WARSAW UNIVERSITY OF TECHNOLOGY

Faculty of Mathematics and Informational Science

Deep Learning Methods

Project 3. report

Generative Adversarial Network

Mateusz Majewski **Kornel Mrozowski**

Warsaw 2022L

Abstract

In this report we present our solution to the 3nd project of the Deep Learning class. We tested three different GAN architectures: a simple DCGAN, a DCGAN extended with progressive learning (ProGAN style), and StyleGAN3. We also tried to implement a VAE architecture and then use it to pretrain a GAN, however we failed to make this work. Additionally, in order to combat training collapse, we tested various learning rates.

1 Task description

In this project, we were meant to train a network to generate bedroom images. We used the sampled Bedroom dataset from [8]. The dataset contains various images of bedrooms. The images are colorful (i.e. not black and white) and have varying sizes.

Our task then was to create more images of bedrooms, similar to those in the learning set (though not necessarily in size). Unlike the last time, the task was not based on a Kaggle competition, which means that we had to assess the results ourselves: both quantitatively (by calculating the Frechet Inception Distance) and qualitatively (by using our eyes). While doing so, we had to focus on the generative adversarial network (GAN) area.

Additionally, we wanted to focus on the influence of hyperparameters on the results. Finally, we needed to generate two images from given latent vectors, and then linearly interpolate the latent vectors to generate more images; this was meant to check if the model is able to interpolate between images.

2 Technical matters

As mentioned in the conspectus, we used the PyTorch [4] framework for this project. We took the basic structure of our code from [5]. When implementing VAE, we were inspired by [6]. Finally, to calculate FID, we used the pytorch-fid [7] library.

We preprocess the data by resizing the shorter dimension of each picture to 64 pixels and taking the center 64×64 picture. Additionally, we normalize the pictures. Our data then consists of tensors with value in the $[-1, 1]$ interval.

3 Used network architectures

3.1 DCGAN

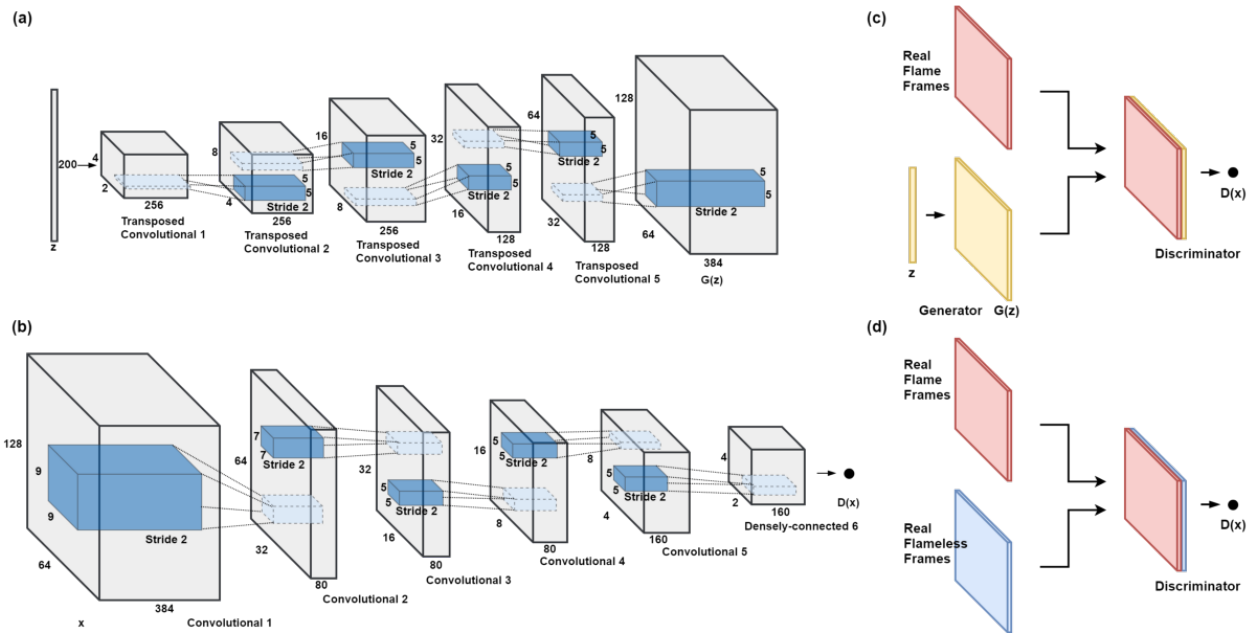

Figure 1: DCGAN generator and discriminator architectures

As mentioned before, we used three GAN architectures. First, we used the exact DCGAN presented in the [5] tutorial. This is a traditional DCGAN, where the generator consists of blocks consisting of convolutional transpose, batch normalization, and ReLU layers, and the discriminator consists of blocks consisting of convolution, batch normalization, and LeakyReLU layers. Each network has five blocks. The last block of generator returns the hyperbolic tangent of the image data instead, and the last block of the discriminator converts all the data into one sigmoid output.

Then we implemented a network using progressive learning. While this is inspired by [3], we did not copy its architecture; instead, we modified our DCGAN as little as possible in order to facilitate progressive learning. This meant that the generator consisted of blocks as before and an additional final block containing a 1x1 convolutional transpose and hyperbolic tangent, and the discriminator consisted of blocks as before and an additional initial block of 1x1 convolution and LeakyReLU.

3.2 StyleGAN3

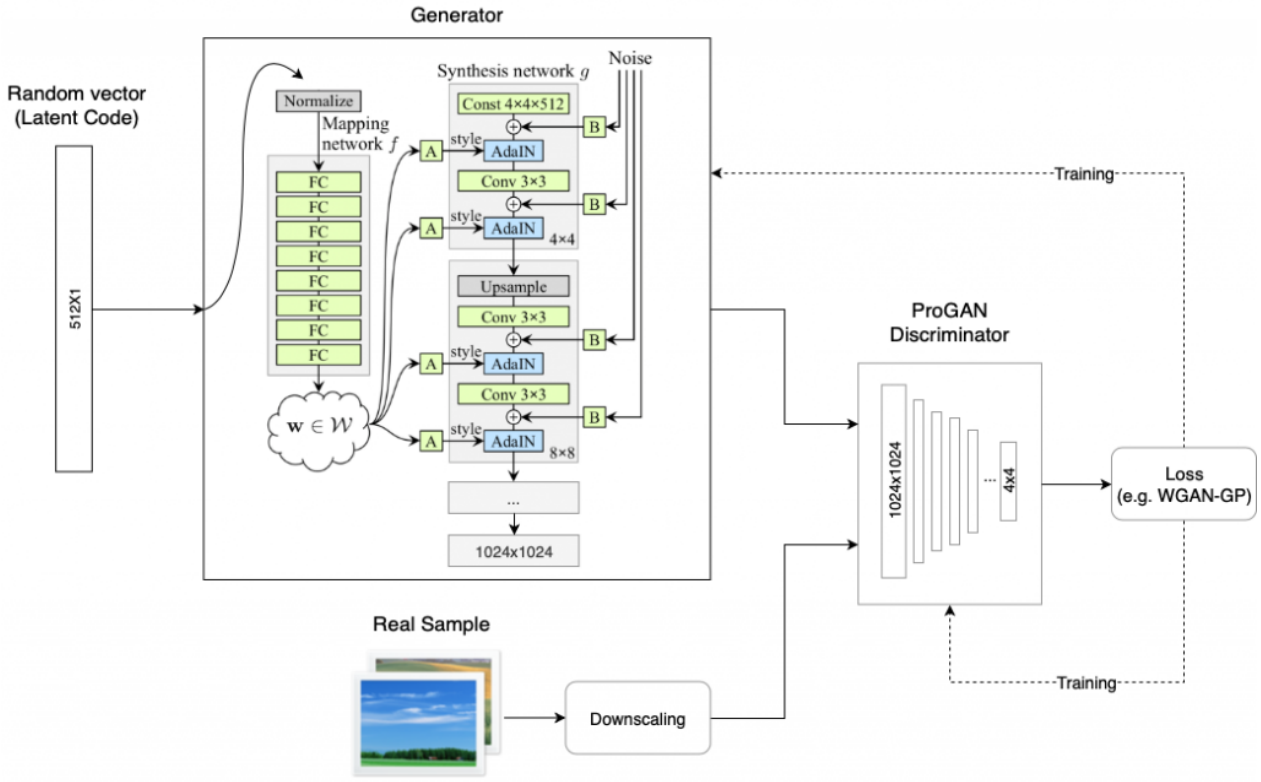

Figure 2: StyleGAN3 generator and StyleGAN2-ADA discriminator architectures with learning process visualized.

We also included a StyleGAN3 architecture [2]. This implementation has been completely based on the official implementation [1]. The StyleGAN3 learns in a progressive way and the discriminator is similar to the one used in the progressive learning experiment (with some minute differences). The heart of this architecture is the generator, which is completely different to the ones seen before. The generator consists of two parts. The first one is the *mapping network*, which takes the latent vector and runs it through several layers of a multilayer perceptron. This results in a vector which represents the style of the image, the *style vector*. Then the *syntesis network* takes the Fourier features of the resulting style vector, and runs the blocks as before. However, each block has a completely different structure: the convolutions are 1×1, so they are not allowed to mix different pixels. This is instead done by up- and downsampling. Additionally, the style vector is reintroduced in each block, and the modulation and demodulation operations are used.

3.3 VAE

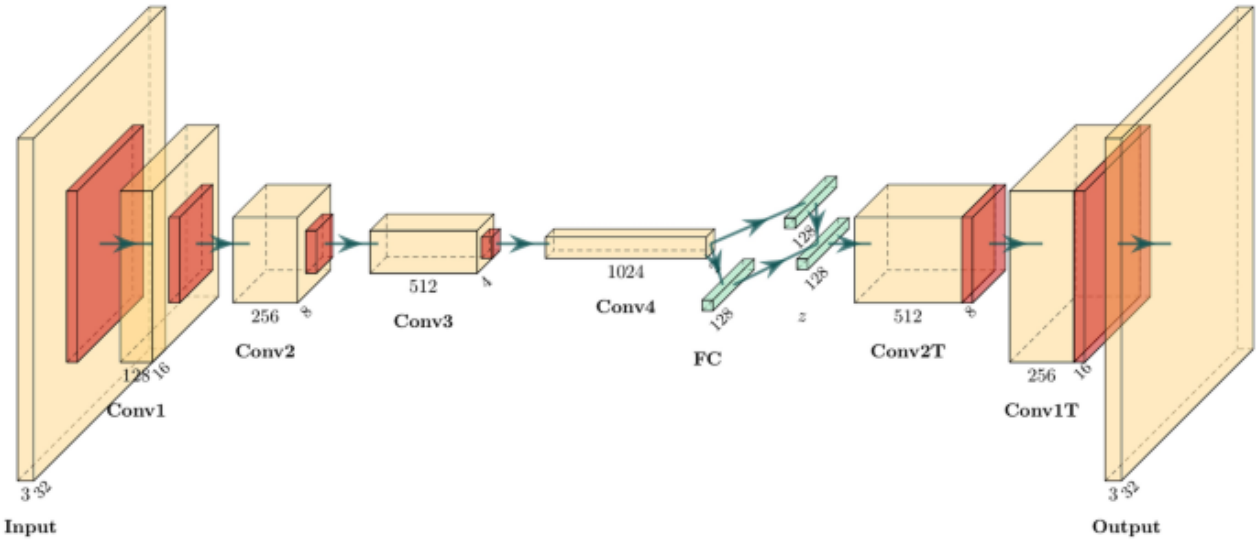

Figure 3: VAE autoencoder

Additionally, we implemented a VAE architecture. We intended it to be almost identical to the DCGAN: the only difference was that the final block of the discriminator and the initial block of the generator became ReLU dense layers of the correct size (as we found all the VAE sources used similar architecture). That was because we wanted to reuse it as a starting point of a DCGAN, a plan that we ultimately discarded due to poor VAE results.

4 Changed hyperparameters

The hyperparameters that we were experimenting with in every network were learning rate and number of filters in convolutional layers. In case of DCGAN with progressive learning additionally with a number of epochs between each progress.

5 Additional data manipulation

We did not perform extra data augumentation; as mentioned in the conspectus, we think we have enough images to not need it.

6 Results

In this section we want to describe the results we obtained. Because we were meant to decide on which experiment we will focus more we decided not to follow blindly on FID score but also take into account the visual impression the images made on us.

6.1 Initialization

After weights initialization in case of every network their output FID score was equal about 470. In other worlds, the output was completely random and lookt linke this:

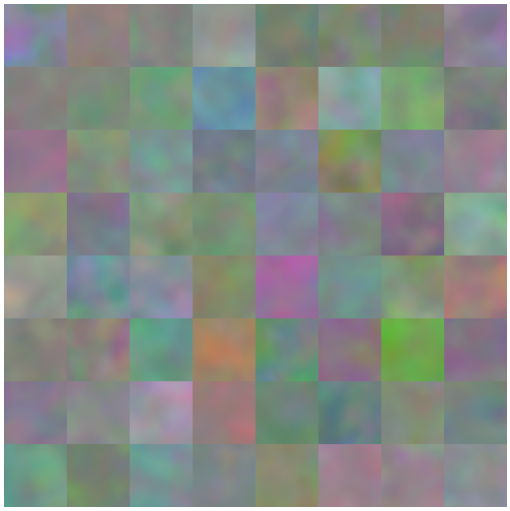

Figure 4: Random network output. FID: 470

6.2 DCGAN

The first results we obtained were initially after 100 epochs (24h of learning) - the results were quite impressive.

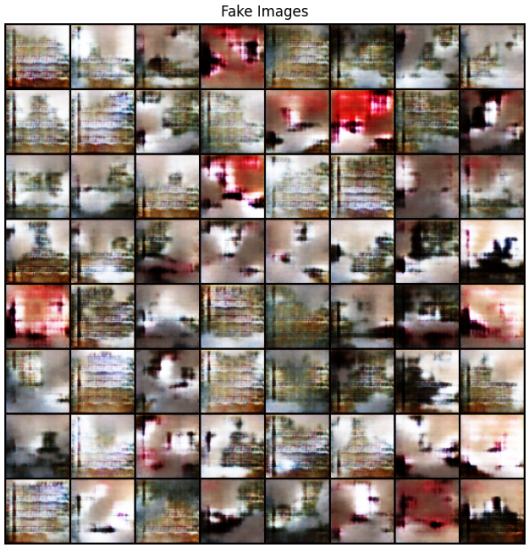

Figure 5: DCGAN output after 100 epochs.
FID: 420

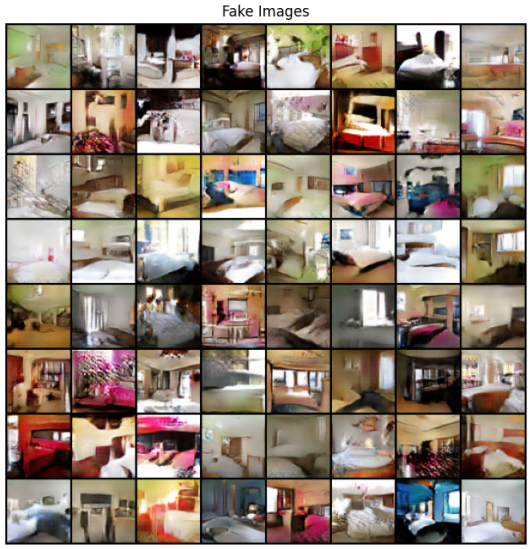

Figure 6: DCGAN output after 5 epochs.
FID: 331

6.3 DCGAN with progressive learning

In case of the progressive learning applied to the DCGAN we couldn't find out what mistake we did so that the network didn't want to learn. We now suppose that it was because of that every progress happens every a specific number of epochs instead of specific number of seen images. In every case the FID score was about 470. The results:

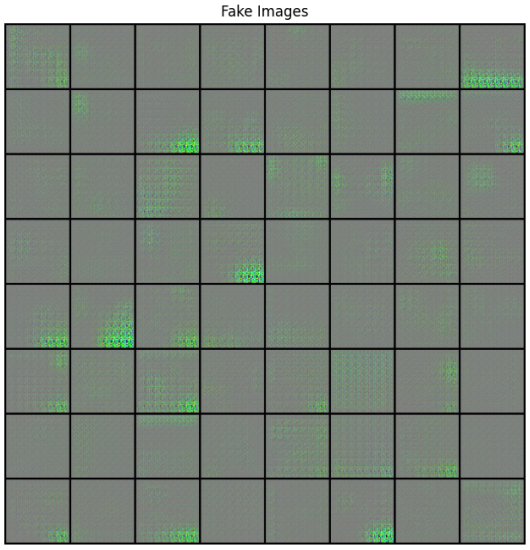

Figure 7: DCGAN with progressive learning output after 100 epochs. FID: 470

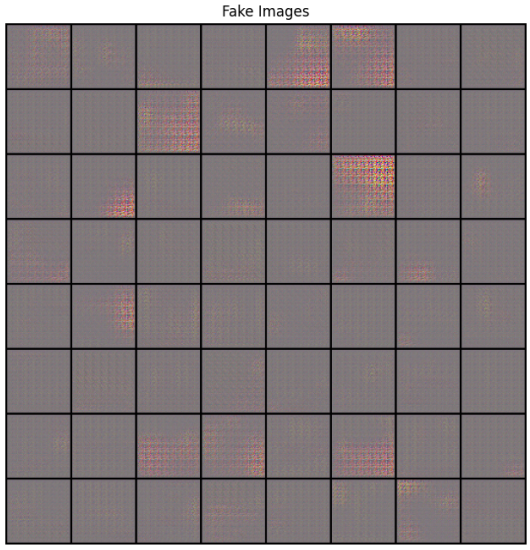

Figure 8: DCGAN with progressive learning output after 5 epochs. FID: 470

6.4 StyleGAN3

In case of StyleGAN3 we performed only one experiment because one epoch lasted for 5 hours and it outperformed all other networks of ours with both: FID score as well as visual aspects.

Figure 9: StyleGAN3 output after 60'000 images seen (20% of one epoch). FID: 424

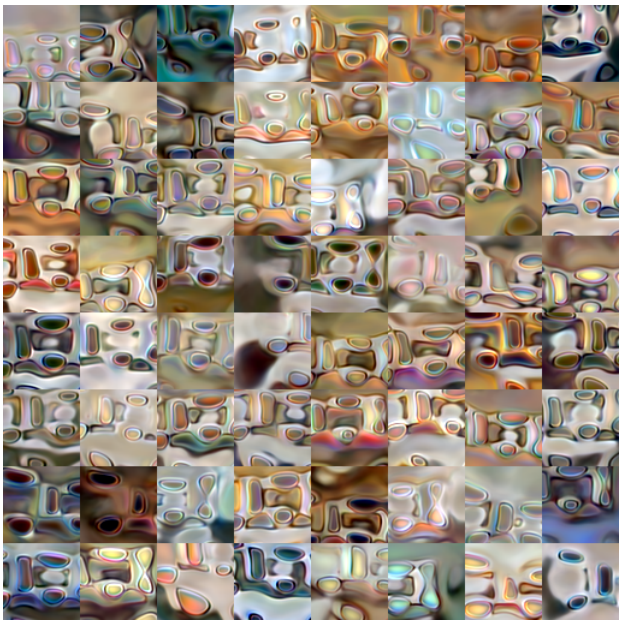

Figure 10: StyleGAN3 output after 120'000 images seen (40% of one epoch). FID: 368

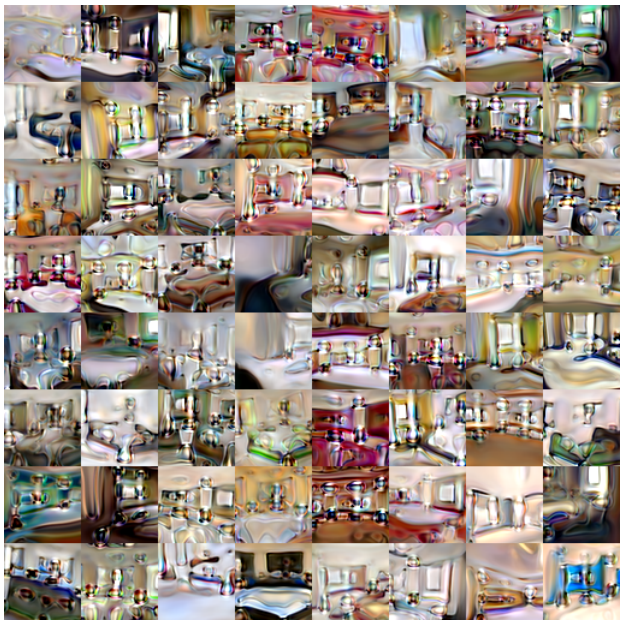

Figure 11: StyleGAN3 output after 300'000 images seen (one epoch). FID: 257

Figure 12: StyleGAN3 output after two epochs. FID: 105

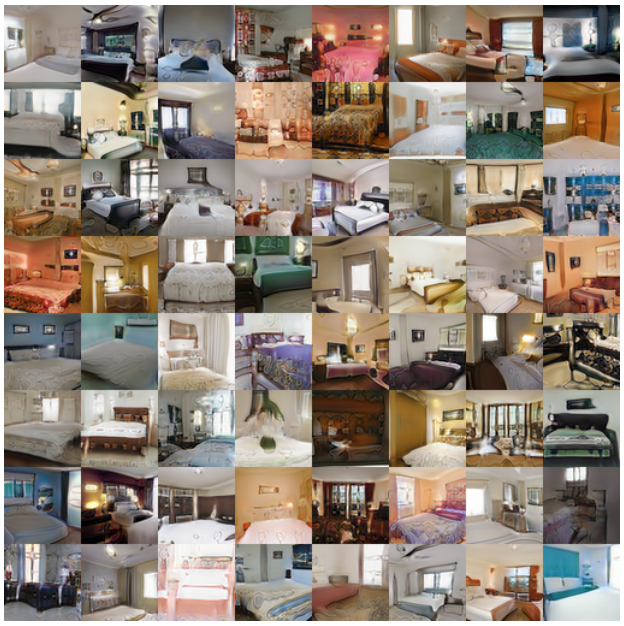

Figure 13: StyleGAN3 output after three epochs. FID: 54

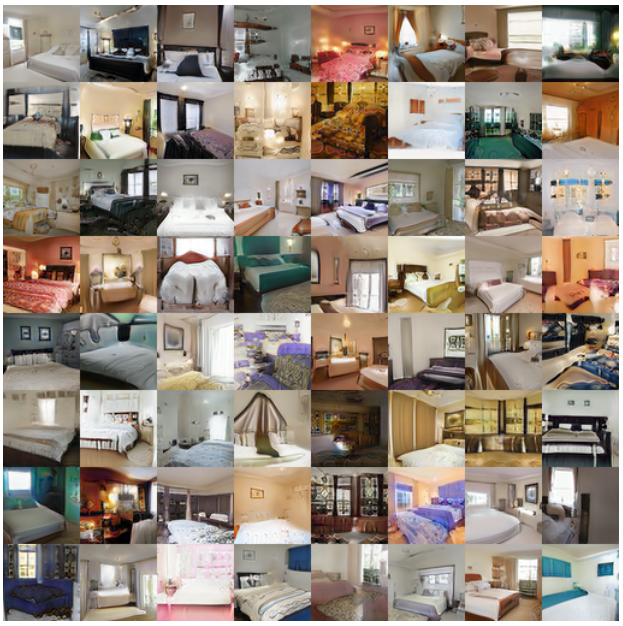

Figure 14: StyleGAN3 output after four epochs. FID: 48

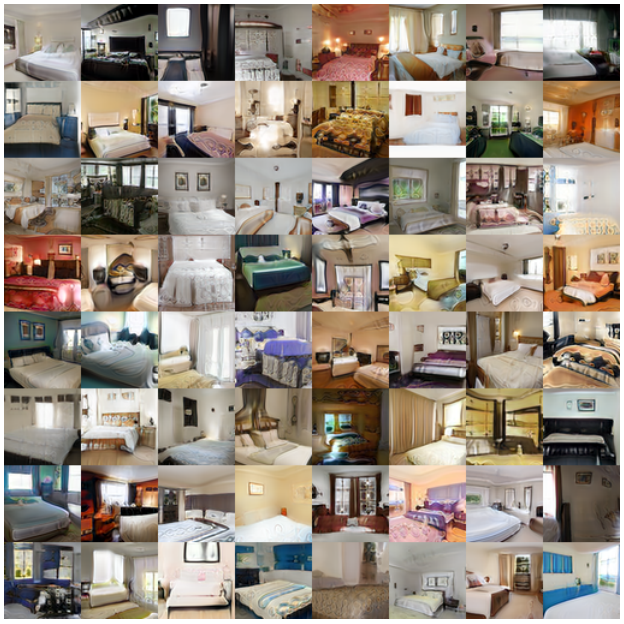

Figure 15: StyleGAN3 output after five epochs. FID: 39

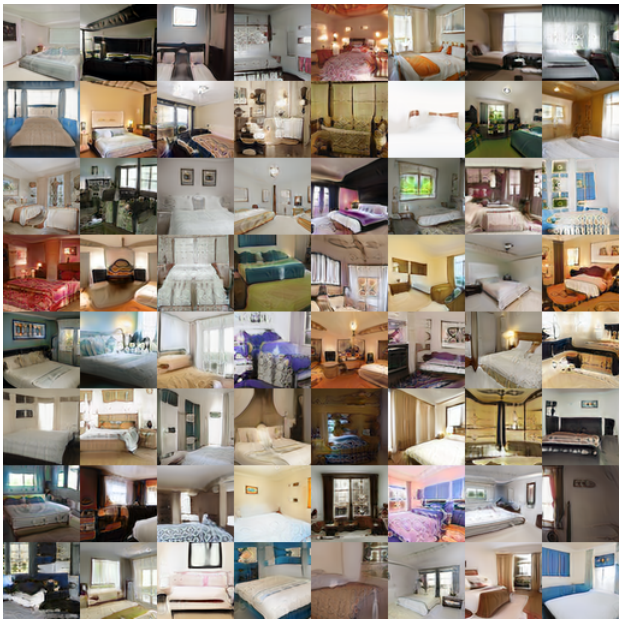

Figure 16: StyleGAN3 output after six epochs. FID: 36

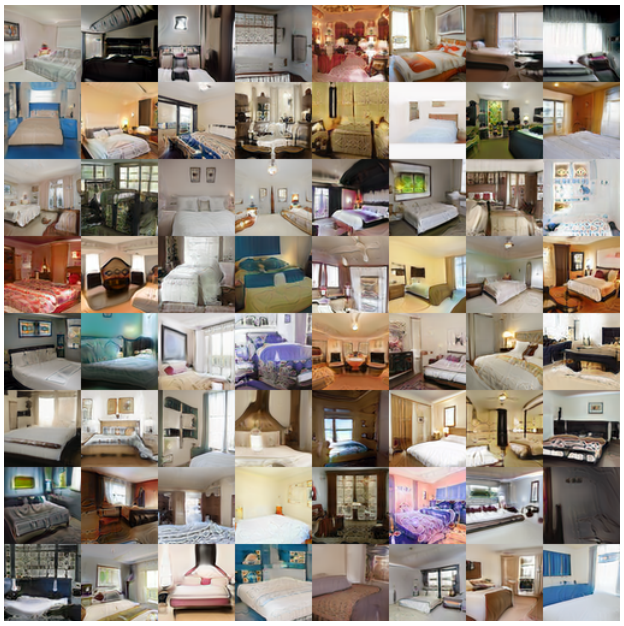

Figure 17: StyleGAN3 output after seven epochs. FID: 34

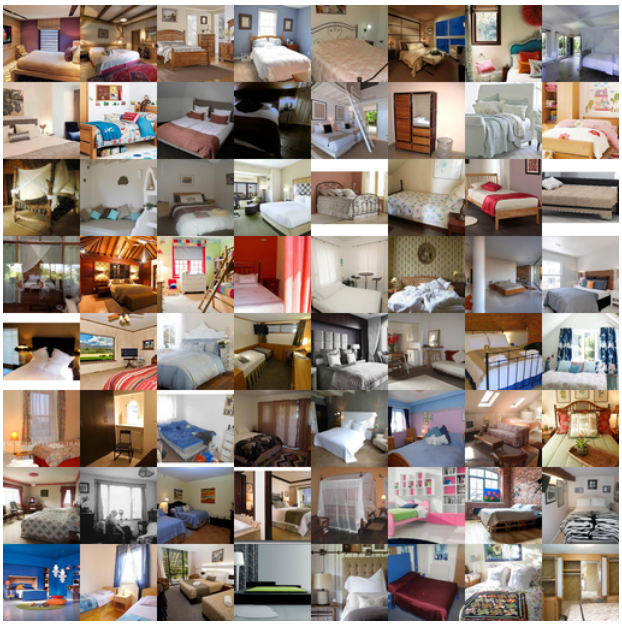

Figure 18: The real images.

6.5 VAE

Results of VAE autoencoder network were independently average no matter how many epochs the autoencoder was trained and how many filters it has in every it's convolutional layers. The results:

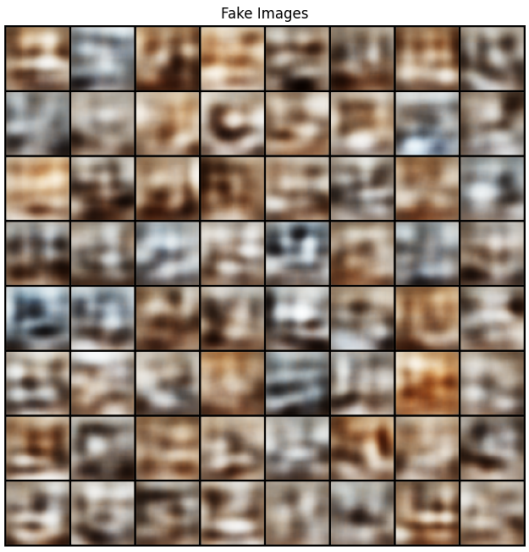

Figure 19: VAE output after 100 epochs. FID: 391

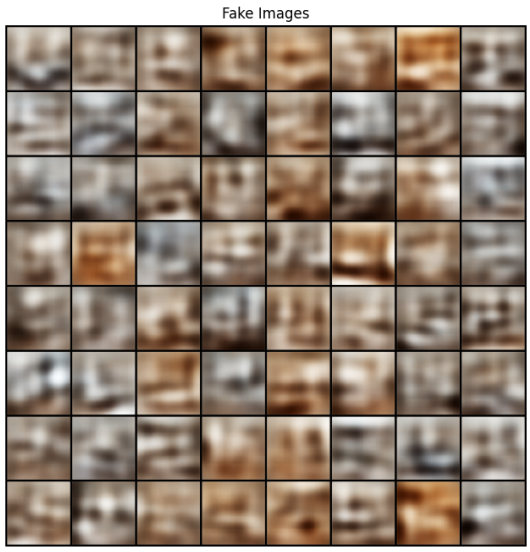

Figure 20: VAE output after 5 epochs. FID: 387

7 Summary

All the networks except DCGAN with progressive learning generated images that colors temperature of their output images were similar to temperature of original images. The VAE output images were the most blurry because of the nature of auto-encoder which first compress images then it tries to reconstruct them. The most realistic pictures were generated by StyleGAN3 network projected and implemented by *Nvidia* - it's FID score did not decay to the score showed in their paper [2] because of lack of computational power and hyperparameters adjustments but decreased to our best score equal 33.

References

[1] *Implementation of StyleGAN3 network*. URL: https://github.com/NVlabs/stylegan3.

[2] Tero Karras et al. "Alias-Free Generative Adversarial Networks". In: *Proc. NeurIPS*. 2021.

[3] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2017. DOI: 10.48550/ARXIV.1710.10196. URL: https://arxiv.org/abs/1710.10196.

[4] *PyTorch*. URL: https://pytorch.org.

[5] *PyTorch DCGAN tutorial*. URL: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html.

[6] *PyTorch VAE tutorial*. URL: https://github.com/Jackson-Kang/Pytorch-VAE-tutorial/blob/master/01_Variational_AutoEncoder.ipynb.

[7] *pytorch-fid*. URL: https://github.com/mseitzer/pytorch-fid.

[8] *Sampled Bedrooms dataset*. URL: https://www.kaggle.com/datasets/jhoward/lsun_bedroom.