Master Thesis : Diffusion models : seek of information and structure in latent space

Auteur : Maziane, Yassine
Promoteur(s) : Louppe, Gilles
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée
Année académique : 2022-2023
URI/URL : http://hdl.handle.net/2268.2/18351

University of Liège
School of Engineering and Computer Science

Diffusion models : seek of information and structure in latent space

A dissertation submitted in partial fulfillment of the requirements for the degree of
Master of Science in Data Science and Engineering

Author
Yassine Maziane

Advisor
Professor Gilles Louppe

Academic year 2022-2023

Acknowledgements

I am deeply grateful to **Allah**, the Most Merciful and All-Knowing, for granting me the ability to pursue studies and complete this thesis. I am forever thankful for His boundless mercy that has encompassed every aspect of my accomplishments.

I wish to convey my heartfelt gratitude to Professor Gilles Louppe for his invaluable teachings and unwavering guidance throughout the course of this work. Your ideas and mentorship have been instrumental in shaping the outcome. Additionally, I would like to express my gratitude to all the professors and assistants at the University of Liege who have provided me with the opportunity to receive this amazing education. Their guidance and expertise have been invaluable and have greatly contributed to the successful completion of this thesis.

Lastly, I would like to extend my gratitude to my parents for providing me with an ideal study environment and for their support during my academic journey. Also, I wish to express my appreciation to my friends for the wonderful moments we have shared and the mutual assistance we have provided to one another. Your presence has not only enriched my academic experience but has also contributed significantly to my personal growth. I am truly thankful for the camaraderie and support that have made this journey both memorable and rewarding.

Abstract

Diffusion models, a prominent subset of generative modeling techniques, have gained substantial attention for their ability to generate high-quality data samples from complex distributions. They work by gradually adding noise to data and learning to remove this noise. They then generate samples by undergoing a series of denoising steps from gaussian noise. This thesis aims at understanding the feature choices a diffusion model makes when generating a sample. We know that the choices must be made between the gaussian noise sampling and the last denoising step however we don't know when exactly. The prevailing consensus in the diffusion model field suggests that the choices are made at the very beginning of the diffusion but there is no reliable evidence for that.

The central research question thus revolves around the timing and specifics of these choices the models make. Specifically, this study investigates whether these choices are instantaneous, reversible, influenced by certain factors etc ...

Through a series of carefully designed experiments, we explore various facets of diffusion models. First, we examine the preservation of features along the forward-backward chain to uncover whether the model aims to replicate initial samples during diffusion. Subsequently, we devote considerable effort to comprehending the destruction of information in the forward process. This exploration stems from the realization that the generative capacity of diffusion models hinges on the analysis of their forward mechanisms. We then investigate the actual feature choices made in the backward chain, along with the timings and contributing factors.

Our findings reveal that the destruction of feature information in the forward process and the creation of features by choice in the backward process are primarily determined by the noise variance schedule. Notably, under the linear schedule, the diffusion chain can be divided into distinct regions. The initial phase, characterized by high information retention, contributes mainly to sample aesthetics without altering essential features. The subsequent region, symmetrically centered in the diffusion process, stands out as the critical juncture for feature choices. Surprisingly, we observed that these choices are not instantaneous but evolve over several denoising steps. This realization bears significance in addressing our research question. The third region on the other is much less interesting. Samples tend to have lost all their information once they reach it and they essentially transition to normal samples in that region.

The partitioning of regions within the diffusion depends not only on the time at which features tend to disappear in the forward process but also on the rate at which they

vanish, as not all features are equally robust against Gaussian noise. While our results provide valuable insights, we acknowledge that further research could fortify our claims and explore additional ideas. This thesis contributes to the discourse on diffusion models by shedding light on the temporal dynamics of feature choices, emphasizing the interplay between noise variance and generative capabilities, and opening avenues for future investigations.

Contents

Chapter 1

Introduction

1.1 Context

Diffusion models are members of the large family defined by generative modelling, the latter includes both VAE's, GAN's but also density estimation techniques and many others. They were initially defined by [1] finding their roots in non-equilibrium thermodynamics but they emerged in several forms and in various frameworks. [2] defined noise-conditioned score network to approximate the score of noisy data, [3] defined denoising diffusion probabilistic models which simply learn a reverse time-dependent distribution of the noise over images and [4] introduced it under the SDE framework.
Essentially all these methods fall in the domain of diffusion models but they are simply perceived from a different angle.
The purpose of diffusion models is to model and generate data samples from potentially complex distributions. They are primarily used for tasks such as image generation, image inpainting but they can also be used for more sophisticated tasks such as representation learning as they learn a usually meaningful representation of the data distribution during the training process. Indeed, the model must capture important features and structures of the data to be able to generate samples that are highly likely to be drawn from the data distribution
Diffusion models' main idea is to gradually noise to a data sample up until it contains no information regarding the distribution it was sampled from, one then creates a sample by feeding this pure noise to a network that was trained to remove a certain level of noise from an image. At each forward pass the image gets less and less noisy such that after a certain number of forward passes the image no longer contains noise and has a large likelihood of belonging to the data distribution the network was trained on. The generative aspect of diffusion models arises from the use of the denoising network, instead of simply returning the denoised output, one can return a sample normally close to it, that is we define a normal around the denoised output and return a realization of that normal distribution.

The observed brownian paths in figure 2.24 exemplify this process. To summarize, the diffusion model's sampling entails drawing Gaussian noise, iteratively denoising with a network, and stochastically selecting the next output from a normal distribution centered around the denoised output, with variance dictated by the noise schedule.

As we'll need at least a shallow understanding of diffusion models to get the point of the

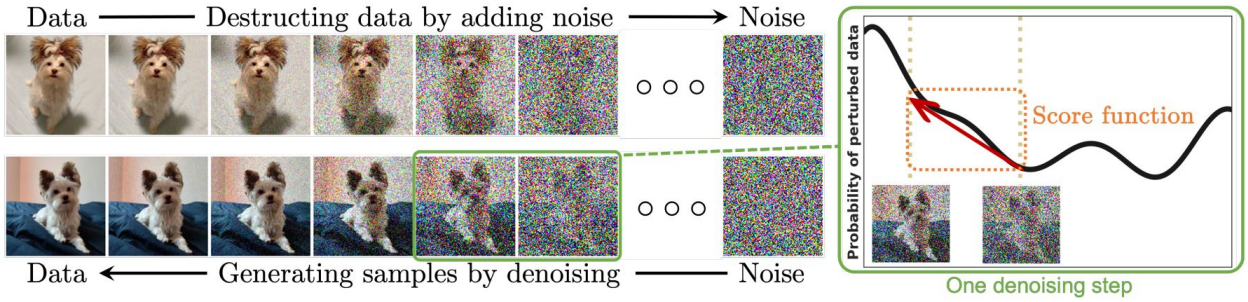


Figure 1.1. Fig. 2. Diffusion models smoothly perturb data by adding noise, then reverse this process to generate new data from noise. Each denoising step in the reverse process typically requires estimating the score function (see the illustrative figure on the right), which is a gradient pointing to the directions of data with higher likelihood and less noise.

research question, we might as well describe their main components in a few lines now. The class of diffusion models generally consists in these several components :

1. Noise schedule : A noise schedule defines the levels of noise that gradually increase in the forward process, i.e, the process in which we add noise. Practically, the added noise of level t follows a multivariate normal with zero mean and a covariance matrix set to a multiple of the identity, the multiple is given by the noise schedule evaluated at t. The noise schedule is designed such that for large t, the distribution of noisy images converges to a standard Gaussian.

2. Forward process : Given a sample x_0 from a data distribution $q(x_0)$, a forward process generates a sequence of random variables $x_1, x_2, x_3, ..., x_T$ by applying a transition kernel $q(x_t|x_{t-1})$. This transition is usually a Gaussian with zero mean and augmenting variance such that at every transition, the samples get more and more noisy.

3. Backward process : The backward process aims at learning the reverse of the forward process, i.e, it aims at removing a certain level of noise from a sample. It is defined by $p_\theta(x_{t-1}|x_t)$ and aims at slowly recovering the data distribution. Where the forward process slowly injects noise to destroy all information and structure of a sample, the backward process does exactly the opposite.

4. Training objective : Ideally, we would like the backward process to approximate as well as possible the reverse the forward process, this would require minimizing the conditional $KL(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$ over all samples from the data distribution. This loss can actually be shown equivalent to minimizing the predicted noise.

5. Sampling procedure : Assuming a trained denoising model p_θ is available, one can generate samples by drawing a standard Gaussian sample and then iteratively denoise it for T steps

A big picture of diffusion models may be looked at on fig 1.1. The forward process is applied in the first row while the backward process is applied on the second row to form data from noise. As we can see on the right, taking a step, in the image space, in the denoising direction is similar to taking a step towards regions of higher likelihood of the

data. Indeed, the less noisy image is more likely to belong to the data distribution, we'll show the previous result in the background section.

Diffusion models are particularly trendy because of their sample quality and ease of training, they are the new state-of-the-art family of deep generative models. They have shown their strength in a wide variety of domains such as computer vision, natural language processing, time series forecasting, representation learning and biomedical imaging.

1.2 Research question

From experiments, we know diffusion models are good at generating images but as these images have some features that the model must have learnt, it had, at a point in the diffusion chain, to choose which features this image will have. That is exactly what our research questions tries to understand : **If some features choices about the generated sample are made, when are they actually made in the diffusion chain ?**

Suppose you trained a diffusion model on the celebA dataset, which comprises images of individuals with varying characteristics such as gender, age, hair color, and more. In order to generate high-quality images, the sampling process of the diffusion model must determine specific attributes, such as gender, age, and other characteristics. However, the precise timing of when these attribute choices are made raises questions.

Are they made when sampling the gaussian noise and thus the whole diffusion chain is simply about making these features appear at 0? Are they made close to 0 such that the whole diffusion is rather about making a generic samples while its features will be decided at the very end ?
Maybe the choice happens several times, first the diffusion model "intends" to generate a man but as the brownian path slowly takes form, it realizes it'll be much easier to generate a woman.

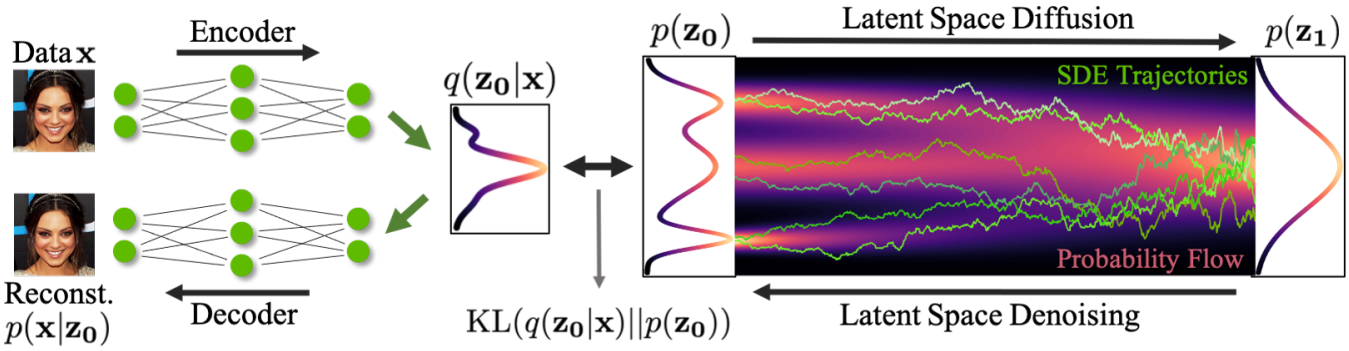


Figure 1.2. Latent diffusion models and brownian paths defined by realizations of the denoised output

On one hand, if such choices were already made at T and fixed through all the diffusion chain, then the whole information regarding the features of the sample could be picked up from the gaussian noise. Quite extremely, we could even suggest there exists a partitioning of the image space where each pixel is a float in [0,1] such that $[0,1]^{64\times64} = A \bigcup B$ with $A \bigcap B = \emptyset$ where if the gaussian noise was in A, the final generated image would be a man but if the gaussian noise was in B, the generated image would be a woman.

On the other hand, if choices were made close to 0 in the diffusion chain, what is the purpose of backward-diffusing a sample several times at large cost if it contains no information whatsoever ? Is it purely for aesthetics ?

So far we only considered diffusion models operating in the image space however some of them operate in a latent space, the latter is simply defined by means of a VAE, this idea was introduced by [5]. Instead of noising images and learning a model to denoise them, one noises encodings of these images which are usually of much smaller dimension. Sampling is then done by drawing a gaussian noise in the VAE space, denoising it for several steps and then finally decoding from latent space to image space. It allows to greatly reduce the computation to generate samples as the denoising steps are done in a smaller space.

On the left of figure 2.24, we can see the encoding part where a VAE encodes a data sample x to a latent z_0. The VAE itself is trained to have good a reconstruction of the samples of the data set but also to have a latent distribution close to that of a normal. We will cover all aspects of VAE such as its design up to its training in the background section but it may be useful to get a shallow understanding of it right here.

As images speak louder than words, we may want to look at fig 1.3 where we see that the input image is fed to an encoder which outputs a mean vector and a variance vector. These two elements will then be used to parameterize a normal distribution whose covariance matrix will have null off-diagonal elements. The latent code is then simply a realization of that normal distribution, that is what makes VAE stochastic compared to classical AE's. Finally, the decoding also stochastically maps the latent to a point in the space in which in the inputs live.
Notice we talked about vectors but VAE may map objects of any dimensions to objects of any dimensions, that is, you may map images to arrays but also vice-versa but usually we like them to conserve the nature of objects they are fed but simply to down-scale them

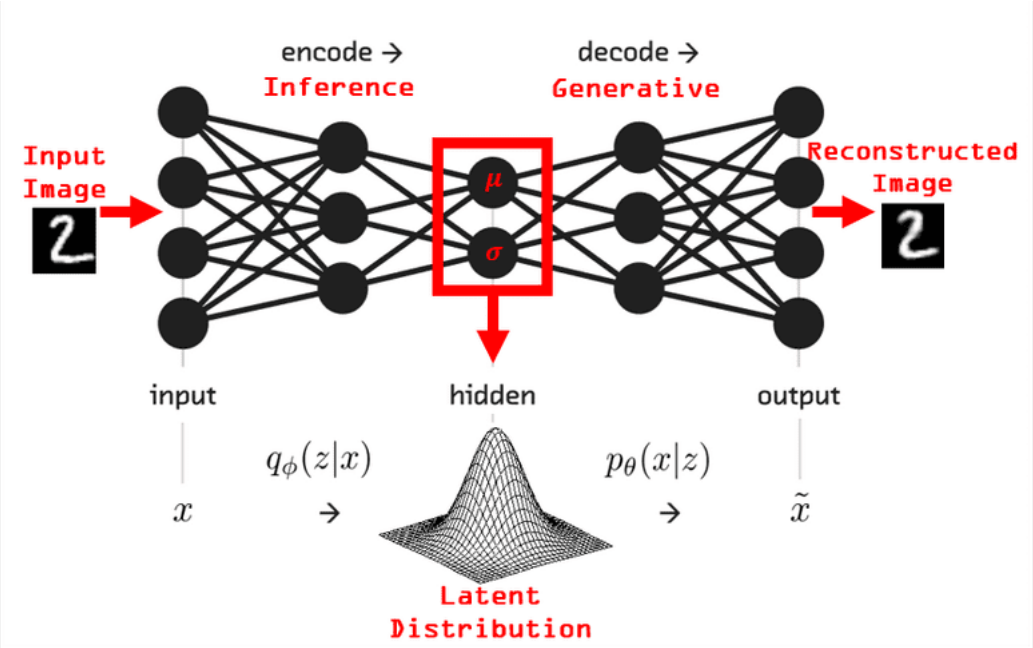


Figure 1.3. VAE encoding an image to a mean and variance vector then sampling from a normal parametrized by these mean and variance and finally decoding the sampled latent

without loosing to much information.

We vaguely introduced VAE's simply to inform the reader that our experiments will also take place in the latent space where information stands in a different form.
This work is not novel in the sense that we don't try to answer a question to which the community has absolutely no clue. Rather, there is no clear consensus on the matter even though many papers, which we'll briefly summarize, argue that the realization of the gaussian noise has very little but importance but what do matter are the denoising steps at the very beginning of the diffusion chain, i.e, close to T.

Chapter 2

Background

As we're doing a thesis on generative modelling, we'll first define what it is, why it matters and describe a few members of that large family. Particularly, since this work is defined in the diffusion and VAE framework, we'll focus onto these two members.

2.1 Generative modelling

Generative modeling refers to a class of machine learning techniques that aim to model and understand the underlying structure of a given dataset in order to generate new data samples that resemble the original data distribution.

One may wonder why one would want to learn to generate purely synthetic data while there is already plenty of real data available in the real world?
The first and most obvious reason is simply data augmentation. Extracting raw data can be time consuming and labelling the data is usually a quite expensive task, especially if it has to be done manually. It is an accepted idea that the more data a model trains on, the better it will perform on new data on average. Thus, increasing the data set size by adding generated samples might be a simple yet effective idea to improve the performance of a model. This is particularly beneficial when working with limited or imbalanced data sets.
The second is that generative models can learn meaningful representations of the input data without the need for explicit labels or supervision. By capturing the underlying structure and patterns within the data, generative models enable unsupervised learning tasks such as clustering, dimensionality reduction, and anomaly detection.
Finally, to quote Richard Feynman : "what I cannot create, I do not understand".

Generative models focus on learning the joint probability distribution of the input data and the corresponding labels $P(X, Y)$ or simply $P(X)$ if there are no labels. Generative models are concerned with understanding the structure, characteristics and patterns of the entire dataset. Essentially they try to learn the features of the dataset that makes it unique in some sense.
A cat dataset is different from the celebA dataset not simply because pixels are different, but rather because the features that define a cat, such as pointy ears, whiskers, and a tail are distinct from the features that define human faces, such as eyes, nose, and mouth.

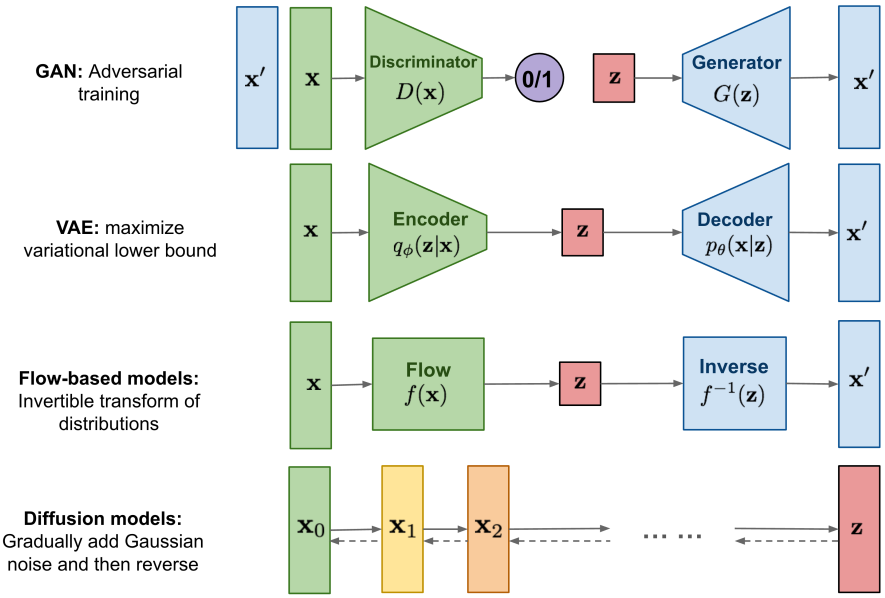


Figure 2.1. Overview of several famous generative models from

Generative models aim to capture these unique features and their relationships within the data, allowing them to generate new samples that possess similar characteristics.

They differ from discriminative models in the sense that discriminative models divide the data space into classes by learning the boundaries, whereas generative models understand how the data is embedded into the space. The former learns the distribution $P(Y|X)$ while the latter learns $P(X, Y)$.

There are a lot of generative models but the most popular in chronological order are kernel density estimators, autoregressive models, VAE's, GAN's and diffusion models. We may observe an overview of the currently most popular generative models on fig 2.1 taken from [1] We now proceed to describing the main members of that family, their description length can be considered proportional to their importance in this work.

2.2 Density estimation and the curse of dimensionality

Density estimation techniques have become outdated and are no longer in use however understanding their major drawback is key to understanding one of the most important designing choice of diffusion models.

When it comes to generating data, the simplest approach would involve sampling from a known Probability Density Function (PDF). However, in almost all real-world scenarios, it is impossible to get the exact PDF. While we might have a general understanding of the distribution shape, we have no closed form for it. That is why modelization plays such an important role in today's science.

[1] https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

As a silly example, consider the distribution of men's sizes. We might expect it to resemble a normal distribution centered around 175 with a small variance, but this is an assumption on both the family of distribution that governs the men's sizes as well as its parameters. Something better one could do would be to gather a dataset of iid samples that consist in men's sizes, postulate a family of distribution and look for the parameters that would best explain the data. This problem is referred to as "probability density estimation".

Suppose we are trying to approximate the men's size density p(x) where x $\in \mathbb{R}$. To get a good level of accuracy in our approximation, we require the data set to contain at least 10 samples per allowed value of the size, in our case, x would be in the discrete set $\{150, 151, ..., 200\}$ rather than in $\mathbb{R}$, we thus do not deal with a PDF but a PMF P(x) . This means the data set should be at least of size 500.

Now suppose we want to model the distribution on both the size and the weight where the weight is in $\{50, 51, \ldots, 100\}$, so we add a dimension to our feature space. We now look for $P(\mathbf{x}_1, \mathbf{x}_2)$ where $\mathbf{x_1} \in \{150, \ldots, 200\}$ and $\mathbf{x_2} \in \{50, \ldots, 100\}$ while aiming to maintain the same accuracy. The problem is that there are many more combinations of values. Previously, we had only 50 possible sample outcomes, but now we have 50 possible weights for a given size, resulting in 50^2 sample outcomes. This means the dataset should be at least of size 2500.

We quickly realize that for typical images that contain not 2 but 3*720*1280 = 2764800 variables, a dataset large enough to learn decent parameters cannot be constructed.

More generally, to parameterize a density over objects that are n-dimensional with each sample taking one out of k values and requiring at least m samples per point in the sample space, one needs mk^n data points in its dataset. Once again, machine learners are hit by the curse of dimensionality : in high-dimensional spaces, the volume of the space increases exponentially with the number of dimensions. This means that the data becomes more sparse and the available data points may not be representative of the entire space. As a result, estimating the density accurately becomes increasingly difficult.

2.3 From AE's to VAE's

2.3.1 AE's

Before diving into the details of VAE's, it may be worthwhile to motivate their existence by describing their predecessor : the Auto-Encoder. An AE is a pair of parametric functions : the encoder $f_\theta(x)$ and the decoder $g_\phi(z)$. The encoder maps points from the original space $\mathcal{X}$ to a latent space $\mathcal{Z}$ where the dimension of the latter is usually smaller than that of the former. The decoder can be seen as the reverse-encoder, it maps a latent z to its counterpart living in $\mathcal{X}$. A simple overview of AE's can be found at 2.2 from [2].

By definition, a good encoder-decoder encodes an input object into an object of smaller size while losing as little information as possible, if all information about the input is lost,

[2]Francois Fleuret, Deep Learning, UNIGE/EPFL

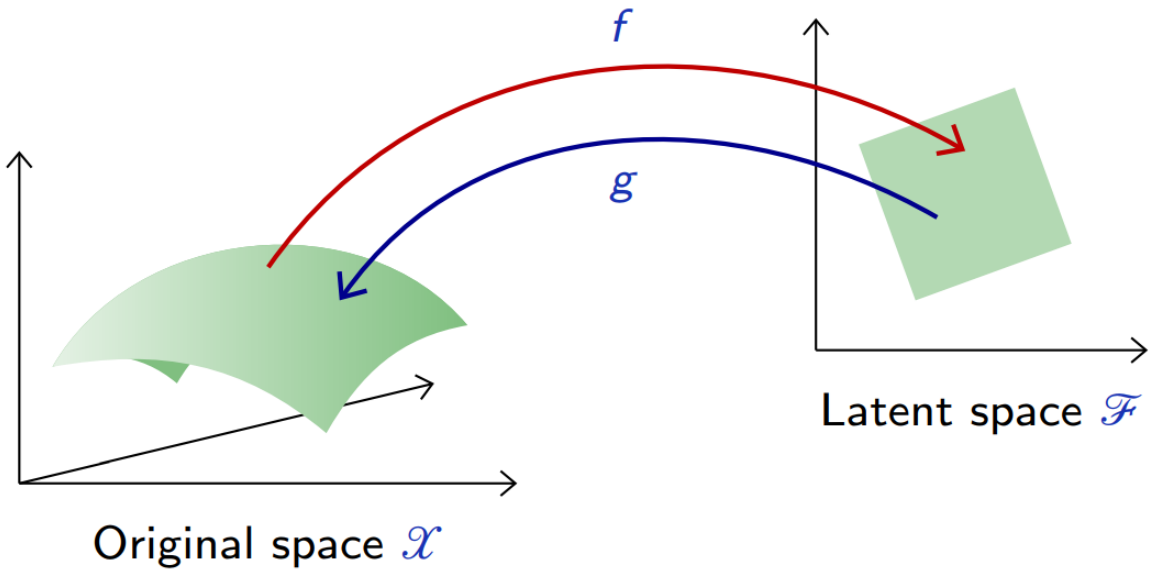


Figure 2.2. Auto-Encoder basics : the encoder f maps points from a 3D-space to points living in a 2D-space and the decoder g does exactly the opposite

then the reconstruction from the decoder will be meaningless and inaccurate, as it won't represent the original input. Therefore, the goal of a good encoder-decoder is to strike a balance between reducing the size of the input representation (dimensionality reduction) and preserving enough information to enable accurate reconstruction. Clearly, a good reconstruction is a necessary condition for an AE to be considered decent.

Let $z = f_\theta(\mathbf{x})$ and $\tilde{\mathbf{x}} = g_\phi(z)$, we can define the reconstructed sample as $\tilde{\mathbf{x}} = g_\phi(f_\theta(\mathbf{x}))$, from that we may define the true reconstruction loss for a data distribution $p(x)$ over $\mathcal{X}$:

$$L_{REC} = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[||\mathbf{x} - g_\phi(f_\theta(\mathbf{x}))||^2\right] \tag{2.1}$$

As usual we cannot compute the true loss and thus one trains the parameters θ and ϕ by using an empirical estimate of that loss :

$$\theta, \phi = \arg\min_{\theta,\phi} \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{x} - g_\phi(f_\theta(\mathbf{x}))||^2 \tag{2.2}$$

Now one has to postulate a model for f_θ and g_ϕ, as usual in deep learning we'll parameterize these two function by neural networks for several reasons :

1. Non-linearity: Neural networks can capture non-linear relationships between variables assuming non-linear activation function, allowing them to model complex patterns and structures in the data.

2. Representation learning: Neural networks are able to to automatically learn useful representations from the data through training. By stacking multiple layers and using non-linear activation functions, neural networks can progressively extract hierarchical features at different levels of abstraction. Furthermore, neural networks are known for being good at manifold learning making them a choice of interest when one essentially seeks to project data on a lower-dimensional embedding

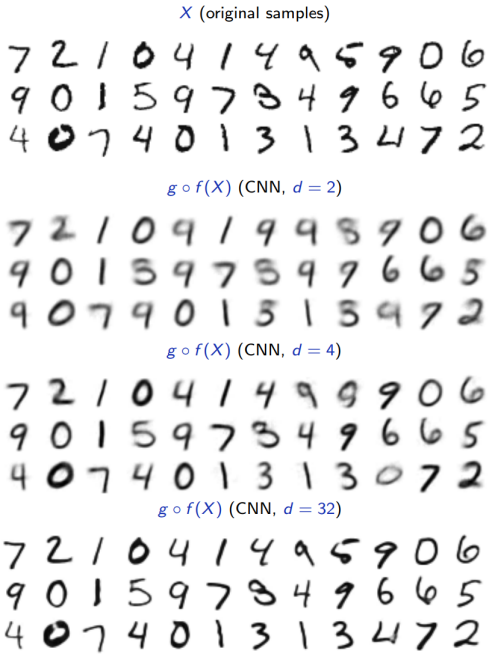


Figure 2.3. Reconstructions of MNIST test samples by feeding them to a CNN-based AE trained on the MNIST dataset, d indicates the number of channels of the latent bottleneck

3. Inductive bias: Based on the input data format, one can choose among various types of neural networks, such as convolutional neural networks for image data, recurrent neural networks for sequential data, or graph neural networks for graph-structured data, making them practical for a wide range of real-world applications.
4. Framework : Powerful computational frameworks, hardware accelerators, and advancements in training algorithms have greatly contributed to the success of neural network-based AEs. These advancements enable efficient training and scalability, making neural networks practical for handling large, high-dimensional data sets.

We may now want to look at the performance an AE could get. As we said previously, a good AE should have very little reconstruction error but obviously the latter depends on the size of the bottleneck, the size of the embedding.

For the sake of the example we'll look at reconstruction for a CNN-based AE trained on the MNIST dataset. Specifically, 3 AE's were trained each with different bottleneck sizes. These sizes are as follows: (2, 1, 1), (4, 1, 1), and (32, 1, 1). We may observe the reconstruction on fig 2.3, the results are very decent. Indeed, using d=2 variables, we still manage to recover the digit class label most of the time from the eye, a classifier would have very decent accuracy on the reconstructions. Furthermore, using d=32 variables, the reconstruction is almost perfect even though it works with a compression ratio of $\frac{32}{28*28} \approx 0.04$ since MNIST samples are 1x28x28. Clearly it seems that the larger the latents, the better the reconstruction. This seems quite expected as the more variables one has, the information one can keep after compression.

We may conclude that AE's are good data compressors however we are not interested in data compression but in generative modelling and we have yet not seen any data generation. To generate data, a simple starting point would be to define a density model

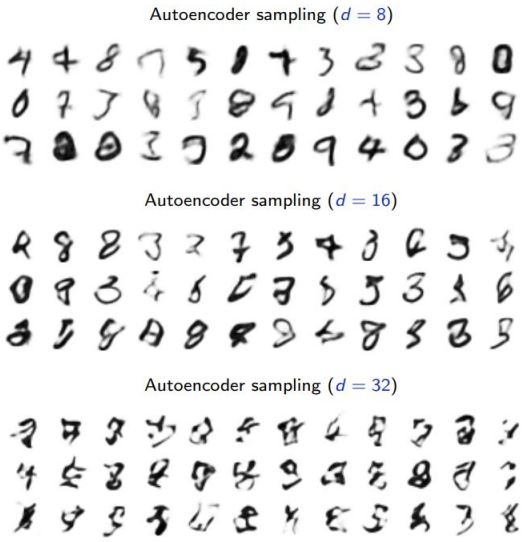


Figure 2.4. Generating samples by drawing a latent from a normal model living in $\mathcal{Z}$ and decoding them to the image space $\mathcal{X}$

q over the latent space $\mathcal{Z}$. Generating a sample $x \in \mathcal{X}$ would simply consist in :

1. Sampling $z \sim q$
2. Decoding $\mathbf{x} = g_\phi(z)$

As we need to specify a model, we'll assume a normal model with diagonal covariance matrix :

$$q(z) = \mathcal{N}(\hat{\mu}, \hat{\Sigma}) \tag{2.3}$$

Where $\hat{\mu}, \hat{\Sigma}$ are estimated on the training data set embeddings.

As we can see on figure 2.4, the results are terrible. There is no need to further describe them however we need to understand why they're being so terrible. We argue the following reasons are

1. The normal model is a strong assumption with very little backup, there is absolutely no constraint in our training procedure that imposes some sort of normality to the latents or any sort of structure at all. The latent space is much irregular as depicted in fig 2.5. Figure taken from [3].
2. The normal mean makes very little when the dataset is multi modal. Suppose the dataset is evenly composed of green images and red images, the mean will consist in a yellow image and most samples will be closed to yellow depending on the variance however there are no yellow samples in the dataset.
3. Our training procedure involved encoding data points and minimizing reconstruction loss, we're just encoding-decoding points with the smallest reconstruction possible. This comes with at a big cost : the latent space severely lacks regularity, it is very poorly organized and structured. A schematic of the problem may be seen on figure 2.6 taken from [4].

[3] https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
[4] https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

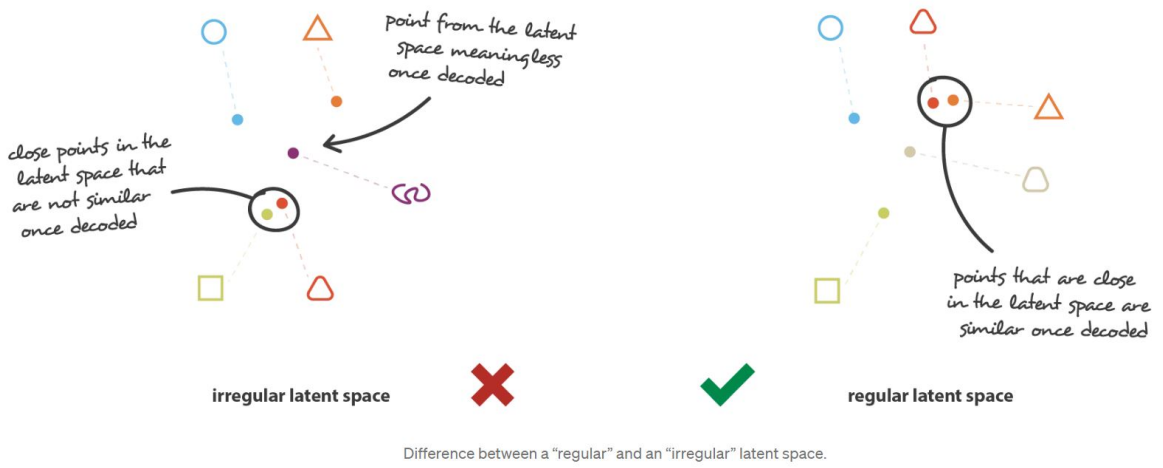


Figure 2.5. Irregularity of the latent space in AE's

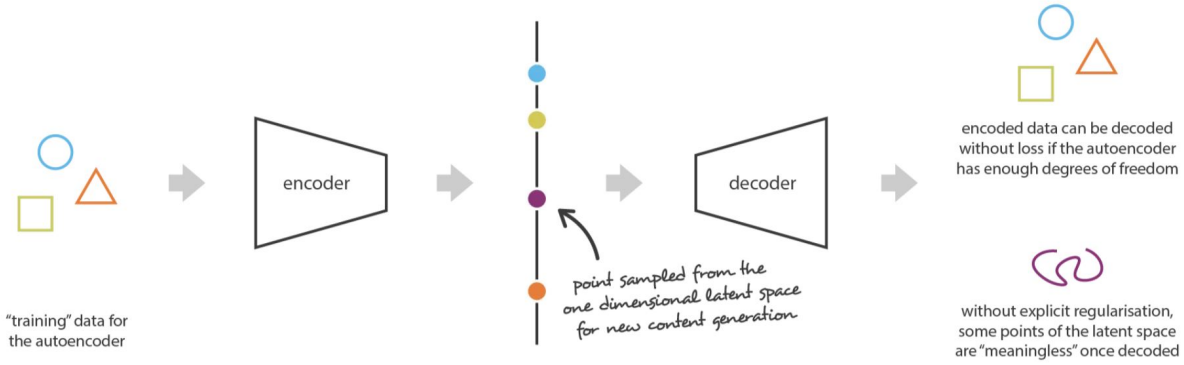


Figure 2.6. Irregular latent space prevents us from using AE for data generation.

4. The only samples $\mathbf{x} \in \mathcal{X}$ that will look decent are those that are extremely close to the training data points and they will simply be decoded as the training data points. This might seem a source for hope however the distance between points grows with the number of dimensions those points have and thus for image data, it is very unlikely that a sampled point ends up being close to a training point. Furthermore, we might argue that outputting almost-identical-to-training-data samples is not generative modelling.

Sometimes taking a look at the problem from a larger perspective helps. We equipped the latent space with an arbitrary distribution and observed that it had poor generative performance. If we want it to have a solid distribution then we're back at the initial problem where we try to generate meaningful samples except we do that in $\mathcal{Z}$ and not in $\mathcal{X}$.

In conclusion, AE sampling performs poorly simply because it was trained to decode specific points to points, as a result the latent space structure is such that decoding a sample close to another has no reason of outputting a similar sample. The latent space has no notion of structure and neighborhoods. Introducing some structure constraint, regularisation constraint or neighborhood constraint is the key step to increase the generative performance of an AE.

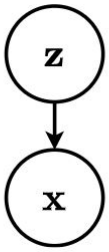


Figure 2.7. Latent variable model of VAE's

This is actually the step Diederik P. Kingma and Max Welling took by incorporating probabilistic graphical modelling and variational bayesian methods to AE.

2.3.2 VAE's

The key change that they added was that instead of having an encoder that maps points in $\mathcal{X}$ to points in $\mathcal{Z}$, VAE's map points in $\mathcal{X}$ to **distributions** in $\mathcal{Z}$. This allows a point in $\mathcal{X}$ to be indirectly mapped to several close neighbour points in $\mathcal{Z}$.

To formulate the ideas clearly, we'll resort to graphical probabilistic modelling. Instead of a simple point-encoder and point-decoder, we'll define their probabilistic versions.
The graphical model on figure 2.7 effectively contains the following relationships between data and latents :

- $p(\mathbf{z})$: the prior latent distribution
- $p(\mathbf{x}|\mathbf{z})$: the likelihood which is directly specified by the model, up to parameters
- $p(\mathbf{z}|\mathbf{x})$: the posterior which is not specified

Under this graphical model and under the assumption that $p(x|z)$ has optimal parameters, sampling works as follows :

1. Sample a latent from the prior distribution : $z_0 \sim p(z)$
2. Decode it to $\mathcal{X}$ through the conditional distribution $p(x|z = z_0)$

We have nonetheless 2 problems to solve to get a working VAE.
First, we need to find the form of the posterior distribution otherwise we'll not be able to encode points in $\mathcal{X}$.
Second, we need to find the optimal parameters of both mappings.
The first problem can be solved using Bayes formula :

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})} \tag{2.4}$$

However $p_\theta(\mathbf{x}^{(i)}) = \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z})p_\theta(\mathbf{z})d\mathbf{z}$ is intractable and thus Bayes helps us by no mean. Indeed it is intractable since to get the density of 1 observed sample $\mathbf{x}^{(i)}$, we have to integrate over **all** possible unobserved $\mathbf{z}$ that could have produced this particular $\mathbf{x}^{(i)}$.
Also, we may already stress that by Bayes the posterior is proportional to the prior distribution, since the latter is quite a regular, we may expect the posterior to be regular as well.

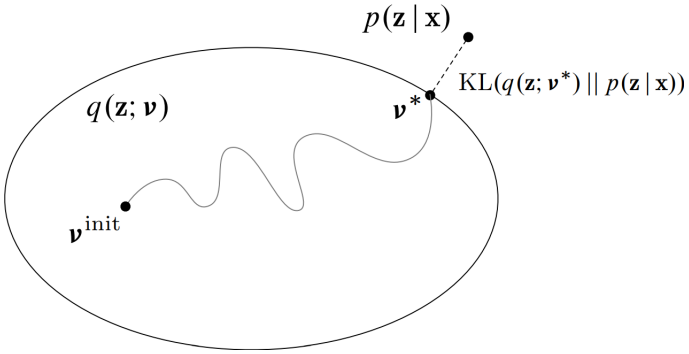


Figure 2.8. The modeler decides the family of distribution that will be used to approximate the posterior and the member of that family is chosen as the one that has minimum KL divergence with respect to the posterior.

Regarding the second problem, at first sight it seems it can actually be solved by maximum likelihood or log-likelihood.
We overwrite the decoder also called generator $p(\mathbf{x}|\mathbf{z})$ by $p_\theta(\mathbf{x}|\mathbf{z})$, by maximum of log-likelihood we have :

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)}) \tag{2.5}$$

By following the graphical model we can decompose the marginal as

$$p_\theta(\mathbf{x}^{(i)}) = \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z} \tag{2.6}$$

And once again we are struck by intractability.
We now resort to the second ingredient introduced by Kingma and Welling : variational inference.
In summary variational inference is a computational technique used in probabilistic modeling and Bayesian statistics to approximate complex probability distributions. It provides a way to estimate the posterior distribution of unknown variables given observed data. In our setup, the posterior $p(\mathbf{z}|\mathbf{x})$ is intractable and we thus decide to assume that it belongs to a certain family of distribution $q_\phi(\mathbf{z}|\mathbf{x})$ where ϕ parameterizes that family.
As we seek to have a surrogate distribution that is as close as possible to the target posterior, we'll look for the optimal ϕ that minimizes the Kullback-Leibler divergence between the two of them as depicted in Figure 2.8 taken from[5].

As a reminder, given two PDF's f(x) and h(x), the KL divergence is defined as :

$$KL(f||h) = \mathbb{E}_f[log\frac{f}{h}] \tag{2.7}$$

Since we're looking for the surrogate distribution with smallest KL divergence to the posterior we need :

$$\arg\min_\phi KL(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) \tag{2.8}$$

By definition of the KL divergence we get :

$$\arg\min_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[log\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}] \tag{2.9}$$

[5]https://glouppe.github.io/info8010-deep-learning/pdf/lec11.pdf

$$\arg\min_{\phi} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[log\ q_\phi(\mathbf{z}|\mathbf{x}) - log\ p_\theta(\mathbf{z}|\mathbf{x})] \tag{2.10}$$

But we cannot minimize this directly as it involves the intractable posterior we're currently trying to approximate, we thus use $p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z},\mathbf{x})}{p(\mathbf{x})}$ and find :

$$\arg\min_{\phi} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[log\ q_\phi(\mathbf{z}|\mathbf{x}) - log\ p_\theta(\mathbf{z},\mathbf{x})] + log\ p_\theta(\mathbf{x}) \tag{2.11}$$

And still we cannot minimize this expression of $KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$ directly because the term $\log p_\theta(\mathbf{x})$ implicitly contains intractable quantities.

To minimize this KL divergence, we need an expression that is either equivalent to it or that bounds it and which does not contain the annoying terms $p_\theta(\mathbf{x})$ or $p_\theta(\mathbf{z}|\mathbf{x})$.

There are 2 paths leading us to the so-called ELBO, Evidence Lower BOund. The first one starts with the likelihood of the data, and the second one starts with the KL divergence between the surrogate distribution and the posterior. Both paths are equivalent in the sense that they end up optimizing the same objective. However, for the sake of conciseness, we'll only detail the first one here. The second path is available at [6].

Reaching the ELBO from the likelihood

We start from the likelihood we have:

$$p(\mathbf{x}) = \mathbb{E}_{p(\mathbf{z})}[p(\mathbf{x}|\mathbf{z})] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \tag{2.12}$$

Taking the log on both sides:

$$\log p_\theta(\mathbf{x}) = \log \mathbb{E}_{q\phi(\mathbf{z}|\mathbf{x})}\left[\frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \tag{2.13}$$

Using Jensen's inequality $\log(\mathbb{E}(y)) \geq \mathbb{E}(\log(y))$, we get:

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{q\phi(\mathbf{z}|\mathbf{x})}[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}] \tag{2.14}$$

By applying the log product rule, we get:

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{q\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}] \tag{2.15}$$

Which can be expressed as:

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{q\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \tag{2.16}$$

The right-hand side of equation 2.16 is termed the $ELBO(\mathbf{x};\phi)$.

We shall note that the ELBO may also be expressed in other completely equivalent forms:

$$ELBO(\mathbf{x};\phi;\theta) = -KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) + \log p_\theta(\mathbf{x}) \tag{2.17}$$

$$ELBO(\mathbf{x};\phi;\theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \tag{2.18}$$

From equation 2.16, we observe that:

[6] https://lilianweng.github.io/posts/2018-08-12-vae/

- Maximizing the ELBO maximizes the log likelihood of $\mathbf{x}$ as it is a lower bound of the left-hand side.
- Maximizing the ELBO encourages the encoder to put latents in a position where their decoding will explain the observed data $\mathbf{x}$ well, as indicated by the first term.
- Maximizing the ELBO makes the surrogate distribution close to the prior latent distribution $p_\theta(\mathbf{z})$. This is beneficial for sampling new data points.

From equation 2.17, we observe that maximizing the ELBO minimizes the KL distance between the surrogate posterior and the true posterior, which was one of the problems we previously pointed out.
From a larger perspective, training requires a loss function to minimize and we thus define the loss function as the opposite of the ELBO :

$$L_{VAE}(\theta, \phi) = -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] + KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \tag{2.19}$$

Traditional VAE

So far we assumed that the likelihood model $p_\theta(\mathbf{x}|\mathbf{z})$ was given, either by domain knowledge or user-specified, now we'll learn the model by parameterizing it it with a neural network.
Specifically, the 3 components of the graphical model are now :

- $p(\mathbf{z})$: the prior latent distribution is a Gaussian
- $p(\mathbf{x}|\mathbf{z})$: the likelihood follows a multivariate normal model with diagonal covariance matrix, the mean and variance are given by a Neural network

$$\mu, \sigma = \mathrm{NN}_\theta(\mathbf{z})$$
$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mu, \sigma^2\mathbf{I})$$

- $p(\mathbf{z}|\mathbf{x})$: the posterior follows a multivariate normal model with diagonal covariance matrix, the mean and variance are given by a Neural network

$$\mu, \sigma = \mathrm{NN}_\phi(\mathbf{x})$$
$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu, \sigma^2\mathbf{I})$$

As we can see on figure 2.9 from [7], both the encoder and decoder map points to normal distribution in the other space and the output simply is a realization of the corresponding distribution.
Regarding training, we might think of simply back-propagating gradients of the ELBO. While it is quite straightforward for the decoder, the gradient of the encoder requires a small trick to be computed.
Gradients of the decoder $p_\theta(\mathbf{x}|\mathbf{z})$ are given by :

$$\nabla_\theta ELBO(\mathbf{x}; \phi; \theta) = \nabla_\theta \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \tag{2.20}$$

Under sufficient conditions :

$$\nabla_\theta ELBO(\mathbf{x}; \phi; \theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\nabla_\theta(\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]) \tag{2.21}$$

[7] https://fleuret.org/dlc/materials/dlc-slides-7-4-VAE.pdf

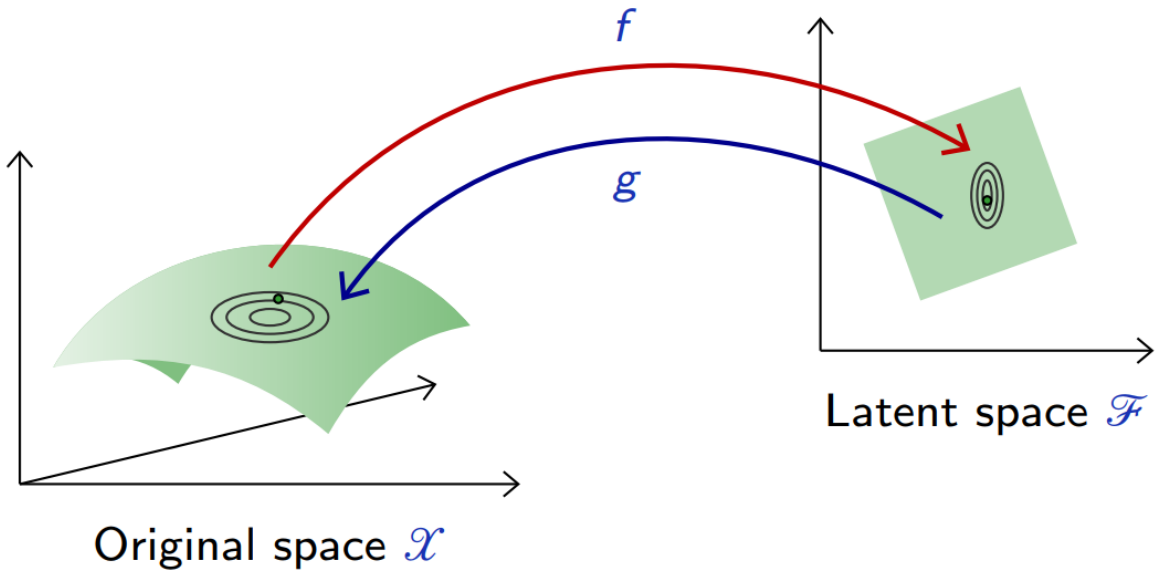


Figure 2.9. Traditional VAE with normal distribution with diagonal covariance matrix

And as the surrogate is independent of θ:

$$\nabla_\theta ELBO(\mathbf{x};\phi;\theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_\theta(\log p_\theta(\mathbf{x},\mathbf{z})\right]) \tag{2.22}$$

The gradients computation are a bit more complex for the encoder as :

$$\nabla_\phi ELBO(\mathbf{x};\phi;\theta) = \nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \tag{2.23}$$

But as the expectation is carried over a distribution directly dependent on ϕ:

$$\nabla_\phi ELBO(\mathbf{x};\phi;\theta) \neq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\nabla_\phi(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]) \tag{2.24}$$

Simply because we cannot back-propagate through the stochastic node z depicted on fig 2.10 taken from [8] . Indeed z is parameterized by ϕ but its realizations are not explicitly dependent on ϕ. Similarly, one could not back-propagate to update parameters of a normal with mean $\mu = 4$ and standard deviation $\sigma = 1.3$. The reason is that a realization, let's say 3.89, has no **explicit** dependence over the parameters of the distribution it was sampled from.

The solution is to express the stochastic variable as a function of a deterministic variable and another stochastic variable ϵ, we'll call this function $g(\phi, \mathbf{x}, \epsilon)$. See fig 2.11 for the updated way of computing the gradients.

Using this trick we get

$$\nabla_\phi ELBO(\mathbf{x};\theta,\phi) = \nabla_\phi \mathbb{E}_{p(\epsilon)}\left[f(\mathbf{x}, g(\phi,\mathbf{x},\epsilon);\phi)\right] = \mathbb{E}_{p(\epsilon)}\left[\nabla_\phi f(\mathbf{x}, g(\phi,\mathbf{x},\epsilon);\phi)\right] \tag{2.25}$$

To conclude this sub-section, we'll look at the added value of using VAE's rather than AE's for sampling. Results may be seen at fig 2.12. Clearly the results are much better but they are far from perfect, there are several reasons from that :

[8] https://glouppe.github.io/info8010-deep-learning/pdf/lec11.pdf

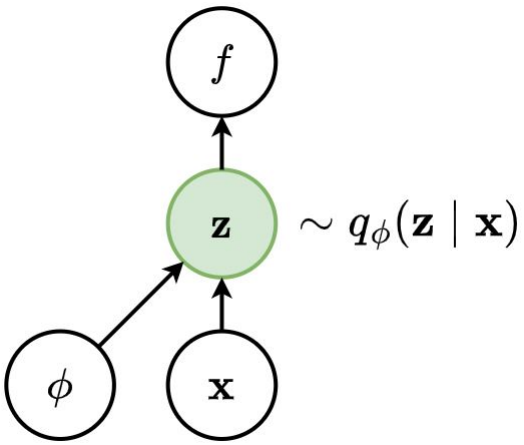


Figure 2.10. Graphical view of the computation of unsuccessful the gradients of the loss function f in the encoder part of VAE's

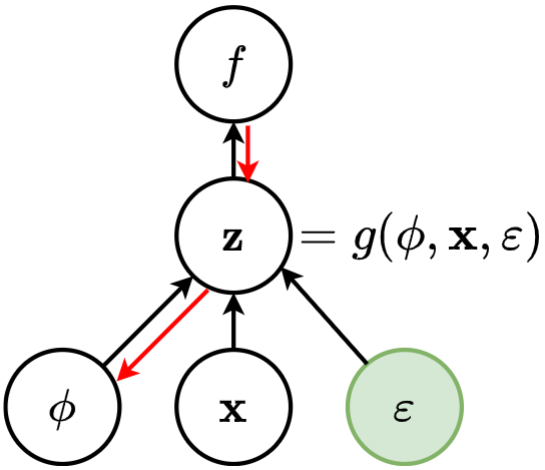


Figure 2.11. Graphical view of the computation of successful the gradients of the loss function f in the encoder part of VAE's

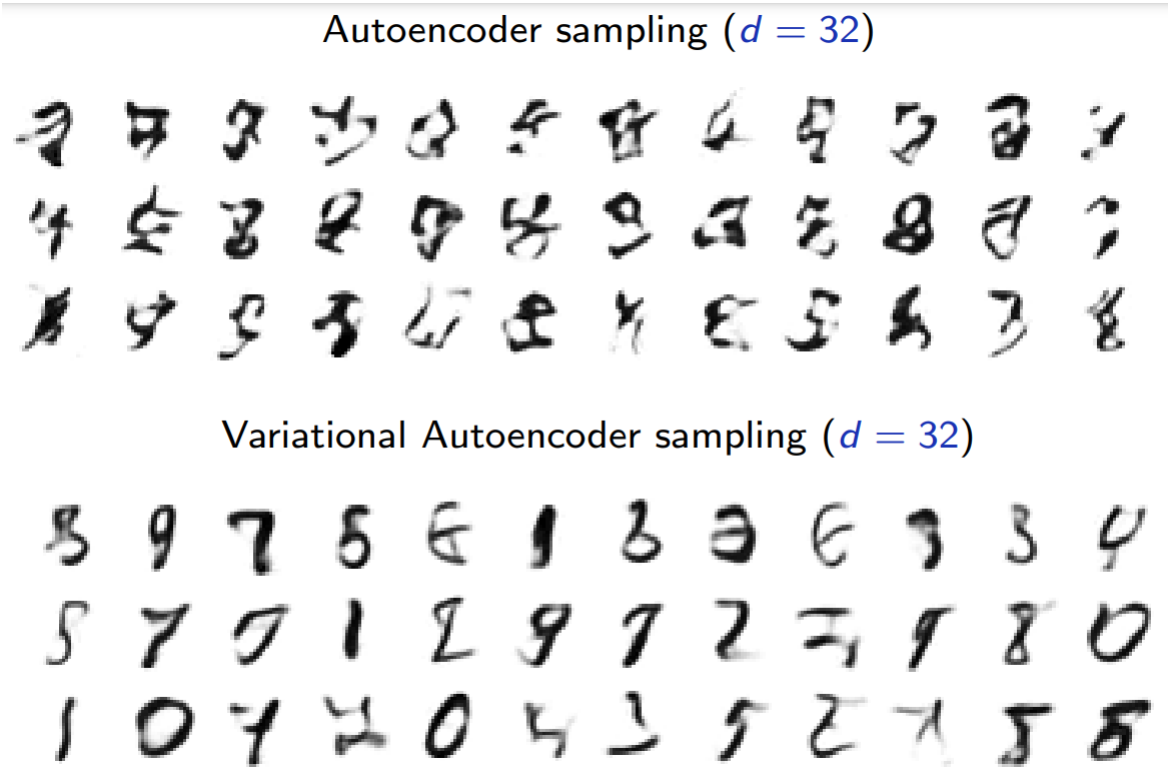


Figure 2.12. Sampling from an AE and a VAE with a CNN trained on MNIST

1. Normal latent distribution : latents are sampled from a simple multivariate normal with null covariance and even though the encoder is trained to be close to that normal by the KL term in the ELBO, using a normal is a very simple modelling choice. This argument is actually what motivated several-layered VAE's and diffusion models
2. Blurriness: one common issue with VAE-generated samples is blurriness. VAE's optimize a reconstruction objective, which encourages the model to generate samples that are similar to the training data. However, this objective can sometimes lead to blurry outputs because it averages over multiple plausible reconstructions, resulting in a loss of sharpness and fine details
3. Mode collapse: VAEs can suffer from mode collapse, which means they struggle to capture and generate samples from all the diverse modes or patterns present in the data distribution

VAE's were introduced in 2013, since then tons of variations emerged with each contributing to either better reconstruction, better sampling or more semantic encoding of training data points. This page inventories almost all of them before 2022 [9].

This section was mainly inspired from the following sources :

1. Lecture 11 of Professor Gilles Louppe : https://github.com/glouppe/info8010-deep-learning
2. Lecture 7 of Professor François Fleuret : https://fleuret.org/dlc/
3. Blog post by Lilian Weng : https://lilianweng.github.io/posts/2018-08-12-vae/

[9] https://github.com/matthewvowels1/Awesome-VAEs

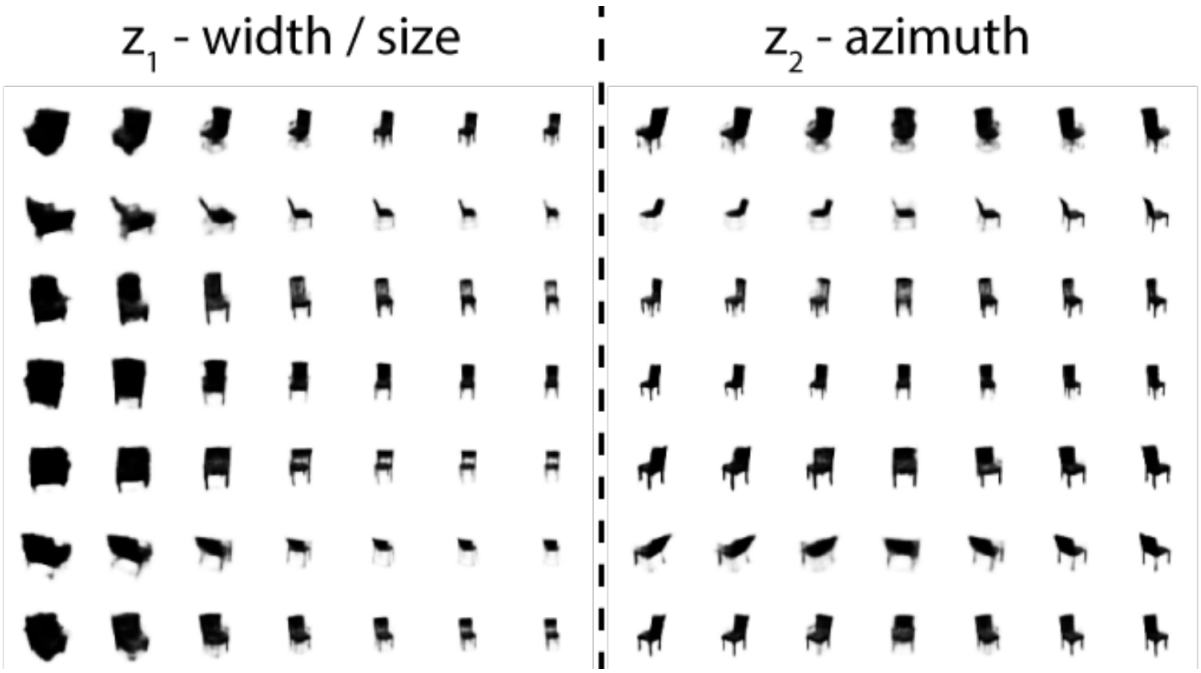


Figure 2.13. A β-VAE was trained on the chairs data set and the first two latents learnt the size and azimuth generative factors. Each row of the left part was constructed by traversing z_1 range of values and then decoding the latent. Similarly the rows of the right part were constructed by traversing z_2 range of values. We observe that when z_1 increases, the resulting chair looks smaller and when z_2 increases, it rotates. Taken from [6]

4. Blog post by Joseph Rocca : https://towardsdatascience.com/understanding-variational-autoe

Variants of VAE

So far we introduced the classical Vanilla VAE but hundreds of variants were defined with each coming up with something new but also limitations. Obviously we'll not cover all those variants as this would be pointless for this work but we'll briefly introduce 2 variants that we'll use in our experiments. These 2 variants are the β VAE and the DIP-VAE.
The $\beta - VAE$ was introduced to learn disentangled representations of the data. If there are groups of variables or variables along that are only sensitive to one generative factor and completely independent of the other generative factors, we call this representation disentangled. In the context of the dSprites dataset version we're using and which we'll describe later, there are 5 independent generative factors. A disentangled VAE which maps images to a latent $\mathbf{z} = [z_0, z_1, z_2, z_3, z_4]$ of size 5 has learnt one generative factor per latent component. That is, z_0 controls the shape, z_1 controls the scale, z_2 controls the x-position etc...
One cool feature that comes up with disentangled representations is attribute manipulation, that is, changing the value of z_i and decoding the changed latent should yield an image whose i-th generative factor has changed. An attribute manipulation can be observed on figure 2.13.

The β-VAE is almost identical to the Vanilla VAE except it puts a weight β to the loss function KL term that penalizes latents that are far from the normal prior. It is thus

$$L_{\beta-VAE}(\theta, \phi) = -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] + \beta KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \tag{2.26}$$

It may seem completely irrational to put a weight on the KL term and expect disentanglement to appear just like that but [7] analyzed and motivated disentanglement in β-VAE using the information bottleneck theory and their arguments are quite solid. We won't enumerate them as this section serves as a mere introduction. We should note that there is a trade-off between reconstruction quality and disentanglement level directly induced by the presence of β.
The second variant we'll consider is DIP-VAE which also aims at disentangling the latents but differently. To keep it short, they simply add to the loss a distance term between the prior $p(\mathbf{z})$ and $q_\phi(\mathbf{z})$.

$$L_{DIP-VAE}(\theta,\phi) = -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] + KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) + \lambda D(q_\phi(\mathbf{z})||p(\mathbf{z})) \quad (2.27)$$

Where $q_\phi(\mathbf{z}) = \int q_\phi(\mathbf{z}|\mathbf{x})p(\mathbf{x})d\mathbf{x}$. The distance function $D(a||b)$ is not taken as the KL divergence but rather a simpler yet effective alternative : a function measuring the distance between the first 2 moments of both distributions. For the sake of conciseness, we'll omit the mathematical details.

2.4 Diffusion models with score-matching and noise-conditioned score networks

VAE's are great for encoding and decent for sampling, the reason they're not so good at sampling new data is that they draw latent samples from a normal distribution. Even though the posterior is trained to be close to a gaussian, it rarely ends up close enough to it to use it as a latent sampler.
A new method termed "diffusion model" was introduced in 2015 by [1], diffusion models can be approached from various angles but the two most popular ones remain score-matching models and denoising models. We here study the approach rooted in score matching and let the second one for the next section.
The goal of this technique is to generate synthetic data given a data set of i.i.d samples $\{\mathbf{x}_i \in \mathcal{R}^D\}_{i=1}^N$ drawn from an unknown data distribution $p_{data}(\mathbf{x})$. The basic idea is to generate samples by using Langevin Dynamics where the score, i.e, the gradient of the log-density is approximated by a deep neural network.

2.4.1 Score vs density

Once again we may wonder why one would approximate the score and not directly the pdf. Let $f_\theta(\mathbf{x})$ be a neural network, as the pdf is a positive quantity we model it by

$$p_\theta(\mathbf{x}) = \frac{e^{f_\theta(\mathbf{x})}}{Z_\theta} \quad (2.28)$$

One could find the optimal parameters θ of the neural network by maximizing the likelihood of a data set however computing the likelihood requires us to know the normalizing constant Z_θ.
As $p_\theta(\mathbf{x})$ constitutes a pdf, it should respect

$$\int p_\theta(\mathbf{x})\mathbf{dx} = 1 \quad (2.29)$$

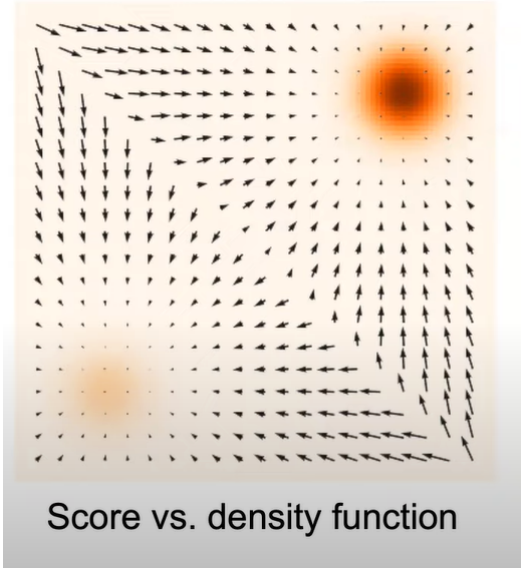


Figure 2.14. Weighted mixture of 2d Gaussians with the density drawn in color and the corresponding scores represented by arrows

And thus

$$Z_\theta = \int e^{f_\theta(\mathbf{x})} \mathbf{dx} \tag{2.30}$$

Equation 2.30 essentially integrates over all possible images or data points and thus makes the computation for the normalizing constant intractable.
A direct way to get rid of this annoying normalizing constant would be to take the gradient of the density or log-density with respect to x. This yields

$$\nabla_\mathbf{x} log(p_\theta(\mathbf{x})) = \nabla_\mathbf{x} f_\theta(\mathbf{x}) - \nabla_\mathbf{x} log(Z_\theta) = \nabla_\mathbf{x} f_\theta(\mathbf{x}) \tag{2.31}$$

The task is now to approximate the gradient of the log-density or more commonly called, the stein-score usually abbreviated score. A toy example is illustrated on fig 2.14.

2.4.2 Score approximation

The score will be approximated by a deep neural network $s_\theta(\mathbf{x}) \approx \nabla_\mathbf{x} log(p_\theta(\mathbf{x}))$ where $s_\theta(\mathbf{x}) : \mathcal{R}^D \to \mathcal{R}^D$. The objective is to minimize the expectation of the $\mathcal{L}_2$ norm of the difference between the true score and the approximated one. Formally

$$\theta^* = \underset{\theta}{argmin} E_{p_{data(x)}}[\|\nabla_\mathbf{x} log(p_\theta(\mathbf{x})) - s_\theta(\mathbf{x})\|_2^2] \tag{2.32}$$

This requires to have access to the score ground truth and thus makes the optimization infeasible as we do not have them. Fortunately, it was shown by [8] that the latter objective is equivalent to score-matching :

$$\theta^* = \underset{\theta}{argmin} E_{p_{data(x)}}[tr(\nabla_\mathbf{x} s_\theta(\mathbf{x})) + \|\frac{1}{2} s_\theta(\mathbf{x})\|_2^2] \tag{2.33}$$

Even though equation 2.33 can be solved by typical gradient descent algorithms coupled with automatic differentiation, it is computationally exorbitant as it requires computing

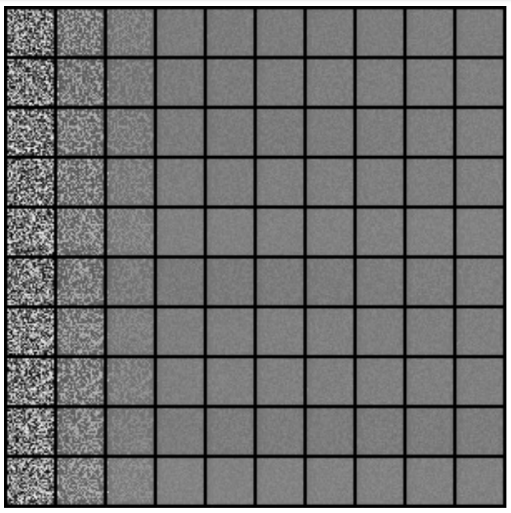


Figure 2.15. Samples generated by Langevin dynamics on top of a network trained to approximate the score of MNIST samples

the jacobian matrix of $s_\theta(\mathbf{x})$ with respect to $\mathbf{x}$. Notice it is the dimension of the data that makes it intractable, in a low dimensional setting, the latter would be totally tractable. Several techniques emerged to deal with this problem such as denoising score matching or sliced score matching which minimizes efficiently computed random projections of the jacobian.

2.4.3 Langevin dynamics

Let us briefly introduce Langevin dynamics which are rooted in statistical physics and named after the french physicist Paul Langevin who was trying to understand the motion of particles suspended in a fluid.
In the context of generative modeling, Langevin dynamics can be used to sample from a probability distribution in order to generate new data points. This approach is often referred to as Langevin sampling or Langevin Monte Carlo.
Langevin Dynamics consist in starting from a random point and going to high density regions by following the vector field of score. Each step taken in the direction of the score comes is added a little bit of gaussian noise so that each path is different from another.
In mathematical terms this writes :

$$x_{t+1} = x_t + \frac{\epsilon}{2}\nabla_x \log p(x_t) + \sqrt{\epsilon_t}\, z_t \tag{2.34}$$

Where ϵ is the learning rate and z_t is sampled from a standard normal
The idea of the technique is then to run a few Langevin steps where each step computes an approximation of the score using the trained score network. Unfortunately, applying this idea straight away does not work as we can see on figure 2.15 where a network was trained to approximate the score of MNIST samples, taken from [2].

This failure is mainly due to 2 reasons the authors identified : inaccurate score estimation in low density regions and slow mixing of Langevin dynamics. For the sake of conciseness, we'll not explain the second reason. To understand the reason score is poorly in low density regions, we might need to first introduce the manifold hypothesis which is quite well accepted in the machine learning community

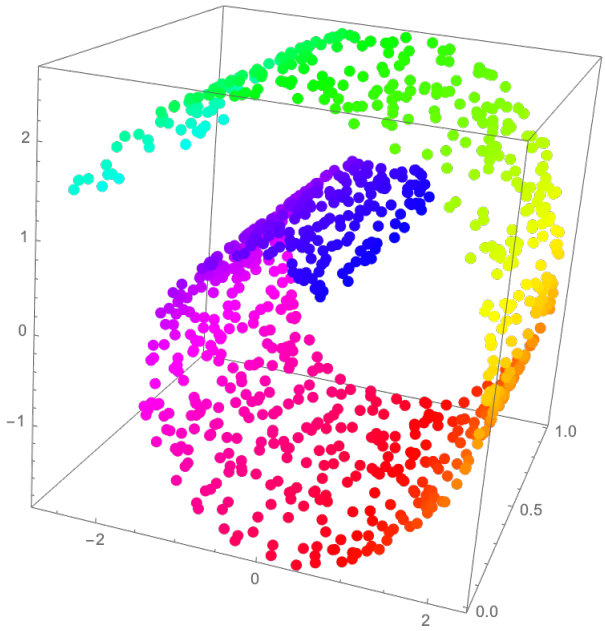


Figure 2.16. 3d points located on a Swiss can be uniquely located by unrolling the Swiss roll into a flat surface and defining 2 coordinates instead of 3

2.4.4 Manifold hypothesis

"The manifold hypothesis states that data in the real world tend to concentrate on low dimensional manifolds embedded in a high dimensional space", "http://colah.github.io/". See fig 2.16 for a toy example on a Swiss roll but the idea is similar for images. Indeed, images definitely do not uniformly fill the space in which they are defined, a very large majority of images defined on $\mathcal{R}^{720\times1280}$ are pure noise to human eyes. The real, meaningful images the data set contains most likely live on a lower dimensional space. Manifold learning is interested in projecting data on lower dimensional space while losing as little information as possible, it can be seen as a non-linear PCA. Deep learning models tend to be very good at implicit manifold learning such as object recognition or data compression. The manifold hypothesis will impact the score accuracy as the gradient is taken on the ambient image space and not on the manifold subspace.

2.4.5 Inaccurate score estimation

The score estimation $s_\theta(\mathbf{x})$ is trained by minimizing an expected loss over the true distribution, in practice it consists in randomly sampling a sample of the data set and computing the loss afterwards. The data set, as stated in the manifold hypothesis, does not fully fill the space in which its samples live but only a tiny fraction of it, hence, noisy images that occupy a majority of that space are never considered. Making a large error on the estimation of the score of these images is thus not a problem as they are very rarely encountered in the data set and thus contribute for a very little part of the total loss. The problem arises when initializing Langevin dynamics, one usually starts with an iterate 0 that is sampled from an uniform prior distribution as we do not know where high density regions are located, if we knew we would sample from there right away. The iterate 0 will then follow a vector field that is highly inaccurate and instead of converging to a high density region will rather random walk in a low density region and finally produce a meaning less output image.

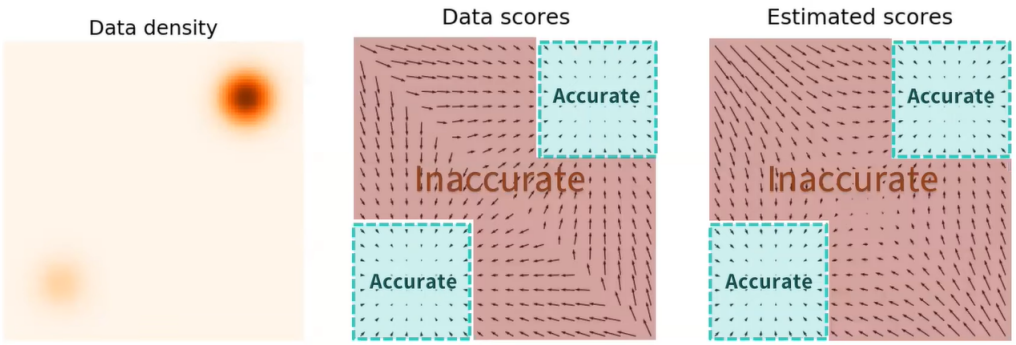


Figure 2.17. The true data scores that have an analytical form for that simple toy example are far from the approximated ones in regions where no data set sample was available

[**song2020generative**]provide a toy example where the true data distribution consists of a mixture of two Gaussians with weights 0.2 and 0.8 and effectively observe on 2.17 discrepancies between true scores that have an analytical form and estimated scores by a neural networks.

2.4.6 Noise conditional score network

The first problem we want to solve is to somehow increase the data in low density regions, the key contribution of [2] is to add a large amount of Gaussian noise to each pixel of the data set. This way, the modes of distribution will widen, become less isolated and thus increase the density of low density regions.
In mathematical terms this leads to :

$$q_\sigma(\mathbf{x}) = \int p_{data}(\mathbf{t})\mathcal{N}(\mathbf{x}|\mathbf{t}, \sigma^2 I)d\mathbf{t} \tag{2.35}$$

which essentially perturbs p_{data} with Gaussian noise leading to a smothering effect. Since the data was perturbed by noise, the estimated scores no longer estimate the true scores but rather noisy scores, thus LD will produce samples that are noisy. Their idea is thus to perturb the data with several levels of noise. That is, a noise dimension is added to the image space and for each sample, one can travel along that noise axis and decide on the noisiness the sample can have. The point is that approximated scores are more accurate for low noise levels but the regions of "decently accurate" score grows with the noise level. An overview of this noise dimension may be observed on fig 2.18. This method was termed "Annealed Langevin dynamics" by [2]

Training thus consists in estimating the score at different noise levels. Instead of defining a neural network per noise level, the authors chose to define one single neural network $s_\theta(\mathbf{x}, \sigma)$ that takes the noise level as input as there can be thousands of noise levels. They decide to train the network using denoising score matching so that $s_\theta(\mathbf{x}, \sigma) \simeq \nabla_\mathbf{x} log(q_\sigma(\mathbf{x}))$ with the noise distribution defined as

$$q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}|\mathbf{x}, \sigma^2 I) \tag{2.36}$$

For a given σ, we want to minimize the loss

$$l(\theta; \sigma) = \frac{1}{2} E_{p_{data(x)}}[\|s_\theta(\mathbf{x}, \sigma) - \nabla_\mathbf{x} log(q_\sigma(\mathbf{x}))\|_2^2] \tag{2.37}$$

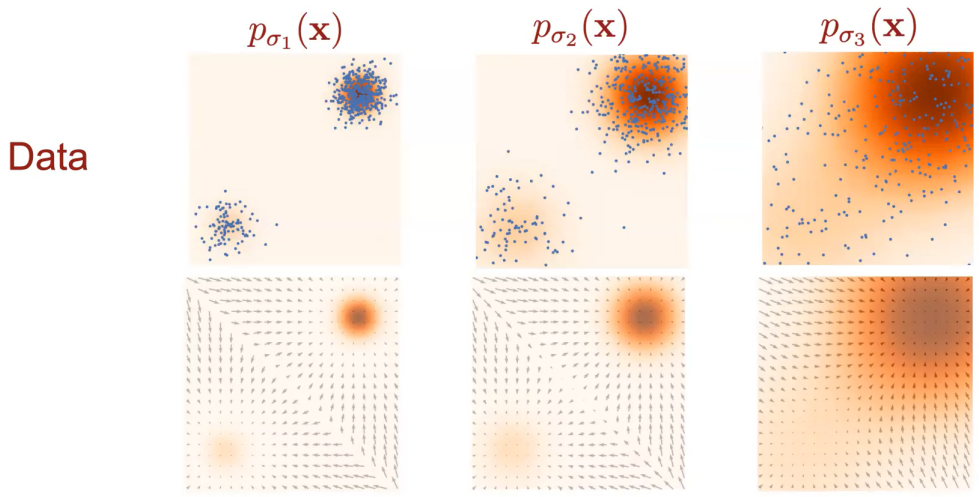


Figure 2.18. Impact of the noise level on the accuracy of estimated scores and on the space span samples cover

Since there is no closed form for $q_\sigma(\mathbf{x})$ but there is one for $q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ the authors actually use the following loss :

$$l(\theta;\sigma) = \frac{1}{2} E_{\mathbf{x}\sim p_{data(x)}} E_{\tilde{\mathbf{x}}\sim\mathcal{N}(\tilde{\mathbf{x}}|\mathbf{x},\sigma^2 I)}[\|s_\theta(\tilde{\mathbf{x}},\sigma) - \nabla_{\mathbf{x}} log(q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}))\|_2^2] \tag{2.38}$$

Notice we take the expectation over $\mathbf{x}$ of the conditional $q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$, this has the effect of leading to an interesting result : in the end we have $s_\theta(\tilde{\mathbf{x}},\sigma) \simeq \nabla_{\mathbf{x}} log(q_\sigma(\tilde{\mathbf{x}}))$!
Since the authors chose a Gaussian perturbation $\nabla_{\mathbf{x}} log(q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \frac{-(\tilde{\mathbf{x}}-\mathbf{x})}{\sigma^2}$ and they finally define the global loss as a weighted sum of individual losses at different noise levels :

$$\mathcal{L}(\theta) = \frac{1}{L}\sum_{i=1}^{L} \lambda(\sigma_i) l(\theta,\sigma_i) \tag{2.39}$$

Inference simply is about making a few LD steps at the very noisy levels, the scores will then be computed at a lower noise level at the exact positions where the previous iterates ended and make a few LD steps and continue that way. Provided that the smallest noise level is close to the data distribution, convergence to the data distribution can be proven. Using this new inference pipeline, results from fig 2.15 have now been greatly improved as we can see on fig 2.19.

2.5 Diffusion models as probabilistic denoising models

2.5.1 Diffusion models as an extension of VAE's

As we repeatedly said, the weak point of VAE's that make them bad at sampling is their latent prior distribution. A multivariate normal with diagonal covariance matrix is an elementary distribution which surely cannot handle the complexity of the real latent data distribution. The solution is to make the prior latent distribution a **learnable** distribution. Classical VAE's can be defined according to the graphical model 2.20 and

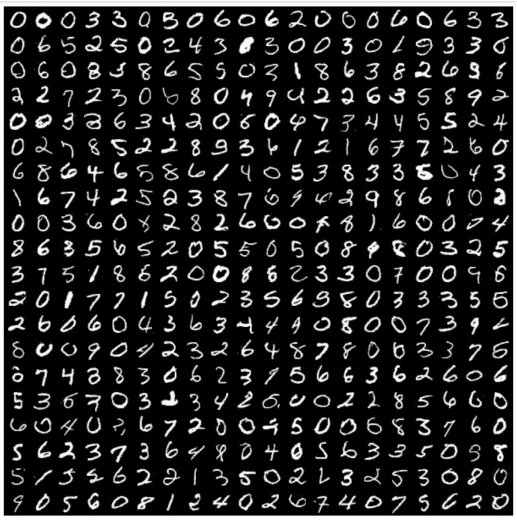


Figure 2.19. Samples generated by Annealed Langevin dynamics on top of a network trained to approximate the score of noisy MNIST samples

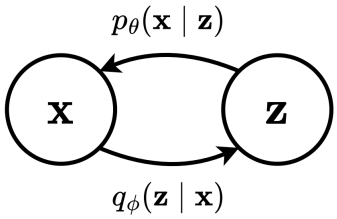


Figure 2.20. Graphical model of a common VAE

diffusion model can be seen as an extension of VAE's where the prior is itself a VAE that will be trained. The prior of this VAE is also itself a VAE and so forth until reaching the hyper-prior which is not a VAE but a usual simple distribution. Thus, diffusion models are Markovian, because only the previous state defines the next, to be more accurate, they are Markovian hierarchical VAE's.

2.5.2 Forward process

In diffusion models, the T encoders and decoders are defined as normal distribution with each having a different mean and a different diagonal covariance. Specifically, the decoders actually define the forward process and the encoders define the backward process from the latents to the data samples $\mathbf{x}$. Note that even though we're talking about encoders and decoders, the data dimension is kept constant here.
An overview of the forward process is depicted on fig 2.22 where an input sample is

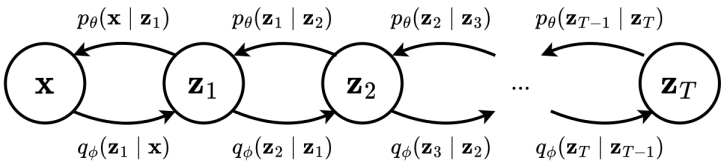


Figure 2.21. Graphical model of a diffusion model where the VAE prior of x is yet another VAE

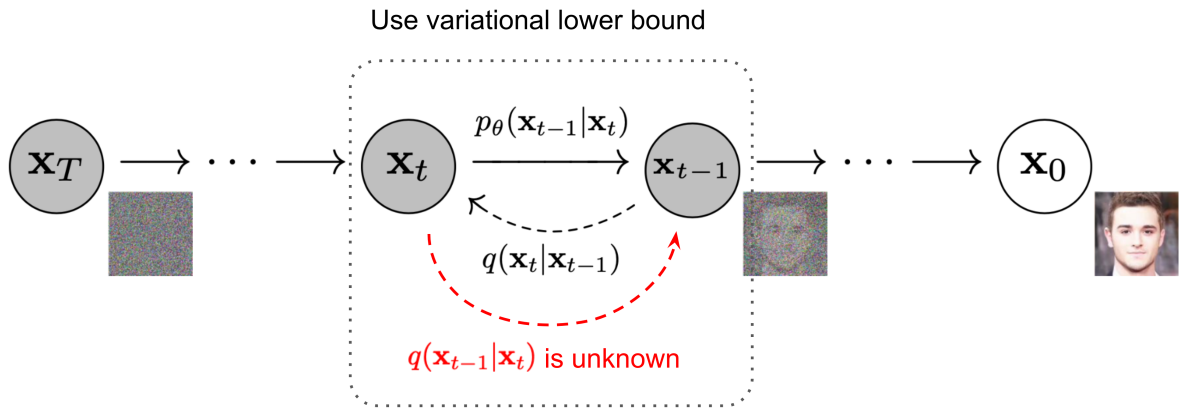


Figure 2.22. The forward process $q(\mathbf{x_t}|\mathbf{x_{t-1}})$ gradually adds noise the the data and eventually turns it into isotropic gaussian noise

gradually added noise such that in the end the sample looks completely like an isotropic gaussian noise. Specifically, we consider a sequence of positive increasing noise scales and for each data point we construct a discrete Markov chain $x_0, x_1, ..., x_T$ such that x_T can be considered fully gaussian for large T.
In mathematical terms we have :

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{2.40}$$

where β denotes a variance schedule and β_k defined $\forall k$ in $[0, T]$ denotes the variance at the k-th time-step. The variance schedule has an important responsibility as it should induce samples to gradually lose information and converge to a mean of 0.
Also, an interesting result stemming from using gaussian kernels is that we're able to sample an x_t for any t in closed form using the reparameterization trick. As shown in [10], let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$:

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \text{ ;where } \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \cdots \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} \text{ ;where } \bar{\boldsymbol{\epsilon}}_{t-2} \text{ merges two Gaussians (*)} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \\
q(\mathbf{x}_t|\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})
\end{aligned}$$

2.5.3 Backward process

The backward process defined by the encoders has the exact opposite role of the forward process, it is tasked with removing a slight level of noise from the sample it is fed. The backward process defines a variational Markov chain in the reverse direction that constructs $\mathbf{x}_T, \mathbf{x}_{T-1}, ..., \mathbf{x}_1, \mathbf{x}_0$ such that they are ideally sampled from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. The distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is only ideal, even though we got $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ we cannot compute by the posterior $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ as it would involve computing the evidence $q(\mathbf{x}_t)$. Mathematically

[10] https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

:

$$q(\mathbf{x}_t) = \int\int q(\mathbf{x}_{1:t}|\mathbf{x}_0)q(\mathbf{x}_0)d\mathbf{x}_{1:t}d\mathbf{x}_0 \tag{2.41}$$

The latter is intractable because several samples could lead to $\mathbf{x}_t$ and one sample could follow different paths that all lead to $\mathbf{x}_t$.
On the other hand, $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is fully tractable as fixing the initial image largely narrows down the possibilities. It has the following form :

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I}) \tag{2.42}$$

where $\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0$ and $\tilde{\beta}_t = \frac{(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}$

The reverse distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is thus approximated by $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. The backward process is fully described by the following equations:

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}(\mathbf{x}_t, t)) \tag{2.43}$$

where $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ is parameterized by a neural network.

2.5.4 Training the denoising network

Training is performed by optimizing the variational upper bound on negative log-likelihood which completely equivalent to the lower bound on positive log-likelihood.

$$\mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] =: L \tag{2.44}$$

Note we're using an upper bound on the negative- The loss defined as the upper bound can be decomposed into 3 terms :

$$L = \mathbb{E}_q\left[D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)\|p(\mathbf{x}_T)) + \sum_{t>1} D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] \tag{2.45}$$

- L_T : the first term expresses the distance between the standard Gaussian at T and the diffused sample x_0 at T.
- L_0 : the last term is minor and not worthy of importance.
- L_{t-1} : the main and middle term expresses the distance between the distribution of the ideally denoised signal and the distribution of the practically denoised signal. Essentially one noises an input sample by one level and asks the network to remove one level of noise.

As both distributions are gaussian in L_{t-1}, the KL loss can simply be expressed as an MSE between means:

$$L_{t-1} = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}||\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)||^2\right] + C \tag{2.46}$$

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Figure 2.23. Training and sampling algorithms of DDPM

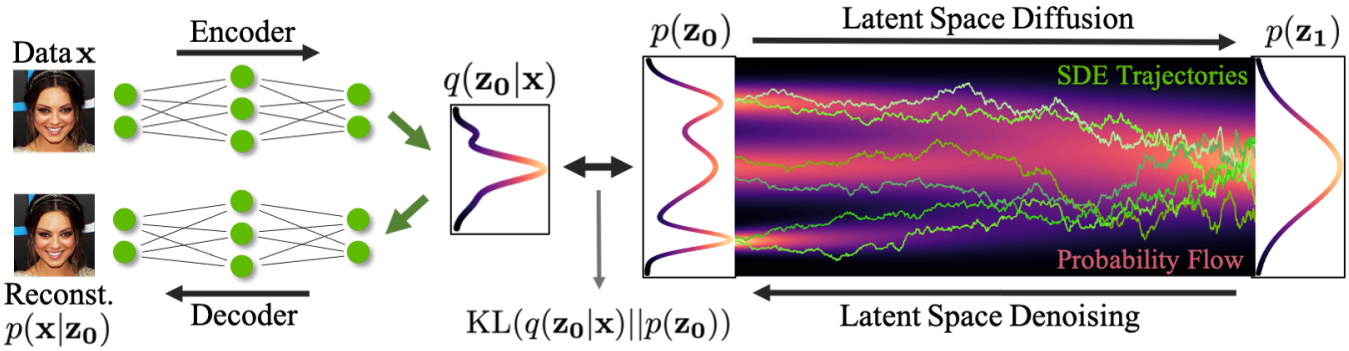


Figure 2.24. Latent diffusion models and brownian paths defined by realizations of the denoised output

Using the reparameterization trick $\mathbf{x}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon}{\sqrt{\bar{\alpha}_t}}$ we can express the loss as

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0, \epsilon} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)} ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2 \right] \tag{2.47}$$

This loss is simply about removing noise added to a clean sample $\mathbf{x}_0$.
The training and inference algorithm are on fig 2.23

2.6 Latent diffusion models

So far we only considered diffusion models operating in the image space however some of them operate in a latent space, the latter is simply defined by means of a regularized Auto-Encoder, this idea was introduced by [5]. Instead of noising images and learning a model to denoise them, one noises encodings of these images which are usually of much smaller dimension. Sampling is then done by drawing a gaussian noise in the VAE space, denoising it for several steps and then finally decoding from latent space to image space.It allows to greatly reduce the computation to generate samples as the denoising steps are done in a smaller space. Even though the idea is quite simple, an overview is shown on fig 2.24.

This background section served as a basis for introducing all necessary tools to develop and perform experiments that will try to answer the research question. This section intended to cover all basic materials and is not exhaustive, if a concept or tool is used afterwards, we shall explain it afterwards.

Chapter 3

Related Work

3.1 Paper 1 : On Analyzing Generative and Denoising Capabilities of Diffusion-based Deep Generative Models

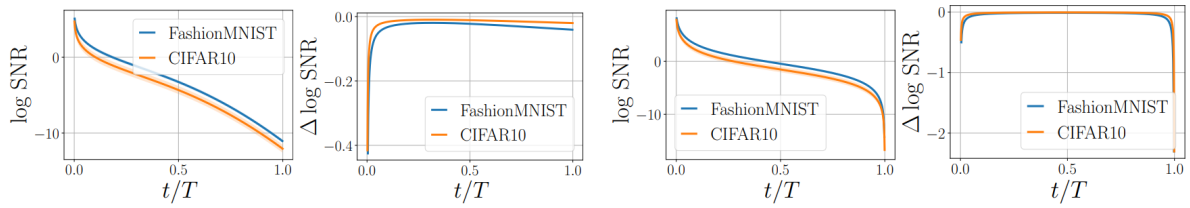
The paper [9] was written by researchers from Warsaw University of Technology : Kamil Deja and Tomasz Trzcinski and researchers from Vrije Universiteit Amsterda : Anna Kuzina and Jakub M. Tomczak.

3.1.1 Paper subject

The paper aims at answering the following question : what is the frontier between the generative and denoising capabilities of a diffusion model. We all know diffusion models are trained with a denoising loss or a score-matching loss that can usually be shown equivalent to a denoising one. Even though they are trained to remove noise from initially well structured images, meaning that for most timesteps the image structure information is not fully lost, they are yet able to generate completely new images. Therefore how do they generate when they were only trained to denoise ? It is a common belief in this field that only the first few backward steps generate most of the information and most of the remaining steps are simply there to give the image a good look. Similarly said in the paper "The more adequate intuition might be that in its initial steps, a diffusion model does not only remove noise but also introduces a new signal according to the distribution learned from the data". The paper aims at studying this frontier a bit more.

3.1.2 Paper contributions

As the backward diffusion chain is essentially trained to revert the forward chain, much of diffusion models generative capabilities can be understood by first analyzing the forward chain. First, they study the noise distribution in the forward diffusion process, i.e, how corrupted does an image get during the forward process. To quantify this amount of noise, they rely on the SNR, i.e, the squared mean of the signal over the variance of the signal. When the SNR is high, much of the signal is present but when it is low, the signal



(a) Linear noise schedule (b) Cosine noise schedule

Figure 3.1. Logarithm of the signal-to-noise ratio averaged over the dataset (solid line) and its standard deviation, and the difference of the log SNR within two consecutive time steps.

is mostly gone. SNR is defined by :

$$SNR(\mathbf{x_0}, t) = \frac{\bar{\alpha}_t {\mathbf{x_0}}^2}{1 - \bar{\alpha}_t}. \tag{3.1}$$

We should note that the SNR is strictly monotonically decreasing.
The authors computed the SNR along the diffusion chain averaged over the whole dataset and reported the log SNR and its discrete derivative on figure .

We can see on the leftmost figure of that the log SNR drops below 0 after about 20 % of the steps, i.e, this is the point after which the signal is dominated by the noise but we thus also know that before those 20 %, the signal is present and strong.

Second, we may wonder whether it'd be possible to reconstruct the original signal given only its noisy version. Surely we cannot exactly reconstruct it as the forward noise is random but a decent reconstruction should be possible. We observe on figure 3.2 that the MAE is rather small for timesteps below 10 %, after which the MAE grows linearly. That is, the denoising model was good at removing low levels of noise but it gets worse and worse with the timesteps.
Based on these 2 observations, the authors suggest to divide diffusion models into 2 parts : a denoiser and a generator. The generator is responsible for transforming the gaussian noise to a corrupted but structured image and the denoiser is responsible for transforming the corrupted image to a clear image, it is aesthetics-designed. They further postulate that only the 20 % steps close to $t = 0$ correspond to denoising ones, that is no information is created over there but rather the image is simply cleaned.

To evaluate their assumption, the authors designed several experiments but one of them makes their assumption pretty convincing. The authors trained a diffusion model on the CIFAR10 dataset and then evaluated its reconstruction error on CIFAR10 and CelebA. Looking at figure 3.3 we observe that for timesteps below 10%, the test reconstruction error on both datasets is similar even though the model knows nothing about CelebA. The reason is that in this region, the model solely denoises corrupted images and does not require any information about the data. After this threshold, the reconstruction error grows fast because the DM has to make generative choices on samples for which it knows nothing. It has no domain knowledge about CelebA and thus makes pretty poor generative choices.

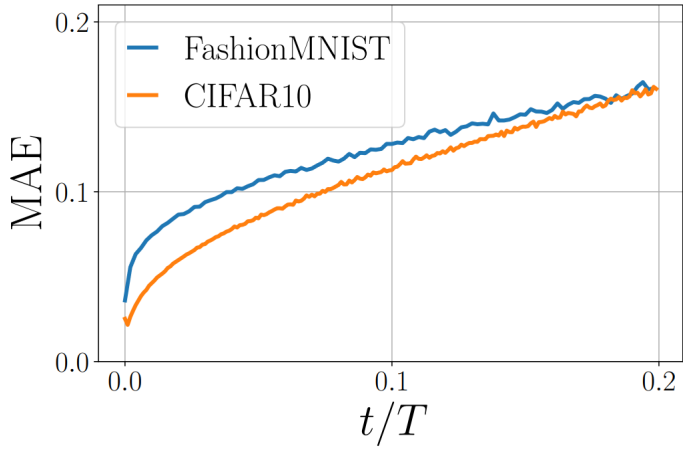


Figure 3.2. The averaged reconstruction error calculated using the MAE at different steps of a diffusion model

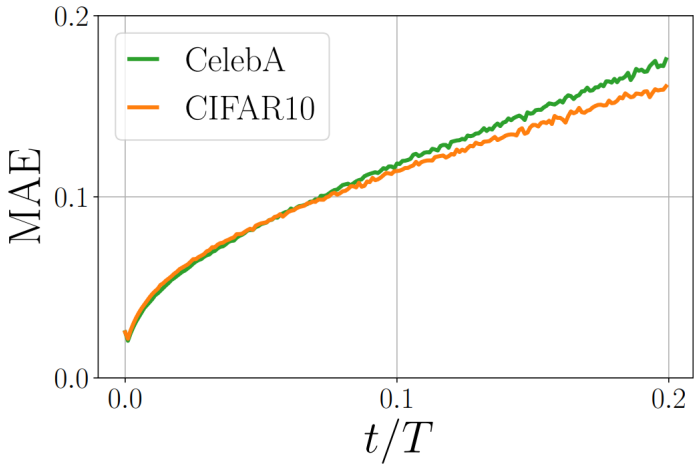


Figure 3.3. The MAE for a diffusion model trained on CIFAR10 and evaluated on CIFAR10 and CelebA

3.1.3 Paper conclusions

It appears that their main assumption was accurate: diffusion models exhibit a transition point at which they shift from a generative to a denoising behaviour. This transition point is likely contingent upon variables like the noise schedule, as well as other factors such as the complexity of the dataset and its domain.

Chapter 4

Experiments

The main objectives of this work are to determine whether diffusion models make choices while generating samples. If so, to investigate the specifics of when these choices occur and who is responsible for making them, whether these choices depend on the space in which diffusion takes place, as well as other questions mentionned in the research question.
A lot of the experiments we'll design take place in a latent space defined by means of a VAE, either vanilla or a variant, therefore we believe it is worth studying them first. These experiments will also usually require a well-trained diffuser, in the generative sense, so we'll also study them a bit. By studying we mean that we want to figure out what factors positively or negatively impact the performances of a VAE and a diffuser.
The considered factors are : latent structure, latent size, stochasticity, regularity and latent representation.
If the reader is used to working with VAEs and diffusion models, this section can safely be skipped.
Before studying VAEs and diffusers, we briefly introduce the main data set with which we'll work : dSprites, a data set of sprites built by deepmind.

4.1 Data set : dSprites

"dSprites is a dataset of 2D shapes procedurally generated from 6 ground truth independent latent factors. These factors are color, shape, scale, rotation, x and y positions of a sprite. All possible combinations of these latents are present exactly once, generating N = 737280 total images". The latent factors can take these values :

- Color: white
- Shape: square, ellipse, heart
- Scale: 6 values linearly spaced in [0.5, 1]
- Orientation: 40 values in [0, 2 pi]
- Position X: 32 values in [0, 1]
- Position Y: 32 values in [0, 1]

Some samples can be observed on figure 4.1.

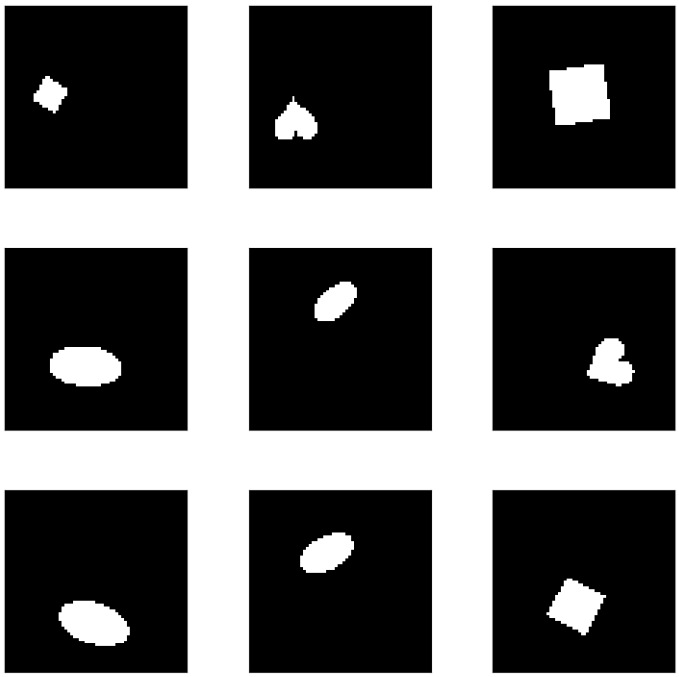


Figure 4.1. dSprites samples

4.2 Pilot experiments : VAE study

To evaluate the performance of a VAE, we need to use some quality metrics or axes to guide us. Unfortunately there is no unique quantitative metric that fulfills this job, there are however many of them that we may use to judge the overall quality of a VAE. The quality of a VAE can be measured by these 3 axes :

1. Test reconstruction error : This axis involves measuring how well the VAE can reconstruct the test data. It is typically evaluated using a test reconstruction error, which can be measured using mean squared error or cross-entropy error, depending on the data type. Practically, we'll consider the reconstruction error of the whole test set.

2. Latent Space Exploration : Assessing the representation learnt by the VAE involves looking at its sampling capabilities and interpolation in the latent space. This evaluation helps understand if the VAE has effectively learned a continuous, regular and meaningful latent space representation, allowing it to generate diverse and realistic samples. An example of interpolation is given on figure 4.2. Practically, we'll choose the 2 first samples of the test set and interpolate 10 latents between them included.

3. Classifier/Regressor Performance : Another way to evaluate the quality of the VAE is to examine how well a classifier or regressor performs when trained on the embeddings produced by the VAE. The VAE's embeddings should ideally capture meaningful and relevant information about the data, leading to solid performance in

downstream tasks. Practically, we'll train a shape classifier. For the sake of conciseness, this axis will rarely be used.

Figure 4.2. A VAE was trained on the celebA dataset, and interpolation can be done by encoding two different images (shown in the last two columns of each row) and then decoding any interpolation between their corresponding latents. Image taken from an article [10]

We stress again that these three axes are not fully sufficient to quantify and draw conclusions on performance of a VAE. Additional axes such as likelihood estimation and interpretability could also be considered depending on the specific use case and requirements. Evaluating a VAE from multiple perspectives is essential to gain a comprehensive understanding of its capabilities and limitations. Now that we have a decent VAE evaluation procedure, we may set up simple experiment that will illustrate the importance of the 5 factors we consider using the dSprites data set.

4.2.1 Latent structure importance

Experiment goal and context

A dSprites sample of shape (1,64, 64) can be encoded into several different latents. Specifically, it can be represented as either a 2D tensor of shape (1, x, y) or a 1D tensor of shape (z), where the number of components of the latents is the same, i.e., z = xy.

On one hand, since the samples are 2D squares, it is reasonable to consider mapping them to smaller 2D squares in order to preserve the structure of the samples. Preserving the structure means retaining the local information and patterns that are crucial for understanding the content of the images. These local patterns include textures, features, and other information that contributes to the overall interpretation of the scene.

To preserve the structure of the images, using Convolutional Neural Networks (CNNs) seems a suitable approach. CNNs are well-suited for this task due to their inherent ability to capture local structures and spatial relationships in images. By sliding small filters over

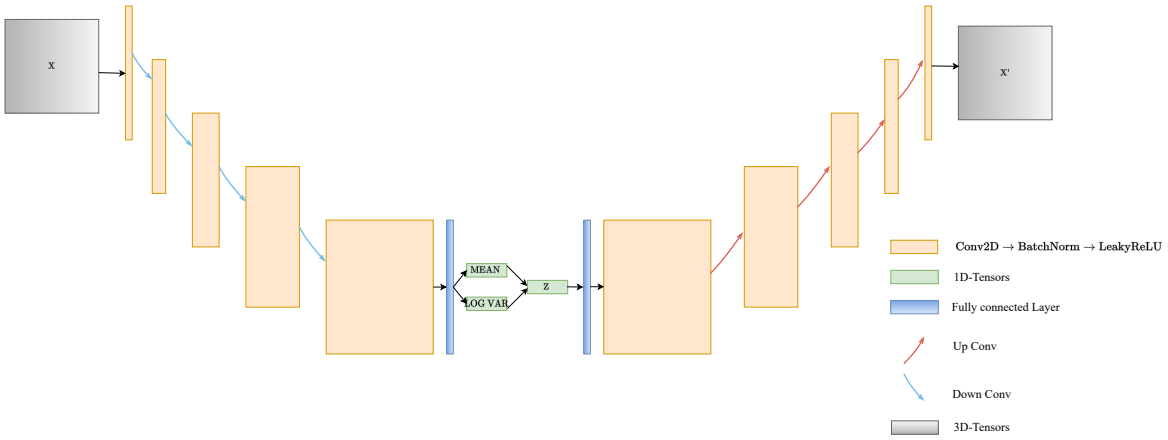


Figure 4.3. VAE that maps 3D input $\mathbf{X}$ and produces a 1D latent variable $\mathbf{Z}$ at the bottle-neck as well as a reconstruction $\mathbf{X}'$. The VAE mainly consists of CNN layers with 2 fully connected layers at the end to extract a 1D mean and 1D log variance that will parameterize the latent normal distribution.

the input image, CNNs can effectively learn and extract meaningful features from local regions, contributing to a more accurate representation of the underlying content.

On the other hand, one could use a simple CNN with fully connected layers at the end to encode image samples to 1D tensors, this comes with the advantage that the latent size is easily manipulable but this also comes at the cost of the network completely destroying the spatial relationships between neighbouring pixels. Note that such a network could also be used to map 2D squares to smaller 2D squares by reshaping the output but this operation would not preserve any structure.

In regard of understanding latent structure importance, we decide to evaluate 3 VAEs :

1. $VAE - 1024$: a VAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (1024)
2. $CONV - VAE - 4 - 16 - 16$: a basic VAE made of a CNN only that maps (1, 64, 64) to (4, 16, 16)
3. $KL - VAE - 4 - 16 - 16$: an advanced VAE made of a CNN that makes use of attention layers that maps (64, 64, 1) to (4, 16, 16)

These 3 VAE all share latents with 1024 components but have very different structure, we shall briefly describe them here. They all have a CNN backbone but those that map 3D inputs to 1D latents naturally have to use fully connected layers at some point. The VAEs mapping 3D inputs to 1D latents that we used are represented on figure 4.3 while the fully convolutional ones that map 2D to 2D or 3D to 3D are represented on figure 4.4. The $KL - VAE - 4 - 16 - 16$ we introduced earlier is similar to $CONV - VAE - 4 - 16 - 16$ except it possesses some self-attention layers, it is represented on figure 4.5.

Results

The test reconstruction errors of the 3 VAEs can be seen on table 4.1.

Samples of $VAE-1024$ can be seen on figure 7.1 and samples of $CONV-VAE-4-16-16$

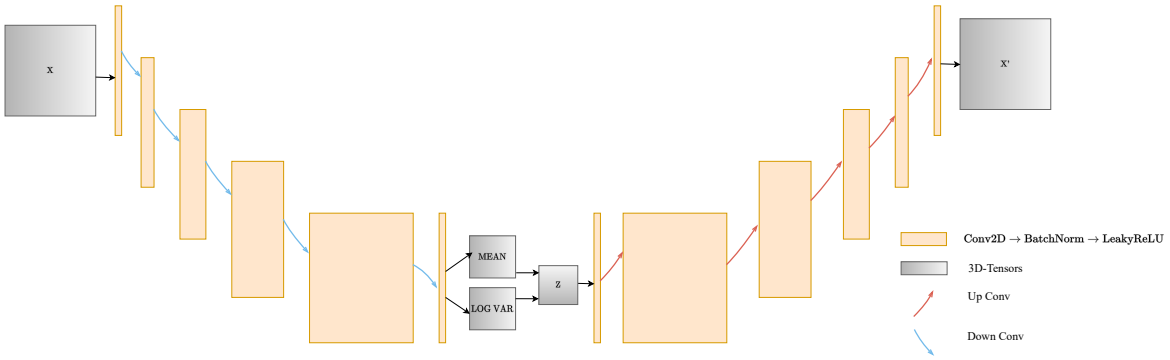


Figure 4.4. VAE that maps 3D input $\mathbf{X}$ and produces a 3D latent variable $\mathbf{Z}$ at the bottle-neck as well as a reconstruction $\mathbf{X}'$. The VAE solely consists of convolutional layers.

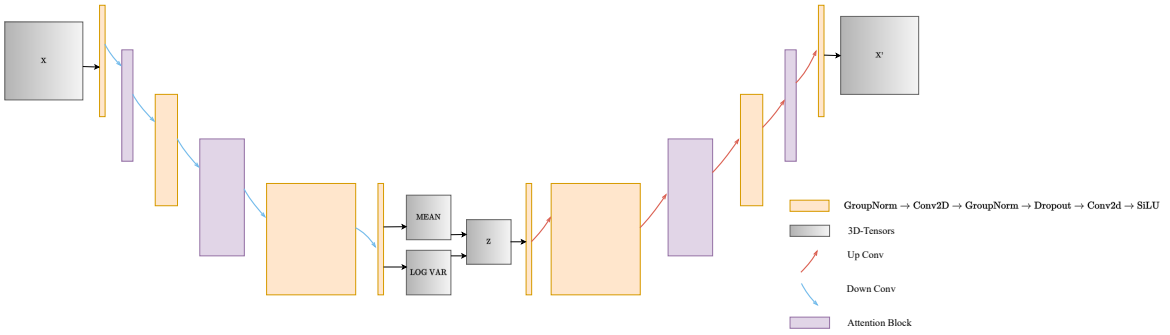


Figure 4.5. VAE that maps 3D input $\mathbf{X}$ and produces a 3D latent variable $\mathbf{Z}$ at the bottle-neck as well as a reconstruction $\mathbf{X}'$. The VAE consists of convolutional layers and self-attention layers.

Model	Test Reconstruction Error
$VAE - 1024$	24.25
$CONV - VAE - 4 - 16 - 16$	20.55
$KL - VAE - 4 - 16 - 16$	3.65

Table 4.1. Axis 1: Test reconstruction errors for different structures of Variational Autoencoders (VAEs)

can be seen on figure 7.2, as $KL - VAE - 4 - 16 - 16$ is a VAE pretrained on a large set of images, its samples are not sprites and thus of low interest. We also put for each model interpolations between 2 images from the test set respectively as 7.3, 7.4, 7.5.

Interpretation

On one hand, the fully CNN-based VAEs performed better, according to the first axis, than the classical one with FC layers at the end which may perturb the overall order of the image. On the other hand, their sampling capabilities and latent space structure seem far below those of $VAE - 1024$. Indeed their generated samples seem not to belong to the data distribution and interpolations between data samples are definitely not smooth. Smoothness is not a sufficient criterion to assess a latent space structure per se. However these discontinuities we observe in 7.4 and the blurriness and multi-modality we observe in 7.5 are clear red flags.
Oppositely, the interpolation capabilities of $VAE - 1024$ are quite good as we see that consecutive interpolates are quite similar which is what you expect for a VAE. Indeed, neighbours should be similar. There is however a small bad feature to note, even though the change of shape between 2 data samples is pretty smooth, the change in position is sometimes too sudden, too abrupt. The third row of 7.12 shows a clear example.
We should also note that we sometimes observe purely black images which are often located in the middle of the interpolation path. We believe these correspond to zones of the space to which no data was encoded and it seems no data even got closed to being encoded there as by the stochastic encoding procedure of VAE, this region would have been populated if it was the case.
From this simple experiment, we are tempted to draw the conclusion that fully CNN-based VAEs are more fit for image reconstruction because they preserve the input structure however they seem to have a poor latent space structure compared to those that contain dense layers.

4.2.2 Latent size importance

Experiment goal and context

The latent size as the name indicates is simply the number of components of the latents and while encoders usually down scale the input data, there is no general rule on the size latents should have.
We could expect larger latents to be able to hold more information and smaller latents to be less expressive as they have less degrees of freedom. Formally, the latent size determines the level of information compression in the model. A smaller latent size forces the VAE to capture only the most essential features of the input images, leading to a more compact representation. Conversely, a larger latent size allows the model to capture more details

and nuances in the data but one should note that if the latent size is too small, the model may not be able to represent the data. Indeed, consider a VAE trained on dSprites with a latent size of 4, by the pigeonhole principle there is at least one latent component that holds for at least 2 generative factors in the ideal case where each latent learnt a generative factor. We can cautiously say that the reconstruction will never be perfect because encoding most likely lost some information that decoding will not be able to "re-create". Note that perfect reconstruction is possible in a simple yet unrealistic scenario : where the latent that holds 2 generative factors learns all pairs of these 2 factors and maps each pair to an index. It essentially boils down to enumerating each pair and it would allow for perfect reconstruction but this would ruin all the generative aspect of the VAE.
We should also note that larger latents are also more prone to over-fitting.
In regard of understanding latent size importance, we decide to evaluate 5 VAEs :

1. $VAE - 256$: a VAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (256)
2. $VAE - 128$: a VAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (128)
3. $VAE - 64$: a VAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (64)
4. $VAE - 16$: a VAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (16)
5. $VAE - 4$: a VAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (4)
6. $VAE - 1$: a VAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (1)

Results

The test reconstruction errors of the VAEs can be seen on table 4.2.

Model	Test Reconstruction Error
$VAE - 256$	16.19
$VAE - 128$	21.18
$VAE - 64$	15.38
$VAE - 16$	52.11
$VAE - 4$	120.41
$VAE - 1$	302.81

Table 4.2. Axis 1: Test reconstruction errors for different sizes of Variational Autoencoders (VAEs)

Samples generated by the VAEs can respectively be seen on figures 7.6, 7.7, 7.8, 7.9, 7.10, 7.11.

We also put for each model interpolations between 2 images from the test set respectively as 7.12, 7.13, 7.14, 7.15, 7.16, **??**.

Interpretation

We can observe that the reconstruction roughly decreases with the latent space size, we can simply explain that by the fact that smaller spaces have smaller variables to hold useful information about a point they wish to encode. The error made by $VAE-1$ is huge w.r.t that of $VAE-256$ simply due to the fact that 1 variable is hardly enough to hold information about 1 categorical variable and 4 quantitative variables.
From the second axis perspective, one can clearly tell that the sampling capabilities seem to decrease with the latent size as well. Samples 7.10 drawn from $VAE-4$ are much more likely to come from the data distribution than samples 7.6 drawn from $VAE-256$. We believe this is due to the fact that the whole data set is encoded to a fraction of the latent space that decreases with the number of dimensions. This could also be due to the larger network being more prone to overfitting or even the smaller one learning a more disentangled representation. As we'll see in 2 sections, learning disentangled representations usually helps.
From these simple observations, we draw the conclusions that larger latent spaces may hold larger amounts of information at the cost of having lower sampling capabilities most likely due to poor disentanglement learning.

4.2.3 Stochasticity and regularity importance

Experiment goal and context

Variational auto-encoders were mainly introduced because auto-encoders latent space had poor regularity and structure. Auto-encoders thus also had poor sampling capabilities but very good reconstruction error as they were trained to minimize it.
Regarding the third axis, auto-encoders provide deterministic embeddings which may not capture all relevant information for downstream tasks like classification and regression. This lack of stochasticity and regularization might result in overfitting to the training data and poor performance on the test.
In regard of validating these hypotheses latent size importance, we decide to evaluate a VAE and an AE with almost similar architecture :

1. $VAE-256$: a VAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (256)
2. $AE-256$: a AE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (256)

The 2 architectures are identical except for the last fully connected layer which is twice larger for the VAE as it should output a mean and a log variance whereas the AE simply outputs the latent itself.

Results

The test reconstruction errors of the VAE and the AE can be seen on table 4.3. Samples generated by the VAE and the AE can respectively be seen on figures 7.6 and 7.18. Note that as AE are not trained to match any normal density, sampling from a standard normal makes little sense. We thus decided to sample from the mean and standard deviation of the test set encodings : $\mu_{\mathbf{z}}$ and $\sigma_{\mathbf{z}}$.

Model	Test Reconstruction Error
$VAE - 256$	16.17
$AE - 256$	1.20

Table 4.3. Axis 1: Test reconstruction errors for an AE and a VAE

Interpretation

We read on table 4.3 that $AE - 256$ has a better reconstruction error than $VAE - 256$ and that is no surprise as it is solely trained to do so. Its latent space representation is however pretty poor as we can see from its generated sampled which appear on figure 7.18. We are however very surprised of its interpolation skills which are depicted on figure 7.19. We would have expected meaningless interpolates while they are pretty decent.
We can safely conclude that VAEs have a better latent space representation than simple AEs.

4.2.4 Latent representation importance

Experiment goal and context

Neural networks are overall very powerful tools however their performance is strongly dependent on the data they're fed. A simplistic network fed with data in a good form might perform better than a complex network fed with data in another less meaningful form. The embedding and representation of the data is a major factor to any task in machine learning.
Therefore we want to know out of the many data representations VAE's can offer, which one is the best one ? In our simple set up we'll simply test 3 VAEs :

1. $VVAE - 256$: a Vanilla VAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (256)
2. $BVAE - 256 - 10$: a Beta VAE, with β set to 10, made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (256)
3. $DIPVAE - 256$: a DIPVAE made of a CNN backbone to extract features and FC layers at the end that maps (1, 64, 64) to (256)

We note that we have also tested a $BVAE - 256 - 30$ with β set to 30 but as soon as the value exceeds 20, the disentangling term is too strong and reconstruction is poor and generated samples have little likelihood of belonging to the data distribution.

Results

The test reconstruction errors of the different VAEs can be seen on table 4.4

Samples generated by $VAE - 256$, $BVAE - 256$ and $DIPVAE - 256$ can respectively be seen on figures 7.6, 7.20 and 7.21.

Interpolations made by the same models can be seen on figures 7.12.

Model	Test Reconstruction Error
$VAE-256$	16.12
$BVAE-256$	12.12
$DIPVAE-256$	6.57

Table 4.4. Axis 1: Test reconstruction errors for 3 different VAEs

Figure 4.6. 2 samples that share the same posX, posY, scale and orientation but different shapes

Interpretation

From table 4.4 it is clear that $DIPVAE-256$ has the best reconstruction error but looking at its generated samples on figure 7.21, they look nothing like the expected data. Conversely, the generated samples of $BVAE-256$ on figure 7.20 are pretty decent. While they have very different sampling skills, they are both very good at smoothly interpolating between data samples as can be seen on figures 7.22 and 7.23. It could be that $DIPVAE-256$ samples were low quality because they were sampled from a standard normal in $\mathbf{Z}$ while the mean and standard deviation of the data set encodings were not close to (0,1) simply because the KL term had little impact during training but after verification, it turns out they are very close to (0,1). We thus have found no honest explanation to this behaviour of $DIPVAE-256$.

After these 2 observations, we are tempted to say that β-VAEs seem to be the best of the 3 VAEs as they have decent reconstruction error and decent sampling skills.

Note : $AE-256$ and $DIPVAE-256$ both had mediocre samples but very good interpolations, this suggests that these 2 criteria we used to assess the latent space structure and representation were actually not both targeting the same feature. If they were, we should not have drawn opposite conclusions from them.

4.2.5 Some notes on VAE

In this section, we share not necessary but useful insights regarding VAEs, particularly we'll look at the latent representation of samples using the different VAEs we've introduced.
We consider for example the figure 4.6 which contains 2 samples that share the same posX, posY, scale and orientation but different shapes. We then look at the different encodings 4.9, 4.8 and 4.7 it admits respectively using $VAE-5$, $BVAE-5$ and $DIPVAE-5$.

We may, not totally honestly, observe that the two samples are less distant in 4.8 than in 4.9. Indeed, the absolute difference element wise of the vanilla encodings is [1.0672, 0.8881, 1.0811, 0.3583, 0.9404] while for the β encodings it is [1.5316, 0.6429, 0.5234, 0.1331,

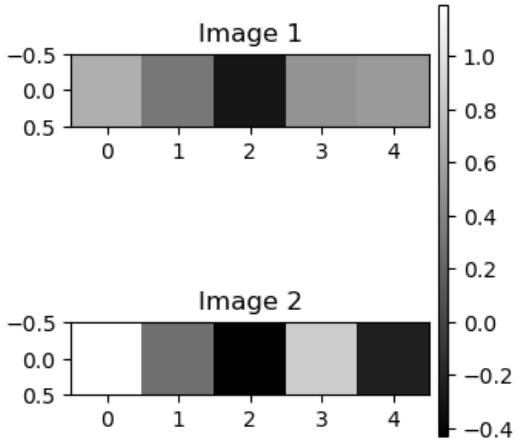


Figure 4.7. Encodings of the 2 samples using $DIPVAE - 5$

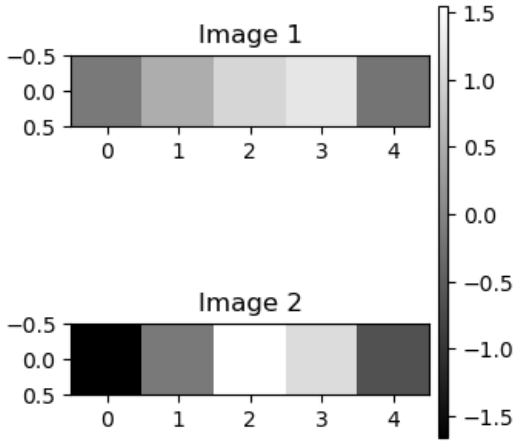


Figure 4.8. Encodings of the 2 samples using $BVAE - 5$

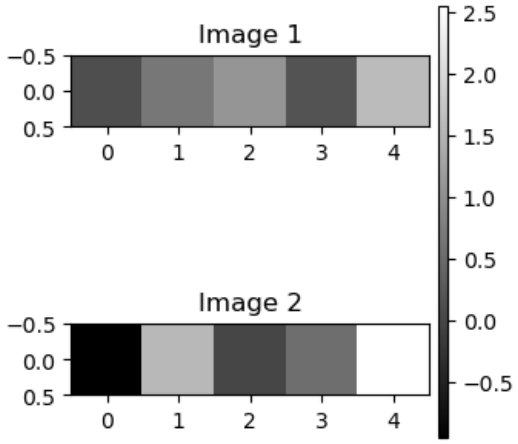


Figure 4.9. Encodings of the 2 samples using $VAE - 5$

Figure 4.10. A square from the test set

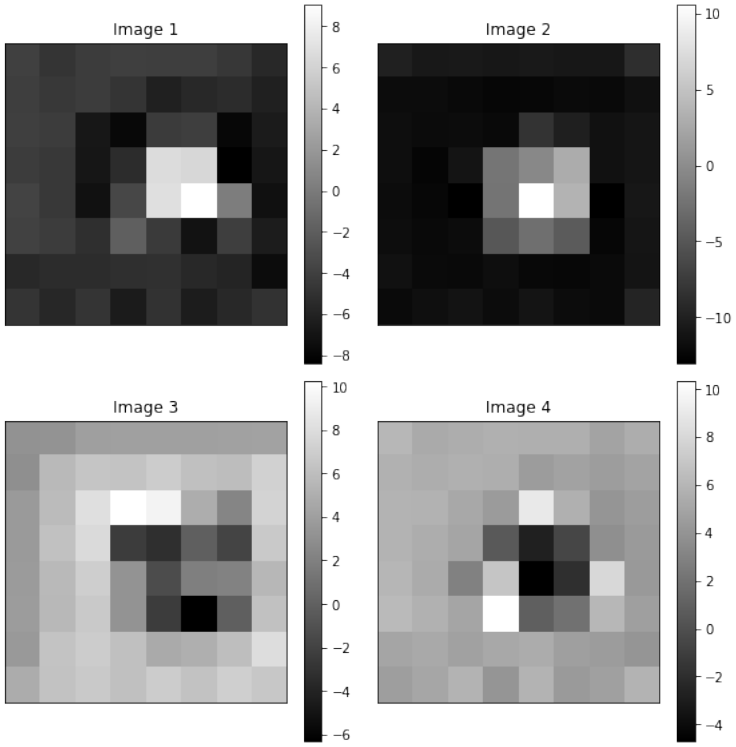


Figure 4.11. 4-channel encoding of the previous square using $KL - VAE - 4 - 16 - 16$

0.4262]. Except for the first component, the β encodings of the 2 illustrated samples are closer. As this pattern was observed very frequently, we can safely say that this is due to the disentangling property that $BVAE - 5$ enjoys. The first latent component is most likely strongly dependent on the shape, although further analysis using disentangling metrics such as β score, Factor score, or Mutual Information Gap is needed to provide more confidence to this claim. We may also want to look at fully CNN-based VAEs to see whether they indeed preserve structure. Consider the square example on figure 4.10 and its encoding on figure 4.11, we can clearly see that that the shape is still present in the encodings themselves even if it may be a little blurry after several convolutional layers.

4.3 Pilot experiments : Diffusion model study

We now proceed to the second parts of pilot experiments which aim at understanding diffusion models a bit more.
To evaluate the performance of a diffuser, we need to use some quality metrics or axes to guide us. Similarly to VAEs, there is no unique sufficient metric to do so. Even worse, generative models tend to be very hard to evaluate. Computing the likelihood of generated samples is usually intractable for all generative model types, if they even admit a likelihood.

We thus choose these 3 essential and informative axes to evaluate diffusion models:

1. Denoising loss: Diffusion models are trained by removing noise from an image to which a certain level of noise was added. The resulting Mean Squared Error (MSE) between the target image and the network output serves as a fundamental metric for assessing a diffusion model's performance.

2. Sampling capabilities: The primary goal of a diffusion model is to generate new and plausible data given a dataset. Evaluating the sampling capabilities involves more qualitative assessments, where visual inspection of generated samples is essential. By comparing the generated samples to the original data samples, we can gain insights into the model's ability to capture the underlying data distribution accurately. A high-quality diffusion model should produce diverse, realistic, and coherent samples that resemble the data distribution it was trained on. Additionally, assessing sampling diversity, i.e., the model's ability to generate a wide range of different samples, is crucial to ensure its capacity for creative and versatile data generation.

3. Modes coverage : Data sets distributions are usually multi-modal where each feature can have several modes, a good diffusion model should have learnt all modes of the distribution and generate them in proportions similar to those of the data set. A simple way to check for modes coverage is to visually inspect the modes of generated samples or feed them to a network trained to predict the modes of each feature given a sample. Practically we'll ask well-trained classifiers and regressors to guess the features of each generated samples and stack those predictions. If the diffusion model has correctly learnt the data distribution and if the regressors and predictors are very good, then we should observe uniform histograms of each stacked feature predictions. We should note however that these modes coverage histograms are a necessary but not sufficient conditions to assure convergence of the diffusion model. Indeed, consider a diffusion model that generates $\frac{1}{3}$ of squares, hearts and ellipses but they are all 3 always appearing at a particular range in the x-position. For example, squares could appear in $[0, 0.33]$, hearts in $[0.34, 0.66]$ and ellipses in $[0.67, 1]$. In this case, the distribution of the x-position and the distribution of shapes would both be uniform over their range however the joint distribution P(shape, x-position) would not! The condition we're using is weak because 5 variables were jointly used in the data generating process but we're only looking at the marginals.

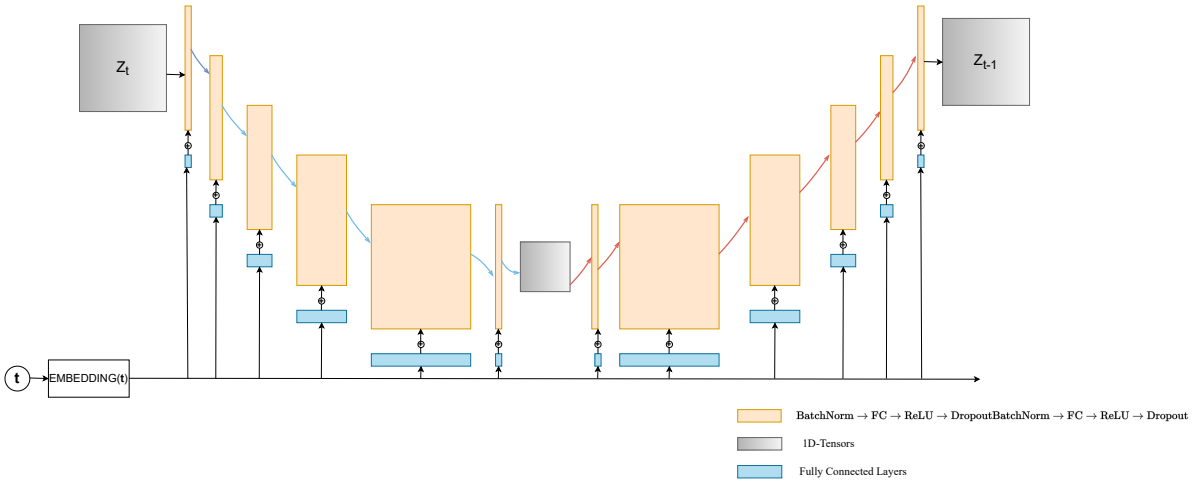


Figure 4.12. Typical diffusion model we used for 1D latent variables as in $BVAE$ or $VVAE$ space. It mainly consists of fully connected layers.

4.3.1 Latent structure importance

Experiment goal and context

In this section, we're interested in learning what structure type is most suited to an appropriate diffuser. What we mean by appropriate is that we want to use diffusers architectures appropriate to the structure of the input data. If it is a 1D tensor with no spatial relationships, then an MLP seems well suited while if the data is image-like, a CNN is a better choice.
In regard of understanding latent structure importance for latent diffusion models, we decide to evaluate 3 VAEs:

1. $DIFF-BVAE-1024$: an MLP-based diffuser working in $BVAE$ space with 1024 latents
2. $DIFF-CONV-VAE-4-16-16$: a CNN-based diffuser working in $CONV-VAE-4-16-16$ space as defined previously
3. $DIFF-KL-VAE-4-16-16$: a CNN-based diffuser working in $KL-VAE-4-16-16$ space as defined previously

Before comparing these diffusion models, we'll first briefly describe their architectures. As the latent variables have different dimensions depending on the latent space in which they live, the diffuser working in that space should naturally be designed in order to take advantage of that structure and dimensional. That is why for 1D unstructured latent variables we built an MLP-like diffuser while for 2D or 3D structured latent variables we used CNN-based diffusion models. They are respectively depicted on figure 4.12 and figure 4.13.

All architectural details are available in the section "Some notes on diffusers".

Results

The denoising test losses of the three diffusion models can be seen on table 4.5

Samples generated by each of the three diffusion models can respectively be seen on figures

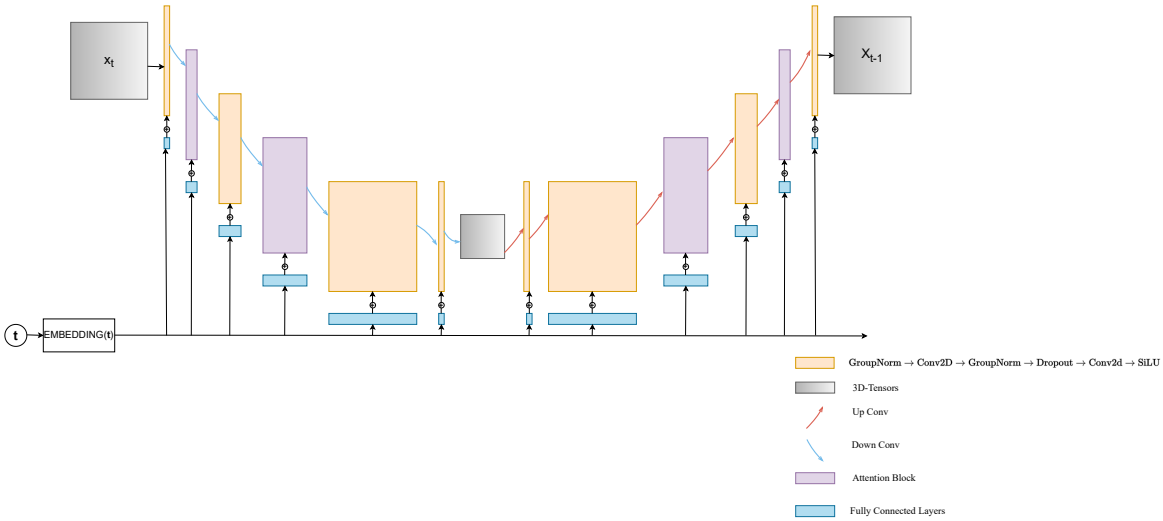


Figure 4.13. Typical diffusion model we used for 3D latent variables as in $CONV - VAE - 4 - 16 - 16$ space. It mainly consists of convolutional layers.

Model	Denoising Test Loss
$DIFF - BVAE - 1024$	4605.60
$DIFF - CONV - VAE - 4 - 16 - 16$	1269.42
$DIFF - KL - VAE - 4 - 16 - 16$	12.81

Table 4.5. Axis 1: Denoising test losses for different structures of diffusion models. These were computed by randomly diffusing test data and asking the diffusion model to remove the noise, the loss is then simply the MSE between the clean sample and the denoised sample.

4.19, 4.15 and 4.16.

The histograms for mode coverage of $DIFF - BVAE - 1024$ can be seen on figure 4.17 while those of $DIFF - KL - VAE - 4 - 16 - 16$ are on figure 4.18.

Interpretation

From table 4.5, it is clear that the KL diffusion model is the one that best removes noise from noisy samples, it however does not guarantee that it will produce the most likely samples. We remind that $DIFF-KL-VAE$ and $DIFF-CONV-VAE$ have the very same architecture, they simply live in different latent spaces therefore, the architecture is not the reason for this out performance.
The samples generated by $DIFF - KL - VAE$ are also the best and by far. $DIFF - BVAE - 1024$ samples 4.19 are however unexpected, they are not even as good quality as the samples **??** generated by $BVAE - 1024$ which samples latents from a normal distribution. We believe this is due to the high dimensionality of the data as we'll see that smaller diffusers produce very decent samples.
Finally $DIFF - KL - VAE$ produces samples whose features are quite uniformly distributed in the data set ranges except the for the shape as the model produces a bit too many squares and not enough hearts.
We believe these 3 cases are not enough to draw a conclusion about the latent space structure importance in LDMs but we might give a few reasons why $DIFF-KL-VAE$

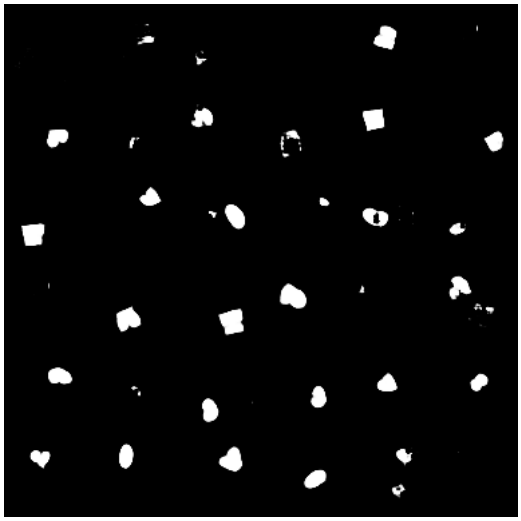


Figure 4.14. Samples generated by $DIFF - BVAE - 1024$

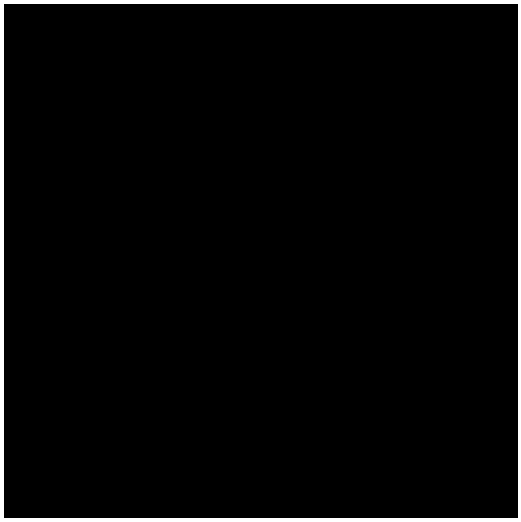


Figure 4.15. Samples generated by $DIFF - CONV - VAE - 4 - 16 - 16$

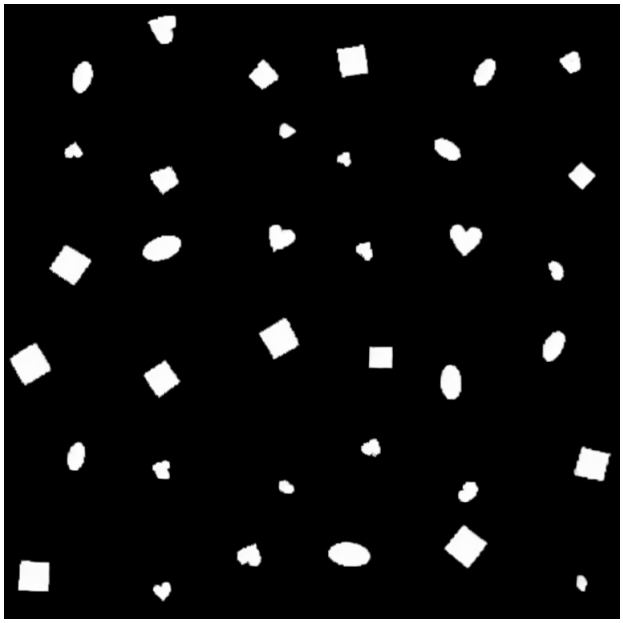


Figure 4.16. Samples generated by $DIFF - KL - VAE - 4 - 16 - 16$

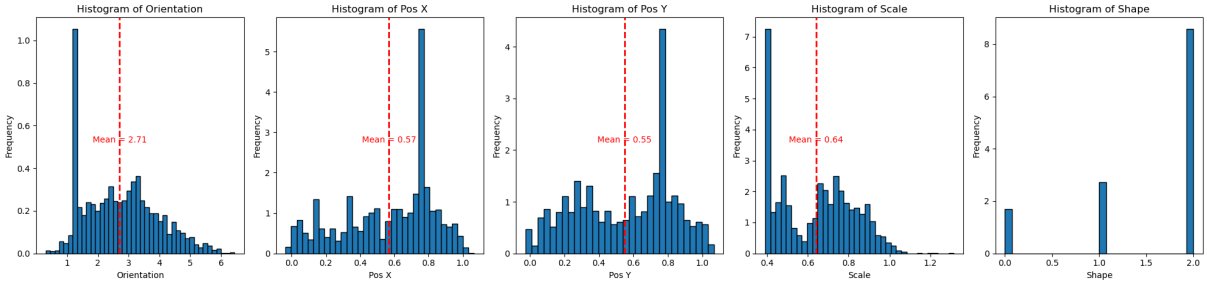


Figure 4.17. Histograms generated by sampling 50k sprites in $BVAE - 1024$ space, decoding them and then passing them to some well-trained classifiers/regressors. As the orientation was poorly understood by the corresponding regressor network, it should not be given attention. Regarding the other variables, we wished for a uniform distribution over their ranges in the dataset, that is, scale should be U[0.5,1].

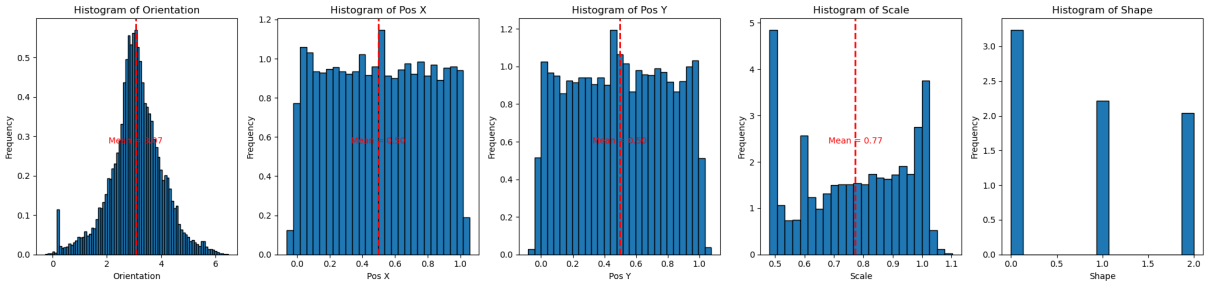


Figure 4.18. Histograms generated by sampling 50k sprites in $KL - VAE - 4 - 16 - 16$ space, decoding them and then passing them to some well-trained classifiers/regressors

is pretty good. We believe the most important reason for this performance is that the $KL-VAE$ is a pretty complex VAE and it was trained with several losses and on several data sets with a very large domain. The losses included GAN losses, LPIPS loss, discriminator loss, KL distance and reconstruction loss. We believe training the VAE with a perceptual loss term as well could make the latent space more comfortable for diffusion models but we leave this idea to future work.

4.3.2 Latent size importance

Experiment goal and context

The latent size is a crucial design choice when building a latent diffusion model for the following reasons :

1. Expressiveness and capacity : The latent size significantly influences the diffuser's capacity to represent data distributions. Larger latent sizes offer higher expressiveness enabling the diffuser to capture complex patterns and detailed features in the data. However there is one problem : large latent sizes can overfit faster than small latents. On the other hand, smaller latent spaces might limit the model's ability to capture fine-grained details in the data.
2. Sample generation : The latent size impacts the diversity and quality of generated samples. Larger latent spaces tend to produce more diverse and varied samples due to their increased expressiveness. This leads to higher quality generated samples in terms of diversity with respect to the data set we're trying to learn. On the other hand, smaller latent spaces might produce more consistent, but potentially less diverse, samples during generation.
3. Interpretability : In some cases, smaller latent spaces can be more interpretable. A reduced latent size might lead to more disentangled representations, where individual dimensions of the latent space correspond to specific generative factors.
4. Computational efficiency: The choice of latent size also directly impacts the computational efficiency of the model. Larger latent spaces require more memory and computational resources during training and inference. Smaller latent sizes can lead to faster training times and lower memory requirements.

In regard of understanding latent size importance for latent diffusion models, we decide to evaluate 3 VAEs:

1. $DIFF-BVAE-1024$: an MLP-based diffuser working in $BVAE$ space with 1024 latents
2. $DIFF-BVAE-256$: an MLP-based diffuser working in $BVAE$ space with 256 latents
3. $DIFF-BVAE-16$: an MLP-based diffuser working in $BVAE$ space with 16 latents
4. $DIFF-BVAE-5$: an MLP-based diffuser working in $BVAE$ space with 5 latents
5. $DIFF-BVAE-5-Untrained$: an MLP-based diffuser working in $BVAE$ space with 5 latents which will not be trained, it may serve as a benchmark

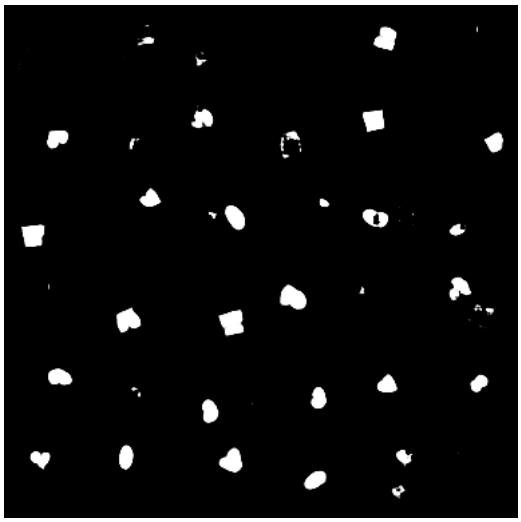


Figure 4.19. Samples generated by $DIFF - BVAE - 1024$

Results

The denoising test losses of the 5 diffusion models can be seen on table 4.2.

Model	Denoising Test Loss
$DIFF - BVAE - 1024$	4605.60
$DIFF - BVAE - 256$	4590.39
$DIFF - BVAE - 16$	4321.80
$DIFF - BVAE - 5$	3934.41
$DIFF - BVAE - 5 - Untrained$	4932.97

Table 4.6. Axis 1: Denoising test losses for different sizes of diffusion models. These were computed by randomly diffusing test data and asking the diffusion model to remove the noise, the loss is then simply the MSE between the clean sample and the denoised sample.

The samples generated by these diffusion models can be seen on figures 4.19, 4.20, 4.21, 4.22 and 4.23.

The modes coverage histograms of each model can be seen on figures 4.17, 4.24, 4.25, 4.26 and 4.27.

Interpretation

From table 4.6, it is clear that the smaller models are better at removing noise and we remind that we report the mean noise over the whole test averaged over the components therefore it is not simply due to them having less components. As we can see on figures 4.19, 4.20, 4.21, 4.22 and 4.23, the smaller models also tend to produce better samples. Note that even the untrained diffusion model $DIFF - BVAE - 5 - Untrained$ produces very good samples. We believe this is due to the VAE more than to the diffuser, we observed while training VAEs that smaller VAEs tended to be closer to the standard normal. Thus whatever the diffuser outputs usually falls within this standard normal range and thus the VAE decodes these latents to likely samples.
Regarding the feature distribution histograms on figures 4.17, 4.24, 4.25, 4.26 and 4.27,

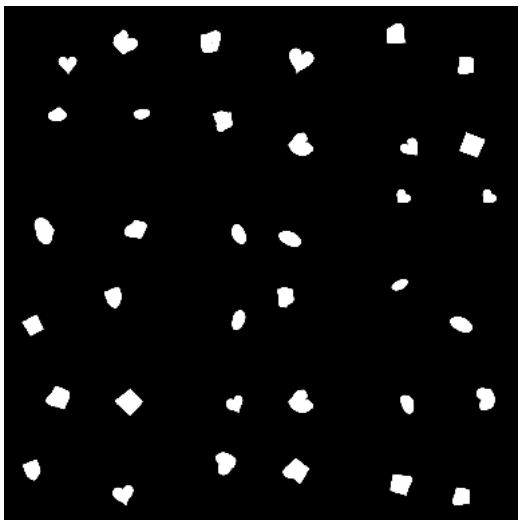


Figure 4.20. Samples generated by $DIFF - BVAE - 256$

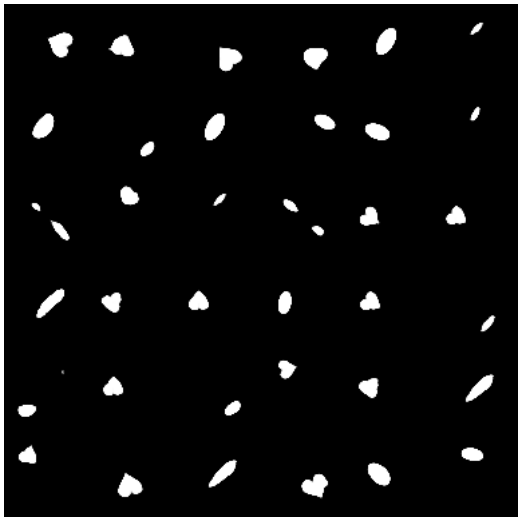


Figure 4.21. Samples generated by $DIFF - BVAE - 16$

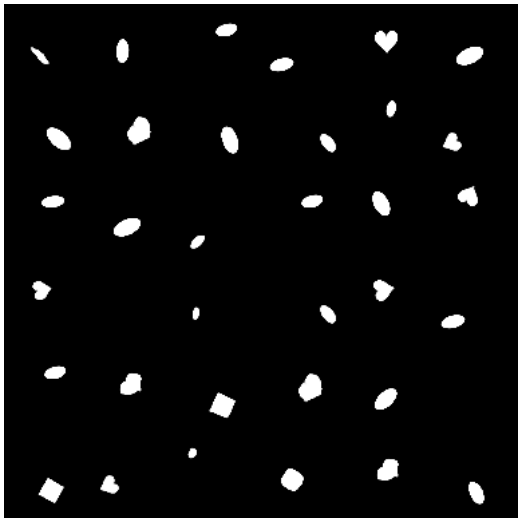


Figure 4.22. Samples generated by $DIFF - BVAE - 5$

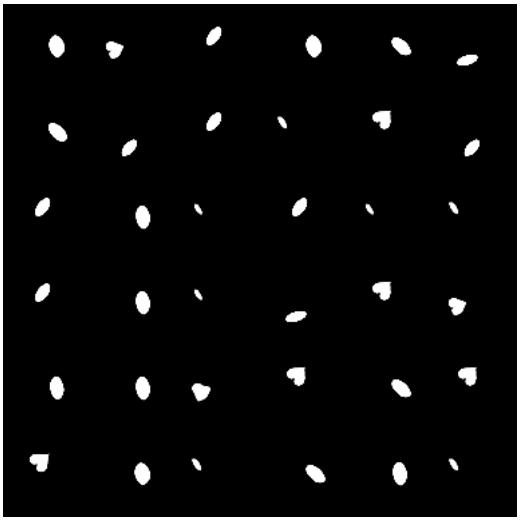


Figure 4.23. Samples generated by $DIFF - BVAE - 5 - Untrained$

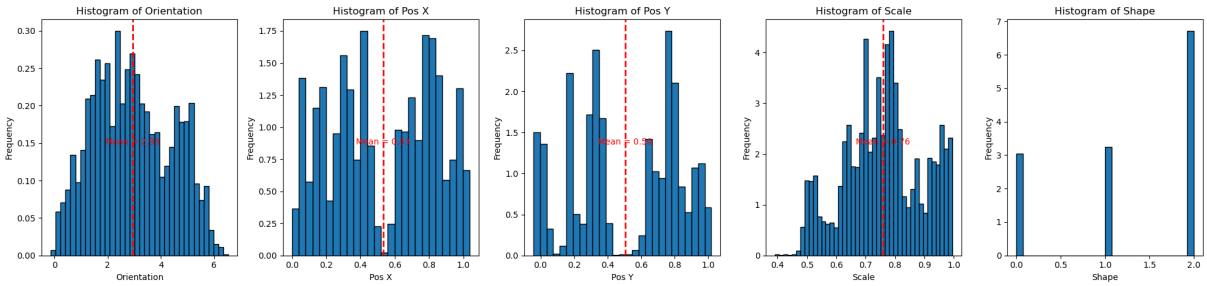


Figure 4.24. Histograms generated by sampling 50k sprites in $BVAE - 256$ space, decoding them and then passing them to some well-trained classifiers/regressors

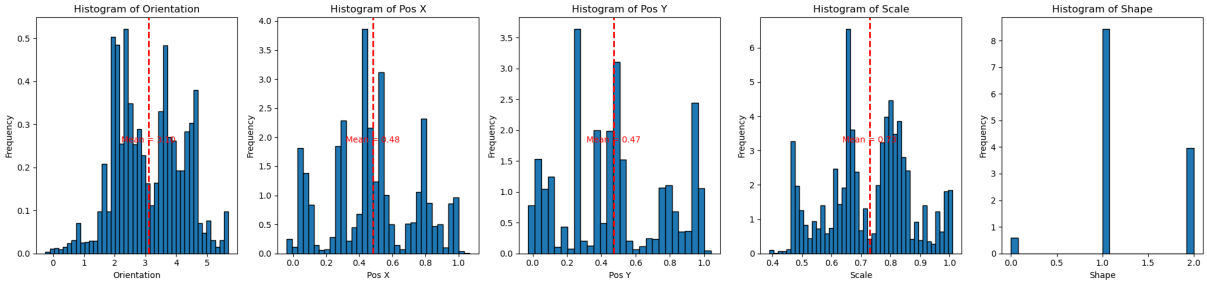


Figure 4.25. Histograms generated by sampling 50k sprites in $BVAE-16$ space, decoding them and then passing them to some well-trained classifiers/regressors

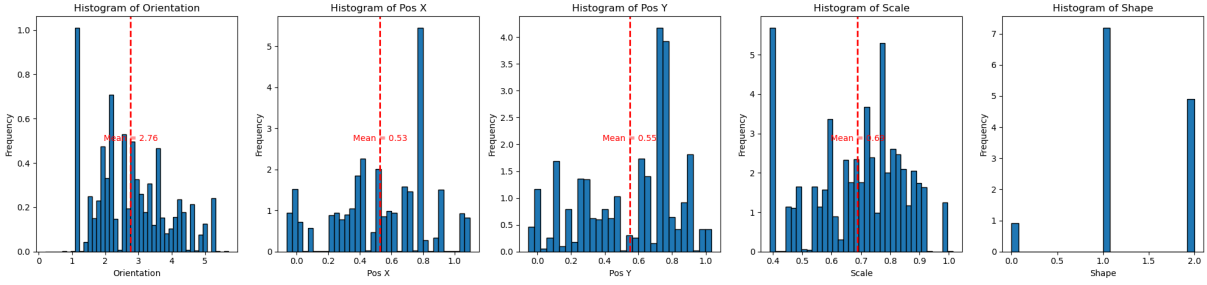


Figure 4.26. Histograms generated by sampling 50k sprites in $BVAE-5$ space, decoding them and then passing them to some well-trained classifiers/regressors

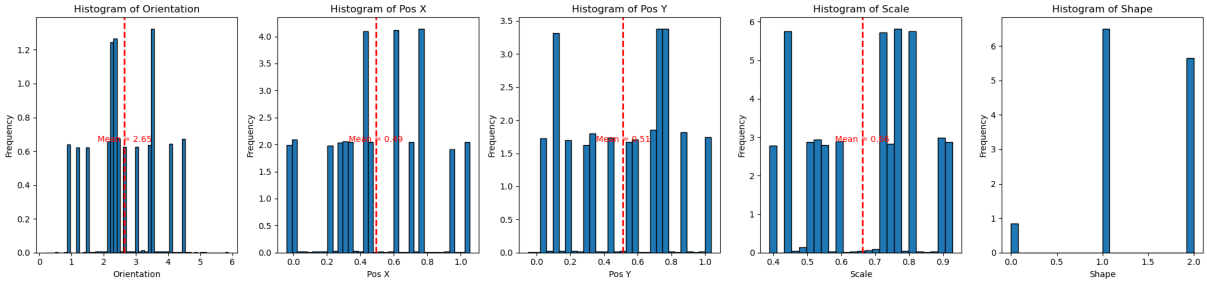


Figure 4.27. Histograms generated by sampling 50k sprites in $BVAE - 5$ space **using a non trained diffuser**, decoding them and then passing them to some well-trained classifiers/regressors

it is clear that no diffusion model has successfully learnt the data distribution as all histograms are either far or very far from a uniform distribution. It is however reassuring that the untrained model has the worst feature distributions 4.27.
We believe the main conclusions to draw from these results are the following. First and most obvious, latent diffusion models performances are heavily dependent upon the quality of the latent space defined by the VAE. Second, we believe the latent regions that lead to meaningful decodings have their size strongly decreasing with the number of latent variables. Note that it is not a simple task to dissociate the culprit when it comes to weak generation as it can be the VAE, the diffuser or even both. Third, smaller VAEs tend to learn more factorized representation of the data, this could highly ease the diffusion models to learn the latent data distribution $p(\mathbf{z}|\mathbf{x})$.

4.3.3 Stochasticity and regularity importance

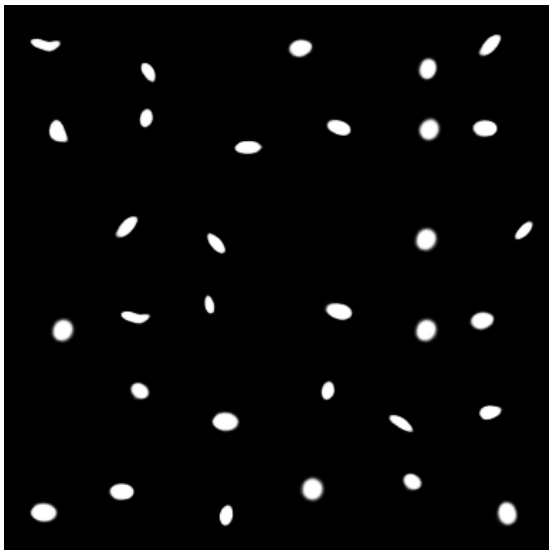
Experiment goal and context

Another interesting question to ask is whether one should use VAEs or simple AEs for encoding and decoding the data. As we said several times, AEs encode and decode point to point, the neighbourhood around a data point might not be encoded close to where this data point will be encoded. This happens simply because there are no neighbourhood constraints and generally there is no structure nor regularity in AEs latent space. Training a diffuser on AEs latent representation of the data set and then using it for inference would, according to us, yield to 2 possible scenarios :

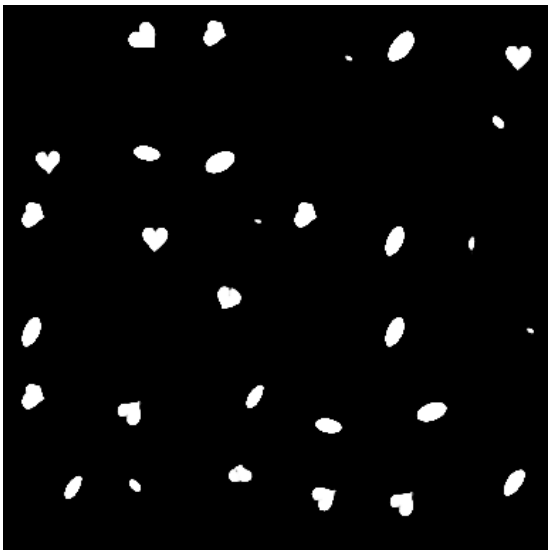
1. The diffusion model ended up generating a latent very close to the latent of a training point, decoding this image would thus yield an image very close to the training point, i.e, the image we were referring to.

2. The diffusion model ended up generating a latent not close enough to any of the training points and decoding it resulted in a meaningless image.

In the first scenario, the generated image is likely to simply belong to the data set and in the second scenario, the generated image has very little likelihood of belonging to the distribution we were targeting. It seems that using AEs is not very beneficial in any of the 2 cases and it also seems that VAEs would perfectly address these limitations. However, despite this, [5] claim to use AEs in their stable diffusion implementation. As we'll see, using simple AEs is not enough. Their latent space was actually decent because



(a) Samples generated by $DIFF-VAE-5$



(b) Samples generated by $DIFF-AE-5$

Figure 4.28. Comparison of generated samples

they didn't use the traditional AE loss but rather a combination of perceptual loss and a patch-based adversarial objective that is described in their paper.
In regard of understanding regularity importance of the latent space for latent diffusion models, we decide to evaluate 2 VAEs:

1. $DIFF-VAE-5$: an MLP-based diffuser working in VAE space with 5 latents
2. $DIFF-AE-5$: an MLP-based diffuser working in AE space with 5 latents

Results

The denoising test losses of the 2 LDMs can be seen on table 4.8.

Model	Denoising Test Loss
$DIFF-VAE-5$	3789.46
$DIFF-AE-5$	4623.00

Table 4.7. Axis 1: Denoising test losses for 2 diffusion models. These were computed by randomly diffusing test data and asking the diffusion model to remove the noise, the loss is then simply the MSE between the clean sample and the denoised sample.

Samples generated by the 2 diffusion models can be seen on figures 4.28a and 4.28b.

The histograms for the modes coverage can be seen on figures ref and ref

Interpretation

From table 4.3, it seems $DIFF-VAE-5$ is the stronger diffusion model yet the samples 4.28a and 4.28b tell otherwise. We may be tricked into thinking that $DIFF-AE-5$ is a better generative model however it actually always generates the same batch of images. It is actually not generative at all as we can see on the histograms 4.30. On the other hand, it is quite surprising that $DIFF-VAE-5$ failed to produce high-quality samples given

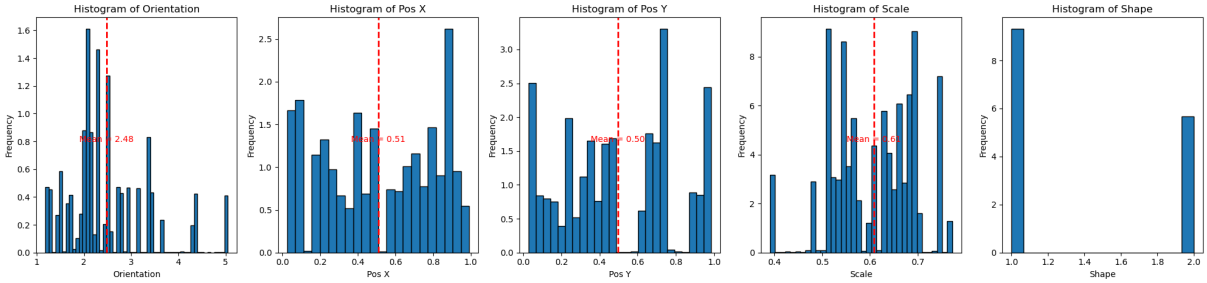


Figure 4.29. Histograms generated by sampling 50k sprites in $VAE - 5$ space, decoding them and then passing them to some well-trained classifiers/regressors

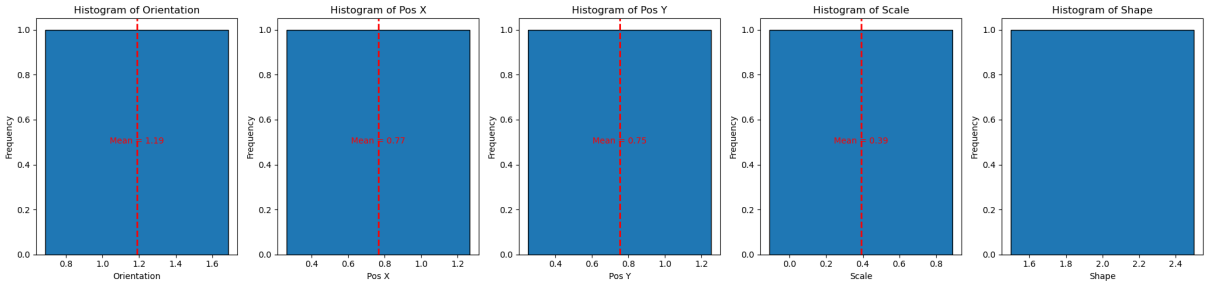


Figure 4.30. Histograms generated by sampling 50k sprites in $AE - 5$ space, decoding them and then passing them to some well-trained classifiers/regressors. We note that this histogram does not generate any square.

that even its VAE was able to. Indeed 4.31 where samples were generated by the VAE led to serious samples. Unfortunately, we found no satisfying answer to these results.

4.3.4 Latent representation importance

Experiment goal and context

Finally we're interested in learning whether it is easier for the same diffuser to learn a vanilla latent distribution or a β disentangled latent distribution. Particularly, it is worth noting that as the β value increases, the capacity of the latent space gets more limited in representing complex patterns as it forces more disentanglement. Thus, there could be a trade-off between disentanglement and expressiveness in β-VAE-based diffusers. In

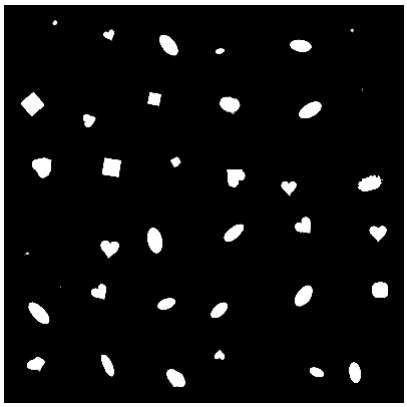


Figure 4.31. Samples generated by $VAE - 5$

regard of understanding representation importance of the latent space for latent diffusion models, we decide to evaluate 2 VAEs:

1. $DIFF-VAE-5$: an MLP-based diffuser working in VAE space with 5 latents
2. $DIFF-BVAE-5$: an MLP-based diffuser working in $BVAE$ space with 5 latents

Results

Model	Denoising Test Loss
$DIFF-VAE-5$	3801.61
$DIFF-BVAE-5$	3934.87

Table 4.8. Axis 1: Denoising test losses for 2 diffusion models. These were computed by randomly diffusing test data and asking the diffusion model to remove the noise, the loss is then simply the MSE between the clean sample and the denoised sample.

Samples generated by these 2 models can be seen on figures 4.28a and 4.22.
Their corresponding histograms can be seen on figures 4.29 and 4.26.

Interpretation

Even though $DIFF-BVAE-5$ has a higher test error, it produces far better samples than $DIFF-VAE-5$. Also, it has learnt all modes of the data distribution although in wrong proportions as it generates very few squares where $DIFF-VAE-5$ does not generate any.
We are tempted to conclude that a disentangled latent space allows diffusion models to better converge to the latent data distribution.

4.3.5 Some notes on diffusers

We spent a long time implementing and training diffusers for this thesis and while doing so we observed many phenomenons that do not fit somewhere else in the writing but which we believe are worth of talking about. That is why in this section, we briefly describe these notes.
This section might be skipped if the reader simply desires to get the research question results.

Architectures of diffusers

As we have shown on the network drawings and said so far, when the latent space is composed of 2d square latents which preserved the images structure we would use an appropriate diffuser, that is, a U-net with time inputs. The main library we used for the project, Hugging Face diffusers, implemented such a U-net2D where the depth, kernel sizes, time embedding type and other parameters could be defined by the user.
Unfortunately, no appropriate diffuser was implemented for 1D latents except a U-net1D relying on 1d kernels however the latter was of no use to us since it assumed spatial relationships between latents components. This assumption is completely wrong in the VAE or $BVAE$ latent space as they contain FC layers at the end. We thus needed to implement a diffuser for that is fed a 1d tensor and a timestep. We thus rapidly decided

to use an MLP or a ResMLP.
Let $f(\mathbf{z_t}, t)$ be the MLP tasked with removing the noise from $\mathbf{z_t}$ to recover $\mathbf{z}$ where $\mathbf{z_t}$ has shape $(LatentSize,)$. There are several ways of injecting the scalar timestep into the network such that it knows the level of noise it should try to remove.

1. Concatenate the raw timestep to the flattened noisy sample before the first layer as shown in figure (ADD FIGURE) but this option poorly works for 2 reasons. First, as the concatenated input gets deeper and deeper in the network during the forward pass, the timestep will tend to be forgotten. This results in its corresponding weights in the network not be updated enough because of the vanishing gradient as the timestep is far from the zone reached by activation functions. A solution to that would be to concatenate the timestep at each and every layer of the network as shown in fig (ADD FIGURE) Second, the timestep information alone is not well captured by the network. Similarly to transformers, positionally embedding the timestep will allow the network to better capture it, even if the information quantity didn't increase by embedding it, it is simply a different representation of the same data that eases the job to the network.

2. The second option and the one we aimed for was to simply add trained positional embeddings to $\mathbf{z_t}$ at each layer . That is we first positionally embed the timestep to an array $embedding(t)$ of shape $(D,)$ where D is the embedding dimension.
We first map the scalar t to an array whose i-th component is

$$\text{emb}(t, i) = t \cdot \left(\frac{-\log(10000) \cdot i}{\frac{D}{2} - i} \right) \quad \forall i \in \{0, 1, 2, \ldots, \frac{D}{2}\} \tag{4.1}$$

$$\text{embedding}(t) = (\sin(\text{emb}(t)), \cos(\text{emb}(t))) \tag{4.2}$$

The task is not to define an MLP $f(\mathbf{z_t}, t)$ anymore but rather $f(\mathbf{z_t}, embedding(t))$. To let the network handle the embeddings as freely as it wants and to fit dimensions, the embeddings themselves are passed through a 2-layer MLP before adding them to $\mathbf{z_t^l}$ where l indicates the l-th layer of the MLP. Said more simply, each layer first transforms the time embeddings and then processes $\mathbf{z_t^{l-1}}$ such that they have the same shape. A detailed view is shown on figure (ADD figure). Given the positional embeddings, a lot of architectural paths were possible, we explored 3 and compared their performances as diffusers

1. $DIFF-MLP$: An MLP that processes $\mathbf{z_t}$ and $embeddings(t)$ to add them together and keep passing them to next layers.

2. $DIFF-ResMLP$: A ResMLP that processes $\mathbf{z_t}$ and $embeddings(t)$ to add them together and keep passing them to next residual layers. Residual layers preserve the size of the input as shown on figure (ADD FIGURE of resMLP).

3. $DIFF-ConcMLP$: An MLP that concatenates trained $embeddings(t)$ to $\mathbf{z_t}$ and keeps passing them to next layers.

The three architectures can be seen on figure (ADD FIGURE).
Show losses of the 3

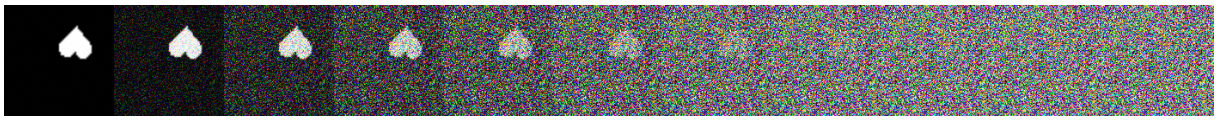


Figure 4.32. The leftmost image corresponds to a sample from the data set and each sample on its right corresponds to a noisier version of it. Specifically, having set T=1000, the first noisy sample has a noise level $t = 100$, the second $t = 200$, etc... The direction of forward diffusion is left to right

Figure 4.33. The leftmost image corresponds to a generated sample and the rightmost image corresponds to gaussian noise. As the direction of backward diffusion is right to left, following this direction one can see all the intermediary samples generated during the backward process. At $t = 600$, the location is not as easy to spot as in the forward.

Some examples of diffusion

We have so far introduced and studied diffusers a bit without even looking at what actually happens under the hood when one forward diffuses a data sample or when one backward generates gaussian noise. In this small section we want to give a more practical look at the intermediary samples as well as the importance of the network schedule.
Consider for example figure 4.32 where a data point was forward diffused for 10 increasing levels of noise. As we can see on the sample in the middle-right, where $t = 600$, the location of the shape can be guessed fairly well but classifying its shape already seems tougher since it quickly looks more and more like a circle or an ellipse.

We now consider the backward example on figure 4.33 where gaussian gaussian was backward diffused in 100 inference steps but we here consider only 10 of them as using 10 inference steps only would lead to low-quality samples. In this direction, it seems the location is a bit tougher to guess from the sample at $t = 600$ meaning that the backward hasn't properly learned the reverse of the forward process even though it was trained to do so. Note that we're learning probability distributions so it does not make much sense to draw such a strong conclusion from samples example. However, we proceed with this approach because we have observed this particular pattern recurring multiple times when re-running the experiment on different samples.

The variance schedule used for the experiments is the following one : 4.34. It is a simple linear schedule going from (0,0) up to (T,0.02). If it were to go to (T,0.001), the variance at the end would not be large enough and the samples at $t = T$ would not be gaussian enough. This would violate the assumptions of diffusion models and hopes of converging to a good generative model would be null.

Training diffusers

In this work we trained a lot of diffusers which led us to making several possibly useful observations.
First, it seems that having a low stable loss is a necessary but not sufficient condition for

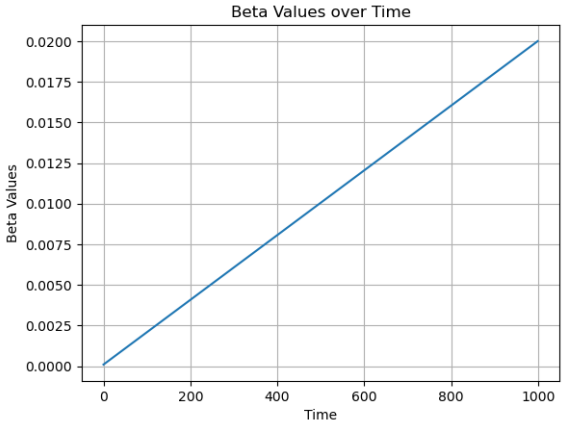


Figure 4.34. β values of the noise scheduler used

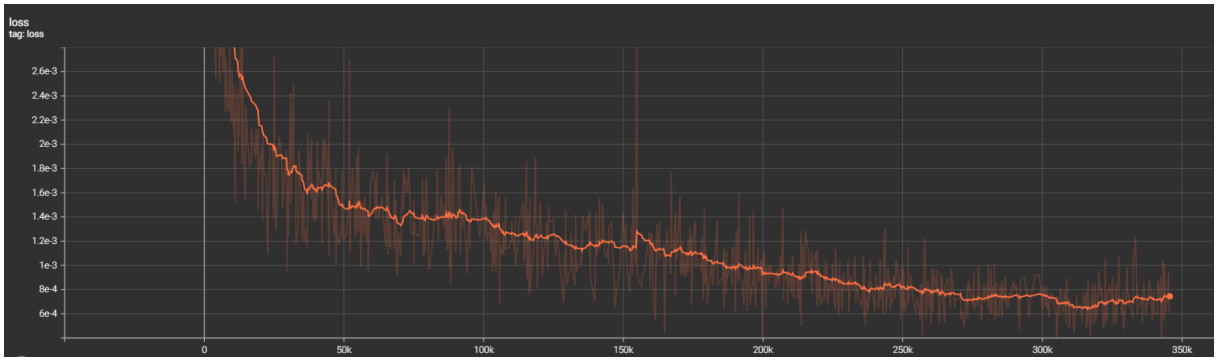


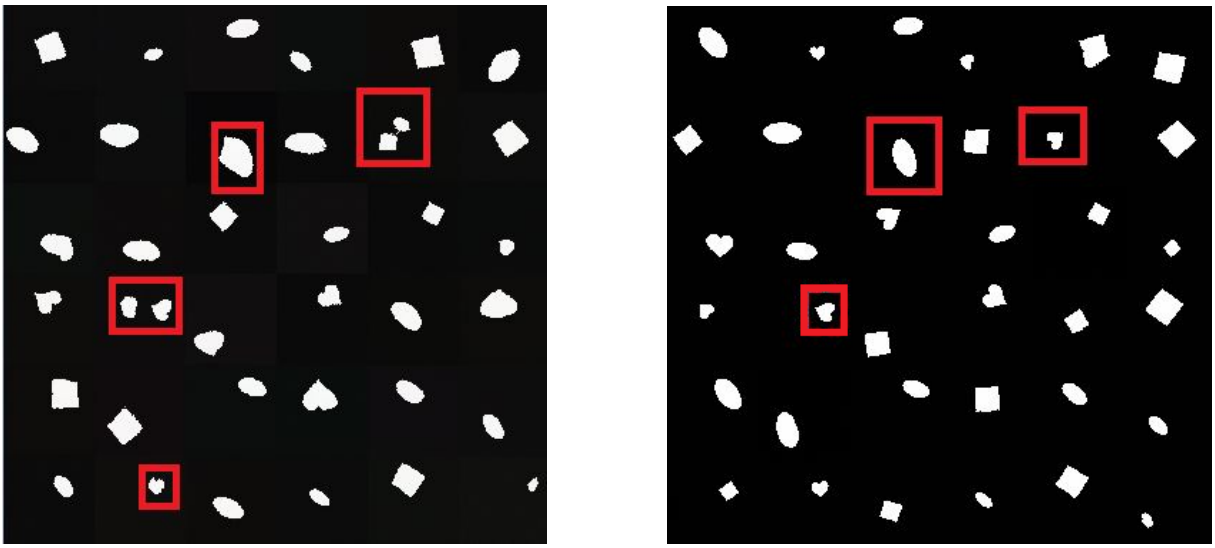
Figure 4.35. Training loss of the model

diffusion models to generate high-quality samples. Indeed consider the example where we train a diffusion model on the dSprites dataset in pixel space, the training loss can be seen on figure 4.35. We now look at the samples generated after 200k and 350k steps where the loss went from 9.8461e-4 down to 7.5258e-4, a relatively small decrease given the loss started at 1e-2. Samples generated after 200k steps may be seen on figure 4.36a while those generated after 350k steps are on figure 4.36b. Clearly, even though the loss has barely changed, the quality of generated samples is much better at 350k learning steps. The denoising loss alone is not a sufficient criterion to assess the convergence and quality of a diffusion model.

show some losses to compare the 3 archis

4.4 Experiments foreword

There is no single experiment within our current knowledge that would be sufficient to answer the research question. Thus, we undertook the task of conducting several experiments. The underlying objective of each experiment is to ascertain whether certain information exists at time 't,' where 't' belongs to the set of time instances 0, 1, 2, ..., T. If a latent variable, in the context of diffusion chain theory, follows a purely isotropic Gaussian distribution, it inherently carries no informational content. However, if the latent variable exhibits even subtle characteristics, such as discernible shapes, reasonable positional estimates, or other features, then we can confidently conclude that it contains



(a) After 200k learning steps (b) After 350k learning steps

Figure 4.36. Diffusion model generated samples

certain pieces of information.

The individuals who evaluate these latent variables and attempt to deduce their features are, in fact, machine learning models like regressors or classifiers. The degree of confidence in their assessments is quantifiable through their average accuracy."

Notice that so far in this paragraph we haven't specified any direction in which diffusion goes. Indeed, our experiments will consider both the forward noisy diffusion direction and the backward generative diffusion direction. That is because we don't care whether the latents x_{700} seem to hold information going in the forward direction where information is destroyed or whether they seem to hold information going in the reverse-time generative direction. At optimality the distribution $p(\mathbf{x_t})$ defined by the forward chain and the distribution $q(\mathbf{x_t})$ implicitly defined by the backward chain should be the same. That is also why one should not be able to tell whether the samples on figure 4.37 are being generated or noised, one simply cannot tell the direction solely looking at the samples.

4.5 Experiment 0 : Feature conservation

4.5.1 Experiment summary

Our initial experiment was designed to determine whether certain information still remains present at time T. The setup is relatively straightforward: using the shape feature in the dSprites dataset, we observed the shape of an image. We then added noise to this image up to time $t = T$, and subsequently reversed the process by diffusing the noisy image back to t=0. If the resulting diffused images frequently reverted back to their original heart shape, it would suggest that the shape information persists at time T.

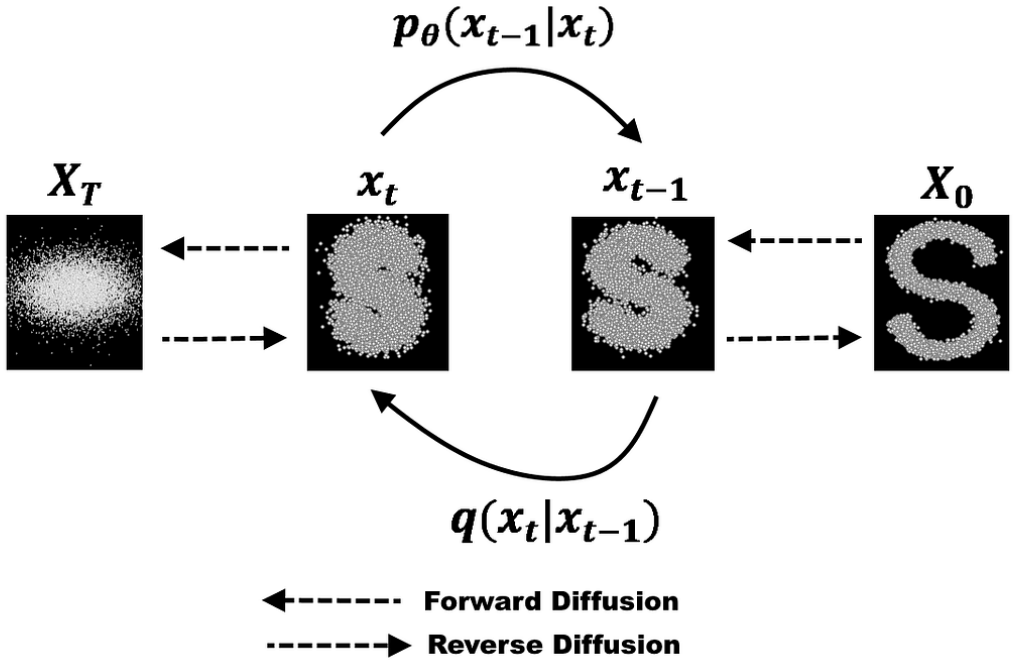


Figure 4.37. Looking at this diffusion chain, one cannot tell whether the images were generated by the diffusion models from left to right or whether they were denoised by the diffusion model in the forward direction from right to left. (Source: [1])

4.5.2 Experiment background

Assuming a perfect diffusion model, the latter should generate samples whose shape feature is uniformly distributed in (Square, Ellipse, Heart). That is, generating a lot of samples and perfectly classifying their shape should generate a shape distribution that is totally uniform. Let $\mathbf{x} \sim q(\mathbf{x_0})$ denote images sampled from the generative diffusion model, if it is optimal then it is identical to the data distribution $p(\mathbf{x_0})$. Under that same assumption, we have $p(Shape = Square|\mathbf{x}) = p(Shape = Ellipse|\mathbf{x}) = p(Shape = Heart|\mathbf{x}) = \frac{1}{3}$ as these are the data set proportions. If diffusing a heart up to T, backward diffusing it back to 0 and it came back as a heart with a proportion larger than $\frac{1}{3}$, then this could suggest that the shape information is somehow still present at T.
If we were to diffuse the data images up to t=10, we should expect each sample to return with the same shape since as we will show, for T=1000, diffusing an image up to t=10 barely modifies it.

Instead of focusing only on when $\hat{t} = T$, we'll test various time steps from $\hat{t} = 0$ to $\hat{t} = T$. We'll also analyze all 5 generative factors of the dSprites dataset: shape, posX, posY, orientation, and scale. Since these are numerical, we'll use MSE instead of accuracy. We'll explore traditional diffusion in pixel space and also in different latent spaces using a VAE or a variant. To ensure statistically significant results, we'll use thousands of samples. Given the volume of samples and the need to predict/regress features, we'll employ a classifier for shape and regressors for the other 4 numerical factor to predict the reconstructed sample's features.

4.5.3 Experiment procedure

Objective: The goal of this experiment is to investigate whether shape information is still present in the latent space S at time $\hat{t}$ after diffusion. We aim to observe the shape of an image, apply noisy diffusion up to time $\hat{t}$, and then perform backward diffusion from time $\hat{t}$ to time 0.

1. **Initialize the experiment:**
   - Prepare the dataset of images from the dSprites dataset with each image coming with its features about shape, posX, posY, orientation, and scale.
   - Prepare a well-trained shape classifier working in pixel space.
   - Prepare a well-trained diffuser working in space S.
   - Prepare a well-trained VAE if S is a latent space.
   - Set the maximum time step t=T to T=1000
   - Choose a noise scheduler, a VAE and a diffusion model trained in the VAE's latent space with that noise scheduler.
   - Set $NumberSamples = 0$, $Square2Square = 0$, $Heart2Heart = 0$ and $Ellipse2Ellipse = 0$
2. **Forward Noisy Diffusion:**
   - Randomly select an image $\mathbf{x}_0$ from the dataset.
   - If $S \neq$ Pixel Space :
     - Encode the data using the chosen VAE resulting in the latent representation $\mathbf{z}_0$.
     - Apply forward noisy diffusion to $\mathbf{z}_0$ from time $t = 0$ to $t = \hat{t}$ using the chosen diffusion model.
     - This results in a new noisy image $\mathbf{z}_{\hat{t}}$ at time $\hat{t}$ after diffusion.
   - Else :
     - Apply forward noisy diffusion to $\mathbf{x}_0$ from time $t = 0$ to $t = \hat{t}$ using the chosen diffusion model.
     - This results in a new noisy image $\mathbf{x}_{\hat{t}}$ at time $\hat{t}$ after diffusion.
3. **Backward Generative Diffusion:**
   - if $S \neq$ Pixel Space :
     - Perform backward generative diffusion on $\mathbf{z}_{\hat{t}}$ from time $t = \hat{t}$ to $t = 0$ using the same diffusion model.
     - Obtain the reconstructed image $\hat{\mathbf{z}}_0$ at time $t = 0$ after the backward diffusion and then decode it using the VAE to $\hat{\mathbf{x}}_0$.
   - Else :

- Perform backward generative diffusion on $\mathbf{x}_{\hat{t}}$ from time $t = \hat{t}$ to $t = 0$ using the same diffusion model.
- Obtain the reconstructed image $\hat{\mathbf{x}}_0$ at time $t = 0$ after the backward diffusion.

4. **Analyzing Shape Information:**
   - Use a shape classifier to predict the shape of $\hat{\mathbf{x}}_0$ and read the shape of $\mathbf{x}_0$ from the data set.
   - $NumberSamples++$
   - If the shape was initially a square and came back as a square :
     - $Square2Square++$
   - If the shape was initially a heart and came back as a heart :
     - $Heart2Heart++$
   - If the shape was initially an Ellipse and came back as an Ellipse :
     - $Ellipse2Ellipse++$
5. **Repeat Steps 2-4:**
   - Conduct the above steps for a significant number of samples from the dataset to obtain statistically significant results.
6. **Return results**
   - Return $\frac{Square2Square}{NumberSamples}, \frac{Heart2Heart}{NumberSamples}, \frac{Ellipse2Ellipse}{NumberSamples}$

Note 1: We here considered the case where we are interesting in seeing whether the categorical variable shape is preserved but we can do exactly the same regarding the 4 other quantitative variables. The only difference is that one has to use a regressor that outputs a scalar value and compute an MSE rather than an accuracy.
Note 2: We could have performed classification in either the pixel space and the latent space, this does not matter as long as the VAE and the classifier are well trained.

4.5.4 Experiment preparations

We decided to perform this experiment in pixel space and in 3 different latent spaces: $BVAE-5$, $BVAE-16$, and $KL-VAE-4-16-16$. As this experiment requires a well trained diffuser, VAE, and good classifiers/predictors, we extensively detail their training process as well as some tests during and post-training to assess their quality in the appendix. In the appendix there is for each diffuser model its architecture and training evidence of the 3 axes we chose in the pilot experiments on diffusers section : loss, sampling capabilities and modes coverage. In the appendix, one can also find the architecture and the training losses for each classifier/regressor as well as some test examples. Finally the appendix also contains the architecture and the training losses of each VAE as well as its reconstruction examples and some generated samples. One can find the models used for this experiment in Table 4.9.

Table 4.9. Models used in experiment 0

Model	**Pixel Space**	**BVAE-5**	**BVAE-16**	**KL-VAE**
Diffuser	DIFF-PIXEL	DIFF-BVAE-5	DIFF-BVAE-16	DIFF-KL-VAE
VAE	None	BVAE-5	BVAE-16	KL-VAE
Shape Classifier	SHAPE-PIXEL	SHAPE-BVAE-5	SHAPE-BVAE-16	SHAPE-KL-VAE
PosX Regressor	POSX-PIXEL	POSX-BVAE-5	POSX-BVAE-16	POSX-KL-VAE
PosY Regressor	POSY-PIXEL	POSY-BVAE-5	POSY-BVAE-16	POSY-KL-VAE
Scale Regressor	SCALE-PIXEL	SCALE-BVAE-5	SCALE-BVAE-16	SCALE-KL-VAE
Orientation Regressor	ORI-PIXEL	ORI-BVAE-5	ORI-BVAE-16	ORI-KL-VAE

One may have observed that in the procedure we specify that if there is a latent space, one shall decode the generated sample to the pixel space and thus all predictors in latent spaces seem irrelevant. We actually sometimes used them in our experiments either this one or the ones and whenever we'll make use of them we'll mention it.

4.5.5 Experiment results and discussion

Before delving into details, let's briefly outline the main conclusions drawn from this experiment.

Firstly, for low levels of noise, the reconstructed samples tend to retain the same features as the initial ones. As the noise levels increase to moderate levels, the initial sample's features gradually diminish. At high levels of noise, the reconstructed samples become entirely independent from the initial ones. Secondly, the time, speed and overall rate at which features tend to lose their information is not the same. Lastly, the space in which samples live directly impact the aforementioned measures.
We now proceed to analyzing the results of the experiment.

Results in pixel space are on figures 4.38, 4.39, 7.26, 7.27 and 7.28.

We observe on figure 4.38 that for all noise levels smaller than $t = 300$, the shape conservation is of 100%, that is all samples that are forward and then backward diffused do not change of shape. The behaviour starting at $t = 800$ is completely different as the initial shape information is not taken into account during the generation. Indeed, consider the blue curve that denotes "Square to square", for $t \geq 800$, it completely falls on the 33.333% square generation plateau. This plateau actually denotes the percentage of squares that the diffusion model generates when backward diffusing from gaussian noise. When the blue curve hits the plateau, this means that whether or not you started from a square in your forward process, that has no importance as a square will be generated 33.333 % of the time. The interval [300,800] is pretty interesting. We could have expected the shape information to abruptly disappear because at some point the gaussian noise was too strong, this would have led to a stair-case plot but this really does not happen. Instead, the shape information tends to smoothly, almost linearly, disappear between $t = 300$ and $t = 800$. For example, at $t = 600$, 50% of squares come back as squares and so do hearts and ellipses. As $50\% \geq 33.333\%$, it may suggest that at $t = 600$, the shape information is still present but it is not the only explanation. It could also be that given a noisy sample originating from a square, the diffusion model thought, from training experience, that the

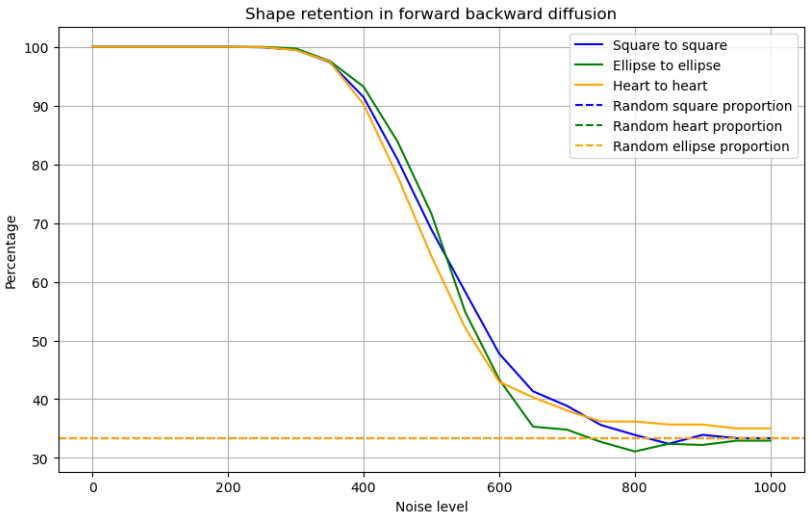


Figure 4.38. Shape retention using a diffusion model in pixel space. Samples were forward diffused up to a certain noise level and then backward diffused, a neural network would then classify the resulting shape. Each continuous curves denotes the fraction of initial samples that conserved its shape during the round trip. The dashed line on the other hands denotes the proportion of shapes that the model naturally generates. If the shape conservation ratio is equal to the generative shape ratio, this means the shape information was lost at this particular noise level or that can also mean it was not taken into account when generating.

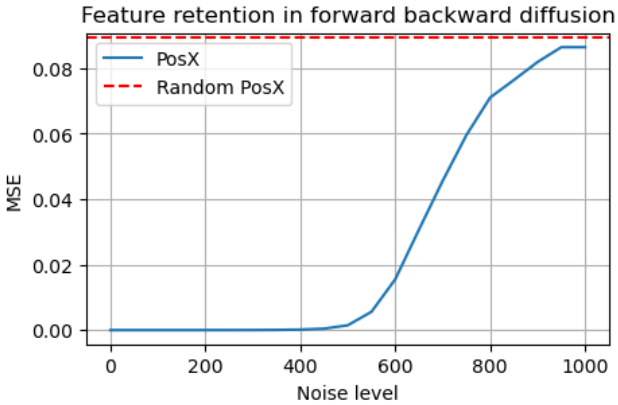
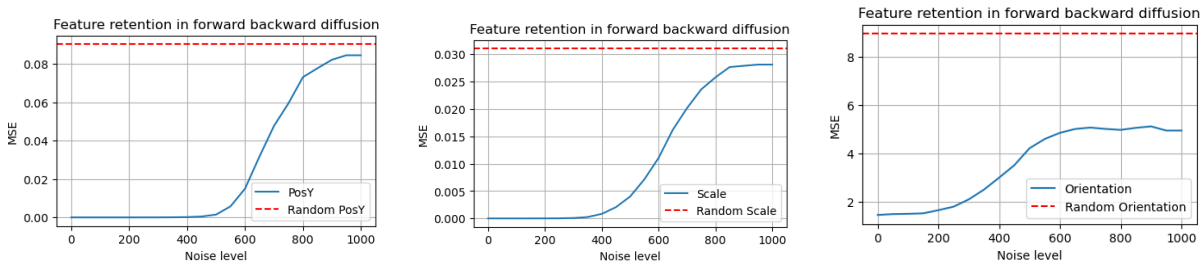


Figure 4.39. X-position retention using a diffusion model in pixel space. Samples were forward diffused up to a certain noise level and then backward diffused, a neural network would then regress the resulting x-position of the sample. The continuous curves denotes the MSE between the initial sample's x-position and the resulting sample's x-position. The red dashed line denotes the MSE committed when generating many random samples from the diffusion models, these have no mutual information with the initial sample.



(a) Y-position retention (b) Scale retention (c) Orientation retention

Figure 4.40. Retention using a diffusion model in pixel space

easiest way to generate a likely sample would be to generate a square.
We believe the reason for this almost-linear behaviour in the [300,800] interval is most likely due to the forward process variance schedule. The variance schedule is linear and noisy samples are a linear combination of the clean sample and the noisy sample with weight given by the variance schedule. As we'll see in the forward experiment, the variance schedule is a main player when it comes to information destruction and changing it will definitely change these curves.
The information on the position can also be somehow quantified as we look on figure 4.39. For $t \leq 450$, the MSE, or rather mean square difference, between the initial sample's x-position and the generated sample's x-position is very small. That is, when we noise a sample with little noise and backward diffuse it, its position does not change much. We could believe that. It is very tough for a diffusion model to change the position of a sprite in a few steps because it would have to recreate a whole sprite at another location. Notice that there is also here a linear increase in the MSE between $t = 500$ to $t = 800$
Consider for example the sixth upper sample of 4.32, asking a diffusion model to generate a sample from there would most likely lead to a sample at the same location because it would be pretty hard for it to remove the current sample and create another. It would, however, not be so hard to turn the heart to a square as one can difficultly tell it is a heart. Noisy samples at $t \geq 500$ tend to all look like blurry circles and its not so hard to generate a heart, square or an ellipse from a blurry circle.
That is the reason why at $t = 500$, the MSE on the x-position is still intact while the shape conservation ratio is at 65 %. It seems that some features information is harder to remove than others'.
Note that the red dashed line represents the error committed by the regressor when the diffusion model generates random sprites, i.e, the regressor has no better chance to guess the initial sample's x-position than to output the mean, 0.5. Under the data distribution, the MSE committed would be of 0.0887. This is simply the result of an MSE between 32 linearly spaced values in [0,1] and the random guess $x - \hat{position} = 0.5$. This value is very close to the following one $\mathbf{E}_{\mathbf{x}\sim p_{data}}[(\mathbf{x_{x-pos}} - 0.5)^2] = \frac{1}{12} = 0.08333$ that results from assuming a uniform distribution. The reason the dashed line is not exactly at $\frac{1}{12}$ is simply because the diffusion model has not perfectly learnt the data distribution and the features distributions are thus not identical to those of the data distribution.

The conclusion we draw regarding the plots for the y-position 7.26 and the scale 7.27 are the same as those for the x-position. We note that the plot 7.28 is definitely unexpected, we won't give it much thought as the orientation regressor is the most likely culprit.

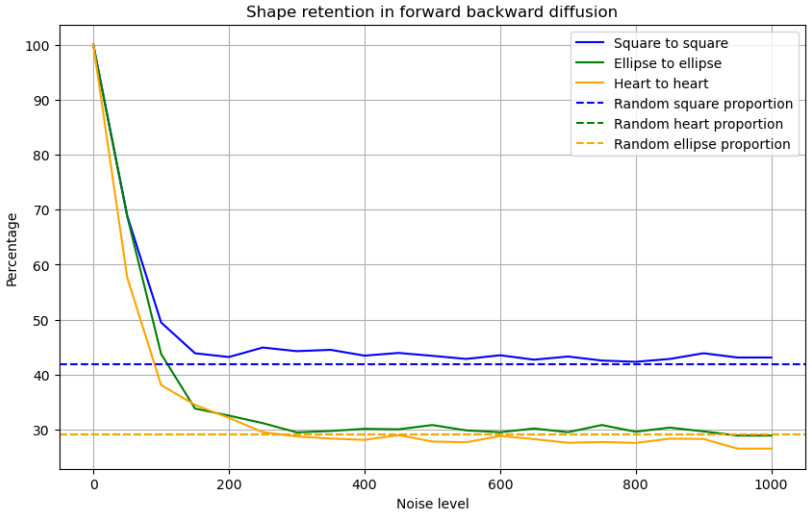


Figure 4.41. Shape retention using a diffusion model in $KL-VAE$ space. The procedure to generate this plot is identical to that described in 4.38 except that diffusion takes place in a latent space and classifiers/regressors also work in the latent space.

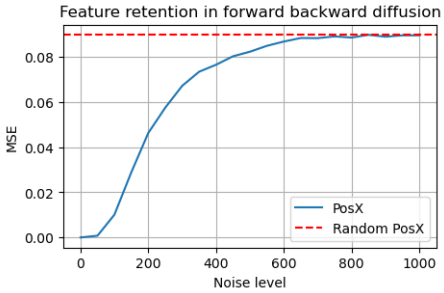
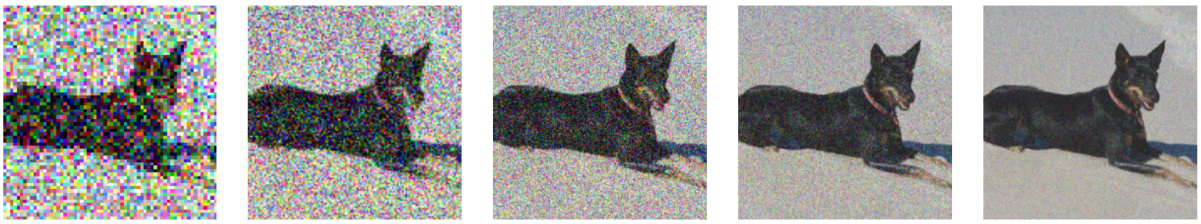


Figure 4.42. X-position retention using a diffusion model in $KL-VAE$ space. The procedure to generate this plot is identical to that described in 4.39 except that diffusion takes place in a latent space and classifiers/regressors also work in the latent space.

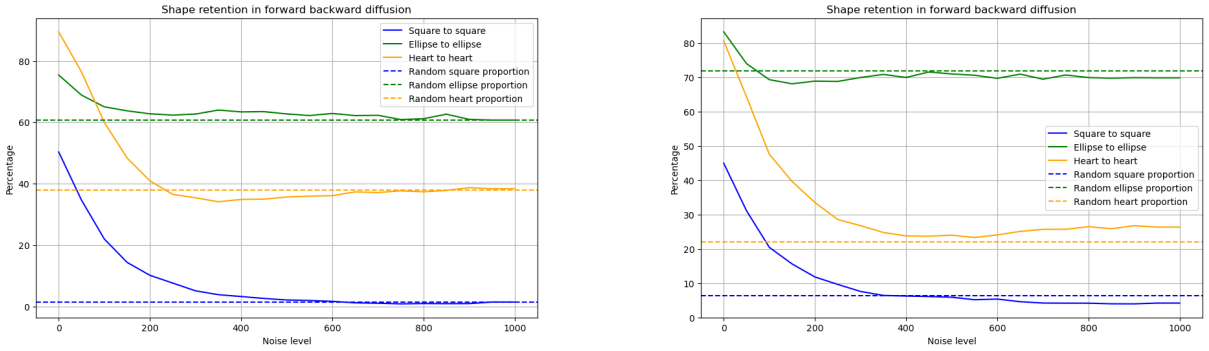
Indeed the orientation regressor training has not gone well and its predictions are not reliable. We detail this in the appendix.
We might wonder whether we'll draw the same conclusions from latent spaces. For the sake of conciseness, we'll only here show plots of the shape and the x-position, others will be in the appendix.
Results in $KL-VAE$ space are on figures 7.24, 7.25, 7.26, 7.27 and 7.28.

It is evident from figure 7.24 that in the $KL-VAE$ space, the shape information is lost much earlier—specifically, at $t = 250$, it becomes entirely indiscernible. Below, we attempt to provide well-founded reasons for this observed behavior.
We note that the diffusion model we used for this experiment, $DIFF-KL-VAE$, has not fully learnt the latent data distribution as it does not generates the 3 shapes in equal proportions, indeed it produces 41% of squares.
Regarding this gap between the $KL-VAE$ results and the pixel results, we know from [11] that large-sized images are less impacted by gaussian noise as neighbour pixels are usually redundant, see example the impact of noise on several images of different sizes on figure 4.43. The $KL-VAE$ space is (4,16,16) while pixel space is (1,64,64), that's a reduction factor of 4. Furthermore, we've seen on figure 4.11 that the 4 channels seemed



(a) 64×64 (b) 128×128 (c) 256×256 (d) 512×512 (e) 1024×1024

Figure 4.43. Larger images tend to be less impacted by independent gaussian noise



(a) Shape retention in $BVAE16$ space (b) Shape retention in $BVAE5$ space

Figure 4.44. Shape retention comparison in different latent spaces

to store rather similar information.

As the size is the most likely reason for this difference in behaviour, we'll run the experiment with 2 BVAEs, one of size 16 and the other of size 5.
Results for the experiment performed in $BVAE16$ space are on figure 4.44a while results for $BVAE5$ are on figures 4.44b.

Even though the experiment was repeated several times, we believe the shape curves 4.44a and 4.44b are not enough reliable for us to draw a conclusion. Indeed, on figure 4.44b the heart shape information seems to be lost at $t = 500$ and the square shape information at $t = 400$ while in the larger space on figure 4.44a, the heart shape information is lost before the square shape information. This might be due to the diffusion model not being trained enough.

The main conclusion we draw from experiment 0 are the following ones :

- All features tend to have a certain time threshold before which the addition of noise barely impacts them and the reconstruction leads to almost identical samples as the initial ones.
- After this threshold, the information loss tends to be linear until it reaches the random plateau, most likely due to the variance schedule.
- Features tend to be lost at different times and different speeds.
- The space in which samples live greatly impacts the rate at which the forward process destroys information.

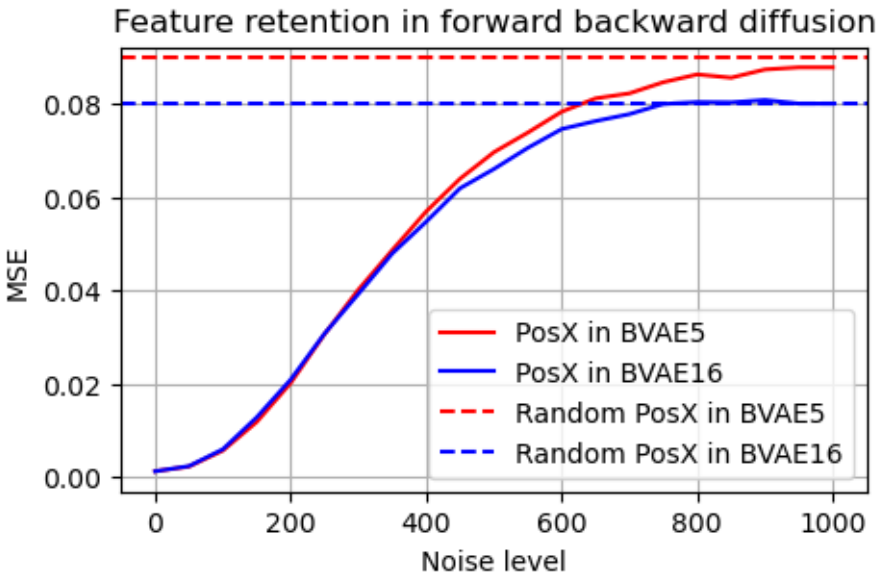


Figure 4.45. X-position retention in $BVAE16$ space and $BVAE5$ space

As our experiments are not fully sufficient to support our claims, we'll design others to test them and potentially even change them based on how the next experiments turn out.

4.6 Experiment 1 : Forward

4.6.1 Experiment summary

The second experiment we designed aimed at learning whether some information was **still** present at some time $\hat{t}$. A simple way to see whether any kind of information regarding the initial sample at time $\hat{t}$ is to see whether some information regarding the features of the initial sample is **still** present at $\hat{t}$. That is, we'll pick a sample from the data set, diffuse it up to time $\hat{t}$ and visually inspect its features to see whether we can accurately predict them from the noisy sample. For example, consider the initial sample $\mathbf{x_0}$ to be a heart located in the top left corner and its diffused version $\mathbf{x_{600}}$, if one can confidently and accurately predict the initial sample's features given only its diffused version, then it is conceivable to believe that $\mathbf{x_{600}}$ **still** contains information of $\mathbf{x_0}$. Surely, instead of visually inspecting noisy samples, we'll train models to predict the features of $\mathbf{x_{600}}$.
Consider for example the following 4.46 forward diffusion chains from data set samples where 2 samples are progressively diffused. At $t = 600$, it is pretty hard to dissociate the heart from the square but if some discriminative information is still there, a classifier might find it.

4.6.2 Experiment background

None

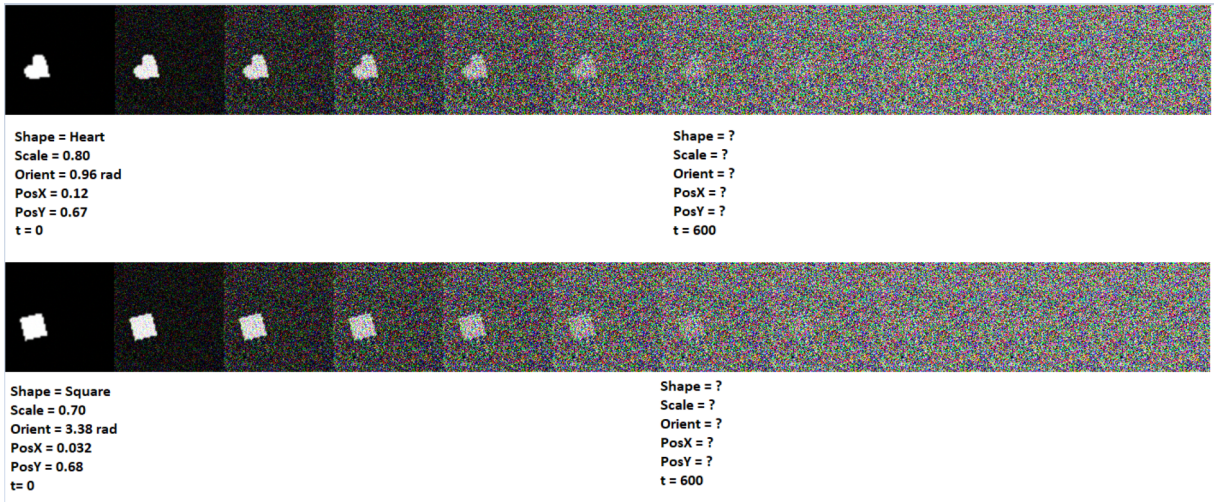


Figure 4.46. 2 data set samples are forward diffused and yet their features can, to the human eye, be estimated decently for up to $t = 600$.

4.6.3 Experiment procedure

Objective: The goal of this experiment is to investigate whether some information about features is still present in the latent space S at time $\hat{t}$ after diffusion. We aim to train classifiers/regressors to predict the features of the initial clean sample given only its diffused version. If the models perform fairly good, then it could suggest that the feature information is still there.

1. **Initialize the experiment:**
    - Prepare the dataset of images from the dSprites dataset with each image coming with its features about shape, posX, posY, orientation, and scale. Split the dataset into a train-validation-test split, respectively made of 80%, 10%, and 10% of the shuffled dataset.
    - Prepare a well-trained VAE if S is a latent space.
    - Set the maximum time step $t = T$ to $T = 1000$.
    - Choose a noise scheduler.
    - Define a neural shape classifier working in S tasked with outputting the most probable shape of a noisy sample it is fed.
    - Define 4 other neural regressors working in S that will be tasked with outputting a scalar value which should be as close as possible to the feature they are trained to predict.
    - Define an ADAM optimizer for each network and set the number of epochs to 20.
2. **Forward Noisy Diffusion:**
    - For each image $\mathbf{x}_i$ in the training set:
        - If $S \neq$ Pixel Space:

        * Encode the data using the chosen VAE resulting in the latent representation $\mathbf{z}_i$.
        * Apply forward noisy diffusion to $\mathbf{z}_i$ from time $t = 0$ to $t = \hat{t}$ using the chosen diffusion model.
        * This results in a new noisy image $\mathbf{z}_{i,\hat{t}}$ at time $\hat{t}$ after diffusion.
    - Else:
        * Apply forward noisy diffusion to $\mathbf{x}_i$ from time $t = 0$ to $t = \hat{t}$ using the chosen diffusion model.
        * This results in a new noisy image $\mathbf{x}_{i,\hat{t}}$ at time $\hat{t}$ after diffusion.

3. Training Models to Predict from Noisy Samples:

- For each image $\mathbf{x}_i$ in the training set:
    - If $S \neq$ Pixel Space:
        * Read the image shape from the training set and feed the shape classifiers both $\mathbf{z}_{i,\hat{t}}$ and the shape label to train the classifier using the cross-entropy loss.
        * Read the image scale from the training set and feed the scale regressor both $\mathbf{z}_{i,\hat{t}}$ and the scale ground truth to train the regressor using the MSE loss.
        * Repeat the previous step for posX, posY, and orientation.
    - Else:
        * Read the image shape from the training set and feed the shape classifier both $\mathbf{x}_{i,\hat{t}}$ and the shape label to train the classifier using the cross-entropy loss.
        * Read the image scale from the training set and feed the scale regressor both $\mathbf{x}_{i,\hat{t}}$ and the scale ground truth to train the regressor using the MSE loss.
        * Repeat the previous step for posX, posY, and orientation.
- Keep training on the training set and validate each model on the validation set after every epoch.

4. Test the Models:

- For each image $\mathbf{x}_j$ in the test set:
    - If $S \neq$ Pixel Space:
        * Read the image shape from the test set, encode it then diffuse into $\mathbf{z}_{j,\hat{t}}$ and feed the shape classifiers both $\mathbf{z}_{j,\hat{t}}$ and the shape label to test the classifier.
        * Read the image scale from the test set and feed the scale regressor both $\mathbf{z}_{j,\hat{t}}$ and the scale ground truth to test the regressor.

        * Repeat the previous step for posX, posY, and orientation.
    - Else:
        * Read the image shape from the test set and feed the shape classifier both $\mathbf{x}_{j,\hat{t}}$ and the shape label to test the classifier.
        * Read the image scale from the test set and feed the scale regressor both $\mathbf{x}_{j,\hat{t}}$ and the scale ground truth to test the regressor.
        * Repeat the previous step for posX, posY, and orientation.
- Keep testing until the whole test set has been covered.
- Return shape test loss, shape test accuracy, scale test MSE, posX test MSE, posY test MSE, orientation test MSE.

Add a figure for this where I show for an image each of its feature and then its noisy versions.

4.6.4 Experiment preparations

We decided to perform this experiment in pixel space and in some latent spaces for a range of time steps b between t=0 and t=T. We may notice that this experiment does not require a generative diffuser but simply, a well-trained VAE, a noise scheduler and some untrained classifier/regressor for each space. These latter models should not be trained as they will be trained for a specific timestep!
The latent spaces we studied are :

1. Vanilla VAE : Latent sizes considered are all powers of 2 from 1 to 256, i.e, 1,2,4,8,16,32,64,128,256
2. β VAE : Latent sizes considered are all powers of 2 from 1 to 256, i.e, 1,2,4,8,16,32,64,128,256
3. DIP VAE : Latent sizes considered are 5,16,256
4. Fully Convolutional VAE : Latent sizes considered are (4,16,16)
5. KL VAE : Latent sizes considered are (4,16,16)

4.6.5 Experiment results and discussion

Before delving into details, let's briefly outline the main conclusions drawn from this experiment.

Firstly, the noise variance schedule strongly impacts the features information conservation in the forward process. Secondly and as observed in the previous experiment, features tend to lose their information at different times, different speeds and the space in which images live greatly impacts these 2 quantities. Lastly, we observed that most of the information is gone before $t = 600$ and thus the remaining timesteps necessity might be questioned.
We now proceed to analyzing the results of the experiment.

As we did for experiment 0, we'll first have a look at the results in pixel space as they are the easiest to interpret. The forward experiments results in pixel space can be seen

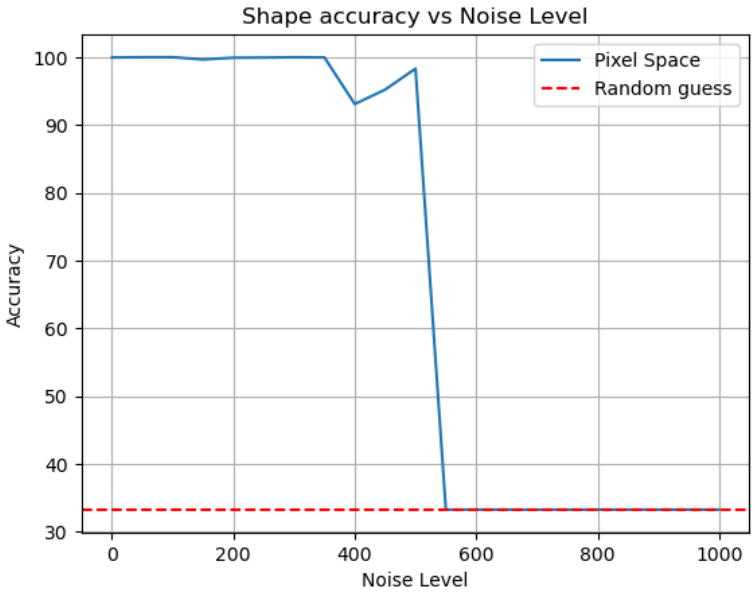


Figure 4.47. Shape classifier performance on data that gets gradually more noisy. For each timestep we train a classifier on the noisy sample shape and report its performance on the test set. As we can see, the accuracy completely drops from near-100% to 33.33% which corresponds to a random guess by the classifier as the 3 classes are balanced in the test set.

on figures 4.47, 4.48, 4.49a, 4.49b and 4.49c.

As we can see on the shape accuracy plot 4.47, the accuracy abruptly drops around $t = 550$, this is an unexpected behaviour as we saw in the previous experiment 4.38 the very smooth behaviour. What is reassuring is that the drop takes place in the interval [300,800] which we previously identified as the one where things happen. Thinking more about it and particularly looking at, for example, the seventh upper sample of 4.32 which corresponds to $t = 600$, the shape our human eyes can observe is barely a blurry circle while on its preceding neighbour on the left, we can still recognize the heart characteristics. Sure at $t = 600$ one can simply see a blurry circle but one can still easily locate it! That is why the x-position loss curve 4.48 only starts increasing at more than $t = 600$, the transition to the random guess plateau is however less steep than the accuracy's for the same reasons we stated previously.
The scale loss curve is however quite intriguing as it is not monotonically increasing whereas the variance schedule is. It is much more intriguing that the scale is badly predicted at $t = 200$ while it seems very simple to guess from the third upper sample of 4.32. Looking at the forward results should help fixing these issues just raised.
Once again we'll simply plot for the latent spaces the shape accuracy plot and the x-position MSE loss as the other variables are quite redundant, their plots are available in the appendix. We remind that we generated these results by following a procedure identical to that used to generate pixel space results except we're first encoding data to the studied latent space.

The results for VAEs with a latent size of 1024 can be seen on figures 4.50, 7.31, 7.32, 4.51 and 7.33.

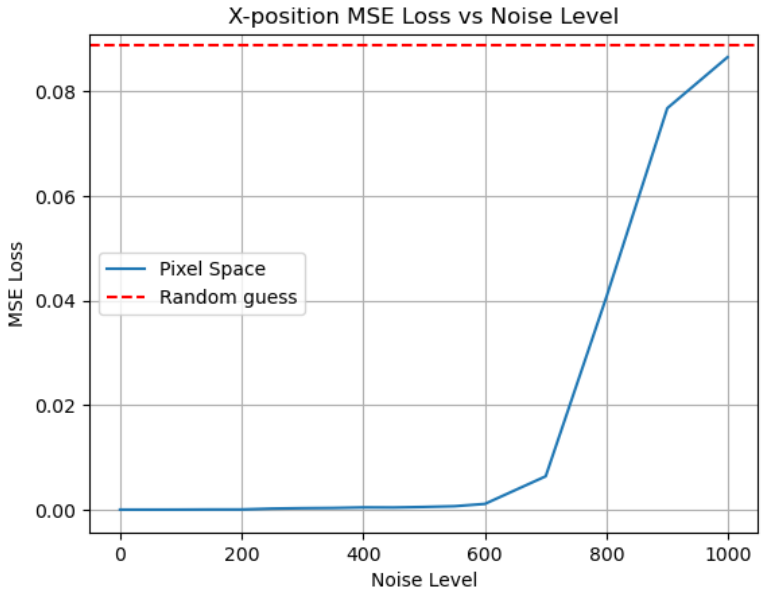
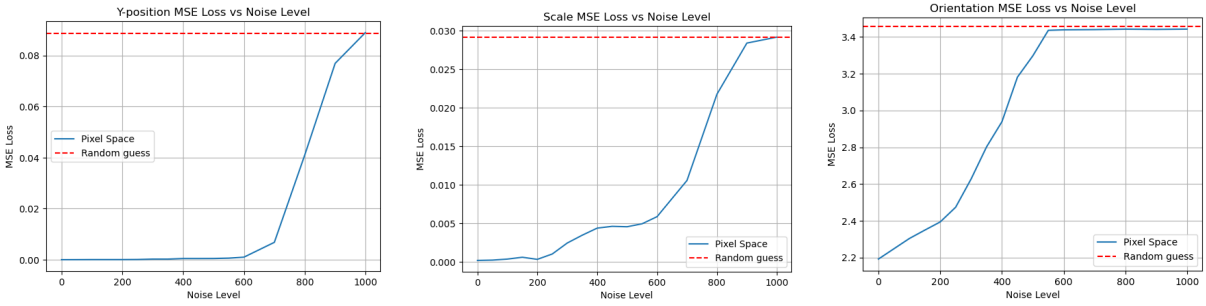


Figure 4.48. X-position regressor performance on data that gets gradually more noisy. For each timestep we train a regressor on the noisy sample x-position and report its performance on the test set. As we can see, the MSE completely starts increasing fast near $t = 600$ and reaches random performance at $t = 1000$.



(a) Y-position regressor (b) Scale regressor (c) Orientation regressor

Figure 4.49. Regressor performance on noisy data

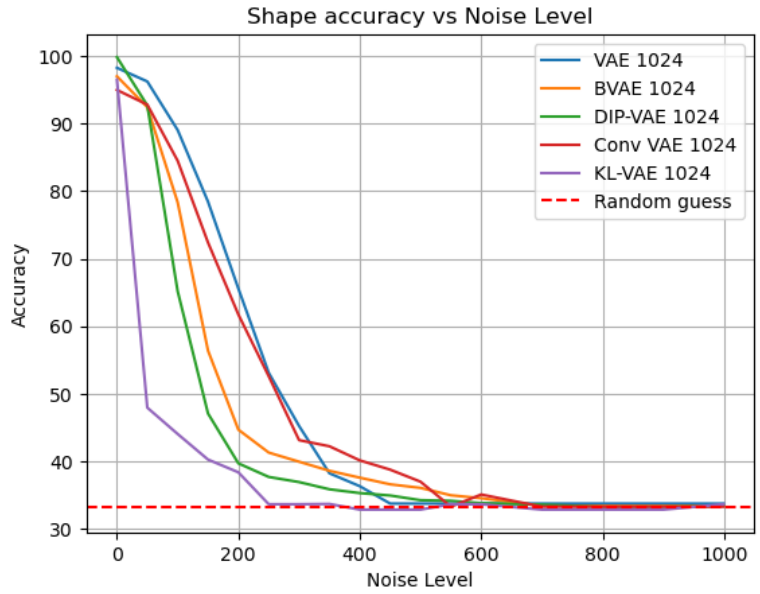


Figure 4.50. Shape classifier performance on data that gets gradually more noisy. For each timestep we train a classifier on the latent noisy sample shape and report its performance on the test set. As we can see, the accuracy curve is strongly dependent on the latent space. It seems latent structure plays almost no role here as the best and worst candidate are respectively the $KL-VAE$ and the $CONV-VAE$

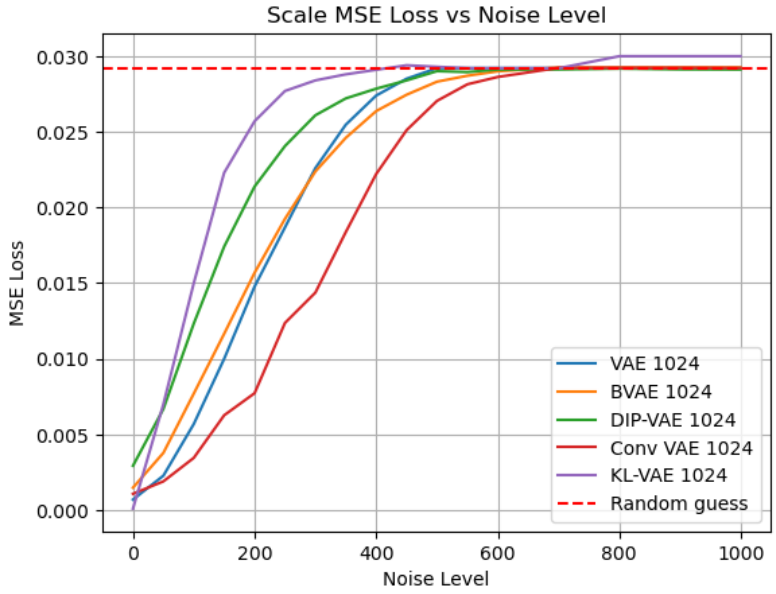


Figure 4.51. Scale regressor performance on data that gets gradually more noisy. For each timestep we train a regressor on the latent noisy sample x-position and report its performance on the test set. As we can see, the MSE curve is different for each latent space and it seems the $KL-VAE$ and the $CONV-VAE$ are here again the worst and best candidates when it comes to resisting to information destruction.

From the shape accuracy plot 4.50 and the scale MSE plot 4.51, we make a few observations :

- All latent shape information is lost at $t = 600$ as in the pixel case 4.47.
- All latent scale information is lost at $t = 600$ which is far earlier than its counterpart in the pixel case 4.49b where information is only fully lost at $t = 900$.
- Information seems to get destroyed earlier in latent spaces.
- Latent spaces tend to all start losing information at the same time near $t = 0$ but they do so at different speeds, furthermore, the information loss is rather smooth. Oppositely, in pixel space there is an information steep drop at $t = 500$ where before it was intact.
- For the most part of the drop, the accuracy curve and the MSE scale loss look linear. That is especially true for the blue plots of $VAE - 1024$.
- In all 5 plots, the VAE performs better than the BVAE which performs better than the DIP-VAE.

As latents are not designed to be human readable, it is tough to look at what noisy latents actually look like. We simply cannot look as before whether noisy ellipses and hearts both tend to look like blurry circles however we can still provide some explanations to these plots.
We believe information gets destroyed sooner in latent space mainly for these reasons reasons :

- Latents are smaller and we've seen that larger objects are less impacted by noise.
- Larger objects are less impacted by noise because they have more neighbours that look like them such that if at least one neighbour can keep meaningful information, it can be enough for a model to find it and use it appropriately for its prediction.
- 1D latents had to go through fully connected layers to exist and thus these neighbourhood local information is something they do not have. There might, however, exist latent data redundancy but simply it won't be found in neighbour components which might be harder for a network to learn.
- We believe Conv VAE works best mainly because it preserves local redundancy.
- We believe BVAE is worse than VAE and DIP-VAE is worse than BVAE for the following reason : they were designed to factorize data. BVAE was designed to have a disentangled representation of the input data, that is, latent components share very little mutual information compared to VAE. This means that when one component is largely noised by mischance, its information is gone and no other component can replace it. It thus makes sense that DIP-VAEs perform even worse than BVAEs as they intended to push disentanglement even further.

To assess whether the size and disentanglement really are important, we'll run the latent forward

1. In BVAE spaces with exponentially growing size, i.e., $BVAE{-}2^n$ with $n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 10\}$
2. In several sizes of VVAE vs BVAE vs DIP-VAE to see whether the disentanglement really is the culprit for faster information loss.

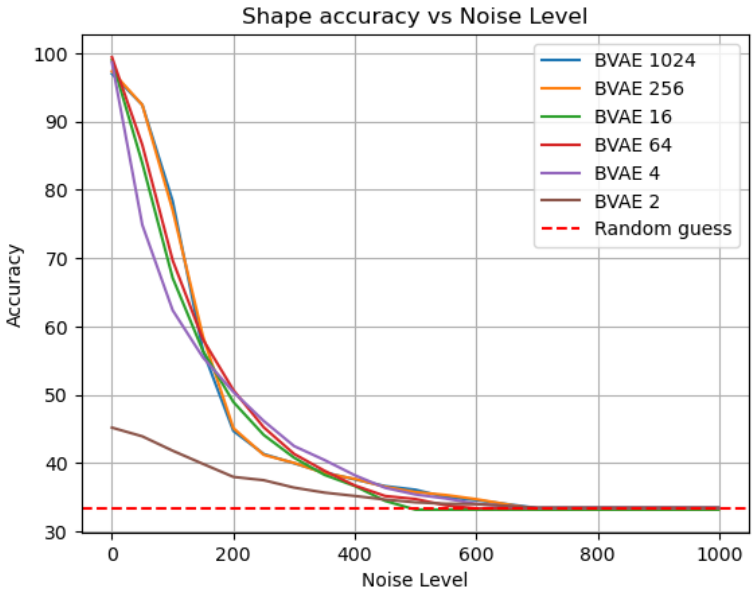


Figure 4.52. Shape classifier performance on data that gets gradually more noisy. For each timestep we train a classifier on the latent noisy sample shape and report its performance on the test set. As we can see, larger BVAEs tend to make a better job at resisting information destruction.

The results regarding the size forward experiments on shape and scale can be seen on figures 4.52 and 4.52. For sake of clarity, we actually restrict to 6 curves per plot and not 10 and we, as usual, depict only the shape and scale plots. The full results are available in the appendix.

As we can see on figure 4.52, the larger BVAEs tend to be better at information conservation, they are better at withstanding gaussian noise. We notice however that smaller VAEs have a higher accuracy in the region [180,300], we believe this is most likely due to statistical fluctuations as this experiment was performed once only for sake of resources. We note that larger BVAEs tend to have their loss more steep while smaller BVAEs tend to have their loss more smooth even though we might be over-interpreting the results here. The scale plot 4.53 essentially tells the same stories. Overall, larger VAEs indeed tend to store information better.
Since we hypothesized that disentanglement could also be a reason for the faster information loss of BVAEs and DIP-VAEs compared to classical VAEs, we'll now compare several sizes of these models to see whether this hypothesis holds. Again, we'll stick to shape and scale for conciseness.
As we can see on the right above plots, for VAEs with a 1024 latent size, the vanilla one outperforms both the β-VAE and the DIP-VAE but we cant guarantee this is also the case for other sizes ?
We plot the shape accuracy and scale MSE losses for VVAE,β-VAE and DIP-VAE 256,16 and 5 on figure 4.54. For each and every forward experiment we ran, the vanilla VAE holds information longer than the β-VAE which holds information longer than the DIP-VAE. This result clearly supports our hypothesis that states that the 2 latter VAEs are not good at holding information for a long time. We stress again this is most likely due to their inherent disentangled latent components which are designed to avoid holding information about their neighbours, this way, there is very little redundancy among latents and thus

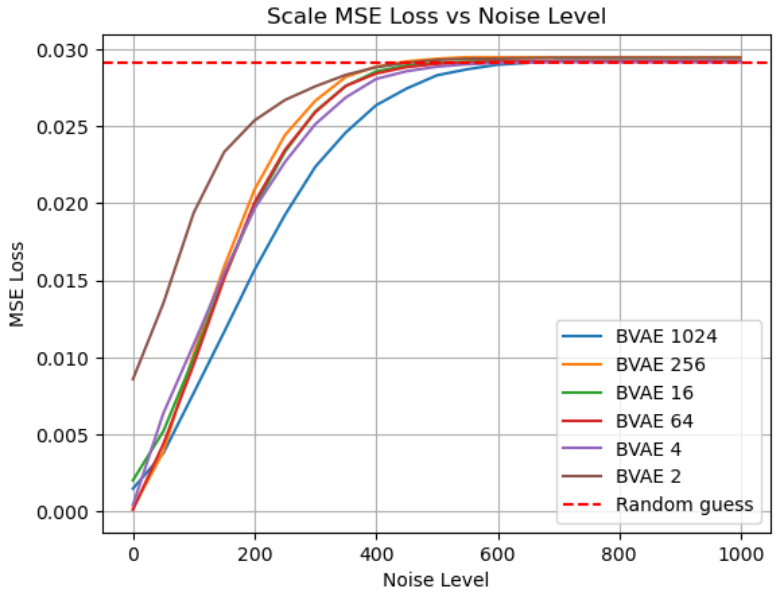


Figure 4.53. Scale regressor performance on data that gets gradually more noisy. For each timestep we train a regressor on the latent noisy sample x-position and report its performance on the test set. As we can see, larger BVAEs tend to make a better job at resisting information destruction. Particularly, the worst and best candidates respectively are the smallest and largest BVAE.

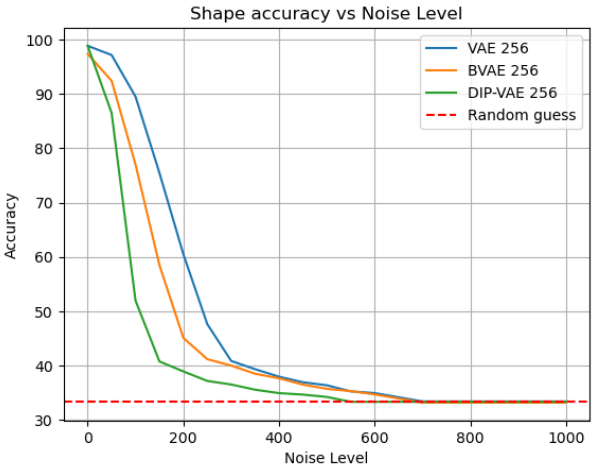
noise destroys overall information much faster.

We've said so far that DIP-VAEs tended to be better at disentangling than β-VAEs which also tended to be better at it than classical vanilla VAEs but we have given no experimental result that would support this claim. We thus report a piece of table given in [12] that largely supports this claim, see table 4.10. β-VAEs tend to disentangle better with larger β values and DIP-VAEs show the same behaviour.

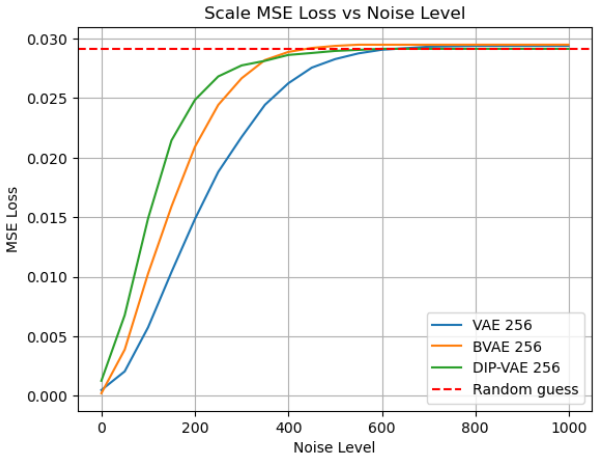
Table 4.10. Z-diff score Higgins et al. (2017), the proposed SAP score and reconstruction error (per pixel) on the test sets for 2D Shapes and CelebA ($\beta_1 = 4$, $\beta_2 = 60$, $\lambda = 10$, $\lambda_1 = 5$, $\lambda_2 = 500$ for 2D Shapes; $\beta_1 = 4$, $\beta_2 = 32$, $\lambda = 2$, $\lambda_1 = 1$, $\lambda_2 = 80$ for CelebA). For the results on a wider range of hyperparameter values, refer to Fig. 1 and Fig. 2.

Method	2D Shapes			CelebA		
	Z-diff	SAP	Reconst. error	Z-diff	SAP	Reconst. error
VAE	81.3	0.0417	0.0017	7.5	0.35	0.0876
β-VAE ($\beta = \beta_1$)	80.7	0.0811	0.0032	8.1	0.48	0.0937
β-VAE ($\beta = \beta_2$)	95.7	0.5503	0.0113	6.4	3.72	0.1572
DIP-VAE-I ($\lambda_{od} = \lambda$)	98.7	0.1889	0.0018	14.8	3.69	0.0904
DIP-VAE-II ($\lambda_{od} = \lambda_1$)	95.3	0.2188	0.0023	7.1	2.94	0.0884
DIP-VAE-II ($\lambda_{od} = \lambda_2$)	98.0	0.5253	0.0079	11.5	3.93	0.1477

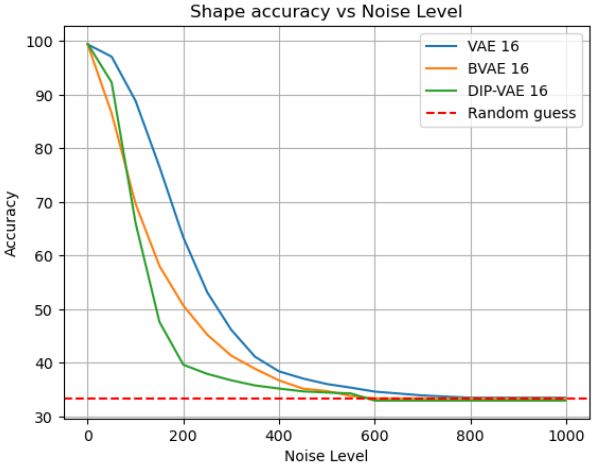
Another point that we may want to study is the impact of the noise schedule on the forward results, indeed, we said that the way information should disappear is a function of the noise schedule. To put this hypothesis to the test, we ran the forward experiment with 3 different noise schedules :



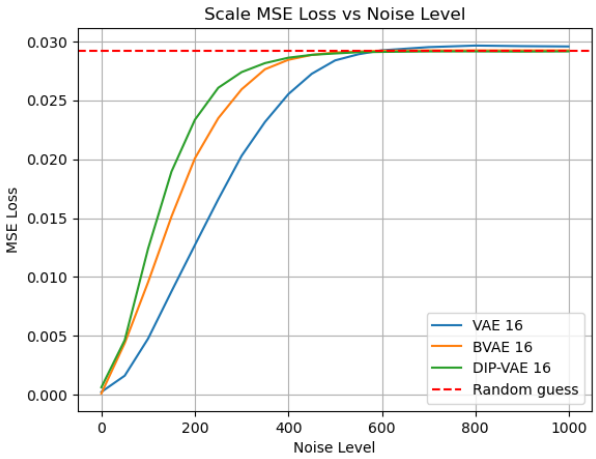
(a) Forward experiment shape classifier performance in 3 VAE latent spaces with size 256



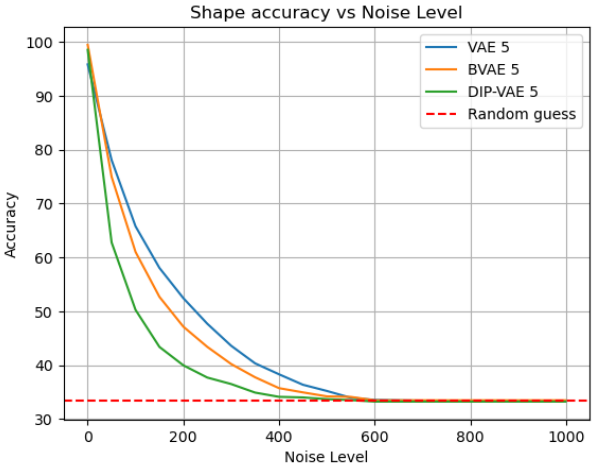
(b) Forward experiment scale regressor performance in 3 VAE latent spaces with size 256



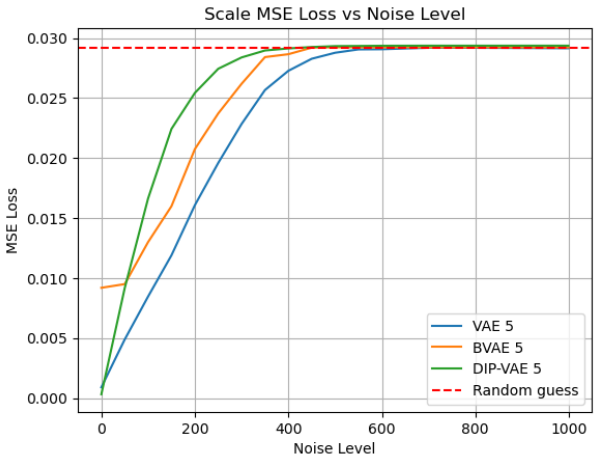
(c) Forward experiment shape classifier performance in 3 VAE latent spaces with size 16



(d) Forward experiment scale regressor performance in 3 VAE latent spaces with size 16



(e) Forward experiment shape classifier performance in 3 VAE latent spaces with size 5



(f) Forward experiment scale regressor performance in 3 VAE latent spaces with size 5

Figure 4.54. Performance of shape classifier and scale regressor in different VAE latent spaces

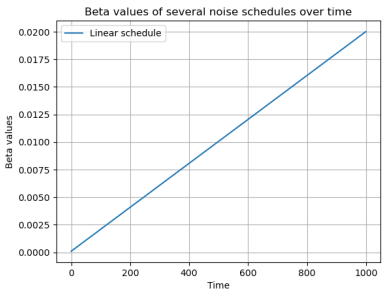


Figure 4.55. A linear noise schedule

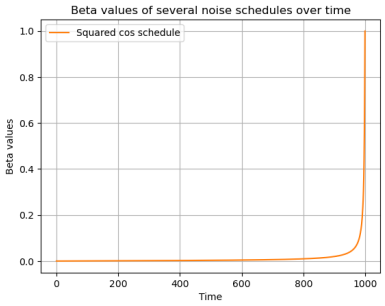


Figure 4.56. A squared cos noise schedule

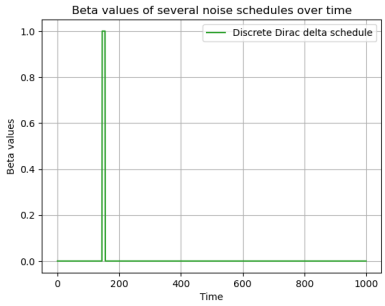
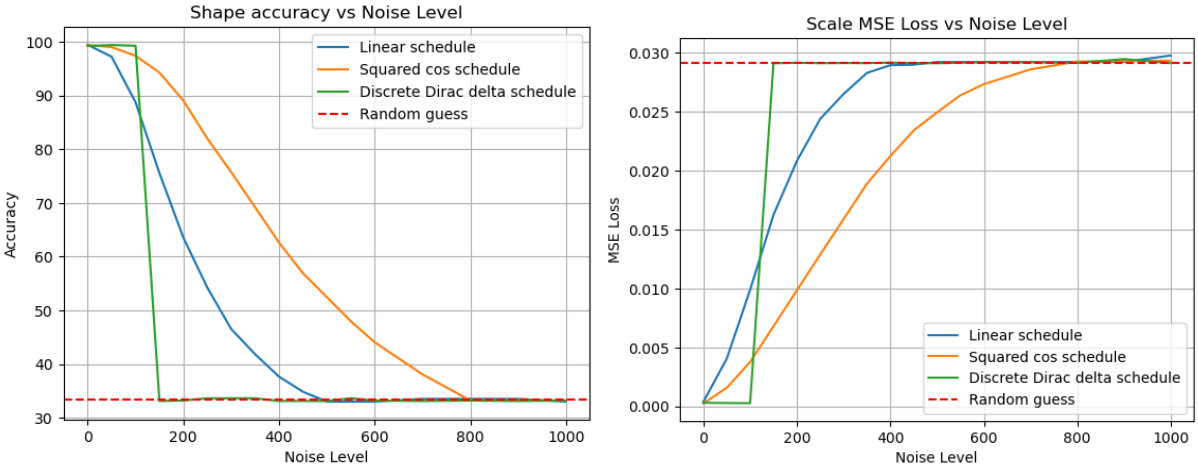


Figure 4.57. A Dirac noise schedule



(a) Shape forward results for 3 different schedules.

(b) Scale forward results for 3 different schedules.

Figure 4.58. Comparison of different schedules

1. The usual linear noise schedule depicted on figure 4.55.
2. A squared cos noise schedule depicted on figure 4.56.
3. A noise schedule with null variance everywhere except in a tight interval $[t_1, t_2]$, somehow a discrete Dirac delta depicted on figure 4.57.

The shape and scale forward results are given on figures 4.57a and 4.57b. As we can see, the Dirac accuracy abruptly drops from 100 % to 33% simply because before the variance spike there was no noise and after it, large amounts of noise were added. We also notice that the squared cos accuracy is almost identically greater than the linear's. This is due to the squared cos variance being almost identically greater than the linear's. Indeed at $t = 450$, the linear schedule variance is of 0.0097 while the square cos schedule variance has to wait up until $t = 800$ to reach that bound.

These simple experiments clearly show that there is a strong dependence between features information preservation and the noise variance schedule. The exact relationship however remains complicated to put into equations.

Finally, since we so far tried to quantify information in the forward chain on a synthetic

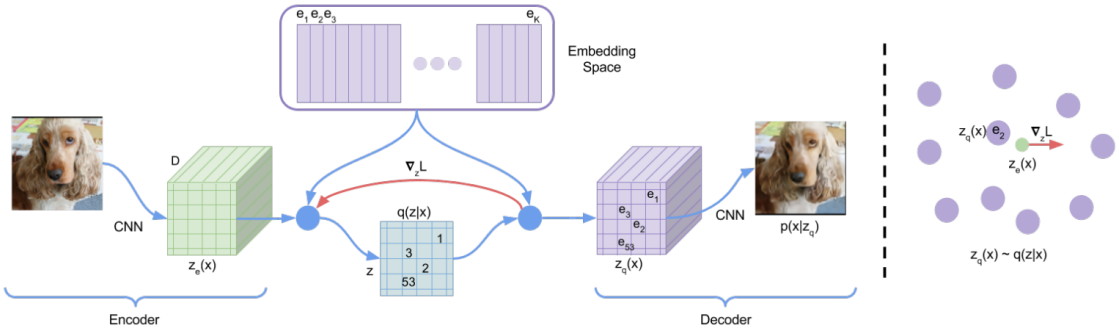


Figure 4.59. "Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(\mathbf{x})$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass." Image and caption taken from [13]

data set, we want to see whether the same behaviours will be observed on a real data set. We thus decided to use a piece of the CelebA data set composed of 20 000 images of men and women with equal proportions of 50-50. This experiment was performed in a VQVAE latent space which is almost identical to a classical VAE except it possesses a nearest-neighbour quantization layer at its bottleneck. A working example of the VQVAE is shown on figure 4.59. The result of the forward experiment is on figure 4.60. We observe that on a real data set, the curves are not as smooth as they used to be, furthermore it seems there are step levels in the descent where the accuracy stays constant for a few noise levels. Overmore, in this case it seems that some information is still present at $t = 800$ while in all previously studied latent spaces, no feature information seemed to be present after $t = 600$. This result supports our claim stating that information retention is highly dependent on the latent space structure, size and complexity.

We here reach the end of the experiment 1 and we may logically wonder how all of these results actually contribute to answering our research question. Initially, our research question considered the choices made the diffusion model : are some choices made, when, why ? So far we've however not studied any real diffusion model since we're simply noising images in the forward process. The reason we spent much energy in this experiment is that as the backward diffusion chain is essentially trained to revert the forward chain, much of diffusion models generative capabilities can be understood by first analyzing the forward chain.
The main conclusions we can draw from these sub-experiments are :

- As we add noise to data, data information gets gradually destroyed and therefore features information gets destroyed as well. However the loss of information is usually not steep and abrupt unless the variance schedule is. More generally, the features information destruction in the forward process strongly depends on the variance schedule.
- Features tend to lose their information at different speeds and at different times.
- The space in which images live is pretty important, either it is pixel space or a latent space. It is important in the sense that its structure, size and overall complexity

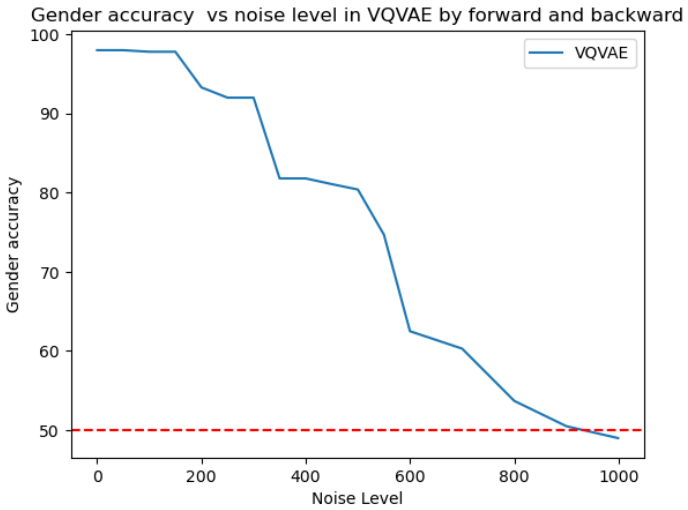


Figure 4.60. Forward experiment on the gender feature of the CelebA data set. Samples were added a certain noise level and a classifier was trained to predict their gender feature based on the noisy samples. Test accuracy is reported on the figure.

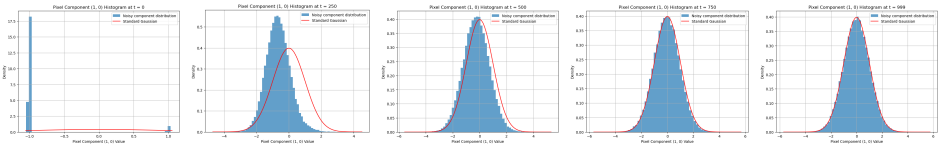
impact the way images handle noise. Smaller spaces tend to lose information faster for example.

- For most of the situations studied, features information tends to stay present up until $t = 600$. We may question the interest of having forward steps further than the point where no information seems to be present anymore. The only usefulness would be to make the signal gaussian if it is not already at the said timestep. We thus check that below.
- If the backward diffusion chain approximates the reverse forward diffusion chain well enough, these timesteps should **not** "create" any information nor make any choices as no information was present there in the forward process.

The next experiment will put the last bullet point to the test and hopefully contribute to answering the research question : when are choices made ?

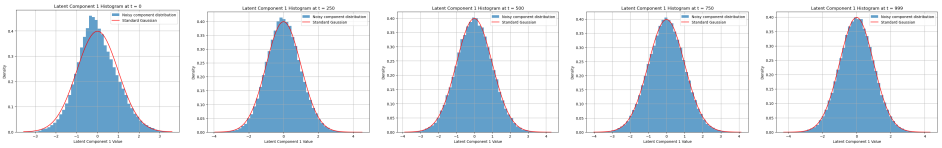
4.6.6 A quick look at normality

We realized during the above experiments that latent signals tended to lose all their information before $t = 600$ and we thus wondered whether the signal was already gaussian at that time, if so, the remaining steps could be useless. We therefore look at the normality of images and latents at several timesteps. We do so by plotting histograms of variables as well as by computing p-values.
We first run this analysis in pixel space and then run it again in the $BVAE - 5$ latent space.
We show on figure 4.61 the evolution of a given pixel distribution with the noise level increasing. We observe that at the beginning the pixel values are in -1,1 since images are black and white but as t approaches T, the noisy component distribution gets closer to a standard gaussian. It might suggest that one could essentially stop at $t = 750$.



(a) t=0 (b) t=250 (c) t=500 (d) t=750 (e) t=999

Figure 4.61. Histograms of the first noisy pixel component computed over the whole data set at different noise levels. As we can see, at the beginning of the diffusion chain, the data is already quite normal. This is due to the KL term in the VAE training loss.



(a) t=0 (b) t=250 (c) t=500 (d) t=750 (e) t=999

Figure 4.62. Histograms of the first noisy latent component computed over the whole data set at different noise levels. As we can see, at the beginning of the diffusion chain, the data is already quite normal. This is due to the KL term in the VAE training loss.

The case in latent space $BVAE-5$ is quite different, as we can see on figure 4.62, the data is pretty normal even at $t=0$, that is due to the KL term in the VAE training loss. Given that latent images achieve the same normality level as pixel images but much before, it may be useless to use the same noise schedule for both. This early normality also explains why the information tended to be lost earlier in latent spaces than in pixel spaces, simply because the latent data is initially more normal.

4.7 Experiment 2 : Backward

4.7.1 Experiment summary

The third experiment we designed aimed at learning whether some information was **already** present at some time $\hat{t}$. A simple way to see whether any kind of information regarding the generated sample at time $\hat{t}$ is to see whether some information regarding the features of the generated sample is **already** present at $\hat{t}$. One should note that in the previous experiment we used the keyword **still** while in this experiment we use the keyword **already**. That is because in the previous experiment we went in the forward noising direction while in this experiment we go in the backward generative direction, i.e, we'll create data. The reasoning is, however, pretty similar. That is, we'll generate a sample from gaussian noise, backward diffuse it up to time $\hat{t}$, and visually inspect its features to see whether we can accurately predict the features of the generated sample given only the still noisy sample. For example, consider the gaussian noise $\mathbf{x_T}$ which when fully backward diffused is $\mathbf{x_0}$ which corresponds to a heart located in the top left corner. During the backward chain, we likely generated a yet noisy version $\mathbf{x_{400}}$. If one can confidently and accurately predict $\mathbf{x_0}$'s features given only $\mathbf{x_{400}}$, then it is conceivable to believe that $\mathbf{x_{400}}$ **already** contains information of $\mathbf{x_0}$. This would suggest that at t=400, a most likely definitive choice regarding the sample's features was already made. Surely, instead of visually inspecting noisy samples, we'll train models to predict the features of

$\mathbf{x_{400}}$.
One should note that the classifiers/regressors at $\hat{t}$ are trained such that their guess of the feature f given $\mathbf{x_{\hat{t}}}$ is as close as possible to the feature f of $\mathbf{x_0}$. Therefore, their considered ground truth is the feature f of $\mathbf{x_0}$ however, we do not have any ground truth data for that particular $\mathbf{x_0}$ as it is not part of the data set! What we'll consider as ground truth will be data predicted by classifiers/regressors well trained on the data set.

4.7.2 Experiment background

None

4.7.3 Experiment procedure

Objective: The goal of this experiment is to investigate whether some information about features is already present in the latent space S at time $\hat{t}$ after backward diffusion. We aim to train classifiers/regressors to predict the features of the generated sample given only its still noisy version. If the models perform fairly good, then it could suggest that the feature information is already there.

1. **Initialize the experiment:**
    - Prepare a well-trained classifier working in pixel
    - Prepare a well-trained diffuser working in space S.
    - Prepare a well-trained VAE if S is a latent space.
    - Set the maximum time step $t = T$ to $T = 1000$.
    - Define a neural shape classifier working in S tasked with outputting the most probable shape of a noisy sample it is fed.
    - Define 4 other neural regressors working in S that will be tasked with outputting a scalar value which should be as close as possible to the feature they are trained to predict.
    - Define an ADAM optimizer for each network and set the number of epochs to 20.
    - Set the number of $TrainingSteps, ValidationSteps, TestSteps = 0.8{*}len(ds), 0.1{*}len(ds), 0.1 * len(ds)$
2. **Backward Generative Diffusion:**
    - Sample a gaussian noise in S called $\mathbf{z_T}$
    - Apply backward diffusion to $\mathbf{z_T}$ from time $t = T$ down to $t = 0$ but store $\mathbf{z_{\hat{t}}}$ during the backward process and $\mathbf{z_0}$ at the end
3. **Gather Ground Truth Data:**
    - Get the ground truth shape label by classifying $\mathbf{z_0}$ with the prepared classifier
    - Get the ground truth scale value, posX value, posY value and orientation value by regressing $\mathbf{z_0}$ with the prepared regressors

4. **Training Models to Predict from Noisy Samples:**
    - For each sampled gaussian noise:
        - Feed the shape classifier both $\mathbf{z}_{\hat{t}}$ and the shape label we just predicted to train the classifier using the cross-entropy loss.
        - Feed the scale regressor both $\mathbf{z}_{\hat{t}}$ and the scale value we just predicted to train the regressor using the MSE loss.
        - Repeat the previous step for posX, posY, and orientation.
    - Keep training for TrainingSteps and validate each model for ValidationSteps after every epoch.
5. **Test the Models:**
    - For each sampled gaussian noise:
        - Feed the shape classifier both $\mathbf{z}_{\hat{t}}$ and the shape label we just predicted to train the classifier using the cross-entropy loss.
        - Feed the scale regressor both $\mathbf{z}_{\hat{t}}$ and the scale value we just predicted to train the regressor using the MSE loss.
        - Repeat the previous step for posX, posY, and orientation.
    - Keep testing for TestSteps.
    - Return shape test loss, shape test accuracy, scale test MSE, posX test MSE, posY test MSE, orientation test MSE.

Note 1: We should note that we could classify at t=0 in any space as long as we got the shape label and feature values. We decided to classify/regress in the S space as its usually a smaller space and thus the forward pass in the network is much faster.
Add a figure for this where I show for an image each of its feature and then its noisy versions.

4.7.4 Experiment preparations

Since we need a good diffusion model, VAE and classifier/regressors, we performed this experiment in 4 spaces : pixel space, $KL-VAE$ space and $BVAE-16$ space. All models can be found in the appendix.

4.7.5 Experiment results and discussion

Before delving into the details, we briefly outline the main conclusions of the experiment. Firstly, the features choices the diffusion model makes while generating a sample are not instantaneous, they are different for each feature and are made during different periods. Secondly, there is a large period near $t = T$ where no choice seems to be made, only a generic sprite is created in that period. The sprite then crystallizes to a certain shape during the rest of the backward diffusion. Lastly, we observed several discrepancies before accuracy curves computed in the forward process and accuracy curves computed in the backward process. Therefore, it is not safe to assume that the diffusion model tends to make choices about features at approximately the same time as the feature was destroyed

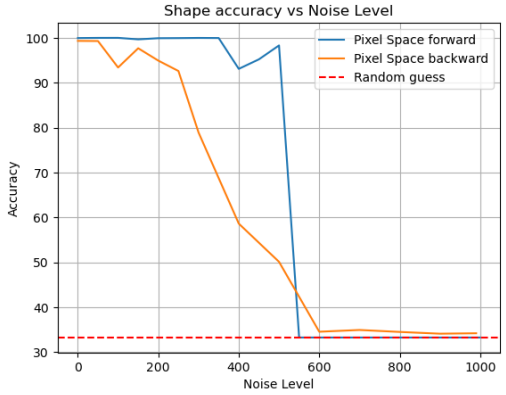
in the forward process.
We naturally decided to first run the experiment in pixel space as it is the most meaningful one to us humans.
The test accuracy and MSEs of the different models can be seen on figure 4.63. On 4.63a, we observe very different behaviours between the backward process and the forward process since the former is smooth while the latter is steep. The binary idea consisting in "The shape is either clearly here or clearly absent because of the noise" is actually quite unfounded even though quite intuitive. Due to the forward variance schedule, the resulting blurriness makes the guess tough but not fully random as we can see on figures 4.32 and 4.33. On these same figures we also observe that the 2 chains seem to be similar and we thus expect the classifiers curve to be the same. We note that if the diffusion model has converged and if the classifiers were perfectly trained, the 2 curves should be identical. We argue the forward classifying curve steepness is due to lack of runs, indeed only 1 was performed. We believe we'd observe a much smoother curve if the experiment was averaged enough.
When it comes the X-position, the curves share the same form except the backward one starts to increase earlier. The observation remains the same for the Y-position and the scale. The orientation curves on the other other tend to be coherent but we cannot trust these are the orientation regressor had weak performance.
Once again, it makes sense that choices regarding each feature are not made at the same time, for example, we can safely say that at $t = 400$ the position of the sprite is fully decided as they regressors have near-perfect prediction but at $t = 400$, the shape accuracy is only of 58 %. That is, the shape has not been fully decided, it could be heart but also an ellipse...

The main question we need to answer now concerns the discrepancy between the forward and backward curves, why do they look similar yet they seem to be shifted from one another ? We believe the culprit is simply the diffusion model training, the backward process seems to not have fully learnt how to reverse the forward process. To assess the likelihood of this assumption, we designed a third experiment which will simply quantify the distance between the forwarded diffused samples and backward generated samples at a certain noise level.

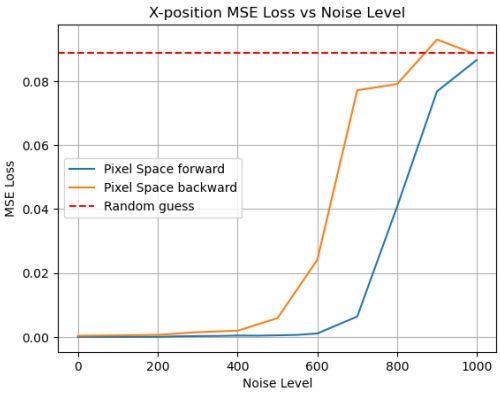
Still we might want to look at the results in latent spaces. The shape and scale results obtained in the $KL-VAE$ space can respectively be seen on figures 4.64 and 4.65. Once again, we observe a small shift between those 2 curves, it is even more present in the shape plot which, as said in the caption, is an unexpected behaviour.

We now look at the reassuring results of the scale and x-position predictions in $BVAE-16$ space on figures 4.67 and 4.66.

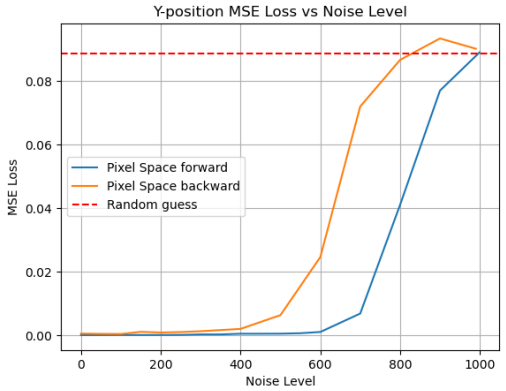
They are reassuring in the sense that as the diffusion model worsens, the forward and backward curves diverge more significantly. This observation supports our earlier statement that the discrepancy between the curves might stem from the imperfect training of the diffusion model.
Finally, we also performed the experiment for the celebA dataset in a VQVAE space to see whether our results would extend to more complex datasets, plot is available on figure 4.69. We observe that once again the overall form of the backward and forward curves is



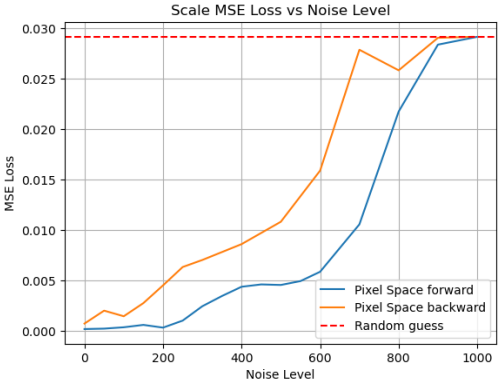
(a) Shape classifier performance on generated data at noise level t. Generative direction goes backward in time. For each timestep, we train a classifier on the noisy sample shape we generated and report its performance on the test set. From $t = 600$ to $t = 1000$, the shape hasn't been decided yet, resulting in an accuracy of 33.33%, which corresponds to a random guess by the classifier since the three classes are generated with equal proportions by the generator. In the interval $t = 0$ to $t = 600$, the behavior is significantly different from the forward process, exhibiting a smoother and less steep accuracy curve.



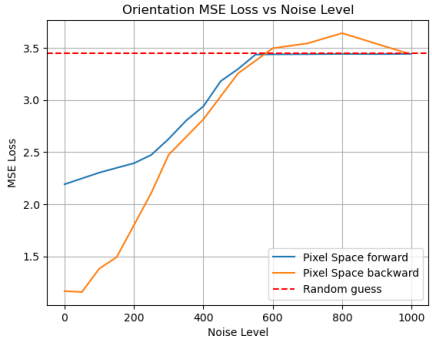
(b) X-position regressor performance on generated data at noise level t. Generative direction goes backward in time. For each timestep, we train a regressor on the noisy sample x-position and report its performance on the test set. Notably, the backward and forward processes yield nearly identical curves, with the former showing a slightly earlier increase.



(c) Y-position regressor performance on generated data at noise level t.



(d) Scale regressor performance on generated data at noise level t.



(e) Orientation regressor performance on generated data at noise level t.

Figure 4.63. Regressor and classifier performances on generated data at different noise levels.

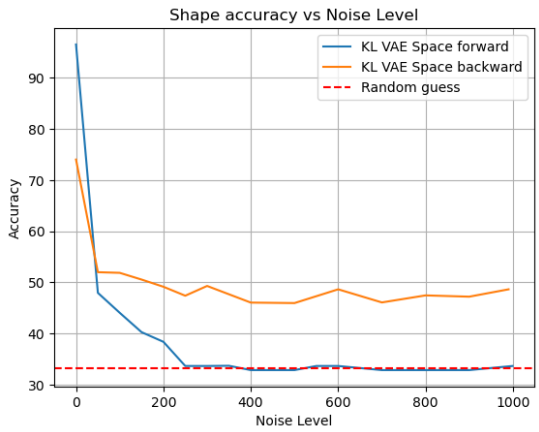


Figure 4.64. Shape classifier performance on generated data in $KL-VAE$ space at noise level t. Generative direction goes backward in time. For each timestep, we train a classifier on the noisy sample shape we generated and report its performance on the test set. The curves have relatively the same form, but the backward curve reaches a plateau of 50% accuracy fairly soon. It is surprising that it reaches 50% accuracy at $t = 1000$ where samples are almost fully Gaussian.

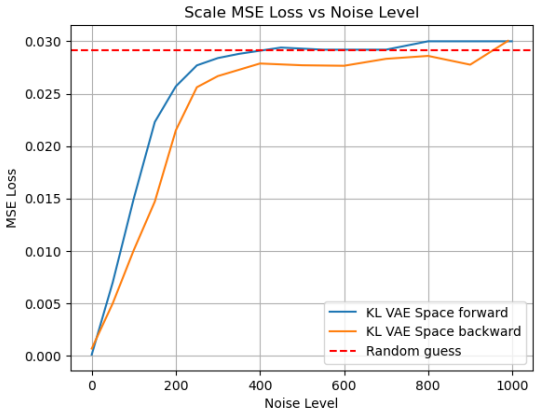


Figure 4.65. Scale regressor performance on generated data in $KL-VAE$ space at noise level t. Generative direction goes backward in time. For each timestep, we train a regressor on the noisy sample x-position and report its performance on the test set. Both curves have the same form but suffer from a little shift.

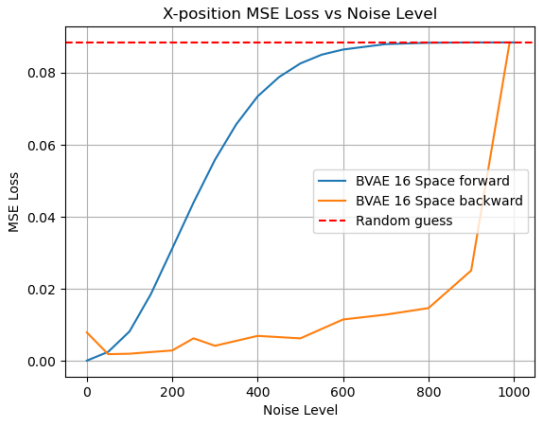


Figure 4.66. X-position regressor performance on generated data in $BVAE-16$ space at noise level t. Generative direction goes backward in time. For each timestep, we train a regressor on the noisy sample x-position and report its performance on the test set. Both curves have the same form but suffer from a little shift.

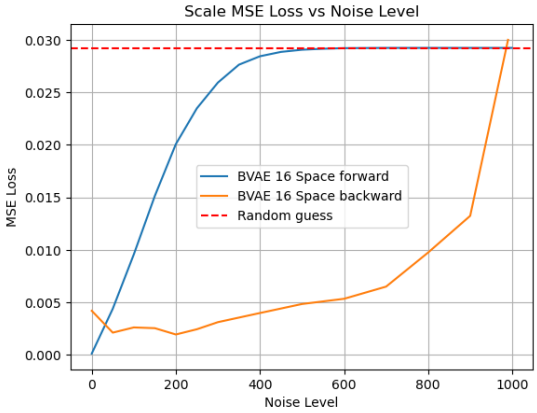


Figure 4.67. Scale regressor performance on generated data in $BVAE-16$ space at noise level t. Generative direction goes backward in time. For each timestep, we train a regressor on the noisy sample x-position and report its performance on the test set. Both curves have the same form but suffer from a little shift.

Figure 4.68. We observe that for both features, the forward and backward curves are far apart even if they start and end at the same point. This is most likely due to the poor training of $DIFF-BVAE-16$. These curves are reassuring because we know from the generated samples and training curve that the underlying diffusion model is bad, and we also know that the 2 curves should be identical only under perfect convergence. This is even more reassuring as we know that the pixel diffusion model is better than the KL-VAE diffusion model which is better than the BVAE-16 diffusion model and each of the forward and backward curves are getting worse with the diffusion model quality. In other words, the better the diffusion model, the closer the curves should get.

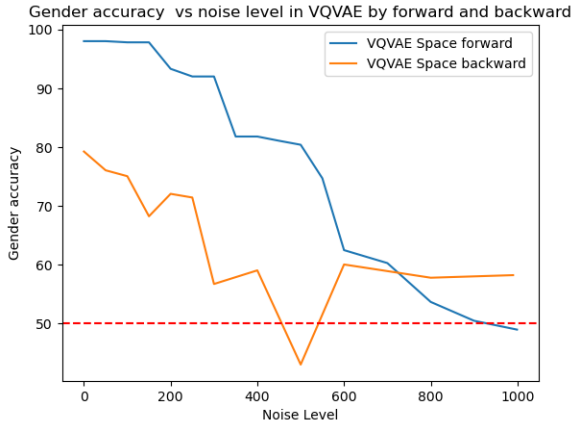


Figure 4.69. Shape classifier performance on generated celebA samples data in $VQVAE$ space at noise level t. Curves have same form but are shifted from one another.

the same except that there is, once again, a shift between those 2.

The main conclusions we draw from this experiment are the following ones :

- Feature choices are not instantaneous and they are not made at the same time nor at the same speed. Rather, the diffusion model tends to generate a still noisy sample and needs several timesteps to define the sample features. For example, the sprite's location seems to be decided first.
- For timesteps in [700,1000], there is almost no feature information, that is no feature choice was made but only a general sample that can crystallize into many different sample is present.
- Under imperfect diffusion model training, it is not safe to assume that the choices the diffusion model tends to make choices about features at around the same time than the time the feature was destroyed in the forward process. Indeed under perfect training the forward and backward process share the distribution and if, for example, the shape information was lost immediately at $t = 456$, then the backward process should choose the shape information exactly at $t = 456$.

We stated earlier that the reason the forward and backward curves were shifted was due to the imperfect diffusion model training. To further analyze these notable discrepancies, we propose the third experiment which essentially tries to classify between noisy samples generated during the forward process and noisy samples generated during the backward process.

4.8 Experiment 3 : Real vs fake

4.8.1 Experiment summary

At convergence, diffusion models should have fully learnt the data distribution on which they were trained. That is, one should not be able to tell whether a given image was sampled from the diffusion model or from the data set, they should be seen as identically distributed even though they are not. This actually does not only concern final samples but also temporal latents generated during the diffusion process. One should not be able

to tell whether $\mathbf{x_{400}}$ is actually a data set sample that was noised or a gaussian noise which was denoised down to $t = 400$. Rather, one should not be able to dissociate the two marginal distributions $p(\mathbf{x^f_{400}})$ and $p_\theta(\mathbf{x^b_{400}})$ which respectively denote the distribution of noisy samples at $t = 400$ and the distribution of generated samples at $t = 400$.
The experiment for a given timestep is then pretty simple. It consists in training a classifier that will try to dissociate noisy data samples obtained by the forward process from noisy generated samples obtained by the backward process. If the classifier has an accuracy significantly higher than 50%, then it would likely suggest that the 2 distributions contain different information making them different. It can also be a decent diagnostic to assess the convergence of the diffusion model.

4.8.2 Experiment background

None

4.8.3 Experiment procedure

Objective: The goal of this experiment is to investigate whether some classifier could dissociate generated data from true data at a given time $\hat{t}$ in the diffusion chain. If the classifier performs fairly good, then it could suggest that the backward process has not learnt how to revert the forward process well enough.

1. **Initialize the experiment:**
    - Prepare the dataset of images from the dSprites dataset with each image coming with its features about shape, posX, posY, orientation, and scale. Split the dataset into a train-validation-test split, respectively made of 80%, 10%, and 10% of the shuffled dataset.
    - Choose a noise scheduler.
    - Prepare a well-trained diffuser working in space S.
    - Prepare a well-trained VAE if S is a latent space.
    - Set the maximum time step $t = T$ to $T = 1000$.
    - Define a neural classifier working in S tasked with outputting the most probable real or fake class of a noisy sample it is fed.
    - Define an ADAM optimizer for each network and set the number of epochs to 20.
2. **Forward and Backward Diffusion:**
    - Apply forward noisy diffusion to the real data samples up to time $t = \hat{t}$ to obtain noisy data samples denoted as $\mathbf{z^f_{\hat{t}}}$.
    - Apply backward generative diffusion on sampled gaussian noise down to time $t = \hat{t}$ to obtain generated noisy samples denoted as $\mathbf{z^b_{\hat{t}}}$.
3. **Classifier Training:**
    - Prepare labels for the classifier, using 0 for real data samples and 1 for generated samples.

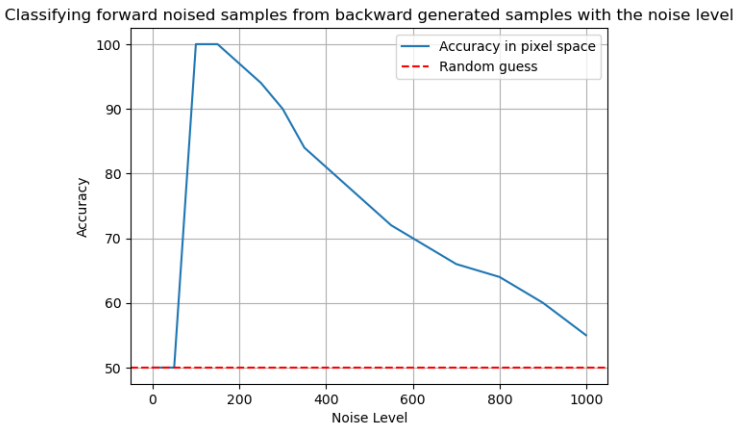


Figure 4.70. At each timestep t, a pixel space classifier was trained to dissociate between forward noised samples up to level t and backward generated samples down to level t. Accuracy on the test set is reported on the plot.

- Train a classifier using the noisy data samples $\mathbf{z}_{\hat{\mathbf{t}}}^{\mathbf{f}}$ and $\mathbf{z}_{\hat{\mathbf{t}}}^{\mathbf{b}}$ along with their corresponding labels.

4. **Evaluate Classifier:**
   - Create a test set containing both noisy data samples and noisy generated samples.
   - Evaluate the trained classifier on the test set to measure its accuracy.
   - Return the accuracy

Note : We should note that if S is a latent space we can either decide to classify in the latent space S or first decode the samples and classify in pixel space. As long as the VAE is well trained, it has no significant impact.

4.8.4 Experiment preparations

We decided to perform this experiment in pixel space, $KL-VAE$ space and $BVAE-16$ space for a range of time steps between t=0 and t=T.

4.8.5 Experiment results and discussion

The plot reporting the classifying accuracy between noised samples and generated samples in pixel space can be seen in figure 4.70.

We make 3 observations :

1. The accuracy at $t = 0$ is of 50%, that is the data distribution and the generated distribution cannot be dissociated by the classifier. This is encouraging as it suggests there is no visual signal present on generated samples that would spill the beans.
2. Close to $t = T$, the accuracy is not significantly far from 50%.
3. The most intriguing observation is that for timesteps $\in$ [200,800], the accuracy is of almost 100%, that is samples are clearly different, or at least there is a visual hint that allows the classifier to dissociate them. This also means that the backward

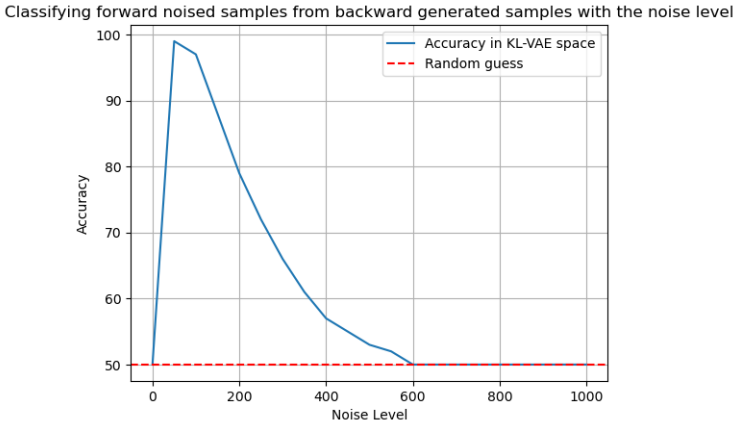


Figure 4.71. At each timestep t, a $KL-VAE$ space classifier was trained to dissociate between forward noised samples up to level t and backward generated samples down to level t. Accuracy on the test set is reported on the plot.

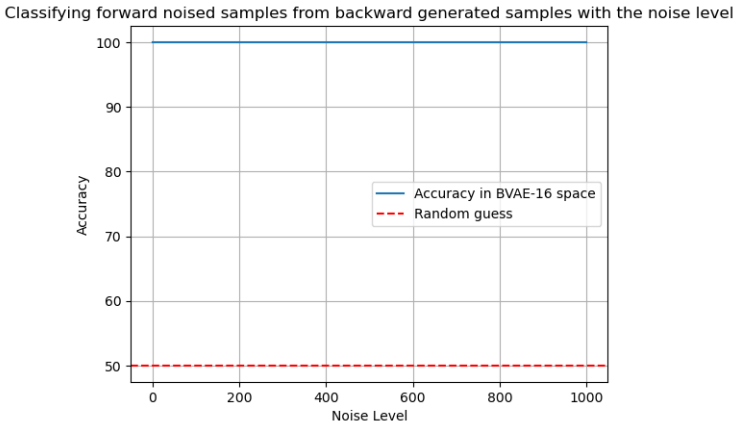


Figure 4.72. At each timestep t, a $BVAE-16$ space classifier was trained to dissociate between forward noised samples up to level t and backward generated samples down to level t. Accuracy on the test set is reported on the plot.

has not fully learnt how to revert the forward process and this may explain the discrepancies we saw in experiment 2.

The plot reporting the classifying accuracy between noised samples and generated samples in $KL-VAE$ space can be seen in figure 4.71.

The observations remain the same except here full 50% accuracy is achieved already at $t = 600$, we believe that is because in this smaller space, noise propagates faster and normality is reached sooner.
Finally, the plot reporting the classifying accuracy between noised samples and generated samples in $BVAE-16$ space can be seen in figure 4.72. From the figure, it is very clear that the diffusion model has not learnt how to revert the forward process. The discrepancies in the experiment 2 plots 4.67 and 4.66 are most likely due to that.

We note that using such a discriminator to assess the performance of a diffusion model might be a good idea but an even better idea would be to use the discriminator results to make the diffusion model generate closer-to-data samples in GAN way. This idea was actually implemented in [14].

Chapter 5

Conclusion

Our research question aimed at answering the specifics of the features choices a diffusion model makes when generating a sample. Particularly, we were interested in knowing when these choices are actually made, whether they're instantaneous, reversible and we also wanted to know what factors contributed to these choices.

We first studied the feature conservation in the forward-backward chain to see whether the diffusion model would tend to recreate samples similar those that it forward diffused. Secondly, we spent a long time studying information destruction in the forward process. The reason we spent much energy in this experiment is that as the backward process is essentially trained to revert the forward process, much of diffusion models generative capabilities can be understood by first analyzing the forward process. Thirdly, we studied the actual choices made by the diffusion model itself in the backward chain along with their timings and factors that influence these choices. Finally, we evaluated how well the diffusion model learnt to revert the forward process.

Each experiment served to either yield novel insights or provide corroborative evidence for assertions based on the observations derived from preceding experimental outcomes. Mainly we learnt that the the features information destruction in the forward chain and the features information "creation" by choice in the backward chain are primarily determined by the noise variance schedule. Under the usual linear schedule we can divide the diffusion chain for a particular feature in 3 regions. In the first region that starts at $t = 0$, the information retention is pretty high and quite robust to noise in the forward sense and we observed that in this region, when generating a sample the choice was already made. It is somehow a region which modifies the look of samples but does not change its main features. In the second region which is often large and symmetrical with respect to the middle of the chain, that is where information gets mainly destroyed in the forward sense and where choices are made in the backward sense. This is the region that aroused the most interest in us as it directly contributes to answering our research question. We observed that choices were actually not instantaneous but rather the result of many denoising steps. Thus during this region it is not easy to guess a sample's feature before the end of the region. In the third region which goes up to $t = T$, the noisy samples have lost all of their features information, they are however not always fully gaussian and we believe this is the only point of this region. In this region, the backward process

essentially creates a generic sample which can subsequently crystallize its features in the second region.

The partitioning of these regions on the diffusion chain from $t = 0$ to $t = T$ depends on the time and speed at which the studied feature gets destroyed in the forward process. Depending on the samples size, structure and complexity, the regions can be either large or small. Surely, having a large third region is not very useful and we therefore believe that the noise variance schedule should be chosen such that the latter is not too large.
We are overall satisfied with out results but we believe there is still a lot of work we could have done to better support our claims but most importantly we believe there are still many ideas to dig into. We provide some of these ideas in the "to go further section".

Chapter 6

To go further

In this chapter, we enumerate several ideas that we would have liked to experiment with. Unfortunately, due to time constraints, we were unable to pursue them and as a result, we have included these ideas in the "To Go Further" section, intended for future endeavors. We below provide a non-exhaustive lists of tasks which we believe are worthy of time and energy.

- A lot of our experiments rely on the assumption of a perfectly trained diffusion model. This assumption is most often never met and that leads us to not being able to trust the results enough. It would be great to look for better training procedures and to be able to assess the convergence level of the diffusion model.
- We have stated that the forward experiment results heavily depend on the variance schedule and we have shown 3 examples of forward results with different schedules. We believe it would be worth to look at the generated backward results with these 3 schedules as well.
- Since the third region close to $t = T$ is sometimes quite large, it could be a good idea to try to replace all these slightly useless denoising steps by a single model that would generate generic samples which could start their diffusion chain from the second region where generative choices are actually made. This idea was inspired from [9].
- As we saw, smaller samples tend to reach normality faster than larger samples and thus we believe the noise schedule should be dependent on the size of samples but also on their structure and complexity. Note that we do not suggest here to learn the noise schedule here as it prescribed in [15].
- On a completely different note, it may be worth to study the importance of the number of inference steps used when generating samples as we observed that they had a very large impact. It seemed that using more inference steps did not always lead to better samples.

Chapter 7

Appendix

7.1 VAEs

In this section we provide all training losses, samples, interpolations and other metrics for diagnosing VAEs.

7.1.1 Latent structure importance

7.1.2 Latent size importance

7.1.3 Stochasticity and regularity importance

Interpolation of $AE - 256$ can be seen on figure 7.19.

7.1.4 Latent representation importance

7.2 Diffusers

In this section we provide all training losses, samples, modes coverage and other metrics for diagnosing diffusers.

7.3 Main experiments

7.3.1 Experiment 0 : Feature conservation

Plots in the pixel space :

Plots in the $KL - VAE$ space :

7.3.2 Experiment 1 : Forward

Plots for the latents forward :

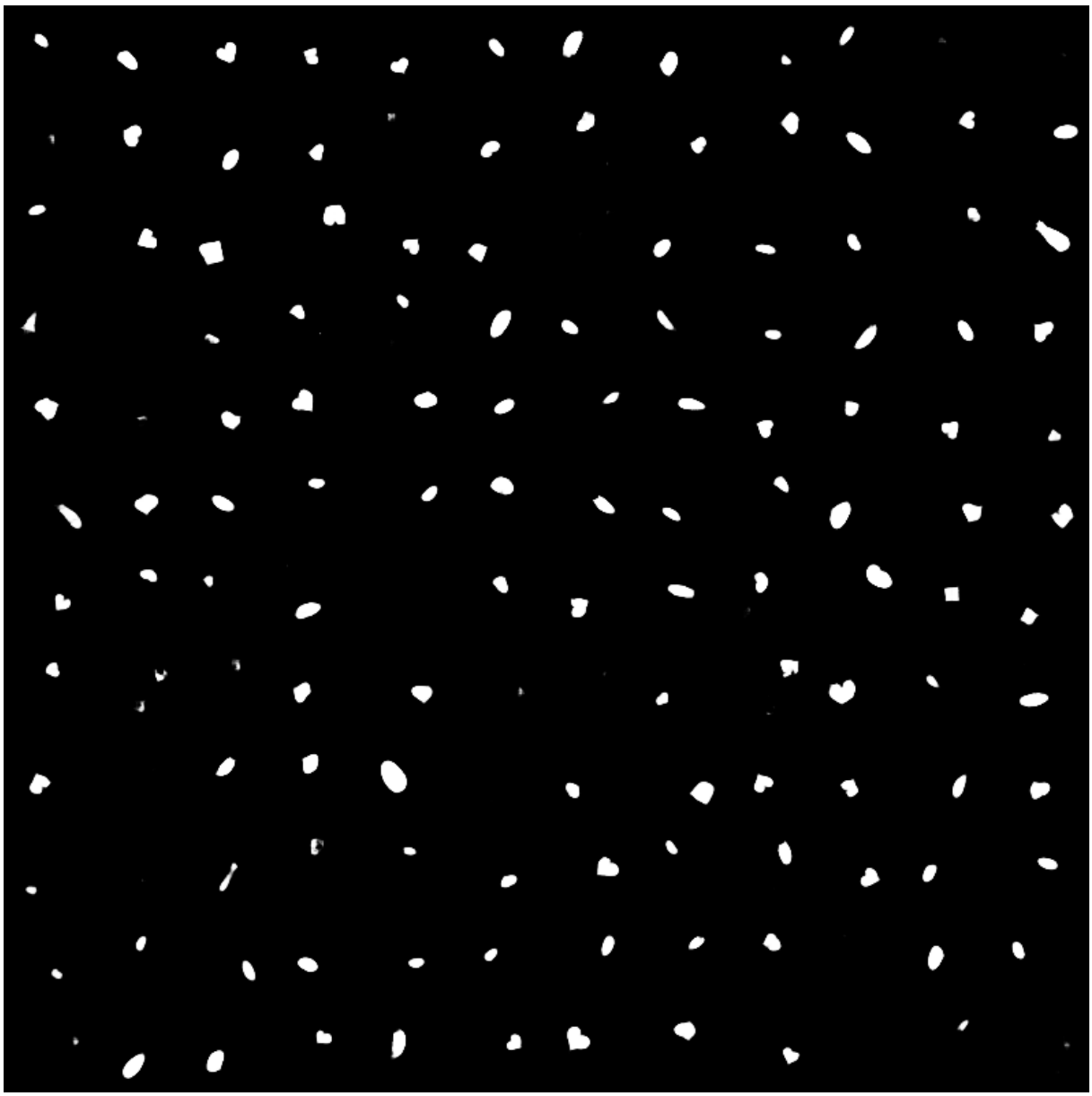


Figure 7.1. Samples of $VAE - 1024$

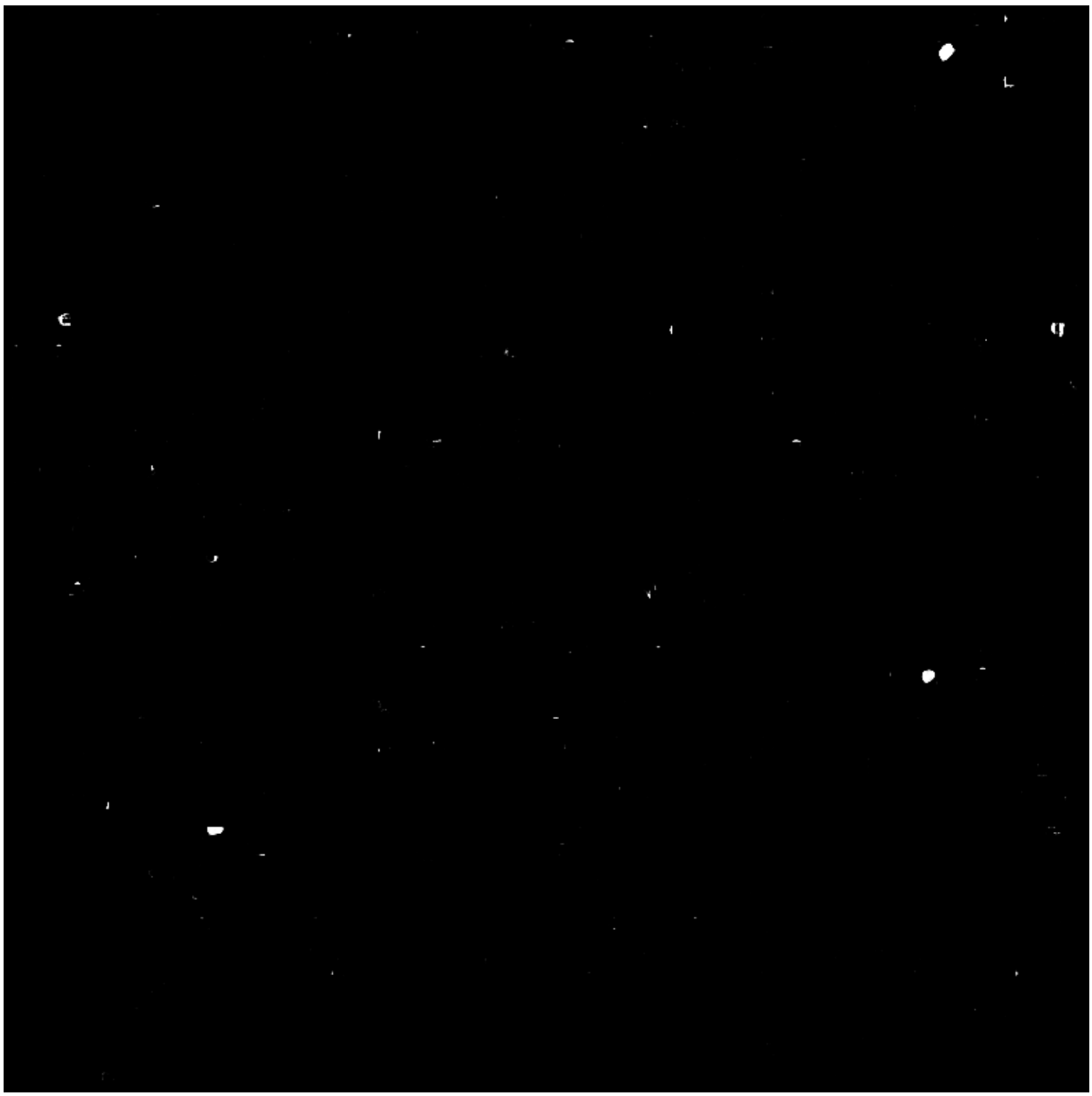


Figure 7.2. Samples of $CONV - VAE - 4 - 16 - 16$

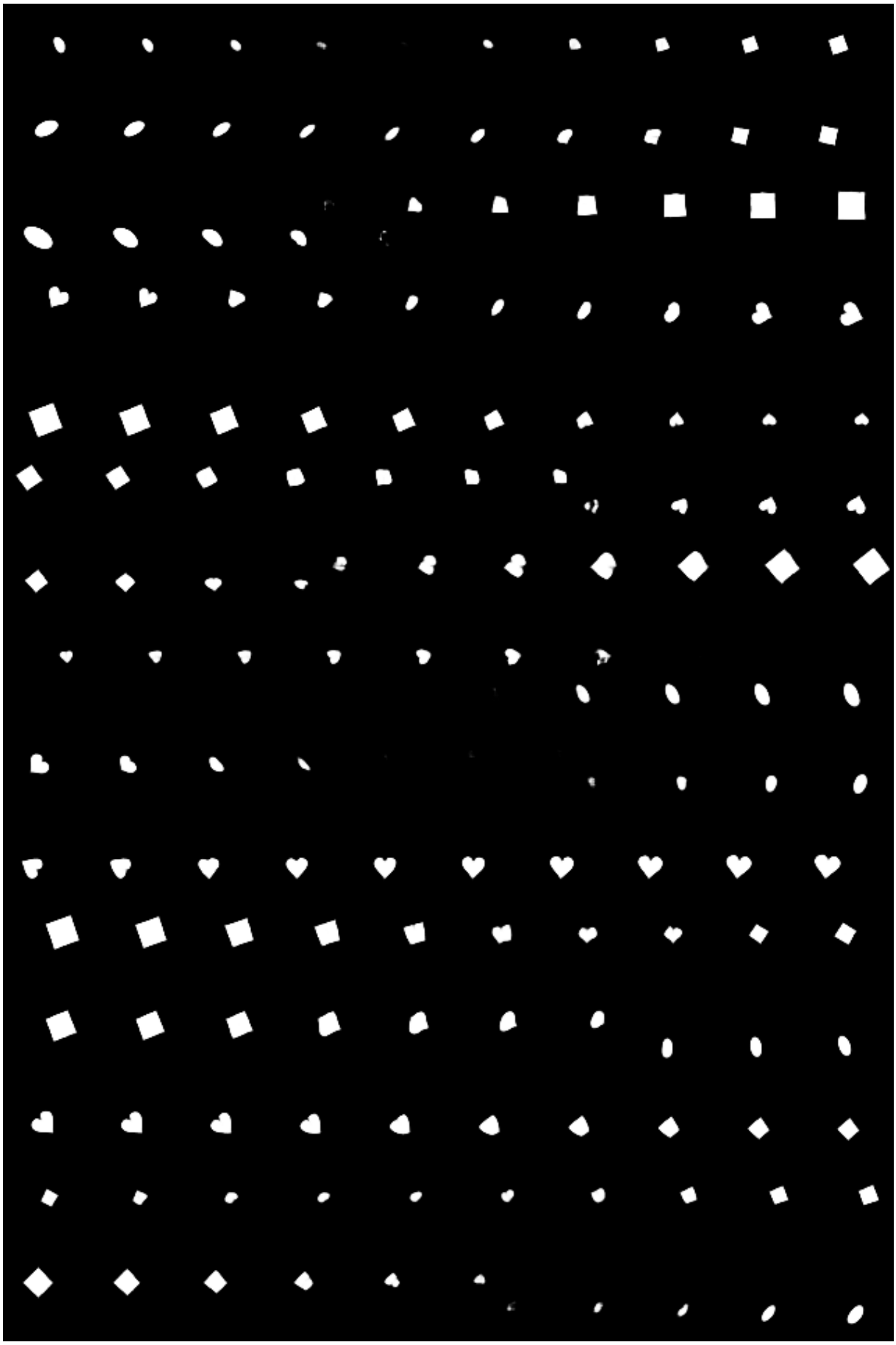


Figure 7.3. Interpolation of $VAE - 1024$

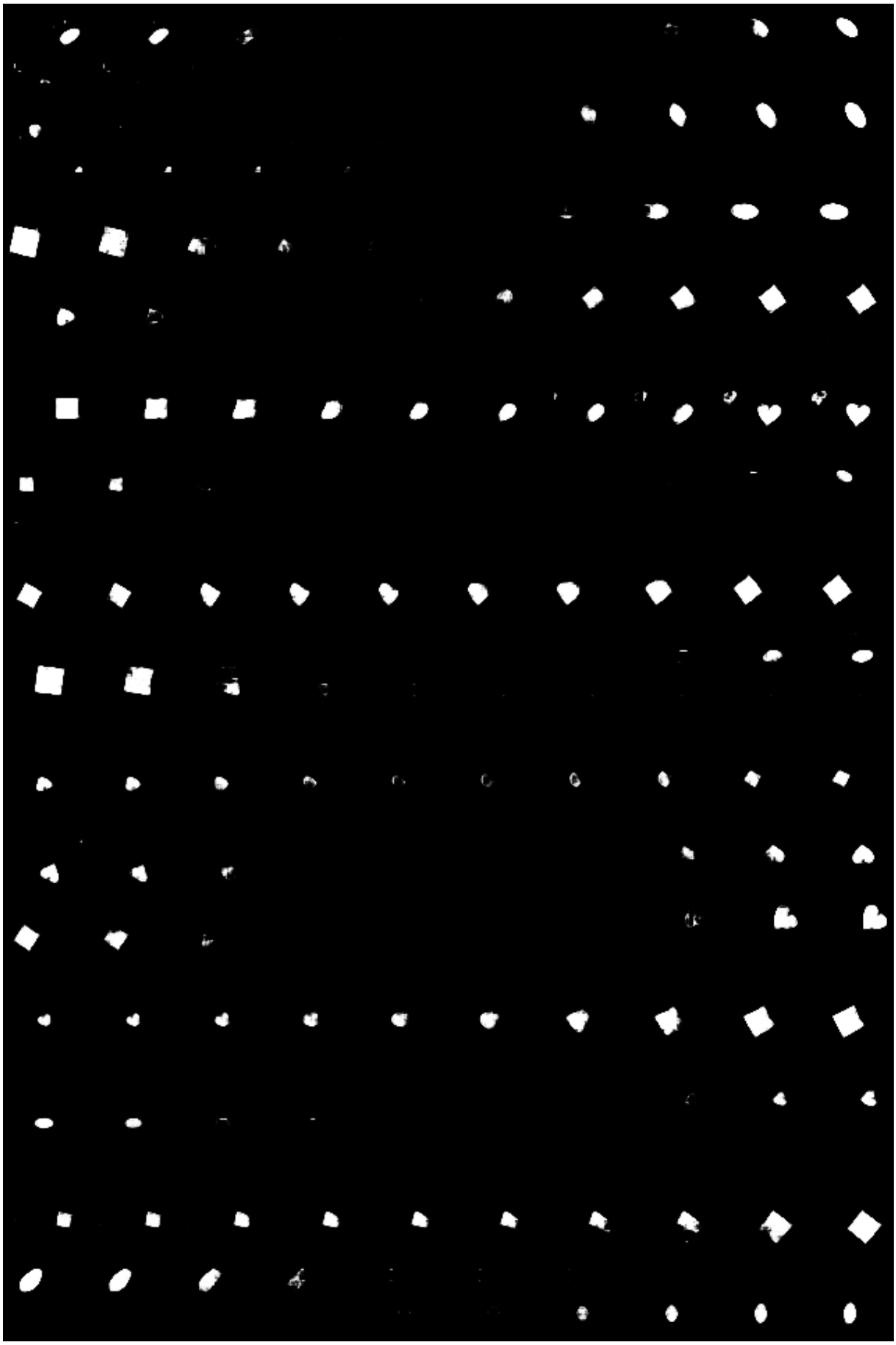


Figure 7.4. Interpolation of $CONV - VAE - 4 - 16 - 16$

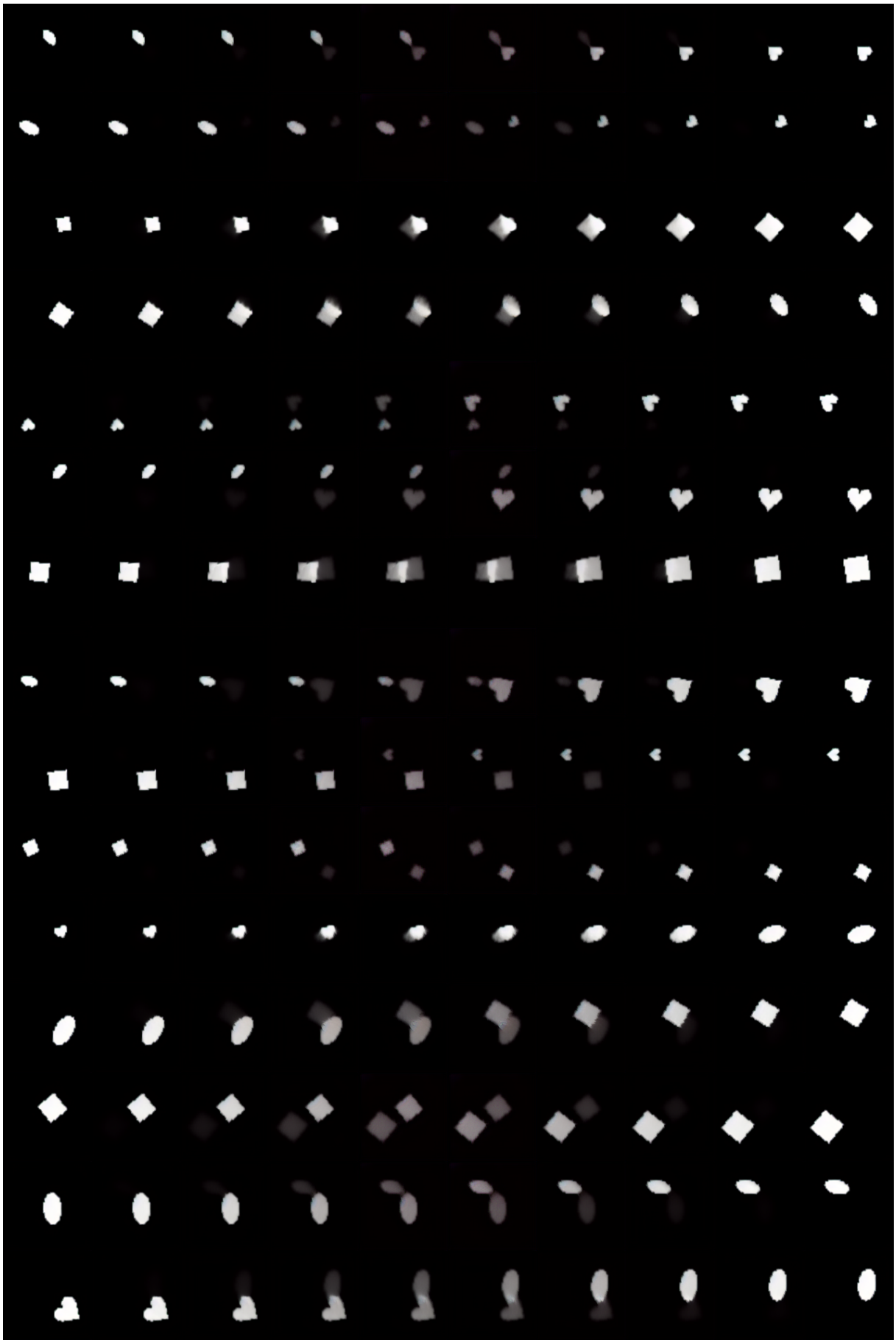


Figure 7.5. Interpolation of $KL - VAE - 4 - 16 - 16$

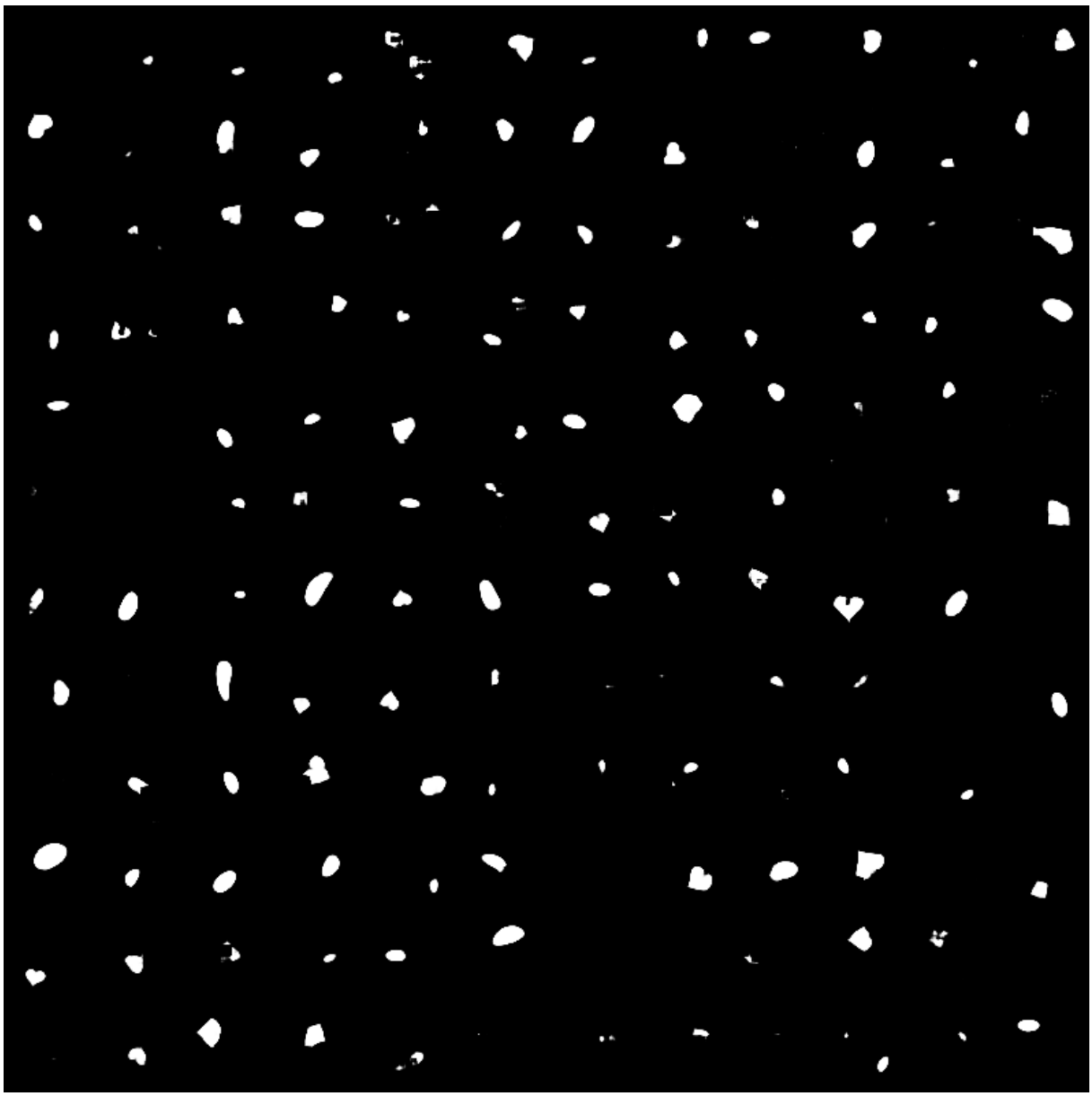


Figure 7.6. Samples of $VAE - 256$

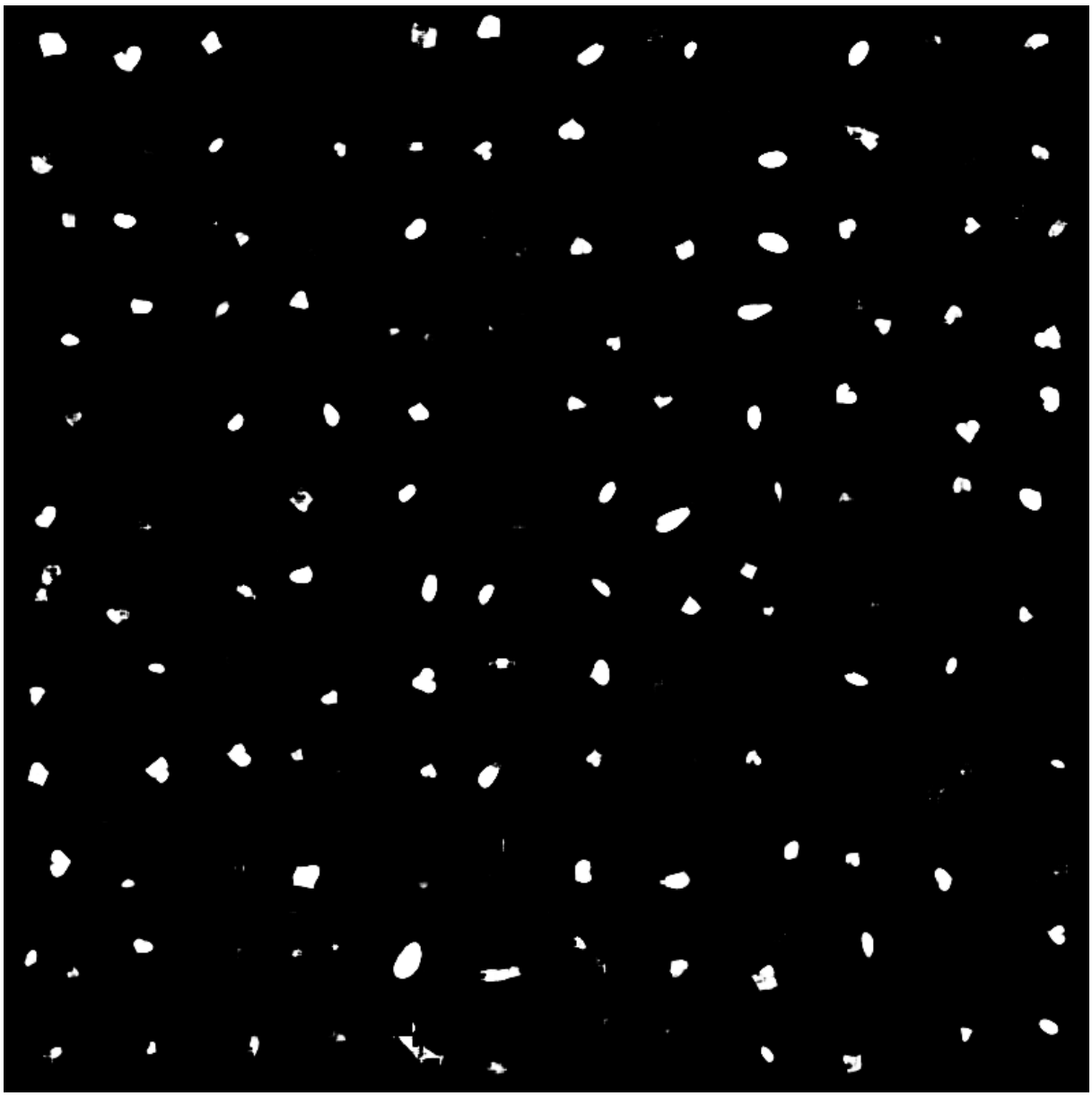


Figure 7.7. Samples of $VAE - 128$

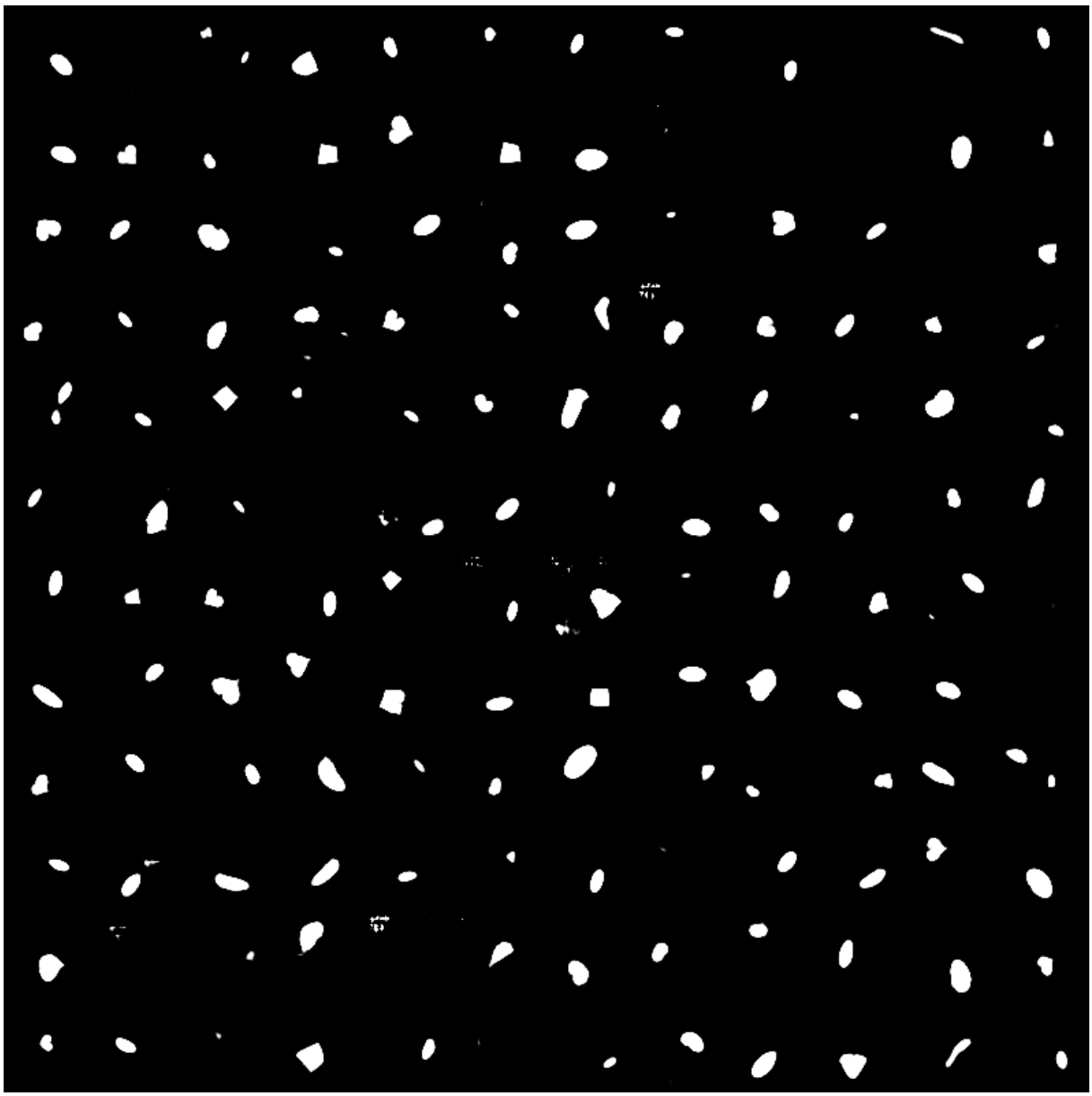


Figure 7.8. Samples of $VAE - 64$

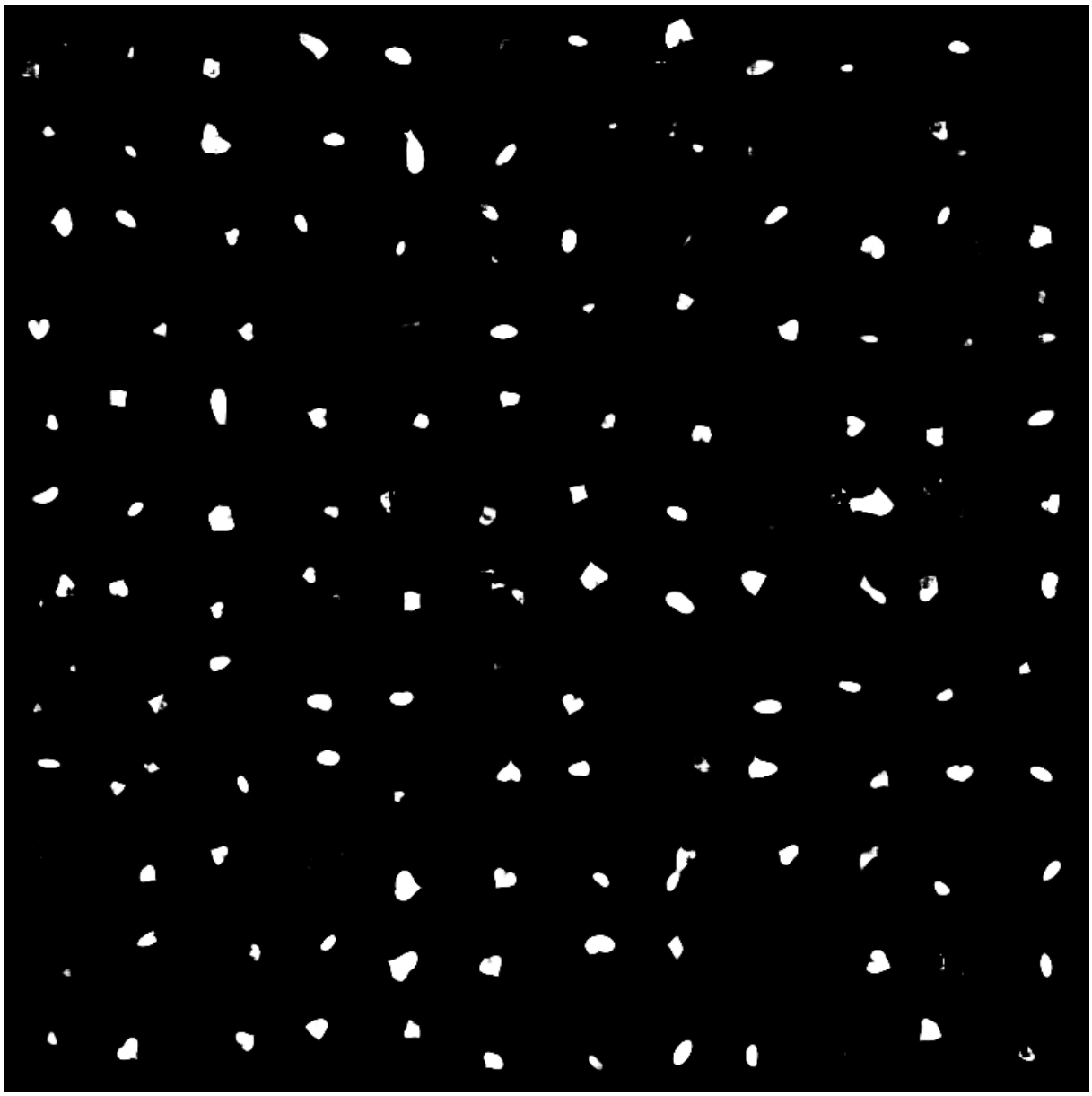


Figure 7.9. Samples of $VAE - 16$

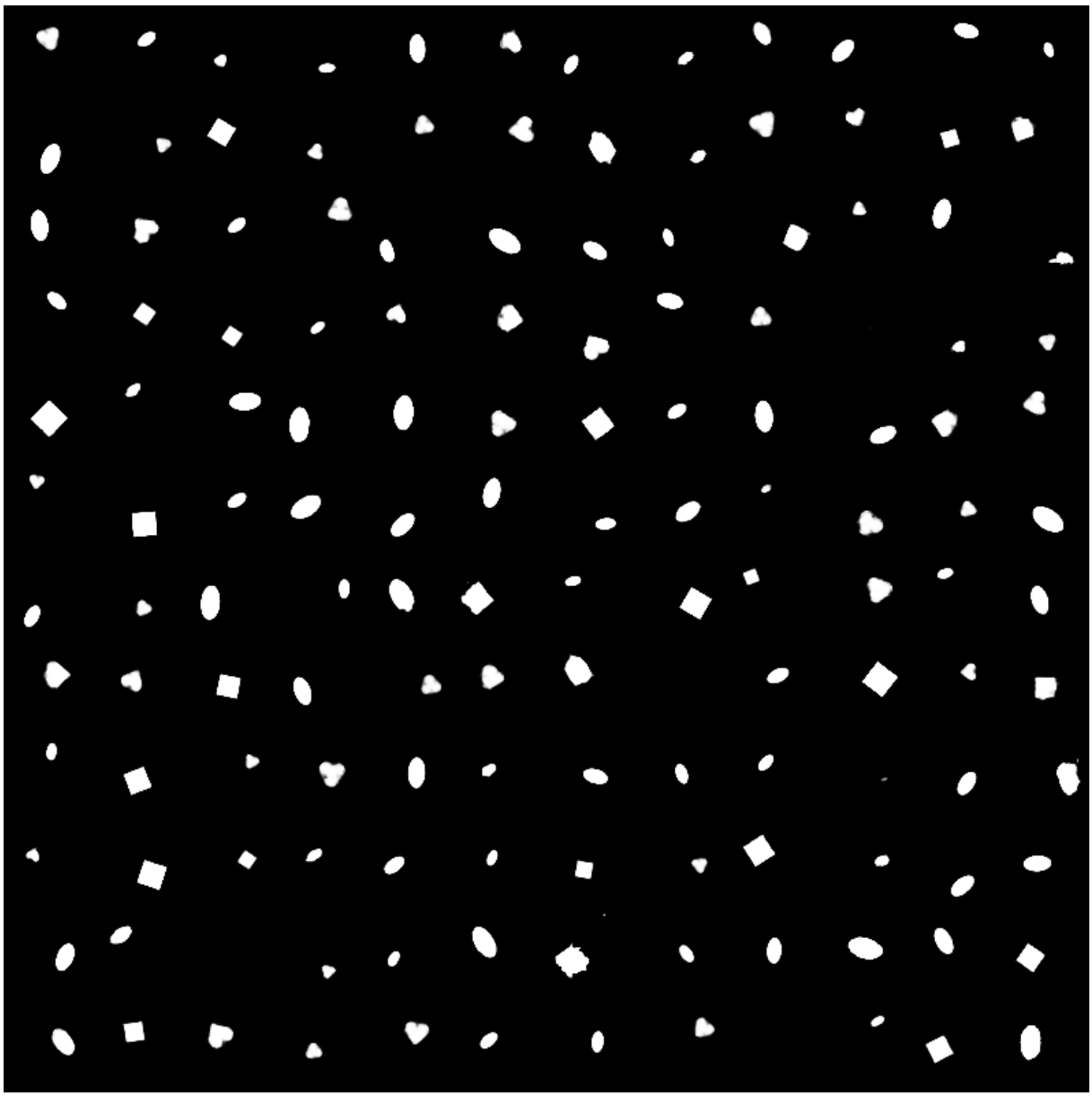


Figure 7.10. Samples of $VAE - 4$

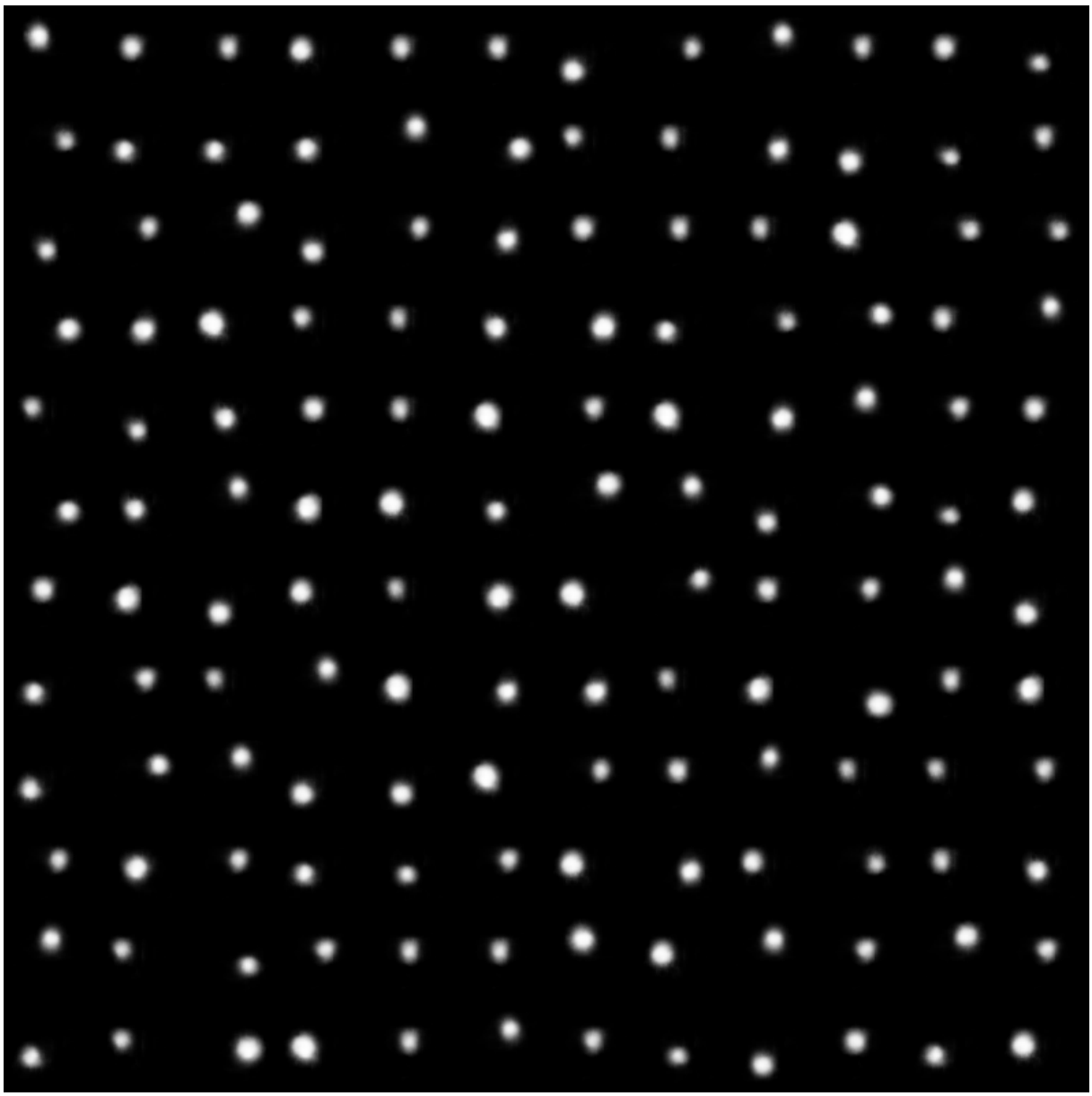


Figure 7.11. Samples of $VAE - 1$

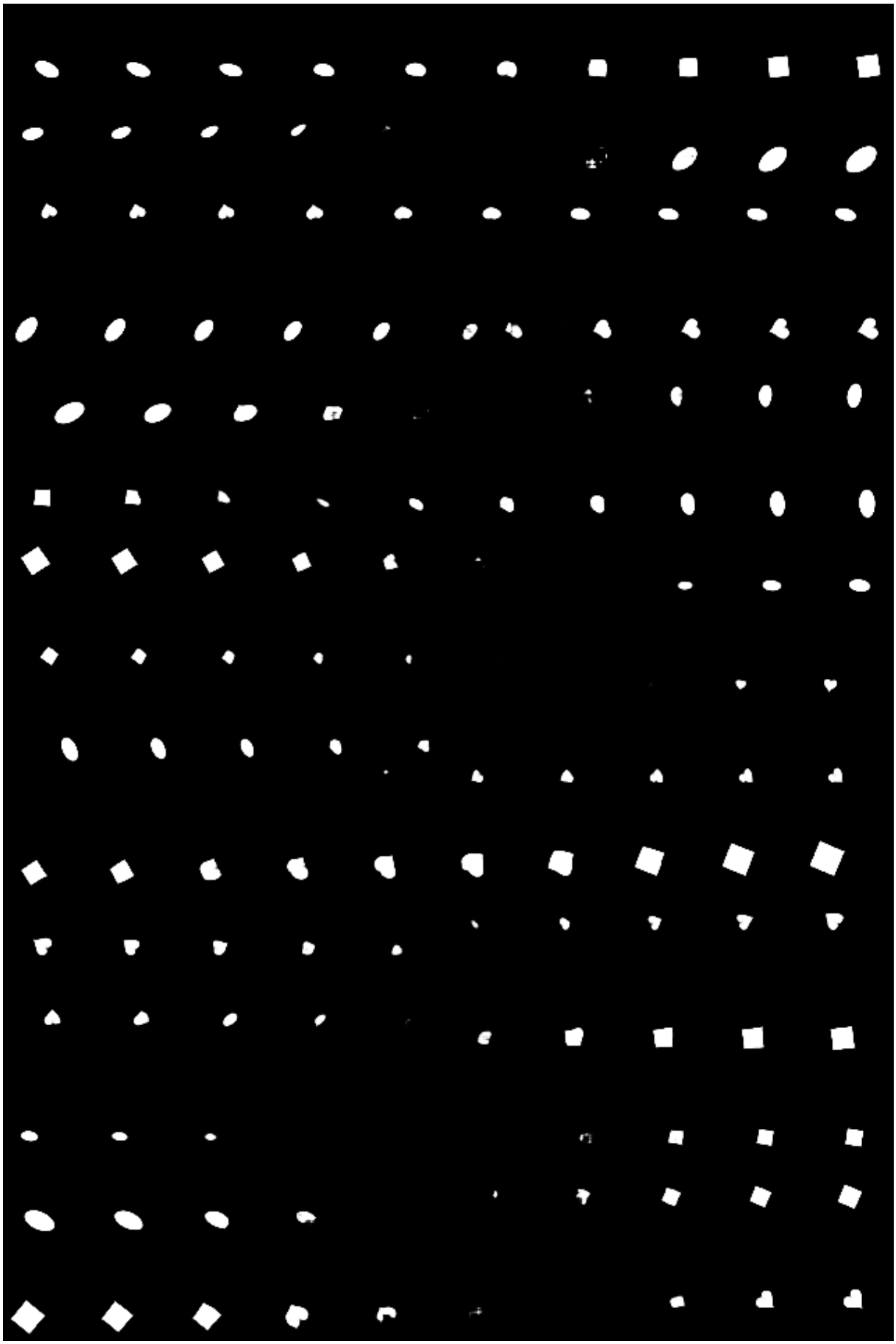


Figure 7.12. Interpolation of $VAE - 256$

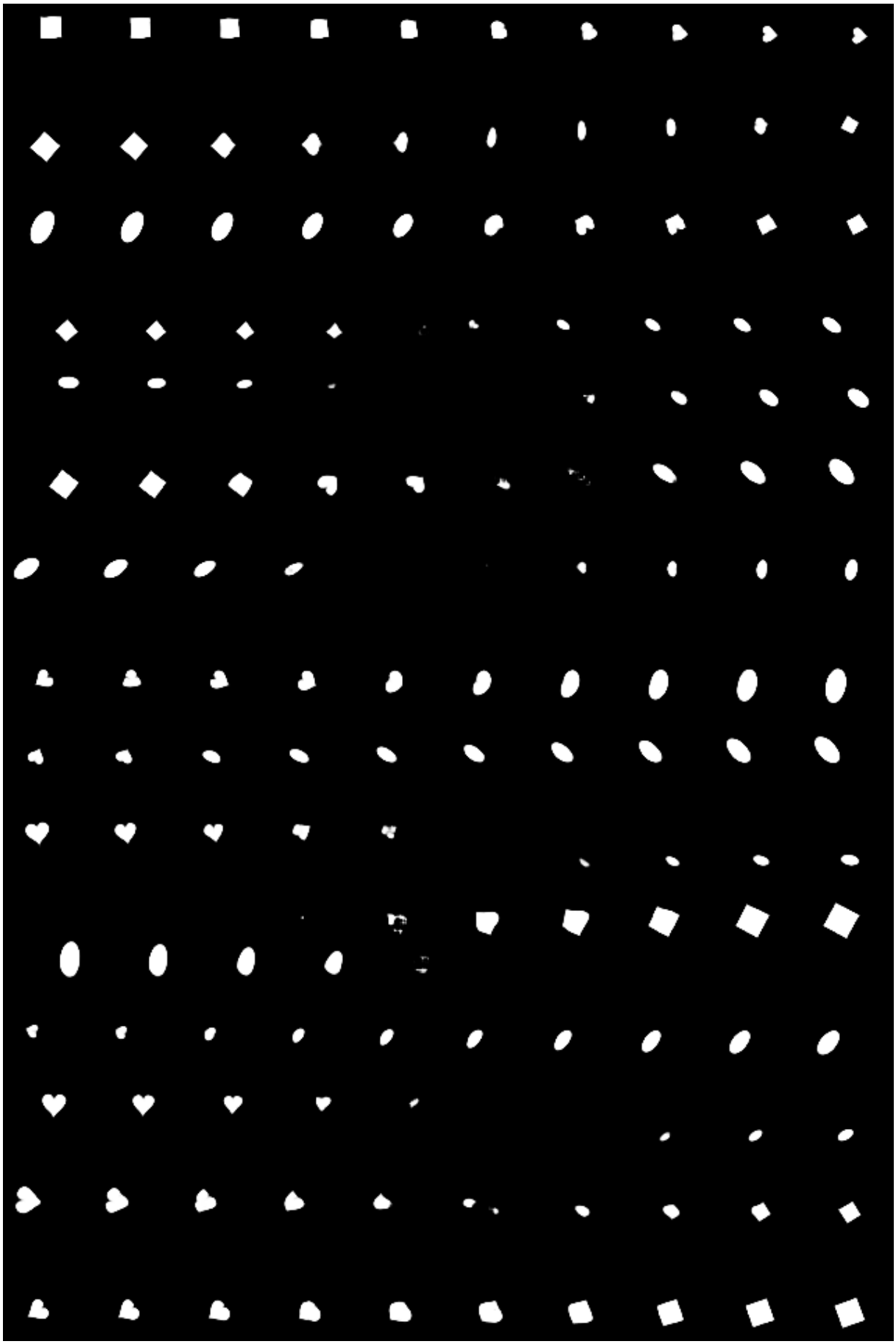


Figure 7.13. Interpolation of $VAE - 128$

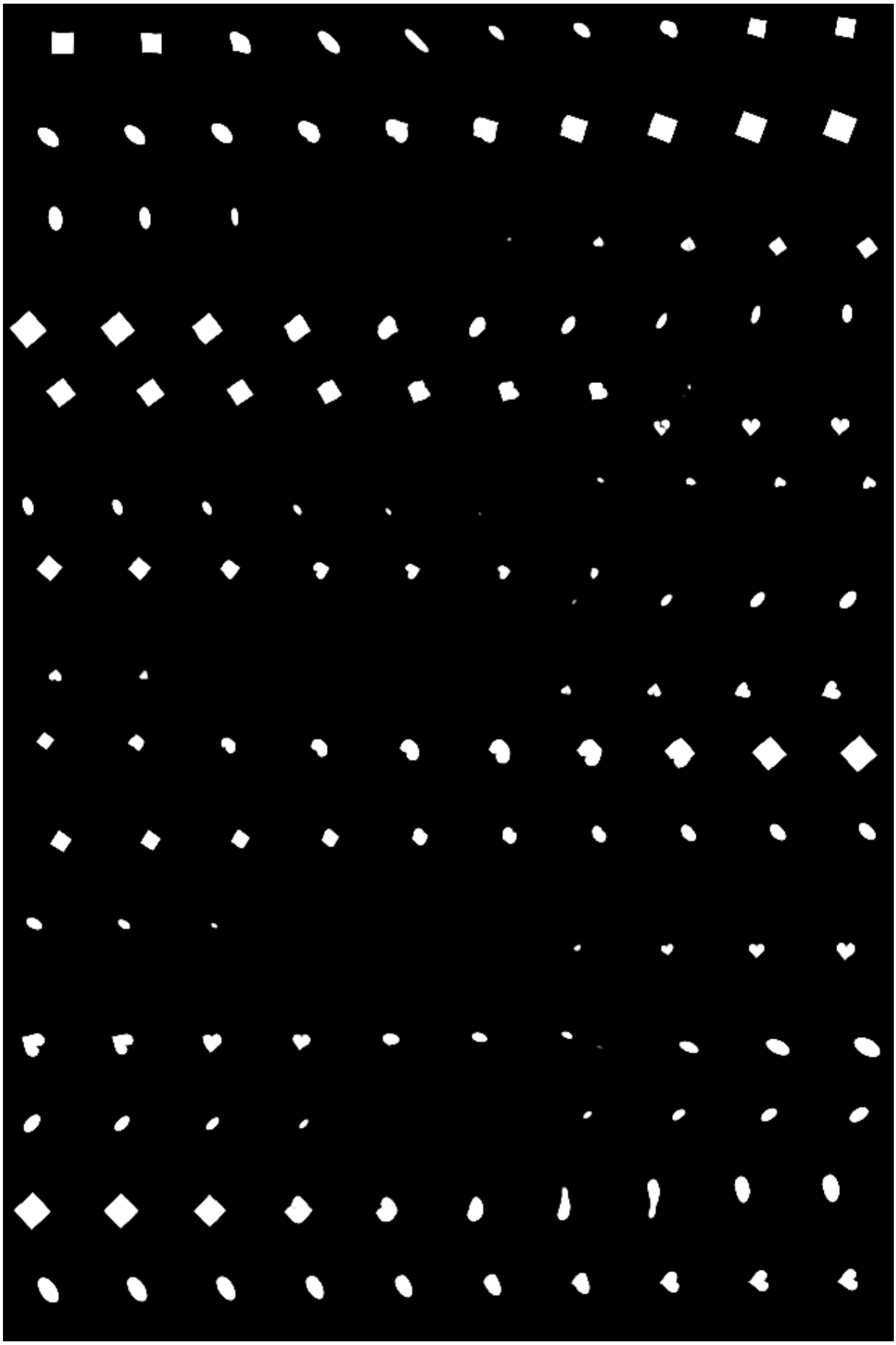


Figure 7.14. Interpolation of $VAE - 64$

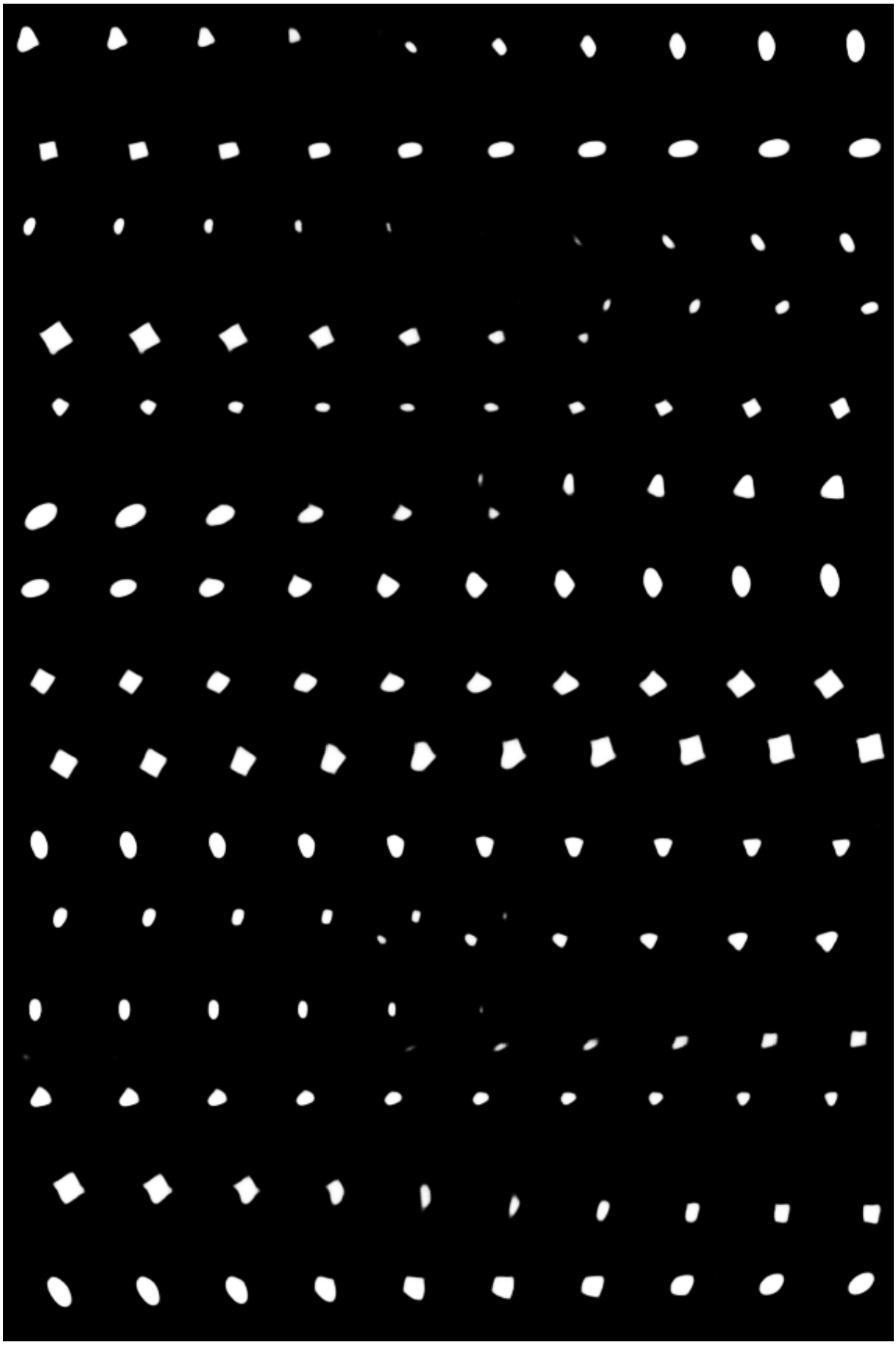


Figure 7.15. Interpolation of $VAE - 16$

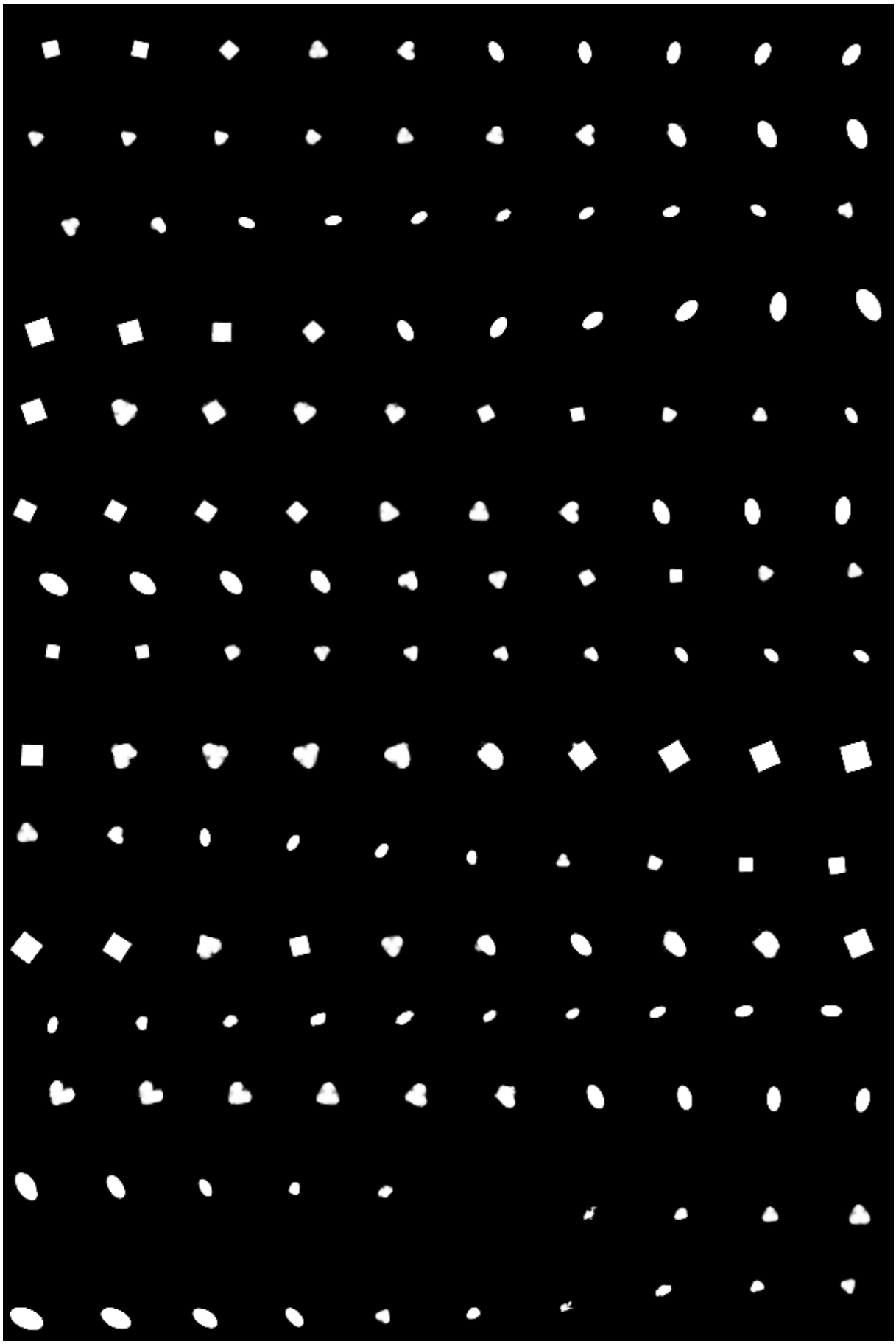


Figure 7.16. Interpolation of $VAE-4$

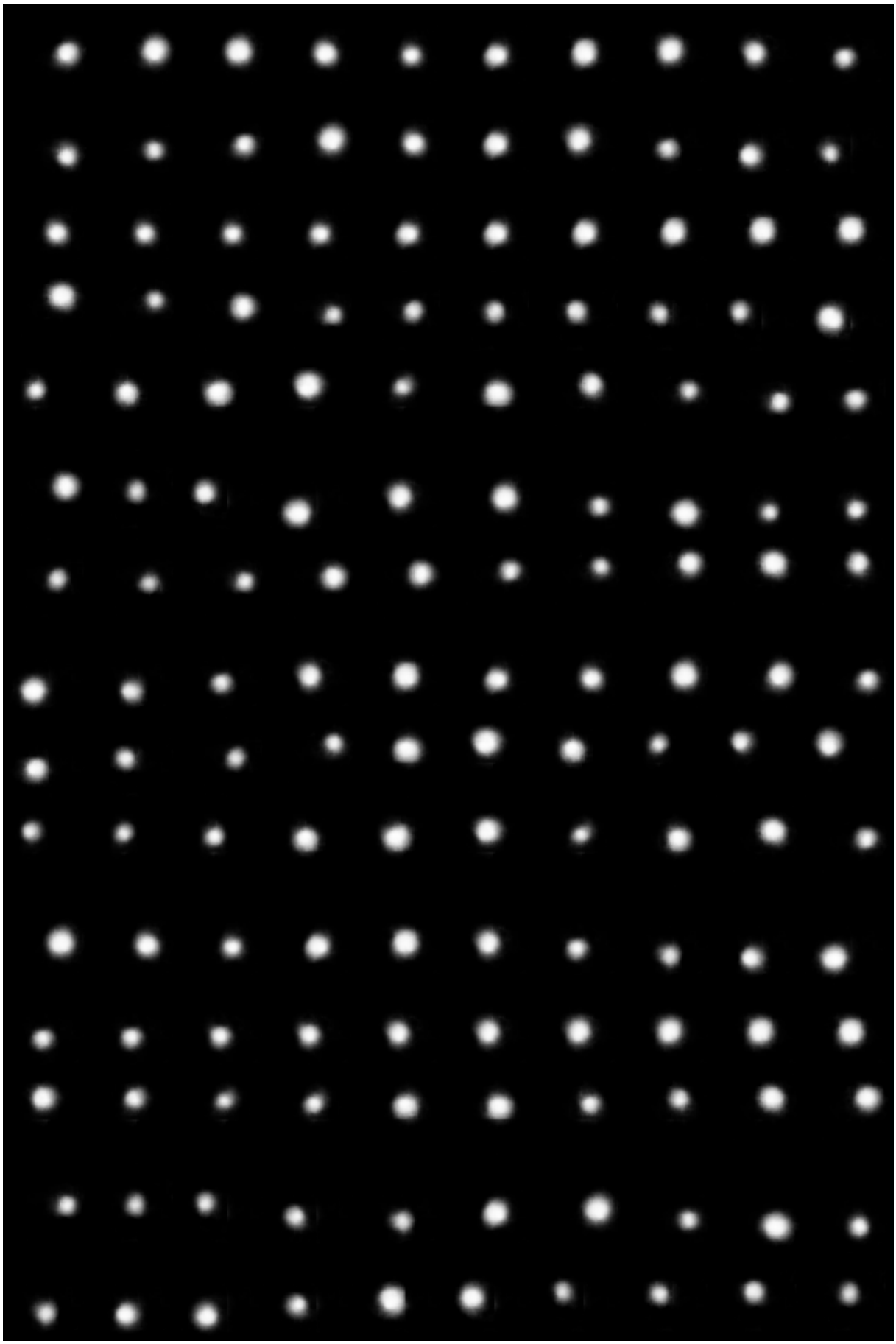


Figure 7.17. Interpolation of $VAE-1$

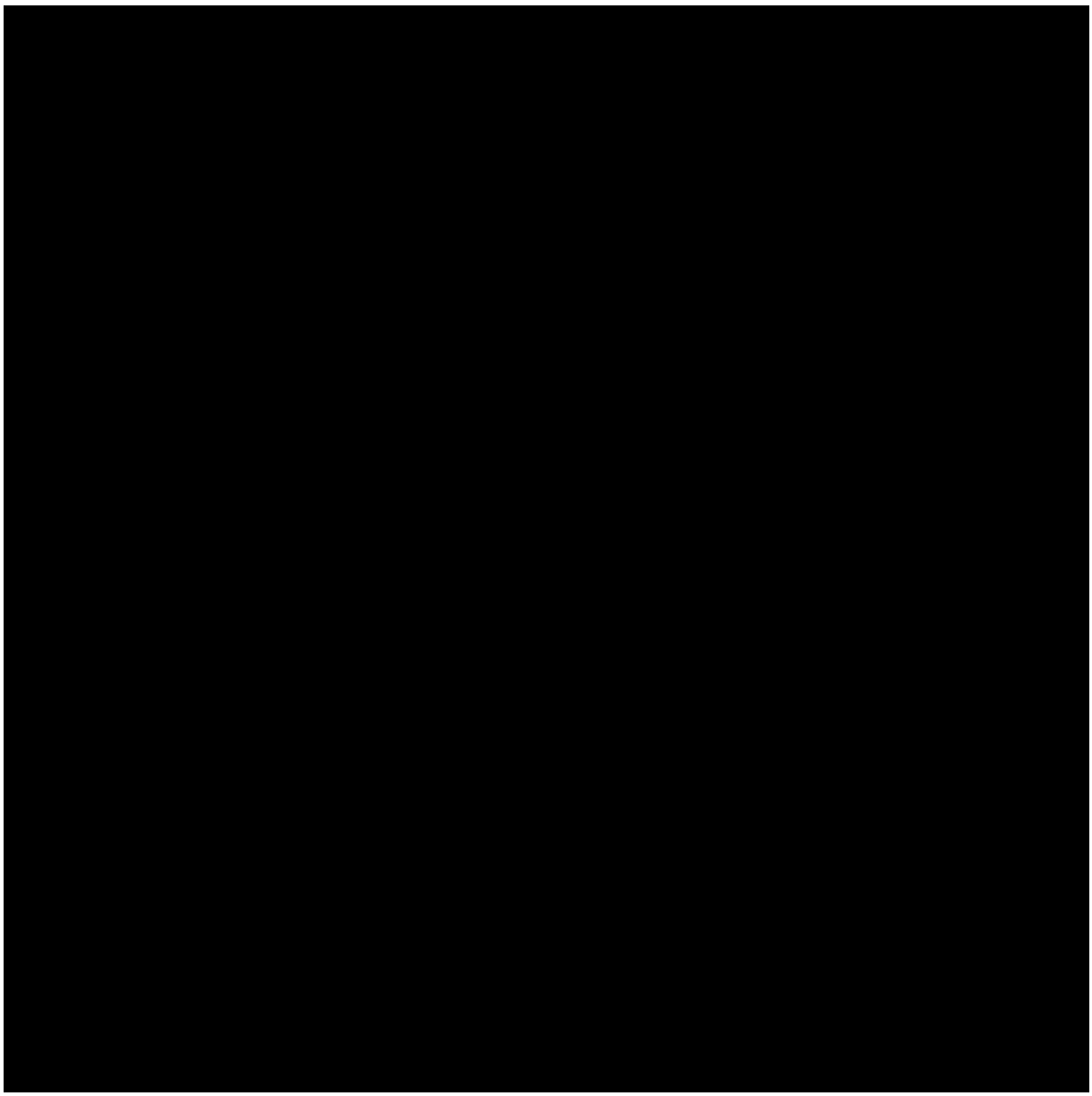


Figure 7.18. Samples of $AE - 256$

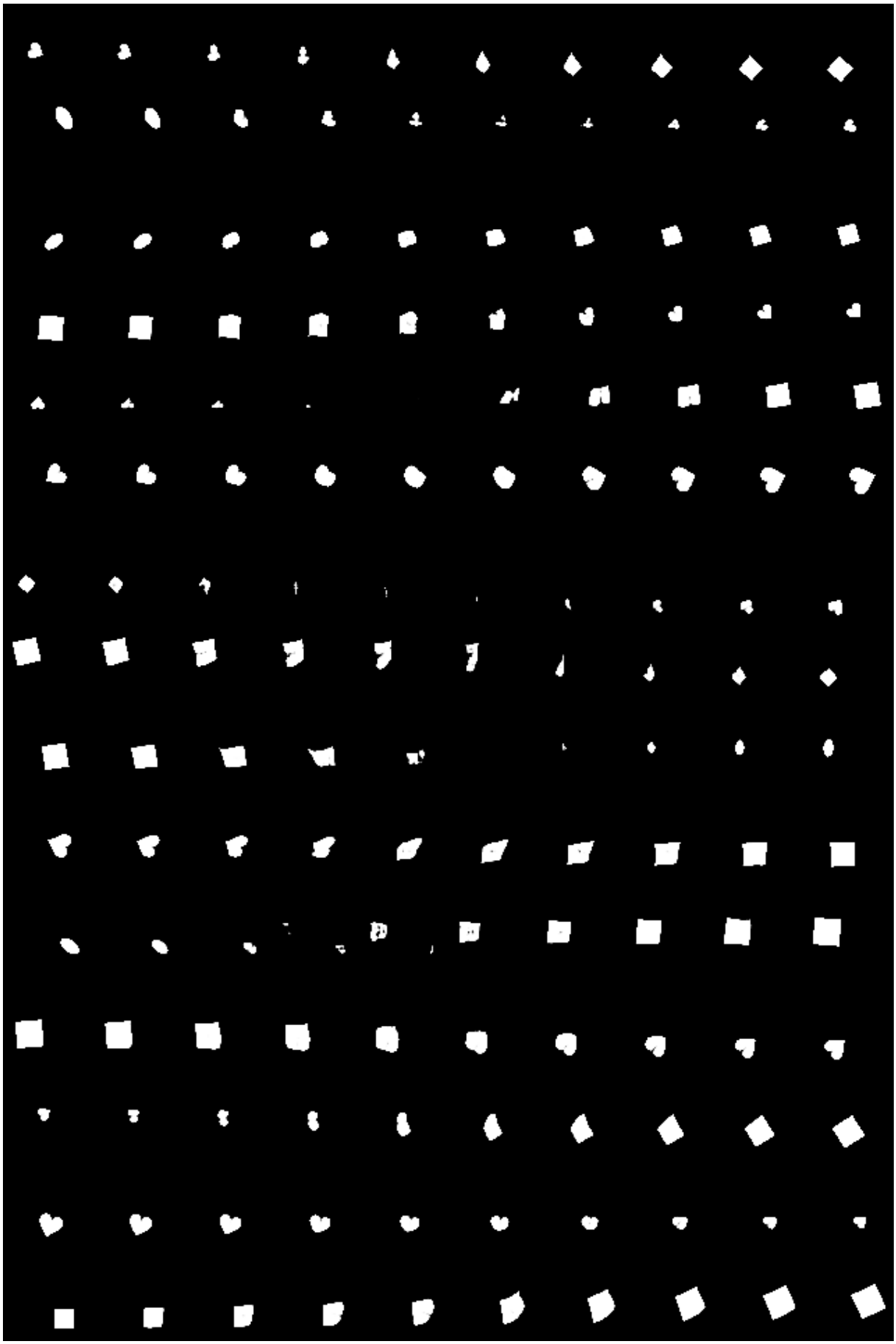


Figure 7.19. Interpolation of $AE - 256$

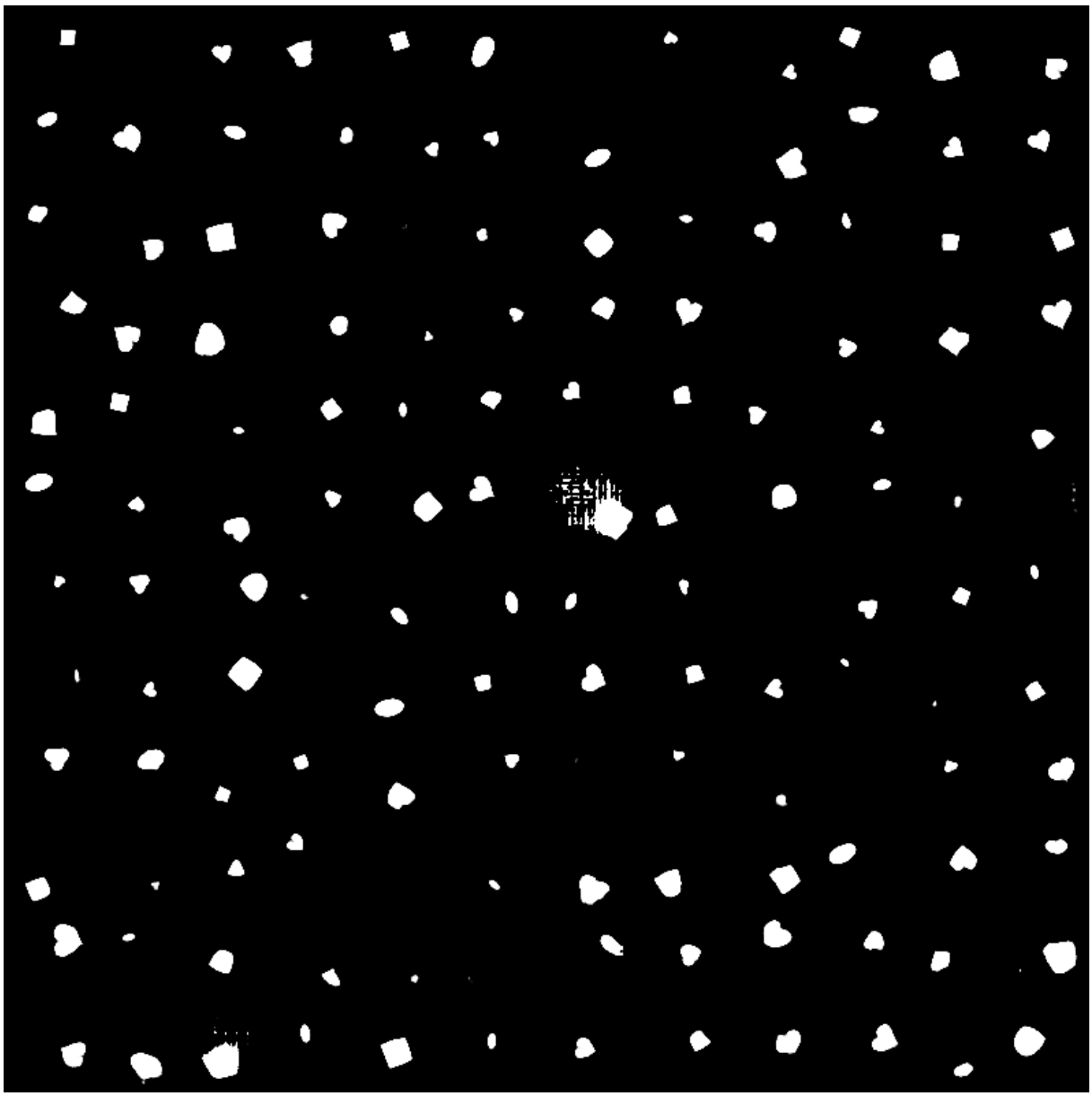


Figure 7.20. Samples of $BVAE - 256$

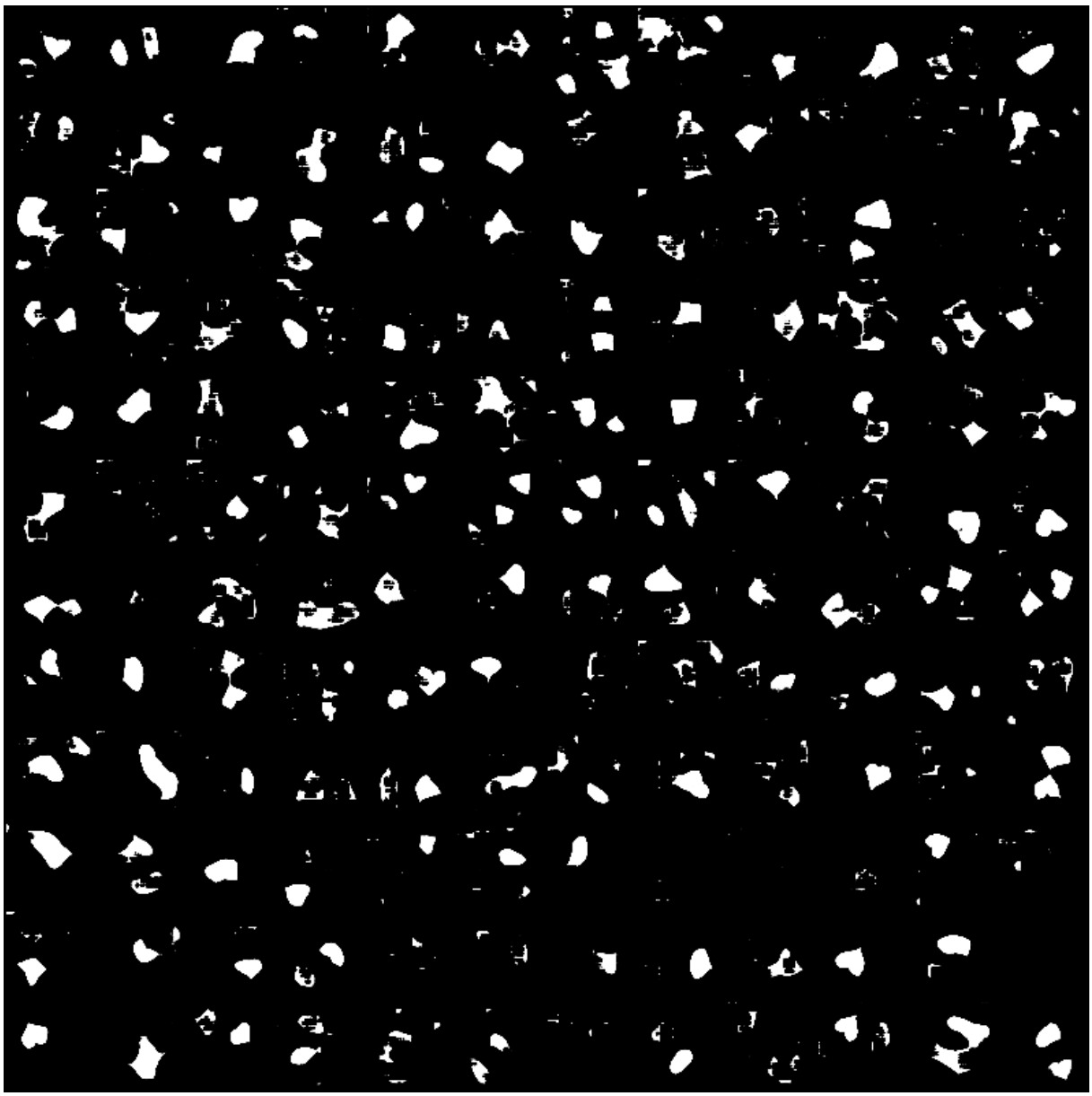


Figure 7.21. Samples of $DIPVAE - 256$

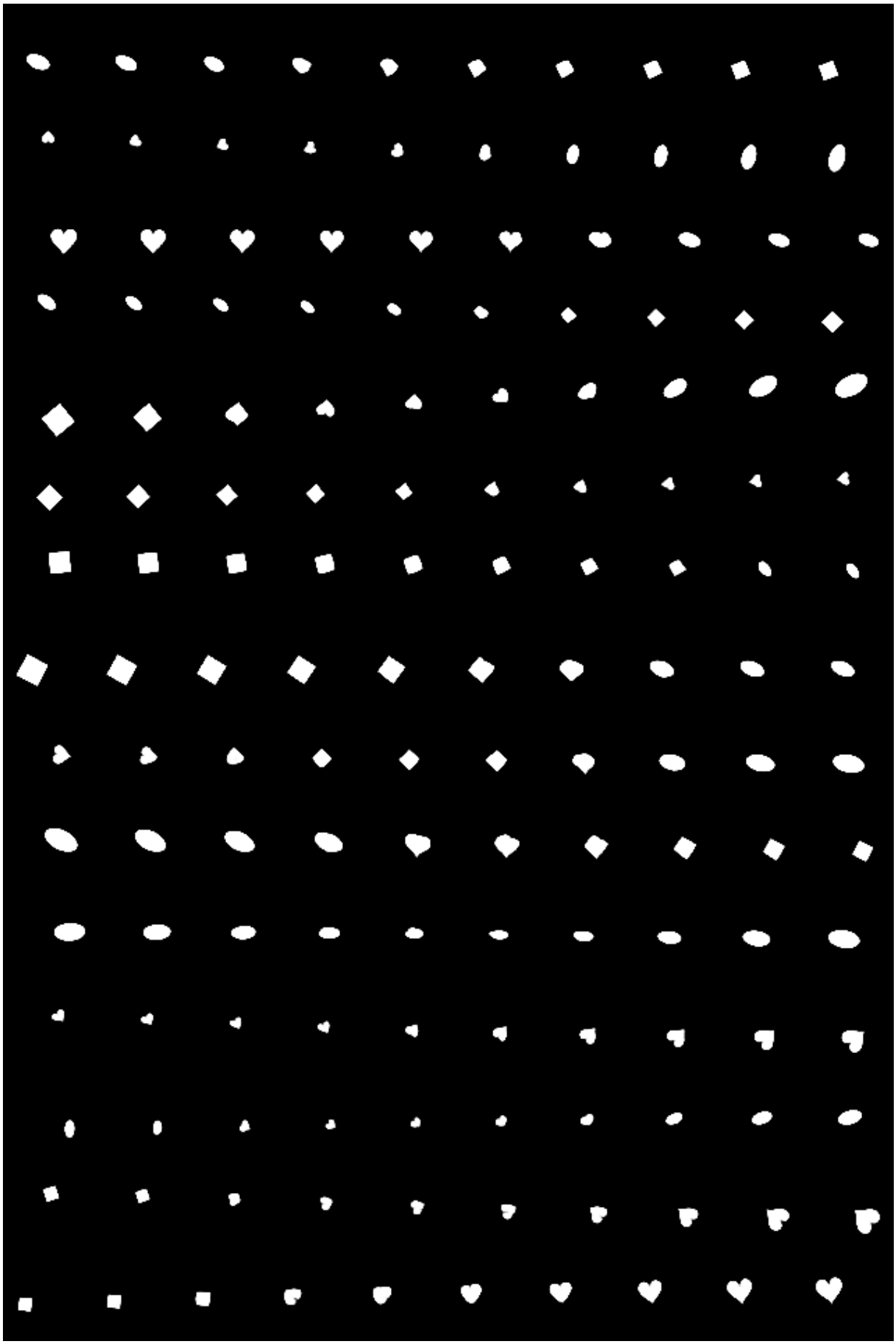


Figure 7.22. Interpolation of $BVAE - 256$

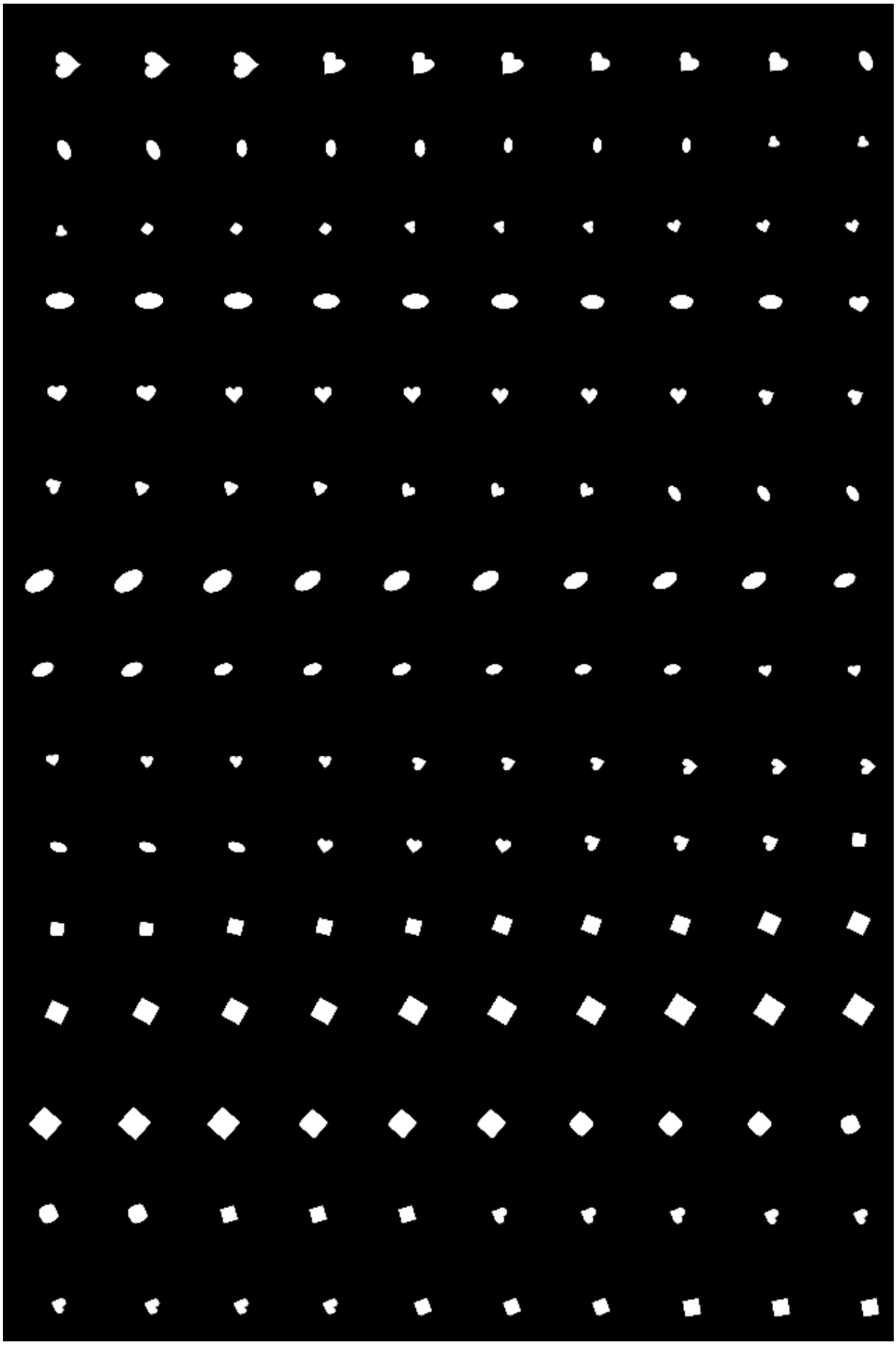


Figure 7.23. Interpolation of $DIPVAE - 256$

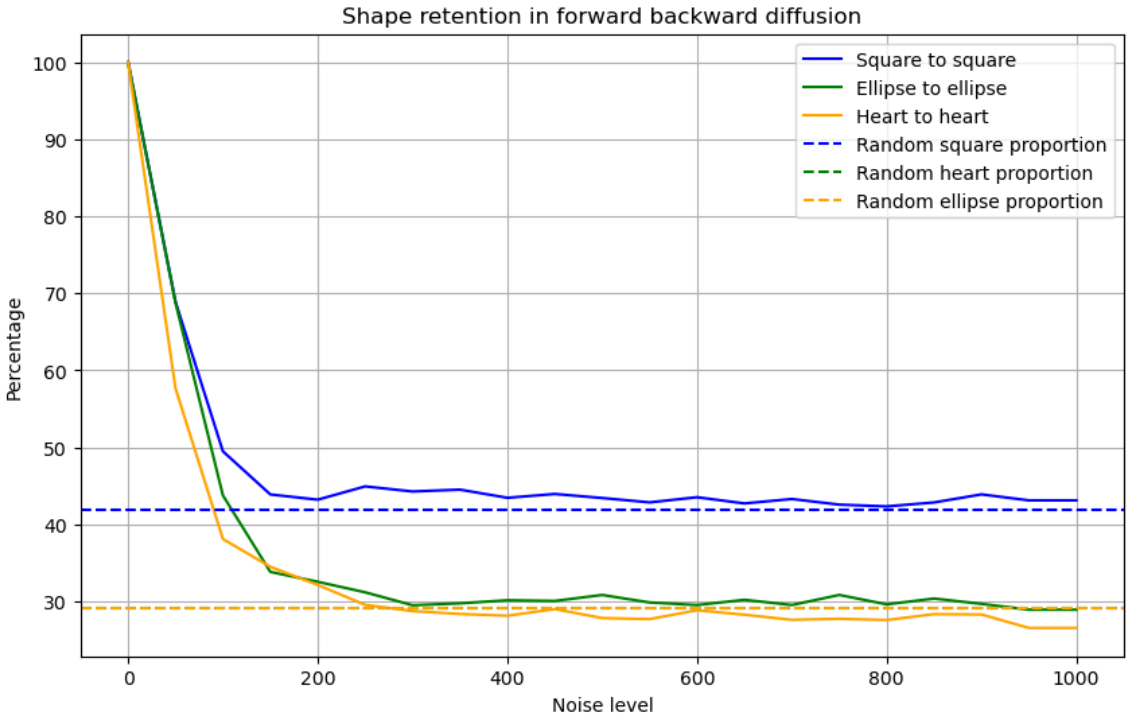


Figure 7.24. Shape retention using a diffusion model in $KL-VAE$ space. The procedure to generate this plot is identical to that described in 4.38 except that diffusion takes place in a latent space and classifiers/regressors also work in the latent space.

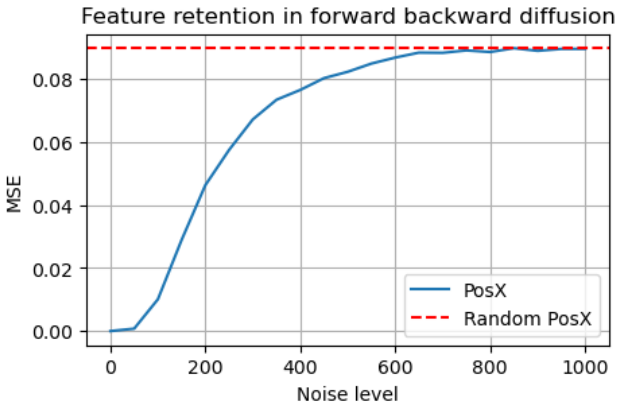


Figure 7.25. X-position retention using a diffusion model in $KL-VAE$ space. The procedure to generate this plot is identical to that described in 4.39 except that diffusion takes place in a latent space and classifiers/regressors also work in the latent space.

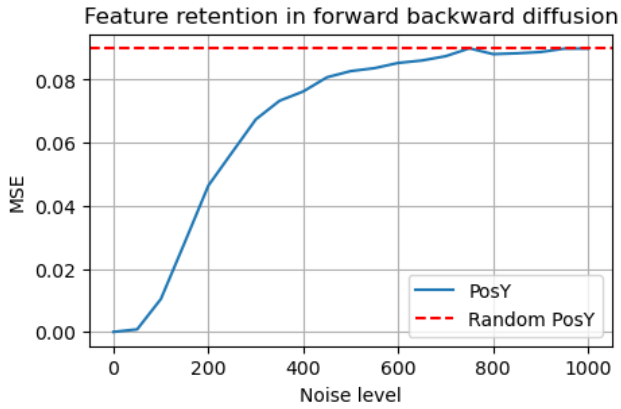


Figure 7.26. Y-position retention using a diffusion model in $KL - VAE$ space.

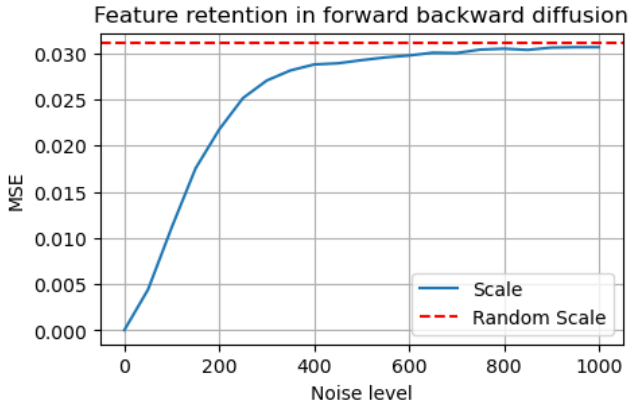


Figure 7.27. Scale retention using a diffusion model in $KL - VAE$ space.

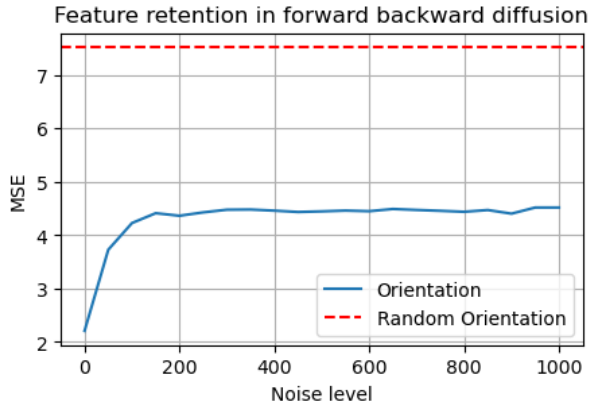


Figure 7.28. Orientation retention using a diffusion model in $KL - VAE$ space.

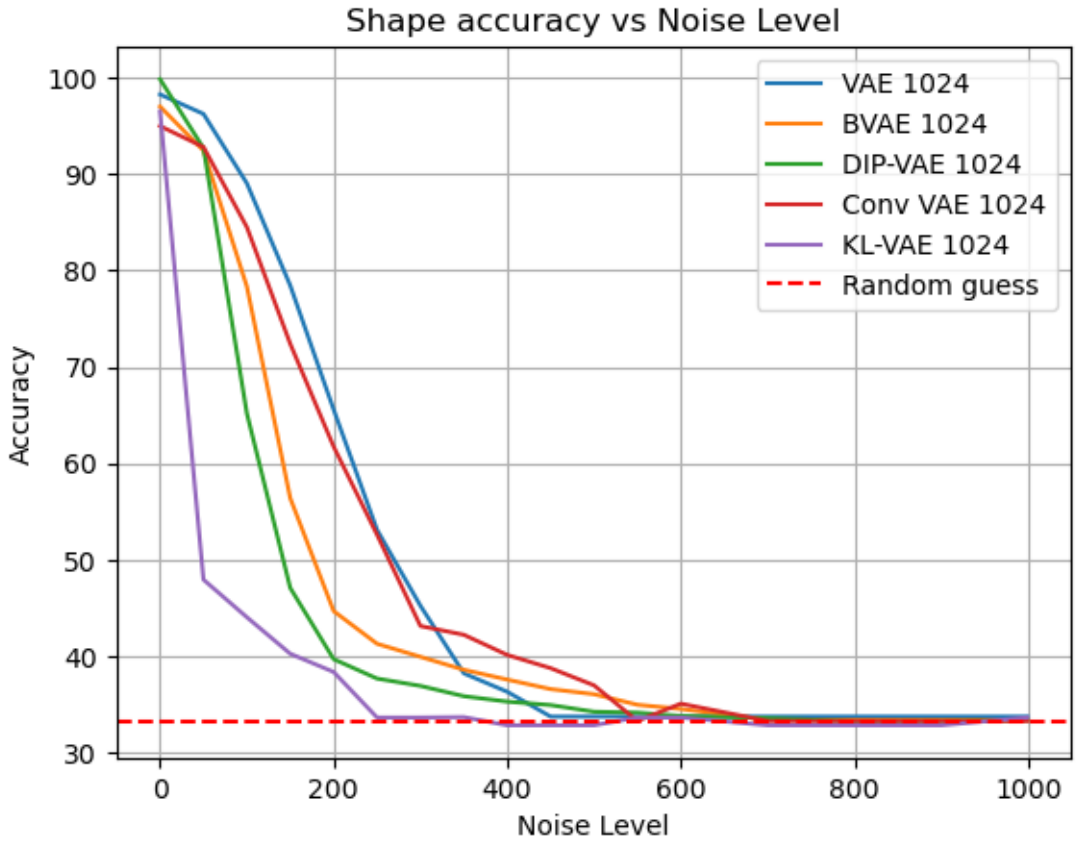


Figure 7.29. Shape classifier performance on data that gets gradually more noisy. For each timestep we train a classifier on the latent noisy sample shape and report its performance on the test set. As we can see, the accuracy curve is strongly dependent on the latent space. It seems latent structure plays almost no role here as the best and worst candidate are respectively the $KL-VAE$ and the $CONV-VAE$

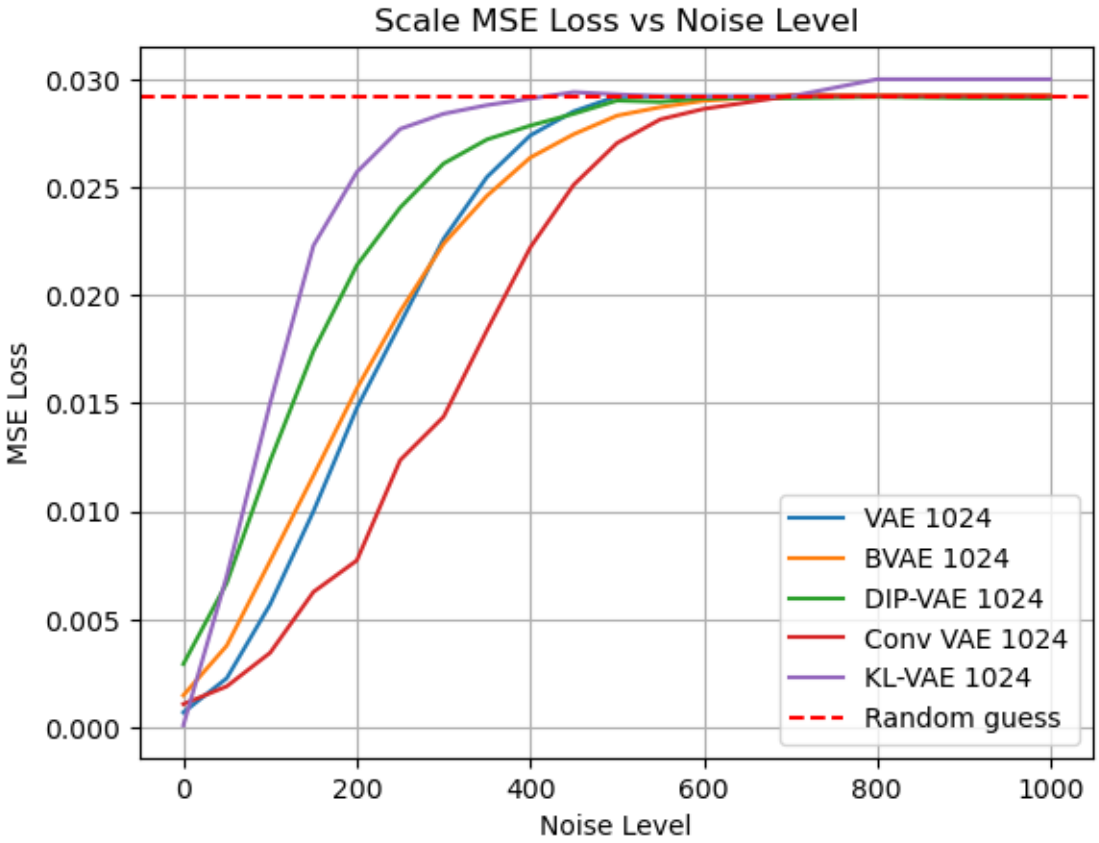


Figure 7.30. Scale regressor performance on data that gets gradually more noisy. For each timestep we train a regressor on the latent noisy sample x-position and report its performance on the test set. As we can see, the MSE curve is different for each latent space and it seems the $KL-VAE$ and the $CONV-VAE$ are here again the worst and best candidates when it comes to resisting to information destruction.

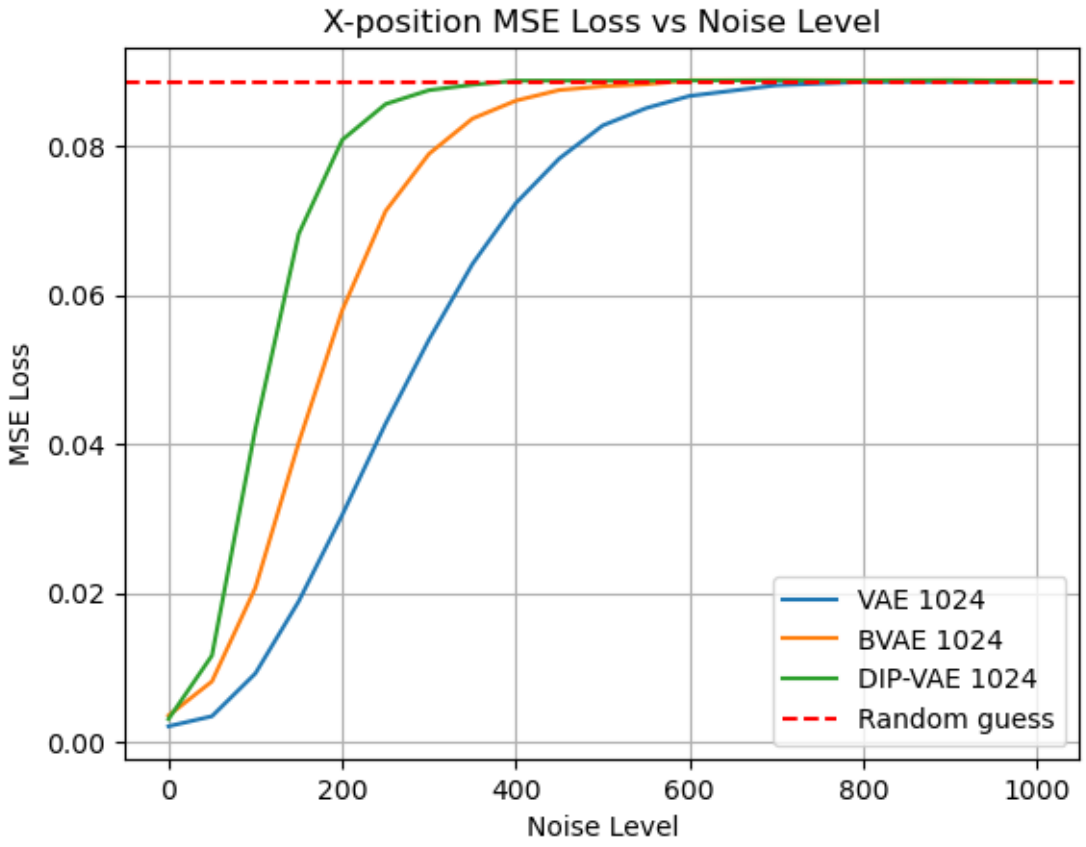


Figure 7.31. X-position regressor performance on latent data that gets gradually more noisy.

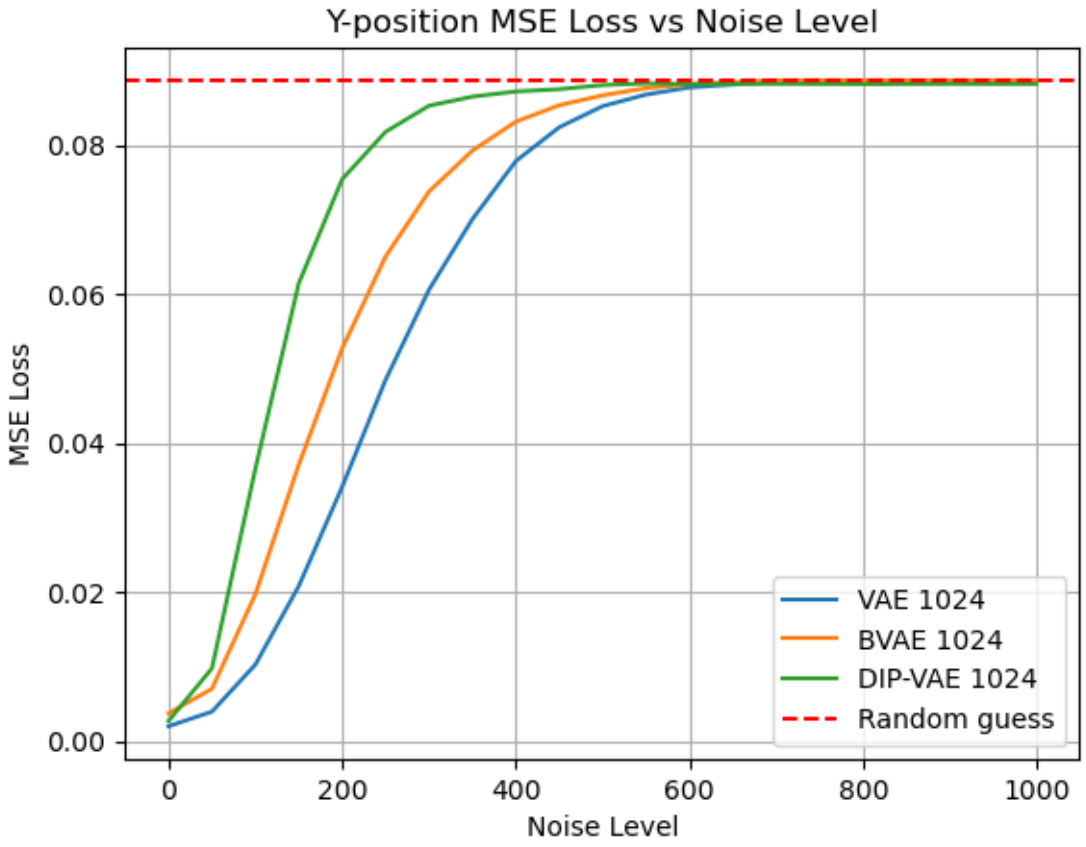


Figure 7.32. Y-position regressor performance on latent data that gets gradually more noisy.

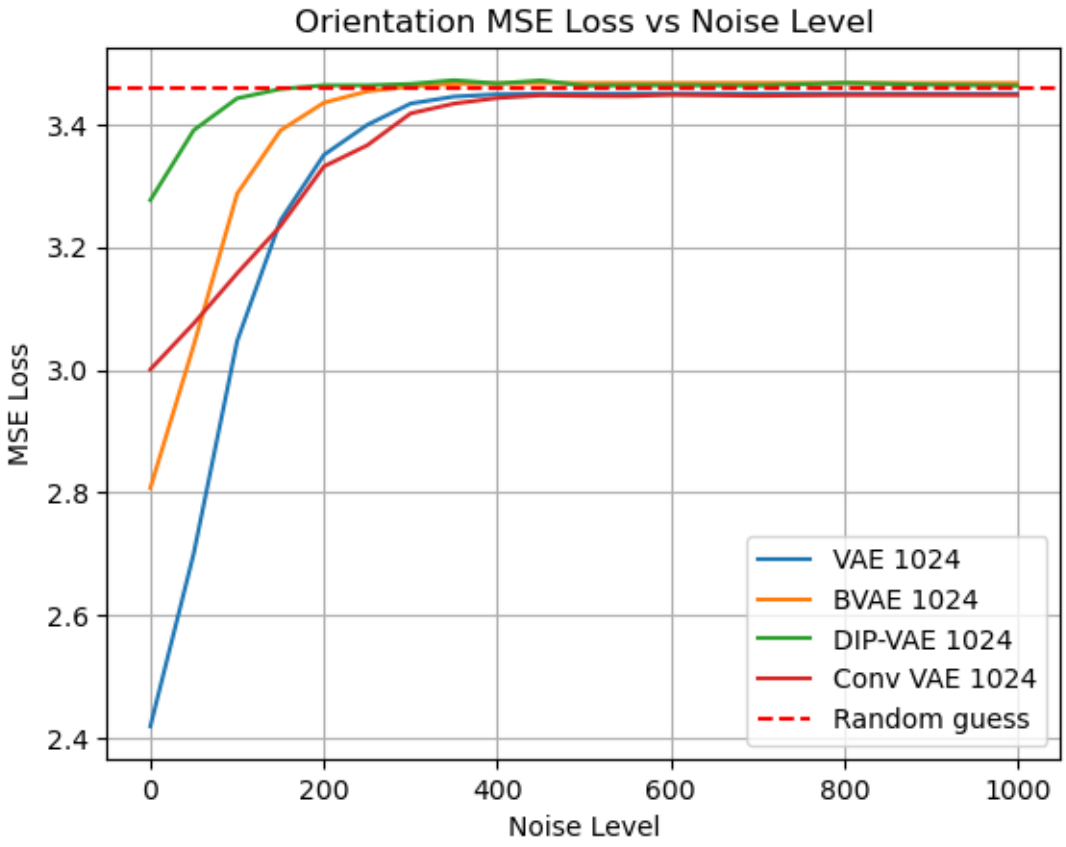


Figure 7.33. Orientation regressor performance on latent data that gets gradually more noisy.

Bibliography

[1] Jascha Sohl-Dickstein et al. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". 2015. arXiv: 1503.03585 [cs.LG] (pages 6, 26).

[2] Yang Song and Stefano Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". 2020. arXiv: 1907.05600 [cs.LG] (pages 6, 28, 30).

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". 2020. arXiv: 2006.11239 [cs.LG] (page 6).

[4] Yang Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". 2021. arXiv: 2011.13456 [cs.LG] (page 6).

[5] Robin Rombach et al. "High-Resolution Image Synthesis with Latent Diffusion Models". 2022. arXiv: 2112.10752 [cs.CV] (pages 9, 35, 61).

[6] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=Sy2fzU9gl (page 25).

[7] Christopher P. Burgess et al. "Understanding disentangling in β-VAE". 2018. arXiv: 1804.03599 [stat.ML] (page 26).

[8] Aapo Hyvärinen. "Estimation of Non-Normalized Statistical Models by Score Matching". In: *Journal of Machine Learning Research* 6.24 (2005), pp. 695–709. URL: http://jmlr.org/papers/v6/hyvarinen05a.html (page 27).

[9] Kamil Deja et al. "On Analyzing Generative and Denoising Capabilities of Diffusion-based Deep Generative Models". 2022. arXiv: 2206.00070 [cs.LG] (pages 36, 104).

[10] Tim Sainburg et al. "Generative adversarial interpolative autoencoding: adversarial training on latent space interpolations encourage convex latent distributions". 2019. arXiv: 1807.06650 [cs.LG] (page 42).

[11] Ting Chen. "On the Importance of Noise Scheduling for Diffusion Models". 2023. arXiv: 2301.10972 [cs.CV] (page 75).

[12] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. "Variational Inference of Disentangled Latent Concepts from Unlabeled Observations". 2018. arXiv: 1711.00848 [cs.LG] (page 86).

[13] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. "Neural Discrete Representation Learning". 2018. arXiv: 1711.00937 [cs.LG] (page 89).

[14] Dongjun Kim et al. "Refining Generative Process with Discriminator Guidance in Score-based Diffusion Models". 2023. arXiv: 2211.17091 [cs.CV] (page 101).

[15] Prafulla Dhariwal and Alex Nichol. “Diffusion Models Beat GANs on Image Synthesis”. 2021. arXiv: 2105.05233 [cs.LG] (page 104).