Deep Learning: Project 3 Report

Maja Andrzejczuk, Maciej Orsłowski

Contents

1 Research Problem

Our research revolved around the topic of image generation. The task was to train neural networks on the LSUN Bedroom dataset [Howard, 2018]. The dataset contains 307495 images (4.8GB) of different sizes, each being a photo in some way related to bedrooms. The goal was to train a network which would generate bedroom images based on this dataset.

In the experiments, we test different architectures, namely Generative Adversarial Network (GAN) and Denoising Diffusion Probabilistic Model (DDPM). We train them using different sets of hyperparameters to find the most optimal training configuration, which would enable us to generate satisfactory images. We test the impact of hyperparameters such as learning rate, batch size, L2 regularisation and momentum. We evaluate the generated images qualitatively, checking if they resemble real bedrooms after every epoch. We also pick the best model based on the Fréchet Inception Distance, which is a state-of-the-art metric commonly used to evaluate the quality of model-generated images.

2 Instruction of the application

Our application is a Python Notebook (.ipynb format). It requires Python 3.10 and the following packages:

- `numpy` (1.26.4),
- `torch` (2.2.1),
- `torchvision` (0.17.1),
- `torcheval` (0.0.7),
- `diffusers` (0.27.2),
- `datasets` (2.19.1),
- `fastai` (2.7.15),
- `pytorch-fid` (0.3.0),
- `tqdm`.

The training is configured to utilise CUDA if a hardware that it runs on supports it, which can significantly shorten training times. The application automatically checks whether PyTorch can use CUDA and sets the device accordingly.

Before running the application, one has to keep in mind that training times are very long, even when using CUDA. Some of the training code can run for many hours, with the entire notebook taking days to complete on an average hardware. To alleviate this, after each epoch training results are saved. These results can then be loaded, which means a user does not have to run everything in the notebook at once to replicate our experiments. The saved architectures for DDPM can be loaded by calling `UNet2DModel.from_pretrained(...)` function, with the path to the folder with saved architectures provided as an argument. In the case of GAN, our custom `load_gan(...)` function, defined in the notebook, can be used, also taking a path to saved models as an argument. The training functions for both networks are also specifically designed to allow resuming of training using `start_epoch` parameter.

3 Theoretical Introduction

3.1 Denoising Diffusion Probabilistic Model

Denoising Diffusion Probabilistic Model (DDPM) is a generative model, which learns to generate images by a process called denoising. During the training process, for each image a noise is added, and then the model tries to remove that noise to obtain the original image. After it is trained, to generate an image, one needs to provide a latent noise matrix, which is a randomly generated matrix. The model then denoises this matrix to generate a new, meaningful image [Ho et al., 2020].

The process of adding a noise to an image is called a forward process, while the process of denoising is called a backward process. Both are divided into steps, the number and configuration of which can be customised. Let us denote T as the number of steps, and x_t as an image at a

step t, where x_0 is an original image and x_T is a final noise image. A step in the forward process follows the following Gaussian distribution:

$$q(x_t, x_{t-1}) = N\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right)$$

where variances $\beta_1, ..., \beta_T$ follow a schedule. The whole forward process is defined as follows:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t, x_{t-1})$$

The backward process can be described analogously with the following formulae:

$$p_\theta(x_{t-1}|x_t) = N\Big(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\Big)$$

$$p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}, x_t)$$

where $p(x_T)$ follows a standard multivariate normal distribution, and θ are the parameters which the model learns. Finally, a diffusion model can be stated as:

$$p_\theta(x_0) = \int p(x_{0:T}) dx_{1:T}$$

3.2 Generative Adversarial Networks

Generative Adversarial Network (GAN) is a type of approach for generating images which follows a game-like approach. GAN consists of two networks: Generator and Discriminator. The goal for the Generator is to generate the images from noise. The Discriminator then has to learn to distinguish if an image is real or if it is generated by the Generator. This functions like a minimax two-player game, where the Generator tries to generate images that are difficult to distinguish from the real ones, while the discriminator attempts to pick the actual real images out.

This process can be defined more formally. Let us denote $p_z(z)$ as the prior probability, x as the data (images), and p_G as the Generator's distribution over data. A Generator network can be defined as a mapping from z to the data space x, marked as $G(z; \theta_G)$. The discriminator network however would be defined as $D(x; \theta_0)$, and return a probability that x is a real image [Goodfellow et al., 2014]. The Discriminator is then trained to maximise the probability of assigning correct labels for real and fake data. On the other hand, the Generator is trained to maximise the probability of Discriminator assigning incorrect labels. This can be summarised with the formula:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} \left[\log\left(D(x)\right)\right] + \mathbb{E}_{z \sim p_z} \left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

3.3 Training collapse and mode collapse

Training collapse, and especially mode collapse, are phenomena that can occur when working with generative networks. The training collapse is a more common concept, and refers to situations when a model gets stuck and cannot progress in training. This can be caused by factors such as vanishing or exploding gradient or overfitting.

Mode collapse is a term related to Generative Adversarial Networks. It refers to a situation, where the images generated by the model are monotonous and do not represent the full variety of the training data. It is often caused by either generator or discriminator overpowering the other.

3.4 Fréchet Inception Distance

Fréchet Inception Distance (FID) is a metric used to assess the quality of images generated by models. As the name suggests, it measures a distance between real images and generated ones. Lower values of FID suggest that the generated images are more realistic.

The FID metric utilised an inception-v3 model, which extracts features from images. These features are numerical representations for each image. For these features, means and covariance matrices are computed. Finally, distances between these means and covariance matrices between real and generated images are calculated, and used to calculate the final value of FID [Lawton, 2023].

4 Experiments

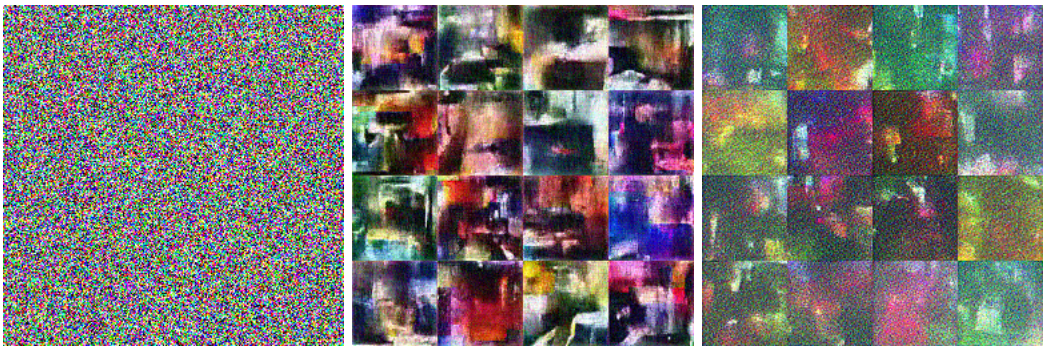
During the search for the optimal solution, we conducted various experiments on two implemented models: the Denoising Diffusion Probabilistic Model (DDPM) and the Generative Adversarial Network (GAN). Upon loading the data, we make sure that all the images are of the same size during training by resizing them to 64x64 pixels. We also apply normalisation to the images to scale them to an interval of $[0, 1]$. Throughout the experiments, we tested different batch sizes, learning rates, regularizations, and even whether changing the beta1 parameter from the Adam optimizer affects the image results. Additionally, we calculated the Fréchet Inception Distance (FID) score for the models we considered the best and conducted experiments with latent noise matrices.

4.1 Denoising Diffusion Probabilistic Model

The first model we tested was the Denoising Diffusion Probabilistic Model (DDPM). We utilised UNet2DModel as the base model, with the architecture identical to the one recommended in the `diffusers` package documentation [von Platen et al., 2022]. The initial training configuration was as follows:

- optimizer Adam with betas: (0.9, 0.999)
- no regularisation
- batch size: 16

We experimented with different learning rates, specifically 10^{-3}, 10^{-4}, and 10^{-5}, with the examples of generated images presented below.



(a) DDPM after 3 epochs, learning rate 10^{-3}

(b) DDPM after 10 epochs, learning rate 10^{-4}

(c) DDPM after 10 epochs, learning rate 10^{-5}

Figure 1: Comparison of generated images using different learning rate.

These models were trained using the Adam optimizer with a batch size of 16. Training with a learning rate of 10^{-3} was halted after 3 epochs due to the model's persistent production of noisy images without any improvement. The learning rate of 10^{-4} was used for 20 epochs, and 10^{-5} for 10 epochs, the latter being shorter due to the lack of visible improvements and the extended training time required. We determined that the best images were produced using a learning rate of 10^{-4}, thus, we proceeded with this learning rate for the next experiment, which involved testing regularization.

We tested regularization by setting the `weight_decay` parameter to 1×10^{-4}, 1×10^{-5}, and 1×10^{-6}. Unfortunately, regardless of the parameter chosen, the images generated with regularization were completely black. This could be due to over-regularization, causing the model's weights to shrink excessively and preventing it from learning meaningful patterns. Additionally, gradient vanishing might occur, leading to minimal weight updates and poor learning outcomes

Training this model was time-consuming (2 hours and 40 minutes per epoch) and the results were unsatisfactory. Therefore, we decided to allocate the remaining time to further training the GAN model, which showed better potential.

(a) DDPM after 5 epochs, regularisation 10^{-4} (b) DDPM after 5 epochs, regularisation 10^{-5} (c) DDPM after 5 epochs, regularisation 10^{-6}

Figure 2: Comparison of generated images using different regularisation.

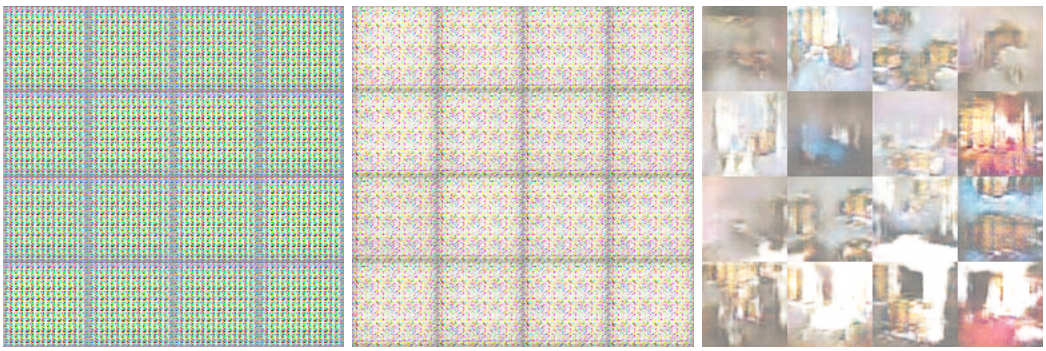
4.2 Generative Adversarial Network

The next model we implemented was the Generative Adversarial Network (GAN). The architecture for the generator comprised four blocks of transposed convolution - batch normalisation - ReLU activation, followed by another transposed convolution and a hyperbolic tangent activation. A transposed convolution layer works like the opposite of a convolution layer, meaning it can be used to convert features into images. The discriminator's architecture starts with a convolutional layer with leaky ReLU activation, then three blocks of convolutional layer - batch normalisation - leaky ReLU, and finally a convolutional layer with sigmoid activation. We used the random noise vectors of size 100 as this is a commonly used value.

The initial training configuration was as follows:

- optimizer Adam with betas: (0.9, 0.999)
- no regularisation
- batch size: 16

In the initial experiments, similar to the approach adopted in the DDPM experiments, we decided to test various learning rates. We experimented with different learning rates, specifically 10^{-3}, 10^{-4}, and 10^{-5}. The only and most promising generator/discriminator pair showed potential with a learning rate of 1×10^{-5}, so we used this parameter value in future training sessions.



(a) GAN after 10 epochs, learning rate 10^{-3} (b) GAN after 10 epochs, learning rate 10^{-4} (c) GAN after 10 epochs, learning rate 10^{-5}

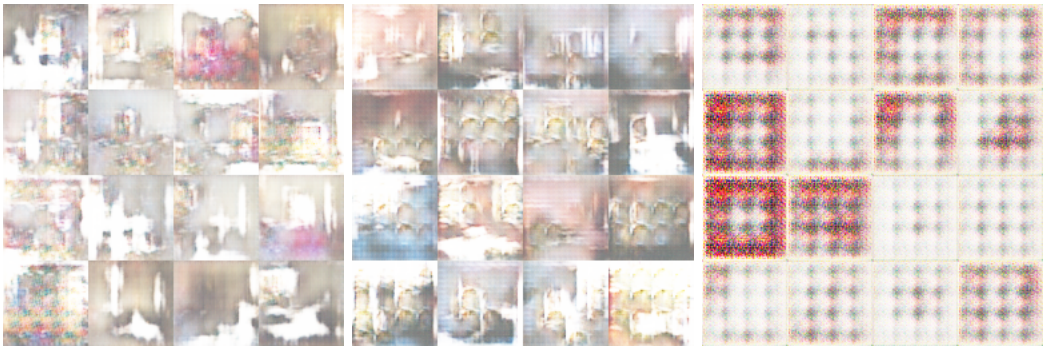
Figure 3: Comparison of generated images using different learning rate.

The next experiment involved utilizing regularization with the `weight_decay` parameter set to 1×10^{-4}, 1×10^{-5}, or 1×10^{-6}. However, as in the case of DDPM, it was also with the GAN model that the regularization caused the model to be unable to learn anything. As seen in 4, all images after a certain epoch become uniformly gray. This may be related to over-regularization, where the weight decay parameter excessively penalizes the weights, leading to the model losing its ability to learn meaningful patterns from the data. Therefore, we completely abandoned it in future tests.

(a) GAN after 5 epochs, regularisation 10^{-4} (b) GAN after 5 epochs, regularisation 10^{-5} (c) GAN after 5 epochs, regularisation 10^{-6}

Figure 4: Comparison of generated images using different regularisation.

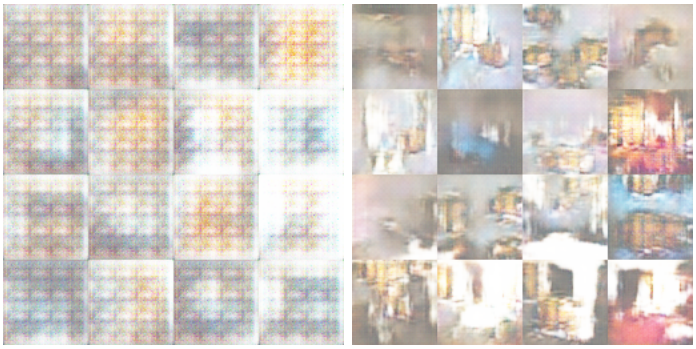
The next conducted test involved examining whether batch size has an impact on our model. We intended to test various batch sizes ranging from [4, 8, 16, 32, 64, 128, 256]. However, after training with batch sizes of 32 and 64, we noticed that the images did not improve during subsequent epochs, so we decided to refrain from testing even larger sizes. We had the best results with the initial batch size equal to 16, so we stayed with it for the last test.



(a) GAN after 20 epochs, batch size 4 (b) GAN after 20 epochs, batch size 8 (c) GAN after 10 epochs, batch size 32

Figure 5: Comparison of generated images using different batch size.

The last test in the GAN involved checking whether changing the default beta1 parameter from the Adam optimizer to a value of 0.5 would help in training. We chose this value because we hypothesized that a lower value might encourage smoother convergence. However, as it turned out, the default parameter achieved better results, so we decided to stick with it.



(a) GAN after 10 epochs, beta1 0.5 (b) GAN after 10 epochs, beta1 0.9

Figure 6: Comparison of generated images using different value of parameter beta1.

4.3 Final model

Due to the significantly simpler training process and better performance results, we have selected the Generative Adversarial Network model as the final architecture for our solution. From experiments, it turned out that the best results are achieved with initial parameters, and such we used for the final training of the model:

- optimizer Adam with betas: (0.9, 0.999)
- learning rate: 10^{-5}
- no regularisation
- batch size: 16

We conducted the model training in two stages. Initially, we trained the model using the above initial parameters for 70 epochs. Sample images generated by the model were saved after each training epoch, and we evaluated the model's capabilities based on these images. We selected the best set of generated images, which, in our opinion, was from epoch 41. Subsequently, we further trained the model starting from the parameters of the model at epoch 41 with reduced learning rates of 10^{-6} and 10^{-7}. We were most pleased with the results of the models:

- generator41 - model trained by 41 epochs on initial parameters (Figure 7a);
- generator43_pt3 - model trained by 41 epochs on initial parameters + 2 epochs with learning rate 10^{-7} (Figure 7b);
- generator45_pt3 - model trained by 41 epochs on initial parameters + 4 epochs with learning rate 10^{-7} (Figure 7c);
- generator47_pt2 - model trained by 41 epochs on initial parameters + 6 epochs with learning rate 10^{-6} (Figure 7d);

which is why we decided to evaluate them using the FID score to pick the final best model.

(a) generator41 (b) generator43_pt3

(c) generator45_pt3 (d) generator47_pt2

Figure 7: Comparison of generated images using different generators - final comparison.

The analysis revealed that the model trained for 45 epochs achieved the best results according to this metric (see Table 1). Comparing these images through visual inspection, we agree with this assessment, as we observe strong features in the images that indicate they represent a room.

Generator	FID
generator41	213.546506
generator43_pt3	210.999851
generator45_pt3	202.652515
generator47_pt2	220.804244

Table 1: FID scores for different generators

4.4 Discussing training collapse and mode collapse

During our research, on several occasions we have encountered the phenomena of training collapse and mode collapse. Most notably, using any regularisation led to the generator producing only gray images (and black in the case of DDPM). However, some of the other hyperparameters, if set incorrectly, could also lead to blurry, noisy and monotonous images. Summarising, based on our research, we can state that in order to avoid training collapse and mode collapse when using GANs, one should:

- avoid regularisation,
- avoid using large learning rate values (anything larger than 10^{-5} can cause issues),
- not use batch sizes larger than 16,
- stick to default beta parameter values for Adam optimiser.

4.5 Experimenting with latent noise matrices

We conducted a detailed analysis of our model by linearly interpolating between two selected latent noise matrices, creating a sequence of ten matrices. Using the best-performing generator model, we generated images from each of these matrices, denormalized them, and converted them into a displayable format. The resulting ten images, including the initial two and the eight interpolated ones, were arranged in a single row to visually illustrate the transformation. This approach effectively demonstrated the model's ability to transition smoothly between different latent spaces, showcasing its robustness and versatility.

Figure 8: Illustration of the generated images in experiment with latent noise matrices.

Using a strategic interpolation method, we visually depicted the transition between latent spaces by blending two selected noise matrices, showcasing the model's ability to navigate complex data distributions effectively. While analyzing the generated images, we noticed a smooth transition between each pair of consecutive images in the sequence. This smoothness suggests a continuous and well-defined latent space representation within the model. It appears that the model effectively interpolates between different latent noise vectors, producing meaningful variations in the generated images as we move along the sequence. This observation underscores the model's ability to navigate the latent space smoothly, indicating its capacity to capture and interpolate meaningful representations of the underlying data distribution.

5 Conclusions

This project helped us gain understanding and experience in dealing with generative networks. We were able to compare two popular architectures - DDPM and GAN. We discovered that despite DDPM being a more complex architecture, its long training times prevented us from optimising it to reach satisfactory results. This meant that it was GAN which was a more suitable architecture for this task. Its relatively short training times (25 minutes per epoch) made it possible to search through various sets of hyperparameters, and then to fully train and experiment with the final model. Regarding the hyperparameters, we also discovered a few interesting facts. The most notable one was that adding regularisation to Adam optimiser in the task of image generation has

a strikingly negative impact on the model outputs. We also eventually found out that in most cases the default hyperparameters worked the best.

Moving on to the results, admittedly, the generated images for the final model are blurry and not fully realistic. However, their resemblance to the pictures of interiors was close enough to make them, in our opinion, satisfactory. We were able to prove this in an extra experiment - as a mean of evaluation, we asked other people (who were not involved in the project) what they see in our generated images. Most of them pointed out that the images resembled house interiors, some of them even stating that they see beds in the images. This reassured us that the models indeed worked and converged to identifiable images.

References

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

Jeremy Howard. LSUN bedroom scene 20 URL https://www.kaggle.com/datasets/jhoward/lsun_bedroom.

George Lawton. Fréchet inception distance (FID), 6 2023. URL https://www.techtarget.com/searchenterpriseai/definition/Frechet-inception-distance-FID#:~:text=Fr%C3%A9chet%20inception%20distance%20(FID)%20is,like%20real%20images%20of%20people.

Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers, 2022.