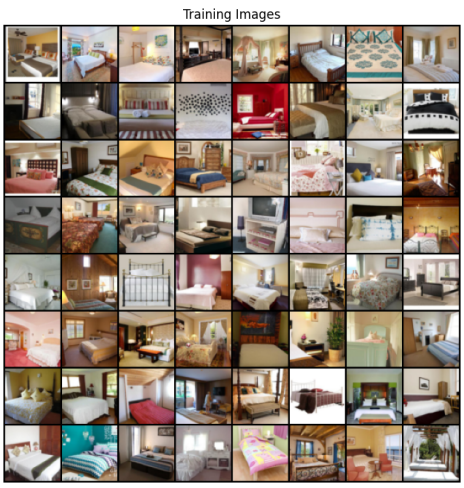

Figure 2: Example images form the LSUN bedroom dataset after preprocessing

5 Metrics

To monitor how the models behave during training we used the following metrics:

- discriminator loss - sum of the value of the criterion on the outputs that the discriminator returns for the real and fake images
- generator loss - the criterion on the output that the discriminator returns for the fake images from the generator
- discriminator response D(x) - mean of values returned by the discriminator in each step, shows how sure the discriminator was that the generated images were real (0 - fake image label, 1 - real image label).

To evaluate the final models we used the official pytorch implementation of Fréchet Inception Distance, using the pytorch-fid package [Sei20].

6 Experiments and results

6.0.1 DCGAN 64x64

At the beginning we tried to use the original DCGAN architecture described in the paper - generating images of 64x64 resolution. We trained the models for 5 epochs using the following configuration:

- size of feature maps in generator and discriminator equal to 64
- size of latent vector of 100
- Adam optimizer for both generator and discriminator with learning rate 0.0002 and beta of 0.5
- Binary Cross Entropy loss criterion for both generator and discriminator

Values of generator and discriminator loss during the training process are shown on figure 3.

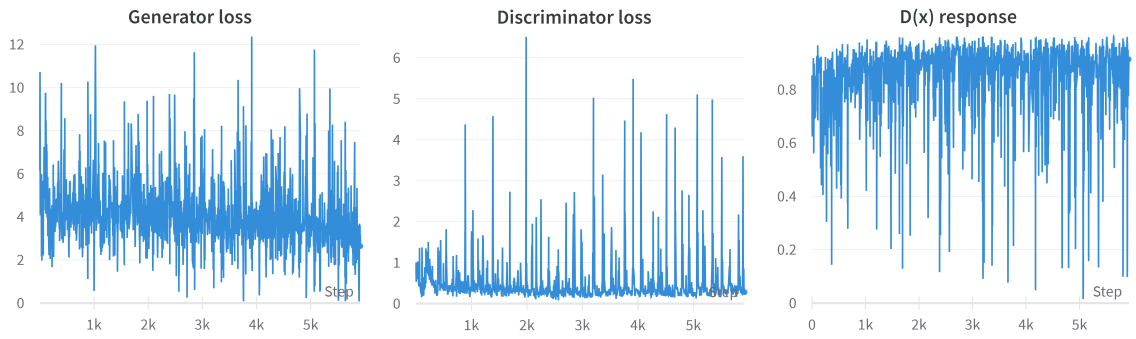

Figure 3: Metrics for DCGAN 64x64 architecture during training process

The loss values in the first model, that can be seen on figure (3, have very high variability. They fluctuate around 0.3 (discriminator) and 4 (generator) and both slowly decrease. It indicates, that the model is actually learning. Discriminator response visible on **??** also has hig variability, but it fluctuates around 80-90%, which indicates that a generator is doing a good job at fooling the discriminator.

On figure 4 we present fake images generated by the model compared to real images. They are a decent quality: they do not have any details, but the shapes present resemble bedrooms to an extend. The FID metric between those sets of pictures achieved a value of 273.99.

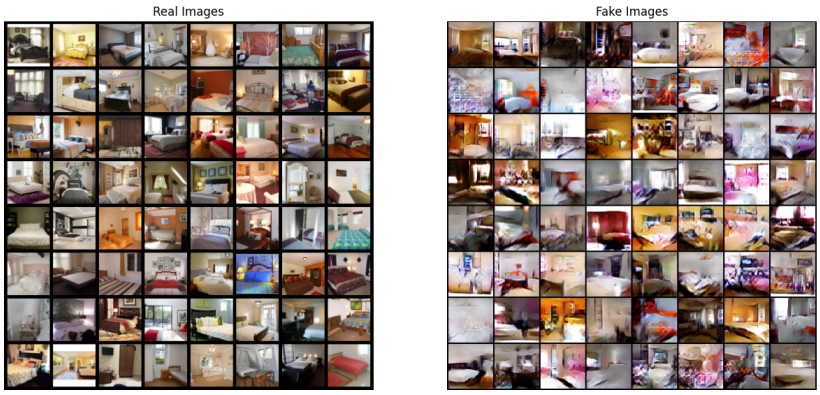

Figure 4: Real and fake images generated with 64x64 model

6.0.2 DCGAN 128x128

For this experiment modified the model architecture to allow generating 128x128 images. It also required changing the size of feature maps in generator and discriminator equal to 128, but apart from that all the parameters were set as in the experiment above.

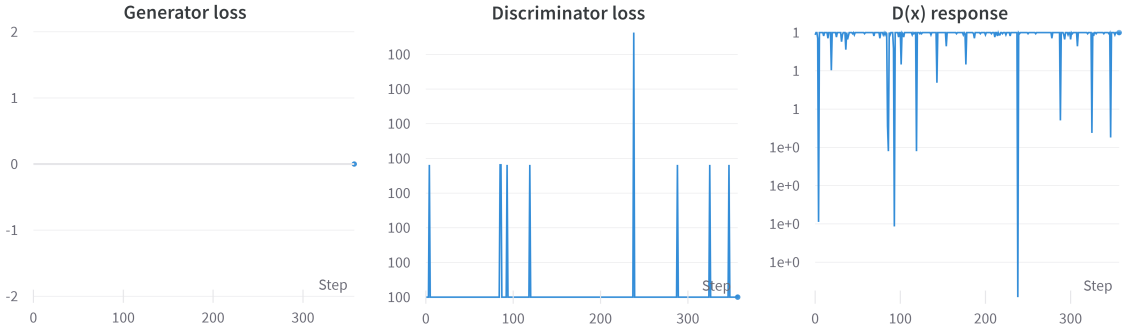

Figure 5: Metrics for DCGAN 128x128 architecture during training process

The loss values, that can be seen on figure (5, indicate that this model collapsed immediately. Discriminator loss is always 100, and the generator loss is always 0, which means that the discriminator is not learning and 'passes' everything that that the generator feeds it. This behavior is confirmed by the discriminator anwsers that the images are real (label 1) almost all the time. Because of this behavious be decided to stop the experiment after about 350 steps.

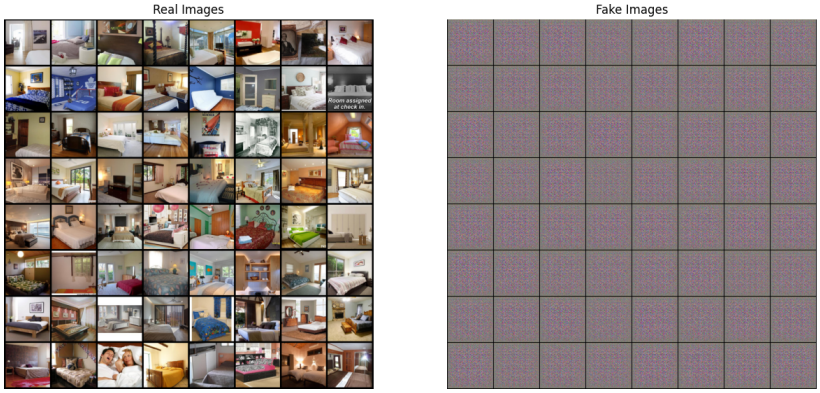

Figure 6: Real and fake images generated with 128x128 model

Unsuprisingly, the images from this model look like pure noise, and their FID value is 419.76.

6.0.3 Fighting model collapse

To prevent the DCGAN 128x128 from collapsing a combination of 3 method, inspired by a list of GAN training hacks from the ganhacks repository [gan], were tested.

1. Adding dropout 0.5 on the Generator model after each hidden layer
2. Changing the optimizer for the Discriminator from Adam to SGD
3. Encoding the labels of real and fake images with 'soft' values: 0.3 and 0.7 instead of 0 and 1.

We tried this approach for both DCGAN 128x128 (where we experienced model collapse) and DCGAN 64x64 (to see if it would improve anything). We also tried training the models for 10 epochs.

6.0.4 DCGAN 128x128 modified

The loss and discriminator response for this model look much better than the 128x128 model without dropout and other techniques aiming to prevent model collapse.

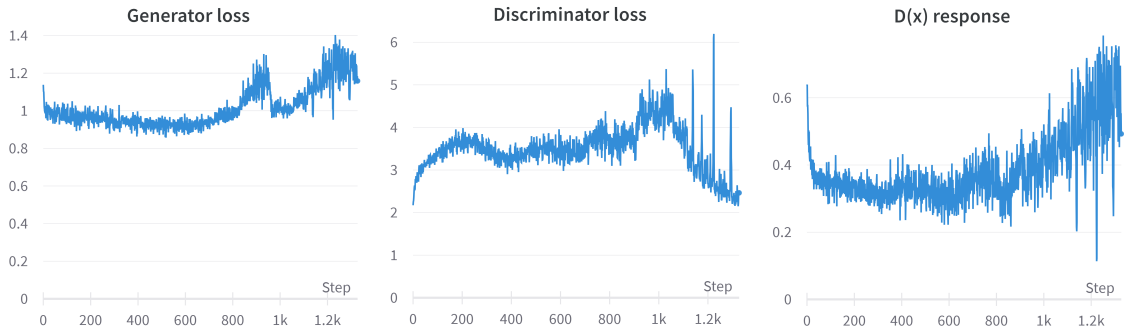

Figure 7: Metrics of a modified DCGAN 128x128 architecture during training process

The discriminator loss shown on figure 7 is slowly increasing and the generator loss fluctuates around 1. The variability of the values is small compared to the previous plots. The discriminator response is decreasing from 0.7 (real label) to around 0.3 (fake label).

Unfortunatelly the training took a very long time, so we were forced to stop the experiment after 1 epoch.

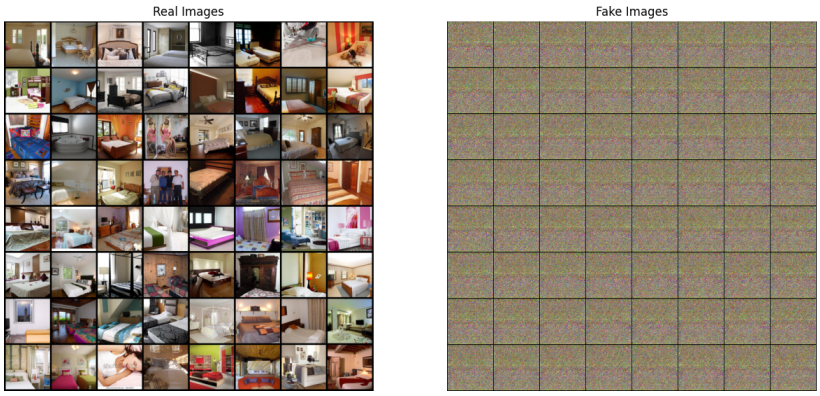

Figure 8: Real and fake images generated with modified DCGAN 128x128 model

The images generated by this model are still very noisy, but they look simmilar to images generated by DCGAN 64x64 after 1 epoch - so we could expect that if we gave this model more time it could

learn to recognize more complex shapes. It's FIS value is 418.39, so slightly better than the DCGAN 128x128 architecture without changes.

6.0.5 DCGAN 64x64 modified

We decided to test the same changes in the architecture and training process for DCGAN model for 64x64 images.

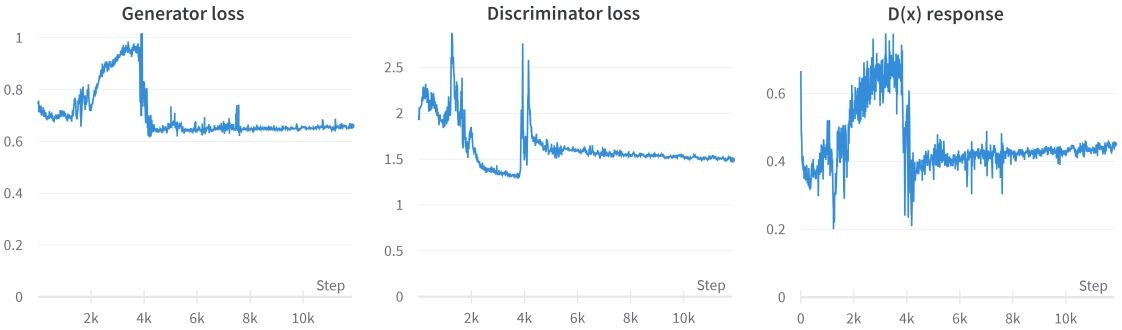

Figure 9: Metrics of a modified DCGAN 64x64 architecture during training process

Figure 9 shows that after about 5k steps, the discriminator loss stabilizes around 1.5 and generator loss - 0.7. Simmilarily, after 5k steps discriminator response stabilizes at a little over 0.4, so the model generally recognizes most of the fake images as fake.

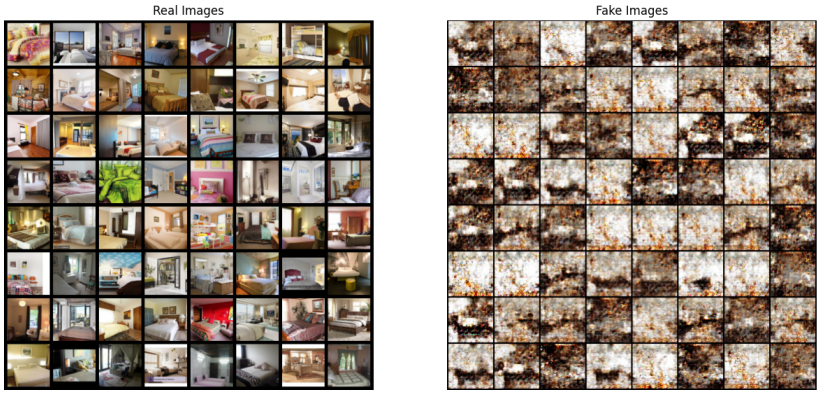

Figure 10: Real and fake images generated with modified DCGAN 64x64 model

However, even though the training process seems to have been stable and succesfull, the images generated with 64x64 architecture with the described changes are not satisfactory - they contains simple shapes that match the general shapes on the bedrrom image, but they lack colour (they are mostly brown, orange and white) and are very simmilar among each other. The FID value achieved by those images are also not impressive - 415.95 is just a little bit better than the pure noise created by the previously presented collapsed model (original DCGAN 128x128).

6.0.6 DCGAN final model

Since the methods aiming to fight with the model collapse significantly decreased the FID score and variability of the DCGAN 64x64, we decided to not use those in the final solution and simply use the original DCGAN architecture and train it for 10 instead of 5 epochs.

Figure 11: Real and fake images generated with final model.

The images generated by this model achieved the FID value of 229.87 and they seem to be the most realistic looking ones so far, with more details than the DCGAN 64x64 trained for 5 epochs.

To further evaluate the model interpolation vectors between two vectors of random noise were created and then images were created using the Generator from the final DCGAN model.

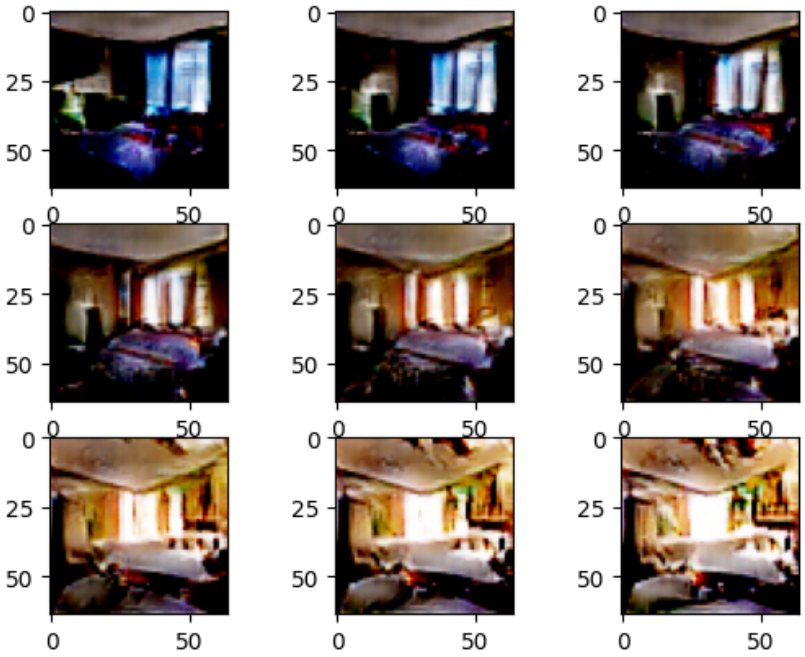

Figure 12: Images created on interpolation latent vectors

The effects of this can be seen in the figure 12. The transition between the images is smooth and it is possible to see diffrent elements of the picture transforming into another - for example the blue window on the first image keeps moving towards the center of the picture and changes to more yellow colour and the ceiling becomes larger and more triangular. This visible transition indicates that the generator has not only learned to generate pictures simmilar to those from the training images, but also is capable of representing the changes in the latency space in the generated images.

6.0.7 Diffusion-GAN results

In the last step we compared obtained results, with images generated by state-of-the-art model trained on our dataset ([WZH+22], [ben23]). Model trained and created by Wang et al. achieved impressing FID value of 1.43. After some problems with environment setup (used libraries, packages and python version) we generated and obtained amazing images shown on figure 13.

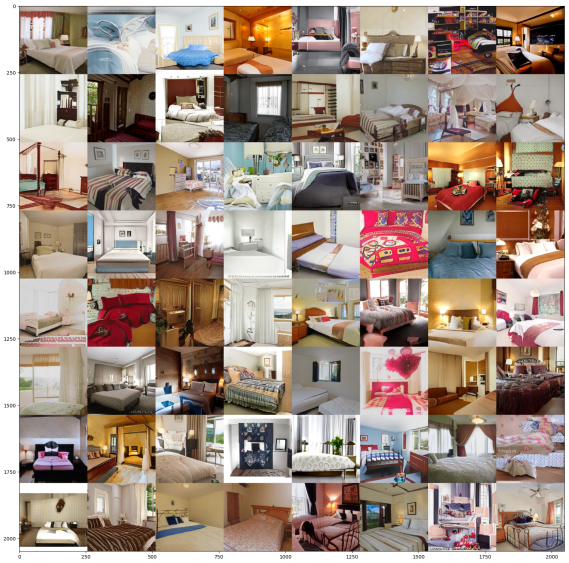

Figure 13: Images obtained from Diffusion GAN model

7 Conclusions

Table 1 shows the summary of tested configurations with execution times and qualitative and quantitative assessment.

Model	Image size	Epochs	Time	GPU	FID	Quality of images
DCGAN	64	5	1h 24m	T4	273.99	decent
DCGAN	128	400 steps (stopped)	35m	T4	419.76	noise
DCGAN dropout	64	10	1h 14m	V100	415.95	mostly noise
DCGAN dropout	128	1	2h 30m	T4	418.39	mostly noise
DCGAN	64	10	1h 9m	V100	229.87	good
Diffusion	256	-	-	-	1.43	realistic

Table 1: Comparison of tested GAN architectures

The main conclusions from conducting this project are:

- For DCGAN 128x128 adding dropout, soft labels and SGD optimizer increased model stability and prevented collapse, allowing the model to learn. However, for DCGAN 64x64, which was training fine, the changes decreased the performance very significantly. This may be because dropout is a regularisation technique and it prevented the generator form learning all the details in the training dataset.
- Computation times for 128x128 images are much higher than the 64x64 images (2,5h/epoch vs 15 minutes/epoch)

- The model with the smallest found FID value is the original DCGAN architecture for 64x64 images, trained for 10 epochs.
- Diffusion could be useful and powerful tool to combine with GANs, improving obtained results significantly.

References

[ben23] Benchmark for lsun bedroom image generation problem, 2023.

[Bie20] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[gan] How to train a gan? tips and tricks to make gans work. https://github.com/soumith/ganhacks.

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[ima] Image generation with gans. https://github.com/m-majchrzak/Image_Generation_with_GANs.

[kag] Lsun bedroom scene 20% sample. https://www.kaggle.com/datasets/jhoward/lsun_bedroom?resource=download.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[RMC15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[Sei20] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid, August 2020. Version 0.3.0.

[WZH+22] Zhendong Wang, Huangjie Zheng, Pengcheng He, Weizhu Chen, and Mingyuan Zhou. Diffusion-gan: Training gans with diffusion. *arXiv preprint arXiv:2206.02262*, 2022.

[YSZ+15] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. Technical Report 1506.03365, arXiv preprint, June 2015.