Diffusion - report

Piotr Bielecki, Tomasz Krupiński

June 2024

Abstract

This document acts as a report of the research and experimentation done for the project regarding image generation using diffusion models. All codes are available at this link: link to github repository

1 Theoretical Introduction

1.1 Denoising Diffusion Probabilistic Models

We follow the theoretical introduction of the model and derivation of loss function, as shown by Nichol & Dhariwal (2021):

> *Given a data distribution $x_0 \sim q(x_0)$, we define a forward noising process q which produces latents x_1 through x_T by adding Gaussian noise at time t with variance $\beta_t \in (0, 1)$ as follows:*

$$q(x_1, ..., x_T|x_0) := \Pi_{t=1}^{T} q(x_t|x_{t-1}) \tag{1}$$

$$q(x_t|x_{t-1}) := \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \tag{2}$$

> *Given sufficiently large T and a well behaved schedule of β_t, the latent x_T is **nearly** an isotropic Gaussian distribution.*

This means, that this process takes us from the data distribution to a known distribution (Gaussian). This is provable for $T \to \infty$, but we have a finite process, so we assume T is large enough, for this to work. Equation 2 means, that the distribution of the next sample (x_t) is going to be a normal distribution:

- centered at $\sqrt{1-\beta_t}x_{t-1}$ - centered at last sample, but downscaled by $\sqrt{1-\beta_t}$
- with variance equal to $\beta_t\mathbf{I}$ - here, the assumption is we use noise with a diagonal covariance matrix

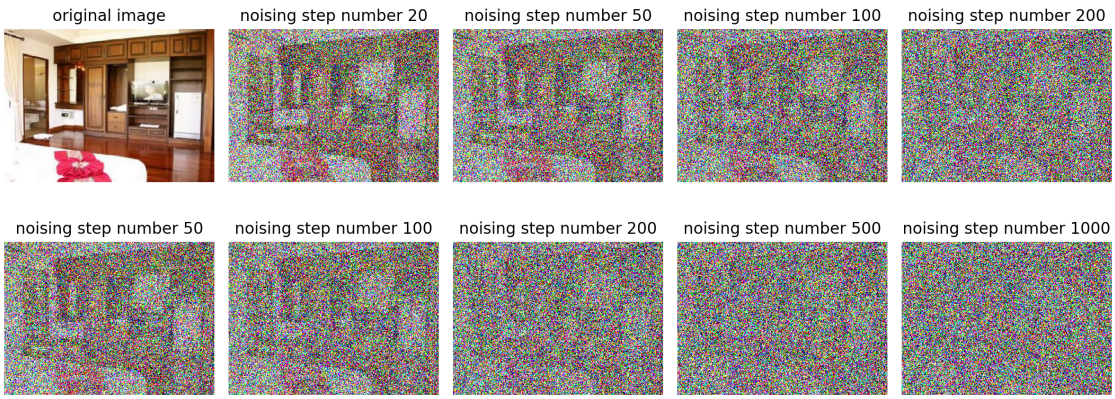

Figure 1: example of the noising process from equation 2

If we could somehow define a reverse process, such that given an image with some noise, the process would be able to tell what image that came from.

> *Thus, if we know the exact reverse distribution $q(x_{t-1}|x_t)$, we can sample $x_T \sim \mathcal{N}(0, \mathbf{I})$ and run the process in reverse to get a sample from $q(x_0)$. However, since $q(x_{t-1}|x_t)$ depends on the entire data distribution(...)*

We cannot simply have the data distribution, because the problem would not have existed in the first place. We would just be sampling from that distribution, therefore:

> (...)we approximate it using a neural network as follows:

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{3}$$

As neural networks are universal function approximators, we could have a neural network, that takes as an input the noised version of the image, and gives as an output - a distribution over images, that could have produced this noised one. The network would produce a mean, and a covariance matrix given the image.
The network is supposed to tell, given a noisy image, what's the gaussian distribution of images where that probably came from. The fact that its a Gaussian distribution is a strong assumption we can make perhaps only because of the "very small steps" in the noising process.

The combination of q and p is a variational auto-encoder(Kingma Welling, 2013), and we can write the variational lower bound (VLB) as follows:

$$L_{vlb} := \sum_{i=0}^{T} L_i \tag{4}$$

$$L_0 := -\log p_\theta(x_0|x_1) \tag{5}$$

$$L_{t-1} := D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)) \tag{6}$$

$$L_T := D_{KL}(q(x_T|x_0) \parallel p(x_T)) \tag{7}$$

Equation 6 essentially means, that we want $q(x_{t-1}|x_t, x_0)$ (the distribution that we want to model) and $p_\theta(x_{t-1}|x_t)$ (the reverse process that the neural network does) to be close to be close one another (D_{KL} - Kullback-Leiber divergence essentially measures that distance in terms of probability distributions).

Aside from L_0, each term of Equation 4 is a KL divergence between two Gaussian distributions, and can thus be evaluated in closed form.
To evaluate L_0 for images, we assume that each color component is divided into 256 bins, and we compute the probability of $p_\theta(x_0|x_1)$ landing in the correct bin (which is tractable using the CDF of the Gaussian distribution).
Also note that while L_T does not depend on θ, it will be close to zero if the forward noising process adequately destroys the data distribution so that $q(x_T|x0) \approx \mathcal{N}(0, \mathbf{I})$

The question now is - how to calculate $q(x_{t-1}|x_t, x_0)$.

As noted in (Ho et al., 2020), the noising process defined in Equation 2 allows us to sample an arbitrary step of the noised latents directly conditioned on the input x_0. With $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=0}^{t} \alpha_s$, we can write the marginal distribution

$$q(x_t|x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)\mathbf{I}) \tag{8}$$

Using Bayes theorem, one can calculate the posterior $q(x_{t-1}|x_t, x_0)$ in terms of $\widetilde{\beta}_t$ and $\widetilde{\mu}_t(x_t, x_0)$ which are defined as follows:

$$\widetilde{\beta}_t := \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t \tag{9}$$

$$\widetilde{\mu}_t(x_t, x_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}x_t \tag{10}$$

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(\widetilde{\mu}_t(x_t, x_0), \widetilde{\beta}_t\mathbf{I}) \tag{11}$$

There are many different ways to parameterize $\mu_\theta(x_t, t)$. The most obvious option is to predict $\mu_\theta(x_t, t)$ directly with a neural network. Alternatively, the network could predict x_0, and this output could then be fed through equation 10 to produce $\mu_\theta(x_t, t)$. The network could also predict the noise ϵ added to x_0, and this noise could be used to predict x_0 via:

$$x_0 = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon\right) \tag{12}$$

1.2 GAN - General Adversarial Network

Generative Adversarial Networks (GANs for short) are a class of machine learning frameworks consisting of two neural networks, the Generator and the Discriminator, that compete against each other.
The Generator is designed to produce data samples similar to the real data by transforming a noise vector z into a data point $\widetilde{x}$, while the Discriminator's role is to classify data as real or fake. It processes an input data sample x or $\widetilde{x}$ and outputs a probability.
The standard GAN loss function, known as the min-max loss function, was described by Goodfellow et al. (2014):

$$\mathbf{E}_x[\log D(x)] + \mathbf{E}_z[\log(1 - D(G(z)))] \tag{13}$$

The generator tries to minimize this function, while the discriminator tries to maximize it.

> *In practice, it saturates for the generator, meaning that the generator quite frequently stops training if it doesn't catch up with the discriminator.*

> *The Standard GAN loss function can further be categorized into two parts: Discriminator loss and Generator loss.*

. Equations 14 and 15 show stochastic gradients for updating the discriminator and generator respectively.

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [logD(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)})))] \tag{14}$$

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\mathbf{z}^{(i)}))) \tag{15}$$

In equation 14 $\log(D(x))$ refers to the probability that the generator is rightly classifying the real image. Maximizing $\log(1 - D(G(z)))$ means maximizing the capability of the discriminator to recognize fake images correctly.

1.3 Mode collapse

Mode collapse happens when the generator focuses on producing a limited set of data patterns that deceive the discriminator. It becomes fixated on a few dominant modes in the training data and fails to capture the full diversity of the data distribution.

1.3.1 Mitigation strategies

- **WGAN** Wasserstein Generative Adversarial Network - utilizes Wasserstein distance instead of cross-entropy. WGANs provide a more stable and informative training signal, allowing for smoother learning and reduced mode collapse. The gradient of the Wasserstein distance enables better convergence, making WGANs effective in handling mode collapse and generating more diverse and realistic samples.
- **Unrolled GAN** - usage of a generator loss function that incorporates not only the current discriminator's classifications, but also the outputs of future discriminator versions. So the generator can't over-optimize for a single discriminator.

2 Experiments

2.1 Diffusion

We tried using a DDPM model with U-net architecture, like most of the implementations seen in the literature. U-net's strenghts for this task may be:

1. they provide representations of different granularity - useful for denoising
2. U-nets have decreases in layer sizes - This forces the encoding to discard some information which should improve denoising properties. The noise cannot be encoded into lower dimensions.

.

We tried using huggingface's diffusion model from their tutorial. Unfortunately, due to computation time we could only make it go on for three epochs. In figures 5 and 6 our resulting images after 1 and 3 epochs, due to the computational cost, even after reducing the dataset to 5% of the original (It took 6-7 hours per epoch + the time to generate images).

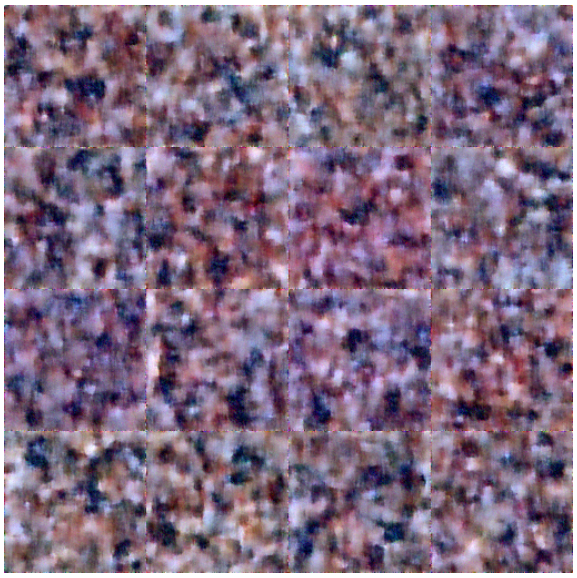

Figure 2: 4x4 matrix of images generated after first epoch

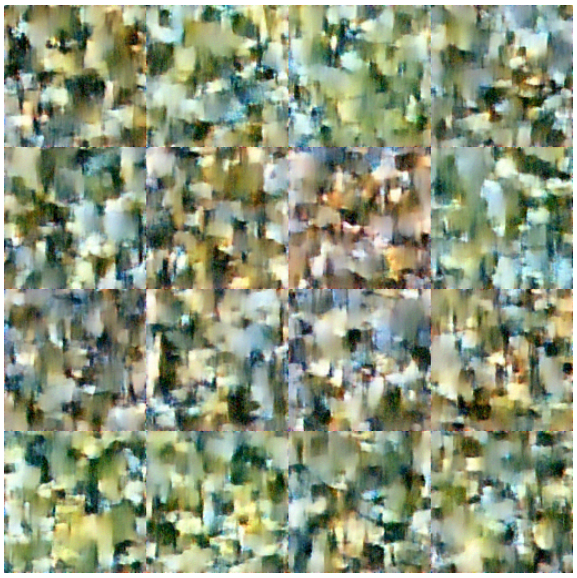

Figure 3: 4x4 matrix of images generated after three epochs

And in figure 4 the noise predicted before training, and after the three epochs: We can notice,

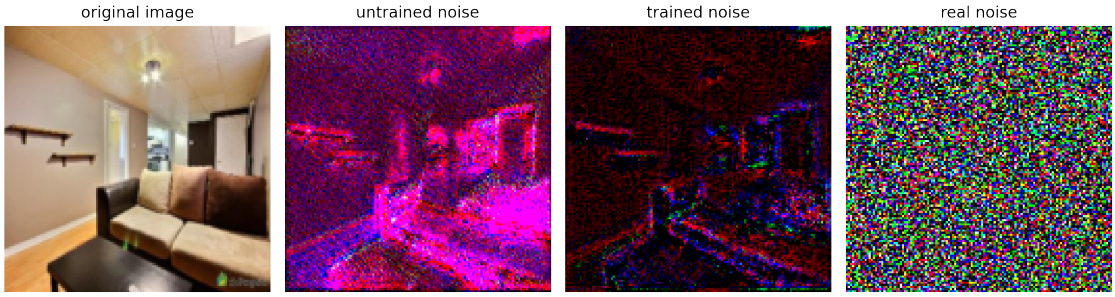

Figure 4: noise predicted by the model along with real noise

the trained model's noise got a lot darker (its values decreased two-fold). It started visually resembling noise a bit more, perhaps.

2.2 GAN

To experiment on GAN, we have used this repository. It provides an already implemented architecture, with the training, evaluation, and data augmentation codes.

The architecture is comprised of the generator and discriminator model. Both of them mainly use convolutional layers. Also, they change the data dimension from 2d into 4d (one dimension is the batch size), in the case of generator and from 4d into 2d in case of discriminator.

Generator	Parameters	Buffers	Output shape	Datatype
mapping.fc0	262656	-	[32, 512]	float32
mapping.fc1	262656	-	[32, 512]	float32
mapping	-	512	[32, 10, 512]	float32
synthesis.b4.conv1	2622465	32	[32, 512, 4, 4]	float32
synthesis.b4.torgb	264195	-	[32, 3, 4, 4]	float32
synthesis.b4:0	8192	16	[32, 512, 4, 4]	float32
synthesis.b4:1	-	-	[32, 512, 4, 4]	float32
synthesis.b8.conv0	2622465	80	[32, 512, 8, 8]	float16
synthesis.b8.conv1	2622465	80	[32, 512, 8, 8]	float16
synthesis.b8.torgb	264195	-	[32, 3, 8, 8]	float16
synthesis.b8:0	-	16	[32, 512, 8, 8]	float16
synthesis.b8:1	-	-	[32, 512, 8, 8]	float32
synthesis.b16.conv0	2622465	272	[32, 512, 16, 16]	float16
synthesis.b16.conv1	2622465	272	[32, 512, 16, 16]	float16
synthesis.b16.torgb	264195	-	[32, 3, 16, 16]	float16
synthesis.b16:0	-	16	[32, 512, 16, 16]	float16
synthesis.b16:1	-	-	[32, 512, 16, 16]	float32
synthesis.b32.conv0	2622465	1040	[32, 512, 32, 32]	float16
synthesis.b32.conv1	2622465	1040	[32, 512, 32, 32]	float16
synthesis.b32.torgb	264195	-	[32, 3, 32, 32]	float16
synthesis.b32:0	-	16	[32, 512, 32, 32]	float16
synthesis.b32:1	-	-	[32, 512, 32, 32]	float32
synthesis.b64.conv0	1442561	4112	[32, 256, 64, 64]	float16
synthesis.b64.conv1	721409	4112	[32, 256, 64, 64]	float16
synthesis.b64.torgb	132099	-	[32, 3, 64, 64]	float16
synthesis.b64:0	-	16	[32, 256, 64, 64]	float16
synthesis.b64:1	-	-	[32, 256, 64, 64]	float32
---	---	---	---	---
Total	22243608	11632	-	-

Figure 5: Generator architecture

Discriminator	Parameters	Buffers	Output shape	Datatype
b64.fromrgb	1024	16	[32, 256, 64, 64]	float16
b64.skip	131072	16	[32, 512, 32, 32]	float16
b64.conv0	590080	16	[32, 256, 64, 64]	float16
b64.conv1	1180160	16	[32, 512, 32, 32]	float16
b64	-	16	[32, 512, 32, 32]	float16
b32.skip	262144	16	[32, 512, 16, 16]	float16
b32.conv0	2359808	16	[32, 512, 32, 32]	float16
b32.conv1	2359808	16	[32, 512, 16, 16]	float16
b32	-	16	[32, 512, 16, 16]	float16
b16.skip	262144	16	[32, 512, 8, 8]	float16
b16.conv0	2359808	16	[32, 512, 16, 16]	float16
b16.conv1	2359808	16	[32, 512, 8, 8]	float16
b16	-	16	[32, 512, 8, 8]	float16
b8.skip	262144	16	[32, 512, 4, 4]	float16
b8.conv0	2359808	16	[32, 512, 8, 8]	float16
b8.conv1	2359808	16	[32, 512, 4, 4]	float16
b8	-	16	[32, 512, 4, 4]	float16
b4.mbstd	-	-	[32, 513, 4, 4]	float32
b4.conv	2364416	16	[32, 512, 4, 4]	float32
b4.fc	4194816	-	[32, 512]	float32
b4.out	513	-	[32, 1]	float32
---	---	---	---	---
Total	23407361	288	-	-

Figure 6: Discriminator architecture

The data augmentation part, is done by a pipeline, that flips, changes colors, changes the hue, tilts and does a lot of different image operations, to reduce the impact of overfitting. It was necessary to use the pipeline, to be able to train the network.

Figure 7: example of data augmentation

To train this network, we used the Google Colab environment, on a small subset of the data (about 1000 images from the whole dataset, but then those images were augmented with a pipeline). This allowed us to use stronger graphic cards to train the network. There were two types of epochs. The evaluation was done every 10 epochs and additionally to training it also computed the FID. The training ones, only trained the model.

Epoch type	elapsed time
evaluation	10 min
training	2 min

The Frechet Inception Distance, or FID for short, is a metric for evaluating the quality of generated images and specifically developed to evaluate the performance of generative adversarial networks. It quantifies the realism and diversity of artificially generated images. The lower, the better. Below in figure 8, FID vs epoch curve, for our GAN model.

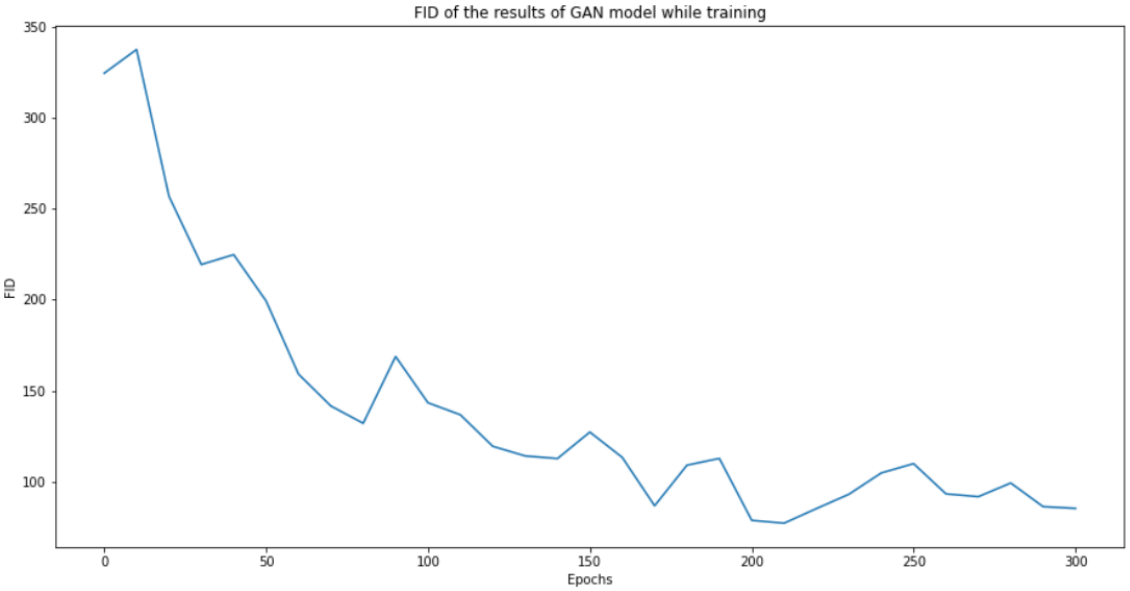

Figure 8: The FID value depending on the training epoch

After the 300 epochs, we have stopped the training. The final FID score achieved was equal to 74, from the initial value of around 300. Even though the final value is still much bigger than the perfect score of 0, the sample images below show that the results were acceptable. 9

Figure 9: example of generated images after 300 epochs

2.3 Interpolation of latent noise for different generated images

Because our diffusion model did not achieve satisfactory results, we only present the GAN's interpolation of the images' latent representations.

2.3.1 Discussion

By interpolating between latent noise vectors, We are essentially probing the latent space of the model. This task provides insights into the model's generative capabilities and the structure of its latent space. The smoothness and coherence of the generated images should reveal the model's capacity to learn continuous transformations and generate realistic intermediate representations.

The transition, displayed in figure 10 seems smooth, but perhaps not strong enough. This may be due to the similarity of the two bedrooms generated (and consequently, perhaps from the small amount of images used for training). The smoother the transition would be, the better the continuity in the model's latent space.

Figure 10: example of the linear interpolation, between top left and bottom right images

3 Conclusions

We have tried to compare two different architectures, the DDPM and the GAN. However, it turned out that the DDPM is too complex in terms of training time and we were not able to achieve satisfactory results. Therefore, the requirements of the project were only fulfiled by the GAN model. This model actually managed to learn to generate images resembling bedrooms. Also, the minimalization of the FID curve over the epochs shows that this model somewhat converged. We were also able to compute the linear interpolation of the latent spaces of this model.

References

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014), 'Generative adversarial networks'.

Nichol, A. Q. & Dhariwal, P. (2021), Improved denoising diffusion probabilistic models, *in* M. Meila & T. Zhang, eds, 'Proceedings of the 38th International Conference on Machine Learning', Vol. 139 of *Proceedings of Machine Learning Research*, PMLR, pp. 8162–8171.
URL: *https://proceedings.mlr.press/v139/nichol21a.html*