Almotion Bavaria

Technische Hochschule Ingolstadt

Fakultät Informatik
Fachgebiet Bildverstehen und medizinische Anwendung der künstlichen Intelligenz

# On Neural Compression using Diffusion Autoencoders

Bachelor Thesis

Dominic Rampas

Supervisor: Prof. Dr. Marc Aubreville
Date: January 18, 2024

Technische Hochschule Ingolstadt

# Speciment for declaration in Accordance with § 30 Abs. 4 Nr. 7 APO THI

---

## Declaration

I hereby declare that this thesis is my own work, that I have not presented it elsewhere for examination purposes and that I have not used any sources or aids other than those stated. I have marked verbatim and indirect quotations as such.

Ingolstadt, 17.01.2024
(Date)

D. Rampas

Dominic Rampas

## Abstract

This work presents an examination of using diffusion models to achieve efficient data compression. As diffusion models have become more dominant in areas such as generative modelling, this work shows that they achieve a strong performance on the task of image-, and video-compression and outperform classical algorithms, as well as established neural compression algorithms. This type of model, referred to as *Diffusion Autoencoders*, is able to achieve spatial compression factors of 32x, while achieving high quality reconstructions that even maintain fine details of the input. The bachelor thesis presents a thorough analysis of individual components of the diffusion autoencoder through extensive experiments, while undermining the solidness of the approach by numerous comparisons and ablation studies. Additionally, an in-depth overview of related work for classical-, and neural-compression, diffusion models and evaluation methods is given.

# Contents

# 1 Introduction

This bachelor thesis investigates the topic of data compression from a machine learning perspective. The key idea being the usage of modern machine learning methods to learn an efficient method of representing data with less information. This topic has a broad application spectrum. With the ever-increasing amount of data produced, efficient storage is important as never before. A typical way to solve the problem of exponential data growth is to increase the storage capacity by vertical or horizontal scaling. However, this is an expensive solution, even with the huge price decreases per kilobyte over the last decades [1]. A more sustainable approach would be to find a way to use the available space more efficiently and store more information with less space. This is the field of data compression and has seen uncountable research activity for years. Initially, as with many other fields, human ingenuity and intuition was used to construct algorithms that compress data. Until today, these still represent the standard approaches that are widely used everywhere. However, with the emerging progress in the field of artificial intelligence, more and more research delves into automating the process of finding good ways to compress data. This field is coined *Neural Data Compression* and has seen big breakthroughs. This work focuses on the *neural* aspect and examines current approaches that work well for representing data in smaller spaces than their original ones. Beforehand, an overview of *Classical Data Compression* is given, which serves as a benchmark to compare neural approaches later on.

Furthermore, over the last years a novel promising class of machine learning methods has evolved, called "Diffusion Models" [2] [3] [4] [5]. This approach has replaced many long-established methods, such as Generative Adversarial Networks in image modeling, by delivering better conditions and results. This work focuses on a type of diffusion model termed *Diffusion Autoencoder* to approach the task of data compression. In the following sections, this thesis delves into the theory behind diffusion models and specifically examines diffusion autoencoders, conducts experiments, evaluates the models, investigates their usability in the real world and gives a future outlook into their potential. While there are many data modalities, this work specifies into investigating visual data. Precisely, diffusion autoencoders for the modality of images and videos are explored. The goal of this work is to give an overview of the field of data compression and to show a new promising way of learning such using state-of-the-art methods from the field of machine learning. Improving neural mechanisms is of great interest, as current state-of-the-art methods have limits in

their compression factors and only enable small compressions. Increasing the amount of compression in existing methods leads to poor results that make it unusable in practice with these settings. For this reason, methods that built upon neural compression in any way, usually only make use of small, well working compressions. Enabling larger compression ratios stays an open research gap and therefore is of obvious importance and applicability. Concretely, established methods perform well up to a spatial compression factor of 8 and quickly degrade above that. With the idea of the diffusion autoencoder however, faithful reconstructions of spatial compressions going up to 32× can be observed. To give an example, encoding a $512 \times 512$ image with a spatial compression of 8, results in a representation of $64 \times 64$, while a 32× spatial compression encodes it to $12 \times 12$. This is a large increase, making further applications building atop this method cheaper and more efficient. The research questions of this work therefore are the following:

1. What are the current limitations of prior neural compression work?
2. How can diffusion models be used to learn efficient neural compression mechanisms?
3. What settings and training setups work best for diffusion autoencoders with a focus on images and videos?

# 2 Related Work

This section will explain & summarize methods and theories relevant to the discussed work of diffusion autoencoders. Firstly, an overview of classical compression will be shown, followed by an outline of the neural counterpart. Afterwards, an introduction to diffusion models is given and eventually a rundown of applicable evaluation metrics is presented.

## 2.1 Compression

Data Compression is one of the most important applications in our world in order to meet the needs of our digital age. Compression originates from the Latin word "compressare" and means "to press together". The goal is to take some form of data and represent it with less total information than it originally had, after applying various transformation steps to it. This assumes that the data has some form of redundancy. Example types of data that are suitable for compression are visual data, such as images and videos. Here compression methods rely on the fact that there is a large redundancy present in the

data. Digitised images can be represented using a $H \times W$ matrix, where $H$ and $W$ stand for the height and width, respectively. Each element in that matrix is referred to as a pixel. Natural images have a strong tendency to have a lot of similar neighbouring pixels. Even stronger tendencies can be seen in videos, where subsequent frames usually contain related information. Pixels are represented as integers which themselves are represented by a number of bits. The amount of information each pixel requires is measured in *bits-per-pixel* (bpp). The standard for most applications is *8 bpp*. This means each pixel can take $2^8 = 256$ different values. This is the case when representing images with a single channel only. However, only gray-scale images can be expressed this way. Coloured images can be simulated by linear combinations of three primary colours [6], most commonly used **R**ed, **G**reen and **B**lue. This requires three channels, resulting in $3 * 8 = 24$ bits-per-pixel. The total amount of storage needed to store a coloured raw image can be calculated with

$$bits = H \times W \times 2^L \times C$$

where $L$ refers to the amount of bits used to represent a single pixel. In our case $L = 8$. Furthermore, $C$ stands for the number of channels needed, $C = 3$ for coloured images. Compression tries to reduce the total bits needed to store a piece of data, while maintaining a way of reversing this procedure. Thus we divide this process of compression into two essential steps, the "compressor" (aka. encoder) and the decompressor (aka. decoder). The following formula, which will be used extensively in this work, describes the compression ratio that a given method achieves.

$$c_r = \frac{bits_{prior}}{bits_{posterior}}$$

There are two distinct types of compressions: lossless and lossy compression. The advantage of lossless-compression is that no information is lost during the process of encoding and decoding data.

$$x = d(e(x))$$

However, this means that $c_r$ is limited and high compression rates can not be achieved. On the other hand, lossy-compression achieves much higher ratios, but the encoded data will have differences compared to the original data when decoded and will only approximate it.

$$x \approx d(e(x))$$

Although information is lost during this process, it is still widely used and popular in many application, especially in visual data, like images and video. This stems from the fact that the human visual system does not perceive certain manipulations to data. They can be easily detected by computer programs, but usually are imperceptible to humans. [7]

## 2.2 Classical Compression

In the following section we will discuss widely used standard compression algorithms for both image and video. We will focus on the following: **JPEG**, **PNG**, **VQ** and **AVC**.

1. **Joint Photographic Experts Group (JPEG):** The original JPEG algorithm was published in 1992 [8] and was a mix of many popular and well working methods at that time. It is a **lossy** compression method specifically designed for digital photography. It achieves good-looking reconstructions with compressions up to a $c_r = 15$. Compression is achieved by a number of consecutive steps. First the image, which typically lies in the RGB colour space, is converted into the YCbCr colour space. This separates the image into luminance (Y) and chrominance (Cb & Cr). The human visual system is more sensitive to changes in the luminance, while differences in the chrominance are harder to spot for us. Usually, an additional step of down-sampling the chrominance channels follows, due to the aforementioned reason. Next the luminance and chrominance parts go through a block-splitting phase, where they are split into $N \times N$ blocks ($N = 8$ typically). The next step is about separating the low-frequencies and high-frequencies in the channels. Low-frequencies contain most of the information (shapes, edges, corners, etc.), while high-frequencies contain finer details. This separation is achieved using a *Discrete Cosine Transform* (DCT), which is applied to each block. This results in DCT-transformed matrices of the same shape, which now represents the different frequencies, starting with low-frequencies in the top-left and becoming smaller towards the lower-right. Afterwards, the main step responsible for the compression is employed: quantization. The JPEG standard introduced different *quantization-matrices* which are used to divide the DCT-transformations of all 8x8 blocks. There are different *quantization matrices* for different $c_r$, however they all follow the same pattern of containing smaller values in the top-left and larger values in the bottom-right. After the division, the values are rounded to the nearest integer, where many of the high-frequencies

become zero. This rounding operation, is the only place in the entire algorithm where a lossy-compression happens (next to the optional chrominance down-sampling), as this step can not be reversed. Following this, the resulting rounded matrix is ordered in a zig-zag pattern, starting from the top-left towards the bottom-right. Essentially, this groups together many of the zeros that are now present in the high frequency details. Differential encoding [9] is used to efficiently compress the data, for example by representing 0,0,0,0,0,0 as 6x0. Finally, Huffman encoding is used to represent the data in a better way while additionally compressing it further. Decoding a JPEG compressed image happens by executing the aforementioned steps in the reverse and inverse order.

2. **Portable Network Graphics (PNG):** Published in 2004 [10], PNG was meant to be a free and open alternative to GIF [11]. It is a lossless compression format for RGB (24-bit) or RGBA (32-bit) colour spaces, and widely used for computer graphic elements. PNG being lossless, means that it's potential $c_r$ is smaller than that of JPEG for photographic pictures, however for images with a lot of uniform colours and plenty of redundancy in general, such as object-centric images with plain backgrounds, PNG can outperform JPEG [12]. Considerably better reconstructions can also be achieved on images with a lot of sharp edges and corners, such as text renders. The algorithm can be desribed by looking at its different intermediate steps: Filtering, LZSS & Huffman Coding. Filtering looks at each row of pixels individually and applies one of 5 filters to it, based on a metric to predict the lowest redundancy after using each filter. These filters can be either *None*, *Sub*, *Up*, *Avg* or *Paeth*. For example *Sub* replaces each element by the difference of the current byte-value of the pixel and its predecessor. Filtering is applied to transform the data to a format that can be better encoded by the subsequent steps. LZSS is a type of compression algorithm that works by back-referencing. This means that whenever elements in a data-stream appear that have been seen previously, a reference to this initial position is made. The third and last step is Huffman Coding which creates a tree of data, using fewer bits to represent elements that occur more often and vice versa. All of these steps are reversible, indicating that no information is lost. This makes PNG a good candidate for high-quality compression. Decoding a PNG file is done by simply reversing and inversing the encoding steps.

3. **Vector Quantization (VQ):** This type of quantization became very popular in

the field of deep learning [13], where it was used in an adapted form. However, this method found its origin in the classical area of compression algorithms [13]. The VQ method is a lossy-compression method which is not as popular as the aforementioned methods for image compression like JPG and PNG. However, it's functionality and idea will play an important role later in this work. The essential idea behind Vector Quantization is to replace blocks of pixels by codes from a learnt codebook. This algorithm requires two steps: training & inference. Initially, a block size of $m \times m$ pixels is chosen ($m \in \{2, 4, 8\}$ is common), which are flattened into a vector. A clustering algorithm is used to determine $k$ centroids. A centroid is a central point in a cluster of data-points. All centroids will be stored in a codebook $Q$ of length $k$. At inference time, an image is taken and each block of pixels is flattened and replaced by the closest centroid from the codebook given some metric like the *euclidean distance*. In order to store a representation of an image, only the sequence of indices of codes from the codebook is stored. This way, an image can be represented as $(H \div m) + (W \div m)$ integers. Decoding of an image can be achieved by looking up the centroid vectors in the codebook based on the indices and reshaping them back into $m \times m$ blocks. Storing compressed images additionally requires to store the codebook. While depending on many factors, a $c_r$ between 10 and 50 is typical. [13]

4. **Advanced Video Coding (AVC):** AVC [14] is a codec for video compression that was released in 2004. It has become the most popular used video codec with more than 90% of the video industry using it [15]. The algorithm compensates for both spatial and temporal redundancy. It is a lossy-compression method, with possible extensions to make it lossless. Typical lossy compression ratios are between $c_r = 50$ and $c_r = 100$ for 720p videos ($720 \times 1280$). The codec defines 3 different frame types:

   (a) **I**ntra-Coded-Frames: Frames that are stored entirely without temporal compression, only spatial compression. They can be decoded individually.

   (b) **P**redicted-Frames: Frames that are decoded using previous P-Frames and I-Frames.

   (c) **B**idirectional-Frames: Similar to P-Frames with the difference of using future frames as well for compression.

   Classifying which frames fall into which category depends on various factors. The simplest approach is using predefined patterns of the frame types. This is known

as Group-Of-Pictures (GOP), which for example determines that an I-Frame is followed by a specific number of P-Frames and B-Frames. This pattern is intrinsically influenced by the specified bitrate and quality targets of the desired compression. A smaller compression may allow for seeing I-Frames more frequently, thus improving the reconstruction quality. More sophisticated encoders [14] use advanced mechanisms to determine the best possible arrangement of the frame types by analysing the frames themselves. Next, block-splitting is applied to P-Frames and B-Frames and motion-estimation [16] is used to predict how the current frame changes from the previous (and subsequent in the case of B-Frames). The resulting motion-vector does not suffice and leads to undesired reconstructions. As a result, the difference (residual) between the original block and predicted block is stored as well. If the motion prediction performs well, the residual consists mostly of zeros which can be compressed efficiently. This procedure is similar to JPG compression as discussed in Section 1. A modified DCT is applied to the residual, separating low- & high-frequencies, which is followed by a quantization. This quantization can be more aggressive as there are typically only few high-frequencies. Motion-vectors and quantized residuals are encoded using entropy-coding, as also done in the JPG algorithm, followed by methods to reduce blockiness-artifacts. These steps are applied to the entire video. The final output is stored in container formats like MP4, MKV or AVI.

In summary, it can be concluded that the classical compression algorithms are very sophisticated and a lot of work has been going into them over the past decades. However, this also comes with the consequence of rising complexity and additional human bias, which will be discussed in the next section.

## 2.3 Neural Compression

Neural Compression is the field of using deep learning methodologies to compress data. The crucial key-idea in comparison to the classical methods discussed above, is to take out the "human in the loop" and reduce human-crafted solutions which may not be optimal, while additionally being biased by our intuition. The goal of neural compression is to provide the neural network with as much freedom as possible in making "choices" how to compress data. Similar outcomes can be observed in various fields of neural network engineering. A good example are the convolutional neural networks (CNN). At the beginning,

humans tried to craft convolutional filters manually or with engineered heuristics [17] [18], to achieve various kinds of image processing tasks such as image classification etc. When convolutional neural networks [19] were introduced, these filters were learned via gradient descent through backpropagation. And until today, CNNs are the chosen standard for such tasks, because the neural network can adjust the filters in a much more sophisticated way than humans could potentially understand, let alone come up with. This philosophy and its benefits of giving the neural model as much freedom as possible can also be seen in neural compression. Furthermore, there exists not only the possibility of achieving higher quality results, the approaches themselves could eventually be easier to understand as well. Of course, a distinction between methodologically understanding something and actually having an in-depth understanding of a method has to be made. While it is possible to learn the fundamental ideas behind neural networks, such as forward- & backpropagation, neural network architectures, optimizers and related parts that belong to the training process, it seems magnitudes more challenging, if not impossible to "read" a neural network and understand the parameters as a whole. Visualizations, theories and experiments can be made, but truly understanding a multi-million dimensional space is simply beyond our capabilities. On the other hand, this is not true for classical approaches where each step in the pipeline can be understood to its fullest. This also resembles the classical black- & white-box idea [20]. However, when only speaking about the methodology side, neural algorithms can be easier and faster to learn than classical algorithms. Many fundamental concepts appear in vastly different tasks and improvements can usually be made by simply "making things larger", like increasing the data-pool or the neural network size. On the classical side, improvements often come from either rethinking entire solutions or building on top of existing frameworks, increasing its complexity. The aforementioned AVC 4 algorithm consists of an extreme amount of little pieces that all contribute small improvements [14] to earlier algorithms.

Neural algorithms can potentially provide two things: higher-quality results compared to classical algorithms, while being easier to understand. Most neural compression methods follow the same concept of having a compressor and decompressor, however these are more often referred to as encoder and decoder. However, while the encoded data of the compressor in a classical setting can still be interpreted, in a neural case this usually is much harder and requires additional methods to get an overview. For this reason, this

space is called "latent space".

Moreover, the goal of neural compression is not always the same as in classical compression. While in the latter case, the goal always is to represent the given data in the smallest possible way without losing the essence of the information, in neural compression there is an increased interest in the properties of the emerging latent space. Specific properties will be discussed in the following sections. One key reason for this is the fact that those latent spaces are increasingly more often used to train models on separate tasks. Such tasks involve any kind of image recognition challenges [21] [22]. One motivation for this is the assumption that the latent space has richer features and can be learnt faster by a model than the original pixel space (in the case of images and videos). Furthermore, also generative models are often learnt in latent spaces. One crucial reason for that is the memory consumption. Generative models are usually parameter-heavy, reaching into the billions [23] [24] [25] [26], which naturally makes them expensive to train. Especially when these trainings are running at a high resolution. Training a model at a spatial resolution of $1024 \times 1024$ is quadratically more expensive than training it at $32 \times 32$. Thus, it is highly desirable to make use of a compressed latent space, where data can be decoded back to a high resolution space afterwards.

The following section will explain various popular neural compression algorithms that have been released and explored in the last years. Specifically, a detailed summary of **Autoencoder**, **Variational Autoencoder**, **Vector-Quantized Variational Autoencoder** and **Vector-Quantized Generative Adversarial Networks** will be given.

1. **Autoencoder (AE):** Autoencoders [27] present the fundamental framework most consecutive methods build upon. The idea is straightforward and draws from classical methods. The AE is a neural network which consists of an encoder and a decoder. The encoder projects the input to a latent space, thereby loosing information in the process of doing so. The compressed representation is fed to the decoder which projects it back into the original space. Encoding and decoding are achieved by using mathematical transformations common to neural networks such as convolutions, feed-forward mappings, activations, normalizations and poolings. These are generally referred to as layers and multiple layers form a block. Downsampling the inputs in the encoder is typically done by either using convolutions with a stride greater

than 1 or pooling layers (average-pooling [28], max-pooling [29]). On the other side, the decoder up-samples the low-resolution embeddings by either transposed convolutions [30] or standard interpolation. The neural network's weights are randomly initialized and learnt with gradient descent through backpropagation. The signal of error is usually a pixel wise metric between original data and the reconstructed data. Standard error signals are the L1-Error or the L2-Error. As mentioned before, these mappings are not easily understandable by the naked human eye.
This approach faces two major problems that make its application limited in the real world. The first issue is that the latent space does not embody any semantics from the original space. Two semantically and visually similar images can be mapped to very different places in the latent space. This does not result in a feature rich latent space. The second problem is that the latent space is not bounded. Encoded images could be mapped to very large ranges. This can result in training instabilities and make it impossible for subsequent models relying on that latent space to learn anything. As a result, a successor of the standard autoencoder approach was proposed, called *Variational Autoencoder*.
**Compression Ratio**: Depending on the number of downsampling layers $f$ and precision $p$ of the latent space, the total compression can be calculated as:

$$c_r = \frac{H \times W \times C \times bpp}{h \times w \times c \times p}$$

where $h = H/f$, $w = W/f$ and $c$ is the number of hidden dimensions. In a typical setup with $bpp = 8$, $p = 32$, $f = 8$, $c = 16$, $W = H = 512$, $C = 3$ the total compression amounts to $c_r = 3$.

2. **Variational Autoencoder (VAE):** The VAE [31] approach extends the classical autoencoder framework. The main difference is about regularizing the latent space to resolve the aforementioned problems of the AE. The encoder's output is changed from predicting the latent vectors directly, to predicting the mean and covariance matrix of a normal distribution. The decoder samples latent vectors from this distribution and subsequently decodes it. The loss is adapted from a simple element-wise difference to also include a regularisation term, that aids in aligning the latent space. This regularisation is expressed as the Kullback-Leibler divergence (KL divergence) between the predicted parameters of the encoders distribution and

a standard-normal distribution.

$$L_{\text{VAE}}(x, \hat{x}, \mu, \sigma) = MSE(x, \hat{x}) + KL(\mathcal{N}(\mu, \sigma)||\mathcal{N}(0, 1))$$

The Kullback-Leibler divergence is a measurement of how unlike two distributions are. The higher the value, the less similar. Minimizing $L(x, \hat{x}, \mu, \sigma)$ ensures that reconstructions are as close as possible and latent vectors are centered around the origin. This way the model can not use the latent space arbitrarily and needs to organise it better, letting the previously missing properties emerge. Both additions are crucial and need to be present. Only introducing the encoder distribution, without the KL divergence, could lead to the same behavior as discussed above, where the predictions of the mean $\mu$ could fall anywhere and the covariance $\sigma$ could shrink, resulting in point-like predictions [32]. Note that the VAE itself can function as a generative model to some extend. After training, it is possible to sample new latent vectors from the standard-normal distribution and decode them back to the original space.

$$\tilde{x} = d(z), \text{where } z \sim \mathcal{N}(0, 1)$$

**Compression Ratio**: The calculation for the compression ratio is the same as for the standard autoencoder.

3. **Vector-Quantized Variational Autoencoder (VQ-VAE):** The VQ-VAE [33] adds vector-quantization to the framework of the AE. Unlike in the classical approaches where the vector-quantization is pre-defined, like the quantization tables in the JPG compression 1, the quantization is learnt during the training as well. This is called the *codebook* $Q$ and contains $K$ vectors, also called *codes* $q$. The encoder and decoder follow the same structure as in the AE. After encoding the input data $z = e(x)$, the resulting latent vectors will be replaced by their nearest counterparts from the codebook.

$$z_q = q(z) = (\arg\min_{q \in Q} ||z_{ij} - q||^2) \in \mathbb{R}^{h \times w \times c}$$

This will be given to the decoder $\hat{x} = d(z_q)$ which projects the compressed & quantized latents back into the original space. Note that although this method contains "VAE" in its name, the original paper [33] neither let the encoder predict a distribution, nor regularise the training with the KL divergence. However, technically

it can still be interpreted as variational, since the latent space is well defined and it is possible to sample from it. While in VAE the latent space approximates a standard-normal distribution, in a VQ-VAE the latent space approximates a discrete distribution provided by the codebook. This is enforced by an added loss term to the standard MSE.

$$L_{\text{VQ-VAE}}(x, \hat{x}, z, z_q) = MSE(x, \hat{x}) + ||sg[z] - z_q||^2 + ||z - sq[z_q]||^2$$

where $sg[\cdot]$ denotes the stop-gradient operator. This ensures that the codebook vectors $q$ will be updated towards the encoder output $z$ and vice versa.
**Compression Ratio**: Given $f$ and the precision of the codebook codes $p_d$, the total compression can be calculated as:

$$c_r = \frac{H \times W \times C \times bpp}{h \times w \times p_d}$$

In a typical setup with $bpp = 8$, $f = 8$, $p_d = 16$, $W = H = 512$, $C = 3$ the total compression amounts to $c_r = 96$. However, note that the codebook needs to be stored as well, which adds $N \times c \times p$ bits to the required storage space. This becomes less noticeable when storing large amounts of data.

4. **Vector-Quantized Generative Adversarial Network (VQ-GAN):** The VQ-GAN [34] method builds upon the VQ-VAE framework and adds a GAN-Loss to the objective. A GAN, **G**enerative **A**dversarial **N**etwork, is a type of neural network in which two models compete against each other: the discriminator and the generator. The rudimentary idea is to have the generator create data, which the discriminator has to detect given a real example and the generated example. If set up well, this leads to both models becoming better at their target task. That is, the generator creates better samples and the discriminator can distinguish real vs. fake better. In the case of the VQ-GAN, the VQ-VAE is treated as the generator. A discriminator is trained to detect the reconstructions from the original images. The belief is that the adversarial loss leads to more visually correct features, because the discriminator can become a better metric than just the standard element-wise MSE and therefore give better signals to the generator on how to adjust. The GAN-Loss can be written as the following:

$$L_{\text{GAN}}(x, \hat{x}) = \log D(x) + \log D(1 - \hat{x})$$

where $D$ stands for the discriminator. This loss is combined with the VQ-VAE loss.

$$L_{\text{VQ-GAN}}(x, \hat{x}, z, z_q) = L_{\text{VQ-VAE}}(x, \hat{x}, z, z_q) + \lambda L_{\text{GAN}}(x, \hat{x})$$

In this loss function $\lambda$ is a weighting factor for how much impact the GAN loss has. Initially, this is set to $\lambda = 0$ in order to give the generator some time to learn how to reconstruct data, otherwise the discriminator would easily dominate the training and there would be no meaningful signals coming to the generator. Finding the right hyperparameter choices is a key-factor in training GANs, which makes training them a lot harder than other methods. However the potential benefit to be gained in the outcomes can be worth to use them.

**Compression Ratio**: The calculation for the compression ratio is the same as for the VQ-VAE.

To summarize, introducing inductive biases into simple models can lead to substantial performance gains in the outcomes. Guiding an autoencoder to have its latent space centered around the origin, forces it to have a better organisation of it, leading to the framework of the VAE. The specified methods are widely used in different fields of applications. In the following sections we will see how these approaches can be extended to yield even higher compression ratios.

## 2.4 Diffusion Models

Diffusion Models [35] [2] [3] [4] [5] have become the default framework for generative tasks in recent years in many fields of applications. Their biggest competitor and predecessor is the Generative Adversarial Network. However, due to many occurring challenges when working GANs, there is an increased tendency towards employing diffusion models over a GAN. Fundamentally, diffusion models learn a mapping between two distributions. In typical setups, one distribution is unknown $p_{data}$ (data distribution) and the other is properly defined $p_{noise}$ (noise distribution). The primary goal is to learn a mapping between $p_{noise}$ and $p_{data}$. GANs work similarly in theory, however have significant differences. The mapping of the two distributions is defined by two processes: the forward process & the backward process. The forward process can best be understood as a continuous interpolation between $p_{data}$ and $p_{noise}$.

$$x_t = a \cdot x_0 + b \cdot \epsilon \tag{1}$$

Here $x_0 \sim p_{data}$, $\epsilon \sim p_{noise}$ and $a$ & $b$ are defined by a *noise-schedule* and depend on the timestep $t \in [0, 1]$. $x_t$ lies on the trajectory between $x_0$ and a specific $\epsilon$. Diffusion models are trained to remove the disturbance introduced in $x_t$ from the known distribution and project it back to it's origin. The forward process and backward process are commonly also referred to as "noising" and "denoising", respectively. A diffusion model can be used to generate new samples from the unknown distribution $p_{data}$ by iteratively denoising samples from the known distribution $p_{noise}$. By the nature of their training, diffusion models work best when used iteratively. The exact formulation for the reverse process is defined by a sampler that defines how to move from $x_t$ to $x_{t-\Delta}$, where $\Delta$ is the step size. The number of iterations can be adjusted freely, however usually higher step counts lead to better results (given some upper limit [3]). This general definition leaves open many specifications, which differ amongst frameworks. Therefore, the following uses a unifying framework over which all common frameworks can be defined through, called GDF (**G**eneral **D**iffusion **F**ramework) [36]. We can define a diffusion model training with the following five components:

1. **Schedule:** The schedule $s(t)$ defines the log-Signal-to-Noise-Ratio (logSNR) at each timestep. The timestep is a continuous value between $t \in [0, 1]$. Note that when $t = 0$, $x_t = x_0$. Likewise when $t = 1$, $x_t \sim p_{noise}$. The noise schedule is a monotonically decreasing function.

2. **Input Scaling:** The input-scaler defines $a$ and $b$ given the logSNR returned by the schedule. The most common input-scaler is called variance preserving [35] [2]. Variance preserving ensures that $x_t = a \cdot x_0 + b \cdot \epsilon$ keeps a variance of $\sigma^2 = 1$, assuming both $x_0$ and $\epsilon$ have a variance of $\sigma^2 = 1$ as well. This plays a vital role for model training, because neural networks work better with normalized inputs, and simplifies the maths. Another popular variant is variance-exploding [5].

3. **Target:** The target $x_{\text{target}}$ stands for the prediction type of the model and is the main error signal in diffusion models. It is used in the loss function and compared to the model prediction

$$L_\theta = d(x_{\text{target}}, f_\theta(x_t, t))$$

where $d(\cdot, \cdot)$ is a distance function, typically the MSE. There are three common prediction types used in diffusion models: $\mathbf{x_0}$**-prediction**, $\epsilon$**-prediction** & **v-prediction** [2] [37]. For $\mathbf{x_0}$**-prediction** the model is optimized to predict the noise free input

$\hat{x_0} = f_\theta(x_t, t)$. On the other hand, **$\epsilon$-prediction** teaches the model to predict the noise in the data $\hat{\epsilon} = f_\theta(x_t, t)$. Note an approximation for $x_0$ can be calculated by:

$$\hat{x_0} = \frac{x_t - b \cdot \hat{\epsilon}}{a}$$

Both types of predictions have extreme conditions at both ends of $t$. While for a **$\mathbf{x_0}$-prediction** model, $t \approx 0$ is easy to learn, the task becomes difficult as $t$ approaches 1. An **$\epsilon$-prediction** faces the same extreme behaviour just the other way around. As a result, the **v-prediction** was proposed by Salimans *et al.*[37] and defined as:

$$v = a \cdot \epsilon - b \cdot x_0$$

This acts as a challenging in-between of **$\mathbf{x_0}$-prediction** and **$\epsilon$-prediction**, because as the noise in $x_t$ increases the model needs to predict $x_0$, while for smaller values of $t$ the model needs to predict $\epsilon$.

4. **Noise Condition:** It is helpful [38] [2] for the model to have knowledge about the current timestep $t$, which can inform the model how much noise is present in $x_t$. Common choices are to condition the model on the timestep $t$, which ranges from 0 to 1, as well as the variance $\sigma^2$ of the noise [5].

5. **Loss Weighting:** The loss weighting $\lambda(\text{logSNR})$ defines an importance weighting for all timesteps and is multiplied onto the loss $\lambda(\text{logSNR}) \cdot L_\theta$. Different theorems [5] indicate that for optimal training of diffusion models, a model should focus more on specific ranges of timesteps. For example one popular loss weighting is P2-Weighting [39] which is defined as:

$$\lambda_{P2} = (k + e^{\text{logSNR}})^{-\gamma}$$

where typically $k = 1.0$ & $\gamma = 1.0$. The P2-Weighting increases importance of higher noise levels. This weighting makes sense when training a model with **$\epsilon$-prediction** objective.

These five criteria well-define a diffusion model training and all popular frameworks can be generalized to it. Note, the main unifying variable is the logSNR. It serves as the center piece of information over which the schedule, input scaling, noise condition and loss weighting are defined and the bridge to translate one category to another.
In the following this work will outline to view diffusion models through the lens of differential equations and afterwards examine two popular frameworks. Specifically, the key

ideas will be discussed and a comparison will be made by looking at the five different categories. Afterwards, a short introduction to sampling methods for diffusion models is given, which are usually independent of the framework that is used and simply describe the process of moving from the known distribution towards the unknown distribution via iterative refinement.

### 2.4.1 Viewing Diffusion Models through Differential Equations

Diffusion models can be viewed as approximating a data distribution $p(x)$. Due to difficulties in a direct approximation [40], one can rephrase the problem into approximating the *score-function* of the distribution $p(x)$ as:

$$\nabla_x \log p(x) \tag{2}$$

A *score-based model* $f_\theta(x)$ is used to approximate the score-function via a neural network. One can use Langevin dynamics [41] to sample from the data distribution iteratively. The starting point is randomly initialized from a prior distribution and optimized towards higher data density. However, the sampling taps into a pitfall when positioned in low data densities. Those areas were not seen during training and thus gradient approximations are inaccurate and lead to bad predictions. To overcome this problem, one can extend the training process and include noise perturbations to the training data via marginal distributions $p_t(x)$. High noise perturbations can cover low data density areas and thus solve the aforementioned problem. $p_t$ can be modeled in the form of $a \cdot x + b \cdot \epsilon$ as mentioned above, where $x \sim p_0(x) = p(x)$. Given that, the noise-conditional score-function can be approximated by a noise-conditional score-based model.

$$\nabla_x \log p_t(x) \approx f_\theta(x_t, t) \tag{3}$$

Furthermore, the diffusion process can be modeled as a stochastic differential equation (SDE) of the general form

$$dx = f(t) \cdot x dt + g(t) dw \tag{4}$$

where $w$ denotes the Brownian motion [42] and $dw$ can be viewed as infinitesimal white noise [43], $f(t)$ is called the drift coefficient and $g(t)$ the diffusion coefficient. When moving forward in time, $dx$ points towards the prior distribution, while when moving backward in time ($dt$ is negative), $dx$ points towards the data distribution. Every SDE possesses a

reverse SDE as found by Anderson [44]. The closed-form solution is given by

$$dx = [f(t) \cdot x - g^2(t)\nabla_x \log p_t(x)]dt + g(t)dw \tag{5}$$

It can be seen that the reverse process involves the score function of $p_t(x)$. As discussed before, we can approximate this by $f_\theta(x_t, t)$. After training, we can use $f_\theta(x_t, t)$ as a proxy for $\nabla \log p_t(x)$.

$$dx = [f(t) \cdot x - g^2(t) f_\theta(x_t, t)]dt + g(t)dw \tag{6}$$

This reverse SDE can now be solved with numerical SDE solvers to generate new data samples. Those will be discussed later. Moreover, one can also solve a probability-flow ODE [5] that takes the form of

$$dx = [f(t) \cdot x - \frac{1}{2} g^2(t) f_\theta(x_t, t)]dt \tag{7}$$

which is equivalent to 6 without the stochastic noise component. One of the benefits of this is that this allows for exact log-likelihood calculations. Moreover, often times numerical ODE solvers, can sample in fewer steps than their SDE counterparts [45].

### 2.4.2 DDPM / DDIM

Denoising Diffusion Probabilistic Models (DDPM) [2] & Denoising Diffusion Implicit Models (DDIM) [3] are widely used frameworks for working with diffusion models, which define the same training process and only differ in the sampling mechanism. The forward process is defined as the following:

$$q(x_t|x_{t-\Delta}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-\Delta}, \beta_t I) \tag{8}$$

Interestingly, a one step forward which mimics 1 can be derived as:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_0, (1-\alpha_t)I) \tag{9}$$

Moreover, the corresponding SDE for the DDPM formulation takes the following form:

$$dx = -\frac{1}{2}\beta(t)x(t)dt + \sqrt{\beta(t)}dw$$

This can be derived from equation 8 by making $\beta_t$ be dependent on $N$ ($N$ stands for the number of timesteps), letting $N \to \infty$, taking the Taylor expansion, dropping the higher

order terms and rewriting the equation.
In equation 9, $\alpha_t$ stands in direct correlation with with the logSNR.

$$\alpha_t = \text{sigmoid}(\text{logSNR}) \tag{10}$$

Therefore, the noise schedule can be defined in logSNR space, while the actual forward process uses the variance $\alpha_t$. The most popular noise schedule is the cosine schedule introduced in iDDPM [38]. It can be approximated by

$$\alpha_t = \cos\left(t \cdot \pi \cdot 0.5\right) \tag{11}$$

and resembles a slowly changing schedule at both ends of $t = 0$ and $t = 1$, while quickly decreasing in the middle part. Note that $q(x_t|x_0)$ is variance preserving by it's definition. DDPM and DDIM can be used with any of the aforementioned targets. Most commonly chosen is the $\epsilon$-prediction, as it usually gives better results [2], however only marginally, than other objectives. Moreover, this gap of performance can be closed by using different loss weightings as shown by Hang *et al.*[46]. The noise condition used in this framework is a Fourier timestep embedding [47], that uses a sinusoidal embedding of $t$ to create an easy-to-understand relative information of the timestep. There are many different loss weightings in existence that are commonly used with this framework, such as *SNR-*, *P2-*, -, *TruncatedSNR-Loss-Weight.*

### 2.4.3 EDM

The paper "Elucidating the Design Space of Diffusion Models" (EDM) [5] introduces a framework in which the forward process is defined as

$$q(x_\sigma|x_0) = \mathcal{N}(x_\sigma; x_0, \sigma^2 I) \tag{12}$$

which yields a sample of variance $\alpha_\sigma = 1 + \sigma^2$. For this reason an input scaling of $c_{in}(\sigma) = \frac{1}{\sqrt{1+\sigma^2}}$ is multiplied to $x_\sigma$, to ensure a variance preserving process over $\sigma$. The denoising model is reformulated to predict a dynamic residual over $\sigma$.

$$f_\theta(x_\sigma, \sigma) = c_{skip}(\sigma) \cdot x_\sigma + c_{out}(\sigma) \cdot F_\theta(x_\sigma, \sigma) \tag{13}$$

While $f_\theta(x_\sigma, \sigma)$ predicts x0, the actual neural network predicts a $\sigma$-dependent residual for x0. Moreover, it can be seen that, since everything is built around $\sigma$, the score-model

is conditioned on $\sigma$ as well. The function $f_\theta(x_\sigma, \sigma)$ approximates the score function of $p_{data}(x)$.

$$\nabla_x \log p_{data}(x) \approx \frac{(f_\theta(x_\sigma, \sigma) - x_\sigma)}{\sigma^2} \tag{14}$$

EDM approaches the problem from the PF-ODE perspective and reparameterizes it to be a function of $\sigma(t)$, instead of the drift coefficient $f(t)$ and diffusion coefficient $g(t)$.

$$dx = -\dot{\sigma}(t)\sigma(t)\nabla_x \log p(x; \sigma(t))\, dt.$$

The noise schedule is defined as $\sigma^2 = {s_{\max}}^{\frac{1}{p}} + t \cdot ({s_{\min}}^{\frac{1}{p}} - {s_{\max}}^{\frac{1}{p}})^p$. The logSNR can be calculated from $\sigma^2$ as $\text{logSNR} = \log \frac{1}{\sigma^2}$. Finally, the loss weighting used in EDM can be defined as $\frac{\sigma^2+1}{\sigma^2}$.

### 2.4.4 Sampling Methods

Samplers are crucial as they determine how to approximate the underlying trajectory and thus have a large impact on the quality of the sampling. Taking too large steps, using bad heuristics etc. will significantly contribute to make the outputs worse than they could potentially be. Any sampling method defines a discretized process to approximate the continuous differential equation (SDE or ODE). The most straightforward sampler is the Euler-sampler [48] which is defined as the following:

$$x_t = x_{t+1} + dt \cdot d$$

where $dt$ stands for the step size and $d = \frac{x - \hat{x}_0}{t}$ is the derivative of the trajectory. The Euler-sampler is a first order sampling method. There are numerous other and more sophisticated samplers [5] [2] [3] [49], often of higher order, that try to provide a better approximation. Since sampling methods are not a direct focus of this work, the reader is referred to the literature.

## 2.5 Evaluation Metrics

This section investigates different categories of evaluation metrics that can be used to assess the quality of reconstructions. Measuring the differences between separate methods is crucial, but also a non-trivial task as many objective measurements do not align with subjective human measurements and opinions [12]. As a result, over the years many distinct evaluation metrics have been proposed, in order to have a standard way of deciding

on the superiority of a method and to make decisions on standard algorithms. Furthermore, it is important to note that the process of evaluation should be executed with great care, as deviations in the setup or the test data can lead to misleading results (when compared to other studies), followed by wrong interpretations. Additionally, evaluation using metrics should always make sure that enough test data is available in order to cover a fair representation of the data manifold. Using only a small number of test data points can lead to wrong estimates. In the following, a closer look will be taken at a few of such metrics, ranging from simplistic and intuitive to more sophisticated ones. There are three types of evaluation metrics that are going to be discussed: analytical-, neural-, and human-evaluation.

### 2.5.1 Analytical Evaluation

1. **Mean Squared Error (MSE):** The most basic evaluation metric is simply comparing all individual pixels between two images. Usually the *L2-Loss* is used as a standard, in order to punish higher deviations from the original, but options such as *L1-Loss* are also common. When using the *L2-Loss*, one sums up all squared distances between all pixels in an image over each channel.

$$MSE = \frac{1}{C \times H \times W} \sum_{c}^{C} \sum_{y}^{H} \sum_{x}^{W} (\hat{x}_{x,y}^{(c)} - x_{x,y}^{(c)})^2$$

   Here $\hat{x}$ represents the reconstructed image and $x$ the original. It is worth mentioning, that a condition for a lossless-compression has to ensure an $MSE = 0$, while for a lossy-compression this condition is loosened to $MSE >= 0$.

2. **Signal-to-Noise Ratio (SNR):** Originally coming from telecommunications to quantify how well a signal was transmitted, it found wide adoptions in many different fields of applications. It can also be used to quantify how well data is reconstructed through measuring the artifacts in an image. The SNR can be calculated by taking the ratio of the mean of the signal (the original image) and the MSE between the original image and the reconstructed image. In the context of images, signal represents the original image and noise stands for the error in the reconstruction.

$$SNR = 10 \log_{10} \frac{\mu_x^2}{MSE} = \frac{1}{C \times H \times W} \sum_{c}^{C} \sum_{y}^{H} \sum_{x}^{W} 10 \log_{10} \frac{(x_{x,y}^{(c)})^2}{(\hat{x}_{x,y}^{(c)} - x_{x,y}^{(c)})^2}$$

This metric is useful because it takes the range of the data into account. Given two types of data that both have very different scales, for example between 0-10 for the first type and 0-256 for the second, even if the relative error is the same, the MSE will be higher for the second image because its range is greater. The SNR compensates that by taking into account the range of the data and acts as a normalization.

3. **Peak Signal-to-Noise Ratio (PSNR):** Instead of taking the ratio between the mean $\mu_x$ of the signal and the MSE, the PSNR takes the ratio between the largest possible signal, $2^8 - 1$ in the case of an 8-bit pixel, and the $MSE$. This puts more focus on the peak error of the reconstruction. It's formula can be written as the following [12]:

$$PSNR = 10 \log_{10} \frac{(2^L - 1)^2}{MSE}$$

Here $L$ stands for the general case of how many bits are used to represent a pixel. PSNR emphasizes the worst-case scenario and if it is high, it ensures with more certainty that the reconstructed image quality is high.

It is worth mentioning why the aforementioned metrics are modelled on a logarithmic scale. According to the *Weber–Fechner law* [50] the human perception of sound and sight is more logarithmic than linear. This means, visual changes and auditory differences do not seem linear to us and are better approximated by taking the log of a changing signal. For example, we perceive a sound as twice as loud not when the sound energy doubles, but when it increases by a factor that is logarithmic.

### 2.5.2 Neural Evaluation

Neural evaluation aims at improving the alignment between metric and human judgement through machine learning. There are many well-established metrics that make use of neural networks. The following examines two very popular methods for image-, and video-evaluation, which will also be extensively used later.

1. **Fréchet inception distance (FID):** The Fréchet Inception Distance (FID) [51] is a measure to assess the quality of generated images. It compares the distribution of generated images to that of real images using deep learning features extracted from an Inception-v3 [52] network. Specifically, FID calculates the Fréchet distance (also known as the Wasserstein-2 distance) between two multivariate Gaussians fitted to

feature representations of the generated and real images. Lower FID scores indicate smaller distances between the generated and real image distributions, and hence, better quality of the generated images. The formula for FID is given as:

$$FID = \|\mu_r - \mu_g\|^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}) \tag{15}$$

where $\mu_r$, $\Sigma_r$ and $\mu_g$, $\Sigma_g$ are the mean and covariance of the real and generated image feature distributions, respectively. FID is preferred over previous metrics like the Inception Score as it considers both the quality of generated images and the diversity of generated samples.

2. **Fréchet Video distance (FVD):** The Fréchet Video Distance (FVD) [53] is an extension of the FID for evaluating the quality of generated videos. It addresses the additional complexity present in videos, such as temporal coherence and motion dynamics. FVD uses a similar approach to FID by comparing the feature distributions of real and generated videos, but it incorporates temporal information by using a 3D Convolutional Neural Network (3D-CNN) [54] to extract features from video clips. This method captures both spatial and temporal aspects of the videos, allowing for a more comprehensive assessment of the quality of generated video content. The calculation of FVD involves fitting multivariate Gaussians to the feature representations of real and generated videos and then computing the Fréchet distance between these Gaussians. The formula in its general form is the same as the FID in equation 15. Like FID, a lower FVD score indicates a closer resemblance between the generated and real video distributions, signifying higher quality of the generated videos.

### 2.5.3 Human Evaluation

Obviously, the evaluation method that aligns best with human judgement, is to collect human judgement. This is a common approach done by many researchers and gives a solid foundation for comparing models. However, due to the subjective opinion of humans, it is necessary to have a representative group of people. For example, if the research team of a method would do the evaluation, there is a high chance that the outcome would be biased towards their own model. Therefore, it is best to choose a random sample of people. If additionally, the number of participating people is large, the evaluation results can be interpreted as faithful and accurate. However, human evaluation is an expensive and time-consuming process. Contrary to analytical-, and neural-evaluation, which is a

rather cheap and fast process. When questioning people, a significant amount of time is already devoted into finding participants, followed by explaining the process, execution and collecting the results. There exists a clear trade-off between quality and time, which needs to be chosen based on the given resources. Furthermore, some tasks are easier to evaluate by numerical metrics, like reconstructions, where even element-wise metrics can give insights into the performance of a model. However, for example evaluating generative tasks is naturally harder to do with simple metrics. This represents another factor to consider.

# 3 Method

Due to the rapid improvements and widened range of applications of diffusion models, it is natural to explore their capabilities in neural compression tasks. One obvious advantage is the iterative refinement process. Methods that exhibit only a single inference step (VAE etc.) might have a hard time decoding all potential available information at once and thus the decoding process does not reach its full potential. In theory, diffusion models could use different parts of the sampling process to focus on different frequencies, starting with decoding the low frequency information and later refining these by adding high frequencies. This main part of the underlying work will investigate how a *Diffusion Autoencoder* (DiffAE) can be formulated. Specifically, the idea will be described and the architecture will be explained. Moreover, a closer look at the diffusion process (both forward and reverse) will be taken, how DiffAEs perform, the benefits and disadvantages, results will be shown, followed by an extensive evaluation study.

## 3.1 Introduction

The diffusion autoencoder follows the same idea as a regular autoencoder. It consists of an encoder and a decoder. The encoder can be any kind of model and is only used to encode the data once. On the other hand, the decoder is the model that will be used iteratively during inference to reconstruct data from the latent space. The reason is that encoding data is much simpler than reconstructing it, thus giving more capacity to the harder task is logical. The encoder will encode the data to a small latent space $z = e(x)$. During training, data $x$ is noised and denoised by the decoder. Contrary, to other works for generative image models based on diffusion [55] [24], the main source for conditioning

the model, does not come from text. The encoded data features are used as the primary condition for the diffusion model. After training, aforementioned numerical SDE and ODE solvers can be used to approximate the reverse SDE. The condition is very information rich and the goal is to create a bijective mapping between condition and output of the sampling process, which is significantly different from other common types of diffusion models where outputs usually follow a one-to-many mapping (e.g. generative text-to-image).

## 3.2 Architecture

There are numerous ways of constructing the encoder and the decoder of the DiffAE. This work will define one possible and working architecture, suitable for multiple modalities. As mentioned above, encoding is a task much simpler than decoding. As a result, the capacity for the encoder can be kept small, while giving more parameters to the decoder. Therefore, an EfficientNet [56] [57] will be used as a main feature extractor for the entirety of this work. Specifically, an EfficientNet-2 in its small (S) variant. This model counts 21 million parameters in total. Note, that the EfficientNet is already a trained neural network for the task of image classification on ImageNet [58]. Thus it can be assumed that the encoded features are already meaningful and can speed up the learning process. Table 2 shows a comparison of using a pretrained EfficientNet and a randomly initialized one for the task of image compression. This work chooses to use the ultimate features, which result in a spatial compression of f32. For example a $3 \times 384 \times 384$ image will be encoded to $1280 \times 12 \times 12$. Afterwards, a projection layer is used to map the 1280 channels to 16.

The decoder specifies a UNet [59] architecture. A UNet itself is made up of an encoder, a bottleneck and a decoder and returns outputs that have the same spatial dimension as the input. This is different from other neural compression methods, in which the decoder takes in the low-resolution latent and returns a decompressed, higher resolution, output. In order to train diffusion models properly, both input and output must have the same dimensions. Otherwise the loss calculation could not work correctly. The encoded latents need to be injected into the model therefore. There are many possible ways to do so. Two primary methods are cross-attention [47] and concatenation. Pernias *et al.*[60] tried cross-attention but encountered complications when trying to decode different resolutions or aspect ratios than shown during training. This work exclusively uses concatenation as a conditioning mechanism, due to its simplicity, minimal extra parameters needed and

its ability to generalize better to unseen resolutions and aspect ratios. Moreover, another required choice is the place of injecting the condition. The models discussed in this work will solely concatenate the condition at the beginning of the decoder. This reduces computational overhead. Finding superior injection-mechanisms is left for future work. The UNet consists of a number of down-sampling and up-sampling blocks that are surrounded by residual convolutional blocks, attention blocks and timestep blocks.
The diffusion decoder contains magnitudes times more parameters than the encoder. This work will show decoders that are between 30x – 300x larger than their encoder counterpart. Neural network trainings become increasingly more memory demanding with higher spatial resolutions. Training large diffusion decoders will suffer these problems, especially when moving to large resolutions. One trick that can be used is to setup a prior neural compression model. For example, it is possible to use a VQ-GAN or VAE to employ a first lightweight compression. The diffusion autoencoder then works in the space of the VQ-GAN or VAE. As discussed earlier, common neural compression methods perform only well when used at small compression rates. Therefore, a spatial compression of f2 up to f4 can be used. This results in near perfect reconstructions, while quadratically shrinking the space that the decoder lives in.

## 3.3 Image Compression

This subsection will outline the training details for the image modality. It will emphasize the specific model setup, hyperparameters and data used for training. Additionally, results will be shown.

### 3.3.1 Architecture

For all ablation studies we follow a specific setup of model settings, which only vary in a few chosen categories to evaluate the impact of different hyperparameters. The diffusion autoencoder will make use of the aforementioned option to use a secondary encoder to reduce the dimensionality of the training space without sacrificing quality. Specifically, an f4 VQ-GAN (Section 4) will be used with a hidden dimension of $c = 4$. The UNet will consist of 4 downsampling layers and 4 upsampling layers with skip connections in between. It will contain approximately 280M parameters throughout all experiments. In order to make training more efficient, a patch size of $p = 2$ will be used. This is implemented through the lossless Pixel-Unshuffle and Pixel-Shuffle [61] operations, which

move spatial information into the channels axis. The UNet is asymmetrical, containing twice as many layers in the upsampler part of it. Primarily, three main blocks will be used: convolutional blocks (with skip connections), attention blocks and timestep blocks. The first two levels only contain convolution-, and timestep-blocks, whereas the lowest two levels also include attention blocks. The reason is due to the quadratic memory growth of attention, therefore only using them in the lowest levels is computationally more affordable. All convolutions have a kernel size of $k = 3$ and padding pad $= 1$. The hidden dimensions for the levels in descending order are [320, 576, 1152, 1152] and the amount of blocks per level are [2, 4, 14, 4]. The latter is to interpret such that the second lowest level will contain most layers, 14 times each block. As mentioned before, the encoded image features will be projected by a single head with two convolutions and concatenated prior to the first layer. A visual depiction of the architecture for image compression can be seen in Figure 1. Weight initialization is applied by using Xavier-initialization [62]. The described architecture results in a total compression of:

$$c_r = \frac{384 \times 384 \times 3 \times 8}{12 \times 12 \times 16 \times 16} = 96.0$$

### 3.3.2 Diffusion Setup

As mentioned before, we need to define five ingredients in order to characterize a diffusion training (Section 2.4): the schedule, the input scaling, the target, the noise condition and the loss weight. In this work, the schedule, the input scaling and the noise condition will be kept constant throughout all experiments as further ablations would go beyond the scope of this work. Only the the target and loss weighting will be used in ablation studies as they are more directly related with the problem at hand. Specifically, the schedule is fixed to the cosine-schedule (Equation 11), the input scaling is chosen to be variance-preserving and the noise condition is done by injecting the timestep $t$ into the model using the Fourier timestep embedding.
Furthermore, the decoding of images via the reverse diffusion process use the DDPM sampler exclusively. Evaluating higher-order samplers etc. will be left for future work.

### 3.3.3 Training Details

It is desired to keep all training hyperparameters constant throughout all experiments. In order to do so, the following section defines the default parameters used in all training

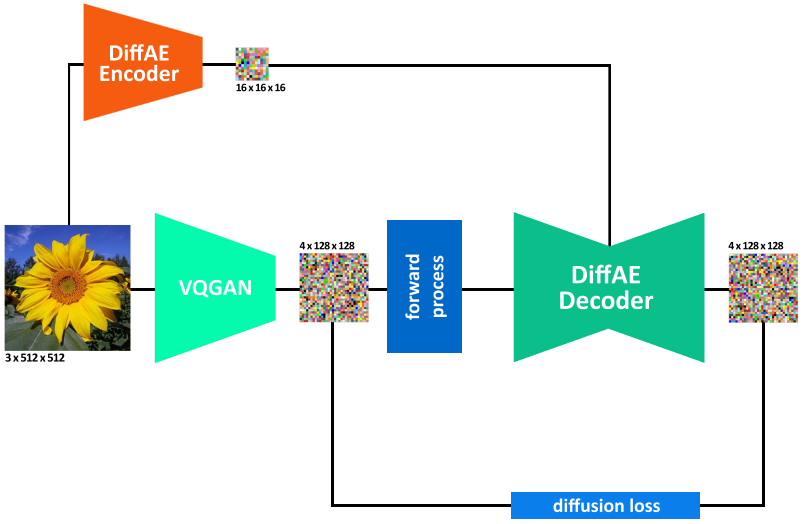


Figure 1: Visualization of the pipeline for the image diffusion autoencoder. Moreover, example shapes are shown for a $512 \times 512$ image. The training image is first projected into the VQGAN space, where the training occurs. Furthermore, the image is also encoded by the EfficientNet [56] and injected as a condition into the decoder part.

runs, if not specified otherwise. Each run is trained for 200.000 iterations. A base learning rate of 0.0001 is used with 10.000 warm-up steps. A batch size of 512 utilized. Training is done on $512 \times 512$ images. Moreover, it is common to use a copy of the base model that is updated with an exponential moving average (EMA) of the base model weights. This usually leads to a more stable model during inference. Each experiment uses 32 A100 40GB GPUs. The experiments were done on the Stability AI [63] cluster.

### 3.3.4 Data

Data is one of the most crucial points in machine learning and especially important in the training of generative diffusion models. A well working setup and model architecture with bad data will perform much worse than a bad architecture with good data. However, in the case of the unimodal compression problem talked about in this work, there are less requirements towards the data. For example, in text-to-image training, it is critical but also very difficult to obtain high-quality captions for the images, as usually default captions (e.g. scraped from the web) are bad [64]. Additionally, such training needs

aesthetic finetuning data too. On the other side, learning a compression does not need multimodal data and the visual data suffices. A large training corpus of many images is only needed. However, it is highly beneficial to have a mix of images from varying categories like photos, drawings, graphics etc. In the last years, a few large-scale datasets were released for that matter [65] [66], which can be used for this task. Specifically, the LAION-5B [66] dataset is used for this work, with aggressive filters (NSFW, watermark, aesthetic). It offers five billion images scraped from the clear-web, including a variety of different image types encountered in daily life. This provides a good starting point to train models from. The images contain captions, however initially these will not be used. Note, that during training only a small fraction of the available data is used, due to two major reasons. Firstly, filters are applied to the data that naturally shrink the dataset size. Specifically, the image-size is filtered to the appropriate training size, watermarked images and non-aesthetic images are filtered out as well. Moreover, a NSFW filter is applied to remove inappropriate images too. The second reason for the dataset shrinking comes from the fact that the total images used during training is far less than what the dataset contains. Roughly estimating, 200.000 iterations multiplied by a batch size of 512 amounts to 102.400.000 total images. This is approximately 2% of the dataset.

## 3.4 Video Compression

Video introduces another axis to the data dimensionality and naturally allows for compression along the temporal axis as well. When combined with the spatial compression, this can lead to larger overall data condensation. This subsection will discuss the possibility of compressing video data using the aforementioned diffusion autoencoders, which theoretically can also be extended to other sequential 2D data types.

### 3.4.1 Architecture

The architecture for the video models follows a similar style as the ones for image compression in Section 3.3.1. The model consists of a stack of 3D residual-blocks and attention-blocks, with standard timestep-blocks for injecting the current timestep. The 3D residual-block is identical to the 2D residual-block, except that it uses a 3D convolution. Furthermore, the base video architecture is not using attention layers at all, as it makes the training faster. This choice will be investigated later. In total, the video diffusion autoencoder contains 734M parameters. The architecture is a UNet as well and downsampling

and upsampling operations are performed by 3D convolutions and 3D transpose convolutions with a spatial stride of 2. Note that the stride is only applied spatially and there is no temporal stride in the UNet in this setup. This means, only the spatial size shrinks during the encoding and decoding, while the temporal is fixed. Exploring the usage of a temporal stride, is left to future work. The EfficientNet encoder is used for the spatial compression. However, a subsequent temporal compression is applied to the latent frames. Depending on the desired temporal compression, a different number of downsampling layers is employed. For example, for an $f_t = 4$ temporal compression, two downsampling layers will be used. A downsampling layer consists of a 3D convolution with a kernel size of 1, mimicking a simple linear projection, followed by an average pooling [19] applied on the time axis. One such layer transforms a $B \times C \times T \times H \times W$ tensor to one of shape $B \times C \times \frac{T}{2} \times H \times W$. Note that for image compression, the EfficientNet embeddings are concatenated to the input after interpolating it to the appropriate latent size. A hypothesis this bachelor thesis makes, is that a linear interpolation for the time axis is sub-optimal and would lead to undesired artifacts, because frames usually do not follow linear changes. However, the number of compressed latent frames (potentially) does not equal the number of (latent) frames that the diffusion autoencoder takes as input. Therefore, a learnt projection is used to upsample the compressed latent frames to the correct (latent) frame number. Specifically, a temporal "deflate" layer, which is simply the reverse of the temporal compression layer, is added for every temporal compression layer. For example, when using a temporal compression of $f_t = 4$, two compression and two deflate layers are used. Note that the bottleneck representing the smallest data size is after the last compression layer. The feature representation after this projection can be used for storage and further trainings. Furthermore, as also done in the Section 3.3.1 for image compression, a VQGAN 4 is used to employ a first-stage compression, in order to make the learning of the diffusion autoencoder more efficient and less memory intensive. However, this time the VQGAN enables a light temporal compression of $f_t = 2$ as well. All in all, depending on the temporal compression factor $f_t$, this architecture employs a total compression ratio of:

$$c_r = \frac{384 \times 576 \times 3 \times 8 \times T}{12 \times 18 \times 16 \times 16 \times t}$$

where $T$ is the number of frames and $t = T/f_t$. For instance, when $f_t$ equals 2, 4 or 8, the compression ratio amounts to 192.0, 384.0 and 768.0, respectively. Furthermore, Figure 2 shows an outline of the video diffusion autoencoder pipeline, showcasing the individual

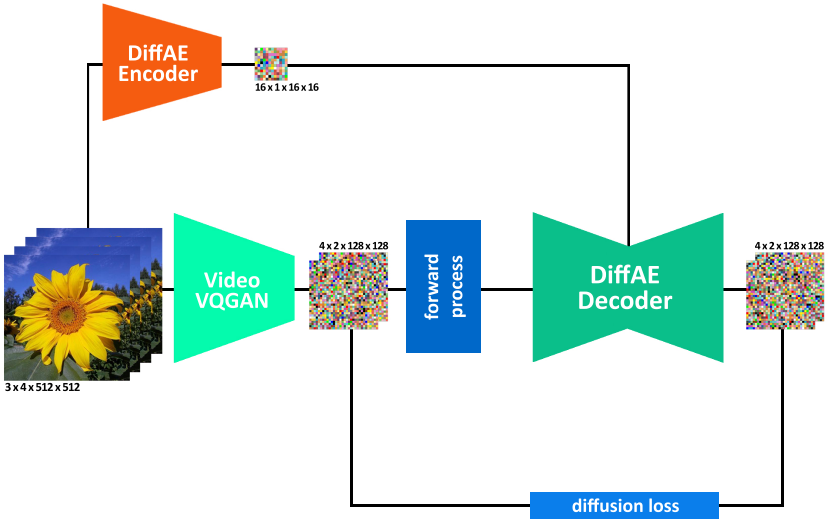


Figure 2: Visualization of the pipeline for the video diffusion autoencoder. Moreover, the example shows a video of resolution $512 \times 512$ with 4 frames. The video is first projected into the Video-VQGAN space, where the training occurs. Furthermore, the video is also encoded by the EfficientNet [56] and injected as a condition into the decoder part. Note, that the figure shows an example of an $f_t = 4$ temporal compression, as 4 pixel-space frames are encoded into a single latent frame.

steps during training.

### 3.4.2 Diffusion Setup

The diffusion setup equals the same as the image compression setup described in Section 3.3.2.

### 3.4.3 Training Details

Since many ablation studies are conducted on the image compression diffusion autoencoders, for video compression, this study chooses to exhibit specific experiments unique to video compression. Primarily, experiments for evaluating different temporal compressions are conducted. A constant learning rate schedule of 0.0001 is used with 10.000 warm-up steps. A total batch size of 128 is employed and a video resolution of $384 \times 576$ is used using 16 frames at 16 fps. Each experiment uses 64 A100 40GB GPUs. All models are

trained for 500.000 steps.

### 3.4.4 Data

Data for video training is as important as for working with images. Moreover, video has many different requirements that make high quality data. For example the fps of the videos. Low fps videos result in non-smooth transitions between frames. Furthermore, the motion happening in a video is also of high importance and influence for a model trained on it. For instance, if a dataset only contains low-motion changes in the videos, the resulting model will not be able to handle fast-motion videos. This would become a problem especially in video compression where the model would only be good at compressing low-motion videos and fail otherwise. Therefore, a mix of a broad range of videos is necessary to achieve good results. The dataset used in this work is a mix of 3 publicly available datasets: WebVid-10M [67], ACAV100M [68] & HDVila-100M [69]. WebVid-10M is a collection of 10 million watermarked videos from shutterstock. ACAV100M and HDVila-100M both contain 100 million ten-second YouTube clips. Except for size and fps filters, no additional filters are used. Note that the total dataset size of a video dataset is largely increased by the fact that there are numerous ways to cut a 10 second clip, contributing to the prevention of overfitting.

# 4 Evaluation

This section will conduct experiments on the aforementioned modalities and discussed settings, such as architecture, diffusion setup, training details and data. Careful comparisons will be made between different hyperparameters of diffusion autoencoders, as well as to other methods.

## 4.1 Image Compression

Experiments concerning image compression will investigate three hyperparameter evaluations (Section 4.1.1), namely the diffusion target, the diffusion loss weight and experiments on the choice of the encoder for the DiffAE. Furthermore, one comparison to a method from classical-, and neural-compression will be made in Section 4.1.2. Finally, further studies on the scaling behavior of diffusion autoencoders will be shown in Section 4.1.3.

### 4.1.1 Hyperparameter-Evaluation

Before conducting evaluations comprising other types of models, it is important to evaluate the possibilities of the diffusion autoencoder itself and find well performing setups and hyperparameters. Thus, this section will explore what works best for the model. Firstly, one captivating ablation is the target of the diffusion autoencoder. The target was mentioned in Section 3. Possible choices for the target are **x0-prediction**, **$\epsilon$-prediction** and **v-prediction**. Table 1 shows a comparison of a default training, as described in 3.3.3, with the only changing variable being the target. Furthermore, the other four parts of the diffusion training are fixed to the cosine schedule as the noise schedule, a variance preserving input scaling, a Fourier timestep condition and a constant loss weighting. The metrics used for evaluation are chosen to be part of the analytical and neural metrics as discussed in Section 2.5. Specifically, the mean-squared-error, peak-signal-to-noise-ratio and Fréchet inception distance. They are chosen as they comprise a diverse set of different kinds of metrics and what each metric focuses on. All evaluations of the DiffAE will be made with the same sampling-parameters if not mentioned otherwise. $N = 30$ inference steps will be used to reconstruct images, a classifier-free-guidance [70] value of $w = 3$. The training noise schedule is used for sampling as well. Lastly the DDPM [2] sampler is used for inference. This first experiment indicates that the $\epsilon$-prediction performs best in terms of FID. Moreover, the x0-prediction performs second best, while the v-objective ranks last. This is in line with previous research [2] that concludes $\epsilon$-objective to be superior to others. However, calculating the MSE and PSNR metrics results in a completely different outcome as Table 1 highlights too. v-prediction suddenly becomes the best and achieves the lowest MSE and highest PSNR, followed by x0-prediction and lastly $\epsilon$-prediction. This comes to a surprise since manually looking at reconstructions shows v-objective generates obvious and dominant artifacts. This is highlighted in Figure 8-11. It can only be concluded that the MSE and PSNR are very unaligned with human perception and focus on different parts. Moreover, prior work also highlights the importance of the *loss weighting* used with the corresponding target. Therefore, the next experiment investigates the combination of each target with an empirically well known loss weighting. Precisely, $\epsilon$-prediction will be combined with a P2 loss weighting which is defined as:

$$\lambda(t) = (k + e^{\text{logSNR}})^{-\gamma}$$

Moreover, a min-SNR loss weight [46] is chosen for a model trained with x0-prediction which is defined as the following:

$$\lambda(t) = min(e^{\text{logSNR}}, 5)$$

Finally, a sech loss weight [71] is used for a v-prediction model:

$$\lambda(t) = \frac{1}{cosh(e^{\text{logSNR / d}})}$$

In summary, the P2 loss weight amplifies the importance of high noise ranges, while the min-SNR increases the weight on low noise timesteps and the sech loss weight puts focus on the middle part of the noising range. The loss weight represents the only variable change in this experiment. Evaluating the three experiments results in a different picture than before (Table 1). The x0-prediction training with the min-SNR loss weight achieves the lowest FID score from all experiments, surpassing even the default $\epsilon$-prediction training. Moreover, the $\epsilon$-prediction with P2 loss weight performs worse than the training with constant loss weight in terms of FID. Lastly, the v-prediction model with sech loss weight only marginally improves the FID score of the default model by a factor of 0.3. However, calculating MSE and PSNR on the loss weight trainings, stays in line with the default experiments. v-prediction with sech loss weighting achieves the lowest mean-squared-error with MSE = 0.1804, while x0-prediction with min-SNR yields MSE = 0.1939 and $\epsilon$-prediction results in MSE = 0.2182. Furthermore, Figure 6 shows the loss distribution over the different timesteps during training for each experiment. This visualizes the extreme sides of each target. It can be seen that x0-prediction has the highest loss at t=1.0, referring to fully noised images. On the other hand, $\epsilon$-prediction performs worst at t=0.0, because it must predict the noise in an input that barely has any. Furthermore, it can be seen how the loss weight affects the training. For example, for x0-prediction, the loss is higher at timesteps close to $t = 1.0$ when using a min-SNR loss weight, while it is smaller in timestep ranges closer to $t = 0.0$, when compared to x0-prediction with constant loss weight. This work hypothesizes that x0-prediction combined with min-SNR loss performs best (w.r.t. FID) because of the following: min-SNR loss weighting increases the weight on smaller noise levels, meaning that the model puts more emphasis on learning the low noise timesteps, while the larger timesteps receive less focus. As these are naturally harder and considering the fact that sampling will occur iteratively, it might be beneficial to let larger timesteps generate only the lower frequencies and let smaller timesteps add the

higher frequencies. However, in a training with constant loss weight, the model might pay too much attention to larger timesteps, resulting in uncorrectable mistakes for the later sampling steps.
Moreover, two different kinds of experiments are conducted regarding the EfficientNet image-encoder. As mentioned before, the **s**mall version of the EfficientNet-2 is used containing 20M parameters. It is an interesting investigation to consider using more parameters for the encoder, aiming for achieving a better encoding representation. Therefore, a diffusion autoencoder is trained using the **l**arge variant of the Efficient-2 with 118M parameters. The diffusion setup follows the x0-prediction with min-SNR loss weight. The resulting metric evaluation can be seen in Table 2. The results show that there is no significant difference in using a larger EfficientNet. While the FID value is lower by a small fraction, the MSE & PSNR score is slightly worse. This could be considered to be pure randomness. The second experiment is using a pretrained EfficientNet vs. a randomly initialized one. All previous experiments have been using the pretrained weights from ImageNet training, because of the believe that these weights already result in meaningful and useful features for the task of image compression and hence serve as a good starting point.

Table 1: Diffusion autoencoder target evaluation results. The table shows results for Fréchet inception distance (FID), mean-squared-error (MSE), peak-signal-to-noise-ratio (PSNR).

| Model | FID | MSE | PSNR |
|---|---|---|---|
| x0-target | 13.9484 | 0.2101 | 6.7951 |
| $\epsilon$-target | **10.6094** | 0.2165 | 6.6640 |
| v-target | 15.9652 | **0.2034** | **6.9365** |
| x0-target (min-SNR) | **10.3849** | 0.1939 | 7.1434 |
| $\epsilon$-target (P2) | 12.0880 | 0.2182 | 6.6299 |
| v-target (Sech) | 15.6047 | **0.1805** | **7.4541** |

Table 2: Diffusion autoencoder evaluation between a pretrained EfficientNet encoder versus a randomly initialized one and training with the Efficient **s**mall versus **l**arge. The table shows results for Fréchet inception distance (FID), mean-squared-error (MSE), peak-signal-to-noise-ratio (PSNR).

| Model | FID | MSE | PSNR |
|---|---|---|---|
| x0-target (min-SNR) (default) | 10.3849 | 0.1939 | 7.1434 |
| x0-target (min-SNR) (Randomly init. EffNet) | 57.0914 | 0.2025 | 6.9480 |
| x0-target (min-SNR) (EfficientNet Large) | 10.3027 | 0.2039 | 6.9223 |

#### 4.1.2 Evaluation against other Methods

As shown in previous research [34], the VQGAN possesses state-of-the-art capabilities compared to all other methods discussed in the Section 2.3 for neural compression. Therefore, in the following, we compare a VQGAN model with a spatial compression of f32 to the explored DiffAE. Specifically, the f32 VQGAN is trained on the same data as the diffusion autoencoder and has seen approximately the same amount of images. One unfortunate difference is that the f32 VQGAN was trained prior to the aforementioned hyperparameter-experiments (Section 4.1.1) and differs in the parameter count and bottleneck dimension. The VQGAN contains 140M parameters and a bottleneck hidden dimension of $c = 36$, while the diffusion autoencoders consist of 280M parameters and $c = 16$ dimensions. Therefore, another diffusion autoencoder (156M parameters, $c = 36$) is retrained that closely mimics the same parameters as the VQGAN. Furthermore, it is important to keep in mind that the VQGAN reconstructs images in a single inference step. On the other side, the diffusion autoencoder can use a varying number of sampling steps, representing a trade-off between latency and quality. In the following we will compare the reconstructions of different amount of inference steps to that of the VQGAN. Moreover, this work chooses to continue with the x0-prediction that is trained with the min-SNR loss weight, because of its good performance. Figure 12-15 shows a visual comparison between the reconstructions of each method, also depicting different amounts of inference steps for the DiffAE. It can be seen that the diffusion autoencoder achieves better reconstructions than the VQGAN when using multiple inference steps. While $N = 1$ shows clearly that the VQGAN performs better, it is a satisfying result that compute can be traded for quality, which outperforms the standard one-step VQGAN method. Furthermore, analytical and

neural evaluation is conducted as well. Table 3 shows evaluations for the VQGAN with its single step inference $N = 1$, while for the diffusion autoencoder $N \in [1, 10, 30]$ is used. It can be seen that the f32 VQGAN achieves an FID score of FID = 22.2873, MSE = 0.1981 and PSNR = 7.0526. The 30-step inference of the diffusion autoencoder obtains scores of FID = 8.8549, MSE = 0.1948 and PSNR = 7.1213. This shows a clear advantage for the diffusion autoencoder in terms of the FID, however only marginal improvement regarding MSE and PSNR. The similar MSE and PSNR values might be due to the fact that blurriness, as often produced by the VQGAN, is on average as correct as detailed generations by the diffusion autoencoder, which if incorrect, result in larger errors. On the other end, when using the diffusion autoencoder for single step inference the results are a worse than those of the VQGAN, as can also be seen in the table. This might be due to the fact that, firstly the amount of times that the diffusion autoencoder sees full noise, $t = 1.0$, during training is much lower than the training steps of the VQGAN, which technically always performs an equivalent step and secondly the diffusion autoencoder learns various other tasks as well, denoising images at lower noise ranges, which requires learning and network capacity as well.

Table 3: Comparison between a VQGAN (f32) and an equally sized diffusion autoencoder with varying inference steps $N \in [1, 10, 30]$. The table shows results for Fréchet inception distance (FID), mean-squared-error (MSE), peak-signal-to-noise-ratio (PSNR).

| Model | FID | MSE | PSNR |
|---|---|---|---|
| VQGAN | 22.2873 | 0.1981 | 7.0526 |
| x0-target (min-SNR) (N=1) | 38.8347 | 0.2488 | 6.0537 |
| x0-target (min-SNR) (N=10) | 8.8687 | 0.1984 | 7.0446 |
| x0-target (min-SNR) (N=30) | **8.8549** | **0.1948** | **7.1213** |

Another important ablation study is a comparison between the considered neural compression method and a method from the classical compression field. As most experiments have been conducted on natural images, it makes sense to use the JPG compression algorithm (Section 1). The JPG algorithm contains a quality parameter (Q), that also determines the compression factor $c_r$. Table 4 shows results for FID, MSE and PSNR calculations on the same dataset as used in the other experiments. While higher numbers for Q lead to low FID values, they only give a small compression factor. For example, high


Figure 3: Visualization of different values for the quality parameter Q for the JPG algorithm and its visual effects on images. Also showing the compression ratio and a comparison to the diffusion autoencoder. Zoom in for better experience and noticing artifacts.

quality (Q=50) enables a compression factor of $c_r = 15$ and results in an FID value of FID = 2.5695. On the other hand, decreasing Q leads to much worse reconstruction quality. For instance, Q=10 gives a $c_r = 46$, but an FID value of 11.1238. This is already higher than what the best diffusion autoencoder (x0-prediction with loss weight) achieves (FID = 10.3849), which additionally has a larger compression factor of $c_r = 96$. Figure 3 shows a visual comparison between different quality factors (Q) and for reference the diffusion autoencoder's reconstruction too. Visual inspection gives notice about the emerging artifacts as compression increases and it can be seen that the diffusion autoencoder does not produce similar artifacts and reconstructions primarily lack details. Against expectations, MSE and PSNR values are all very similar, even though there are clear visual artifacts in the images. This is yet another example that MSE and PSNR differ a lot from human perception. However, this experiment shows visually and in terms of FID, that the diffusion autoencoder can achieve superior reconstructions at high compression rates. However, this comes at the great cost of latency. The JPG algorithm's execution happens in near real-time, while the diffusion autoencoder takes numerous times longer.

Table 4: Evaluation results for the classical compression algorithm JPG. Experiments used a varying quality factor when storing JPG images, resulting in different compression ratios. The table shows results for Fréchet inception distance (FID), mean-squared-error (MSE), peak-signal-to-noise-ratio (PSNR). Evaluated at a fixed $512 \times 512$ image resolution.

| Model | $c_r$ | FID | MSE | PSNR |
|---|---|---|---|---|
| JPG (Q=50) | 15 | 2.5695 | 0.2141 | 6.7152 |
| JPG (Q=25) | 23 | 4.3632 | 0.2159 | 6.6761 |
| JPG (Q=10) | 46 | 11.1238 | 0.2137 | 6.7219 |
| JPG (Q=5) | 93 | 24.5988 | 0.2091 | 6.8156 |
| JPG (Q=1) | 144 | 51.9815 | 0.2163 | 6.6673 |

### 4.1.3 Further Studies

This section conducts further studies into the method of diffusion autoencoders. Specifically, an always-intriguing experiment is scaling laws of models. However, as trainings require a lot of time and compute, perfect ablations for scaling laws are not feasible for this work. However, two interesting experiments were done that give some insight into the behaviour when scaling up this model with the specific architecture and training paradigm. One model has a total of 700M parameters, while the second model has a total of 3 billion parameters. The training config is slightly different than the best one found in the previous section. This is because the training was started earlier than the results of the ablation were obtained. However, the only difference is that the training uses $\epsilon$-prediction with P2 loss weighting instead of x0-prediction with min-SNR loss weight. Moreover, the training was started at $512 \times 512$, but after an initial pretraining phase, moved to $1024 \times 1024$. Furthermore, both larger models were trained for more than 1M updates, while all ablation experiments in this work only trained for 200k steps. Another difference is a varying compression factor instead of a fixed one. During training, images were additionally and randomly scaled down by a factor of 2, resulting in a possible spatial compression ranging between f32 and f64. Another difference is that a second conditioning was introduced along with the EfficientNet embeddings, namely text condition. Because the aforementioned dataset includes text anyways, it is at no cost to condition the model on it as well. This change is followed by changing the percentage the EfficientNet embeddings are shown during training, decreasing it from 95% to just merely 30%. The reason for this

choice, is to teach the model to be able to add potentially lost details through the compression. 70% of the time the model will only see text condition, which brings it closer to the framework of the standard latent diffusion model [55]. Figure 7 shows a comparison between the $\epsilon$-prediction model for all three different models with 280M, 700M and 3B parameters, respectively. Furthermore, Table 5 shows a numerical comparison between the three models.

Table 5: Numerical Evaluation between diffusion autoencoders of various sizes, with one containing 280M parameters, one 700M and one with 3B. The table shows results for Fréchet inception distance (FID), mean-squared-error (MSE), peak-signal-to-noise-ratio (PSNR). Note, metrics for the 700M and 3B model were evaluated at $1024 \times 1024$ resolution, while the 280M parameter model was evaluated at $512 \times 512$.

| Model | FID | MSE | PSNR |
|---|---|---|---|
| $\epsilon$-prediction (P2) (280M) | 12.0880 | 0.2182 | 6.6299 |
| $\epsilon$-prediction (P2) (700M) | 1.5236 | 0.2180 | 6.6357 |
| $\epsilon$-prediction (P2) (3B) | 1.2842 | 0.2182 | 6.6318 |

It can be seen that large improvements can be obtained by upscaling the diffusion autoencoder. It is specifically noticeable in the FID metric and the visual inspection, while MSE and PSNR barely change. However, it is important to mention that FID usually improves at higher resolutions. Therefore, it is possible that the $\epsilon$-prediction model with 280M parameters would achieve lower scores as well.

## 4.2 Video Compression

This section details the process of conducting evaluations on three experimental setups, intended to answer a unique question each, that was not addressed in the same or a similar way in the experiments on diffusion autoencoder for images. Specifically, ablations on reconstructions with different temporal compression ratios are done, followed by a training using higher quality data and lastly evaluating what effects adding more parameters in the form of attention layers has. These will be discussed in Section 4.2.1. Furthermore, an ablation to a classical method for video compression will be discussed in 4.2.2.

### 4.2.1 Hyperparameter-Evaluation

Following the first experiment, three different DiffAEs were trained with temporal compression ratios of $f_t \in [2, 4, 8]$, while all had a fixed spatial compression of f32. All evaluation results can be found in Table 6. As mentioned in Section 2.5.2, the FVD metric is better suited for evaluating experiments involving videos, as it takes into account the temporal axis as well. Therefore, all video experiments replace FID with FVD in their evaluation. An unused video data subset is used which was not seen during training. It can be observed that, as expected, higher temporal compressions result in worse results as indicated by the FVD. The $f_t = 2$ temporal compression model achieves FVD = 245.7596, the $f_t = 4$ model FVD = 330.7555 and the $f_t = 8$ diffusion autoencoder FVD = 397.1898. A visual comparison can be seen in Figure 4 between the different compression ratios. Interestingly, the MSE and PSNR are in a notable different value range compared to the image experiments, where the MSE is consistently orders of magnitude smaller, leading to a higher PSNR too.

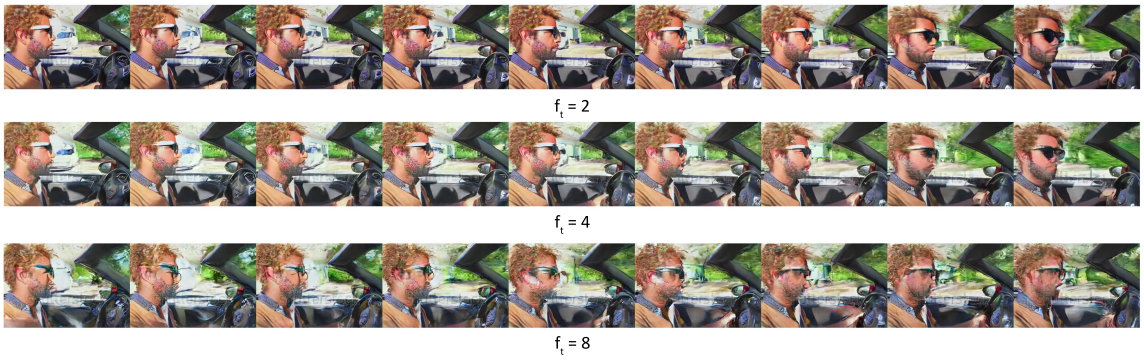


Figure 4: Visual comparison between different temporal compression ratios $f_t \in [2, 4, 8]$. More compression leads to worse reconstructions spatially and temporally.

The second experiment is about investigating the importance of high quality data. For that, one video diffusion autoencoder with temporal compression of $f_t = 4$ is trained only on WebVid-10M, which mostly consists of only low motion stock footage. A second model is trained using WebVid-10M, ACAV100M & HDVila-100M, which comprises a much more diverse dataset ranging from aesthetic stock footage to typical videos encountered in an every-day-life situation. Both models are trained for 500.000 iterations at a resolution of

$384 \times 512$ and later finetuned for 100.000 steps at $512 \times 704$. The results can be seen in Table 6 too, clearly demonstrating the benefit of using better data when looking at the FVD value. An improvement of 25.1372 points can be observed. Moreover, also the MSE and PSNR have improved after training on better data.
The last experiment examines if better results can be achieved by adding more parameters. Specifically, temporal-, and spatial-attention layers are added. Note that temporal-, and spatial-attention is not combined into a single layer as this grows the memory requirements exceptionally high through the quadratic complexity of attention and results in out-of-memory errors. After adding the attention layers, the model contains 1.3B trainable parameters compared to 734M as without this change. Note, that this model was trained on better data as well and finetuned at a higher resolution. It can be seen that, compared to the diffusion autoencoder that was finetuned and used better data, making the model larger, results in an improvement of 10.72 points on the FVD, while the MSE and PSNR slightly worsened.

Table 6: Experimental results for intra-evaluation of video diffusion autoencoders. Specifically, evaluation for different temporal compression ratios, increasing model size by adding attention layers, using more diverse data and finetuning at higher resolution. The table shows results for Fréchet video distance (FVD), mean-squared-error (MSE), peak-signal-to-noise-ratio (PSNR).

| Model | FVD | MSE | PSNR |
|---|---|---|---|
| $\epsilon$-target (f2) | 245.7596 | 0.007497 | 22.7778 |
| $\epsilon$-target (f4) | 330.7555 | 0.008882 | 21.7545 |
| $\epsilon$-target (f8) | 397.1898 | 0.008827 | 22.0215 |
| $\epsilon$-target (f4) (finetuned) | 195.1561 | 0.008483 | 21.9025 |
| $\epsilon$-target (f4) (finetuned + better data) | 170.0189 | 0.006504 | 23.5642 |
| $\epsilon$-target (f4) (larger + finetuned + better data) | 159.2998 | 0.006625 | 23.2281 |

### 4.2.2 Evaluation against other Methods

This section examines the quality between the classical compression method AVC and the discussed video diffusion autoencoder. Similarly, to the comparison between JPG and the image DiffAE in Section 4.1.2, the common metrics (FVD, MSE, PSNR) are calculated for video reconstructions using varying compression factors on AVC. While JPG uses the

quality parameter Q to indicate the level of compression, AVC features a parameter called *constant rate factor* or short CRF. This value ranges between 0 and 51, where lower numbers indicate a lower compression and thus a higher quality of the reconstruction. The common standard for the constant rate factor is CRF=18. The Table 7 shows the metric evaluation for different CRF values. It is recognizable that, as expected, the metric values decline as the compression is increased. For better comparison to the video diffusion autoencoder, Figure 5 visualizes the trade-off between the compression ratio $c_r$ and the FVD metric for both algorithms. Specifically, the figure compares the three base models ($f_t \in [2, 4, 8]$) at a resolution of $384 \times 576$. It can be seen that for lower compression ratios, the AVC algorithm achieves lower FVD values than the video diffusion autoencoder, while as the compression increases, the DiffAE obtains better values. Interestingly, the table also shows that AVC constantly obtains better MSE and PSNR results than the DiffAE. Even the highest compression from the AVC table (CRF = 40) results in a better MSE and PSNR than the lowest temporal compression ($f_t = 2$) of the video DiffAE. The reason for this is very likely that the blockiness artifacts caused by AVC, are not influencing the MSE and PSNR, but have a large impact on the FVD.

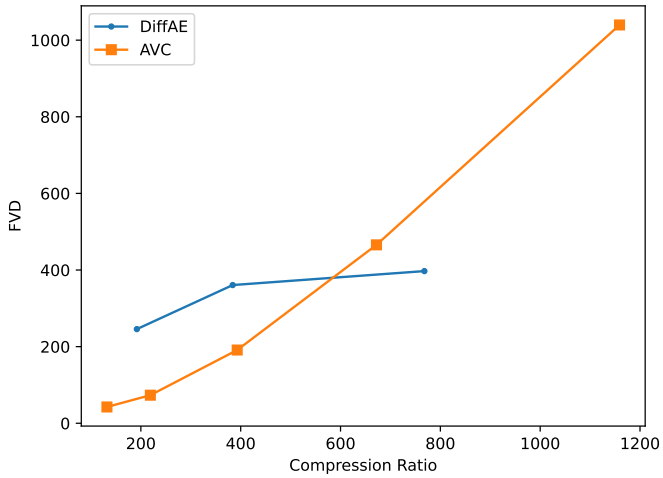


Figure 5: Comparison between AVC and video diffusion autoencoder. The figure shows the trade-off between the compression ratio and the FVD metric. All results were obtained on a pixel resolution of $384 \times 576$.

Table 7: Evaluation results for the classical compression algorithm AVC. Experiments used a varying quality factor when storing videos, resulting in different compression ratios. The table shows results for Fréchet video distance (FVD), mean-squared-error (MSE), peak-signal-to-noise-ratio (PSNR). Note, all metrics are evaluated at a pixel resolution of $384 \times 576$.

| Model | $c_r$ | FVD | MSE | PSNR |
|---|---|---|---|---|
| AVC (CRF=18) | 132 | 42.54 | 0.000285 | 36.1728 |
| AVC (CRF=25) | 219 | 73.30 | 0.000393 | 34.8193 |
| AVC (CRF=30) | 393 | 191.18 | 0.000643 | 32.7789 |
| AVC (CRF=35) | 672 | 465.69 | 0.001068 | 30.6124 |
| AVC (CRF=40) | 1159 | 1039.65 | 0.001768 | 28.4644 |

# 5 Discussion

Diffusion autoencoders show a promising new direction for data compression, achieving high quality reconstruction from compression ratios that were neither achieved by classical-, nor neural-methods before. This bachelor thesis explored the guiding research questions and discussed the current limitations of prior neural compression methods such as the VQGAN and found that its capabilities are exhausted for high spatial compressions of f32. Furthermore, the second research question was answered by showing how it is possible for diffusion models to learn efficient and high compression ratios for images and videos. While there might be multiple approaches to do so, this thesis made use of a small encoder and a large iterative diffusion decoder. It was shown that this works very well for the image modality. On the other hand, while using a diffusion autoencoder for video compression works as well, it still lacks behind classical approaches in both quality and latency. Finally, an in depth hyperparameter analysis was made for answering the third question, which showed that using an x0-prediction combined with a min-SNR loss works best out of all examined options in terms of FID evaluation. For video compression the results also revealed that using a better dataset leads to better reconstructions and, as expected, choosing smaller temporal compressions, leads to better results as well. However, there are still possible improvements to be made. Firstly, it was shown that diffusion autoencoders require more time to reconstruct data because of the large amount of model parameters, as well as the iterative sampling. This leaves these two bottlenecks as poten-

tial starting options for future work. Established methods like knowledge-distillation [72] can be experimented with, in order to learn a model that contains fewer parameters, while maintaining the same performance. On the other hand, the number of inference steps can be reduced by various distillation techniques as well [37] [73] [74]. One promising approach is called consistency distillation introduced by Song *et al.*[75]. This method takes a pretrained diffusion models and makes predictions from different positions on a trajectory consistent. That means that the model now tries to minimize $(f(x_t, t) - f(x_{t+1}, t+1))^2$, given that $x_t$ and $x_{t+1}$ are both diffused with the same noise $\epsilon$. This replaces the original MSE loss between the prediction and the target. Successfully implementing both options could lead to real-time encoding and decoding, making this option more viable for a larger crowd of consumers.

Another critical point is the training data distribution and the unknown performance on different out-of-distribution data points. While methods like JPG perform well on nearly all distributions of images, it is hard to make similar claims about a neural network approach. Usually neural networks tend to perform well on data that lies inside of its training distribution, however performance quickly degrades when moving outside of it. For example, the diffusion autoencoder's ability to reconstruct medical images or cosmic photos of stars is likely going to suffer important losses in its reconstruction. Those could occur in the form of wrongly reconstructing the shape of a cell or the location of a cluster of stars. The importance of these errors highly depend on the context and who is looking at the images. A person conducting medical analysis should not be dealing with potential errors from the decoding stage and should be able to rely on the veracity of it. Therefore, when thinking about deploying such a method it is important to be aware of this weakness and to also inform downstream users of the problems. It is important to keep in mind that one is not dealing with a deterministic approach of decoding data. The diffusion autoencoder solves a differential equation in the reverse process, which by its nature is stochastic. Therefore, different initial seeds will lead to different outputs. While these differences will occur on a tiny scale, the aforementioned example showcases their potential influence.

Evaluation is an important step in the process of developing new methods and aiming for an objective comparison to others. However, evaluation metrics introduce another dimension of problems, especially neural metrics. As a result, it is of utter importance to discuss those and convey realistic expectations to what can be assumed when using them. The FID metric is a controversial metric in the generative AI field [76] [60]. As

mentioned earlier, it's making use of an Inception [52] network that was trained on ImageNet [58]. The Inception network can only be expected to generate rich features for photos that are similar to ImageNet. Those are object-centric natural images. Certainly, this does not represent a large amount of possible images and excludes popular categories such as paintings, drawings and other kinds of artworks. Moreover, humans pay attention to specific details in images such as other humans depicted in them. As ImageNet does not contain many of such photos, expecting truthful feature comparisons becomes questionable. Another important aspect is the resolution that the Inception network was trained on. This bachelor thesis was mostly working with high-resolution images starting at $512 \times 512$, while the Inception network was trained at a resolution of $299 \times 299$. This represents another large bottleneck of this metric, because features at lower resolutions include less details, hence high frequencies are potentially not being considered properly by the network as well. The work by Pernias *et al.*[60] discloses more critics to the metric, such as investigating the data augmentation that was used to train the Inception network. It is obvious that the FID metric introduces many problems, and similar conclusions can be obtained when delving into the FVD metric used for video analysis. However, they are still widely used metrics and finding better ones is a difficult challenge. As a result, this bachelor thesis uses them as well, while pointing out their drawbacks and intending to make the audience aware to critically perceive evaluations using these metrics. Moreover, this work leaves finding better suited metrics for future work.

Furthermore, numerous times throughout the work it was seen that the MSE and PSNR metrics were misaligned with the visual appearance of reconstructed data. Also, subtle differences for the ranking of different methods compared to the FID / FVD were seen, where large decreases in performance according to FID / FVD, barely changed the results of the MSE and PSNR. This highlights the fact that both metrics might not be best suited for the task of visual compression evaluation, as the element-wise focus accepts and punishes different aspects compared to a human. However, in other tasks, these metrics might be very well aligned with human perception. Still, since both metrics are cheap to compute, it is of no harm to use them in combination with other metrics, while being aware of the limitations. Furthermore, it has been demonstrated, that reconstructions are not perfect, and especially high frequencies will persist to be a challenging problem. Scaling up the models further could improve this. However, another option is accepting the possibility of generating novel high frequencies on the fly. Potentially, this trade-off could be left to the

user to decide how much compression is desired and how much information can be lost. Moreover, another way of enabling orders of magnitude more compression could be through quantization, such as vector quantization or lookup-free-quantization [77]. Currently, a diffusion autoencoder projects an image to a latent representation consisting of multiple vectors (e.g. $16 \times 16 \times 16 \times 16$, given an f32 spatial compression with the dimensions being $c \times h \times w \times p$), that each have a dimension $c$ ($c = 16$ in this work) and that need to be stored with a certain precision / number of bits (e.g. fp16). Learning a well working quantization would mean that only $h \times w \times p$ bits are needed for storing a single image, yielding a 16x higher compression. To put this into context, this would allow for storing a dataset of 500 million images of size $512 \times 512$ on a consumer laptop, requiring only 256GB of space and achieving an unbelievable compression ratio of $c_r = 1536$. For comparison, the same dataset stored with the JPG algorithm and a compression ratio of $c_r = 15$, requires 26214GB.

Furthermore, the evaluation has shown that the video diffusion autoencoder is outperformed by AVC, both in FVD and MSE & PSNR. This work suggests that there is a lot of potential for improving the reconstruction quality of video DiffAEs as seen by noticeable improvements by simply using different data. Potentially, improving the compressing scheme can be one starting point. It could be that the model is not able to learn an optimal temporal compression due to the nature of convolutions and pooling layers. The investigation of finding better encodings is left for future extensions of this algorithm. Moreover, the experiment setup considers a very isolated process in which the models were trained on 16fps with 16 frames at a specific resolution (similar argument holds for image diffusion autoencoders). Using different settings, such as less fps or lower / higher resolution leads to out-of-distribution data as well and it is uncertain how the model would react for each possible setting. On the other hand, AVC is constructed to work with varying fps, lengths, resolutions etc., which naturally is more attractive for actual usage. However, it is important to keep in mind that the diffusion autoencoder is a recent approach that not much research has been going into and this study showcases its potential and early experiments. Therefore, it is to be seen if this approach becomes promising to more people, which could lead to rapid improvements and make the method more generally applicable.

Finally, the scaling ablations in Section 4.1.3 have to be considered with care. As mentioned earlier, the evaluations for the smallest model were conducted on a resolution $512 \times 512$, while the larger models were both evaluated at $1024 \times 1024$. This does not

represent perfect conditions for making conclusions. Furthermore, the larger models had slightly changed conditions during training and were trained for significantly longer. Closing this gap, might also result in more similar visual appearance and performance on metrics. However, it is still very likely that larger models perform better, as also shown by comparing the results of the 700M to the 3B parameter model.

# 6 Conclusion

This work extends the research on the diffusion autoencoder, introduced in Pernias *et al.*[60]. An in-depth study is presented concerning critical parts of the framework. Specifically, this bachelor thesis conducts experiments on the modality for image and video. Numerous insights have been gained through the ablation studies, highlighting the usefulness of the method to the discussed problem of data compression. The diffusion autoencoder achieves higher compression ratios than previous neural-compression algorithms, while still maintaining a well defined latent space. It also achieves similar compression and quality in its reconstructions than classical compression algorithms. This way the diffusion autoencoder can be seen as combining the large compression ratios from classical methods and the good properties of the latent space from the neural methods. This requires less space for storing data and allows for using the latent space for downstream tasks. Moreover, this work also hypothesizes more hidden and unexplored potential in this method, specifically for video compression. Future successful research could enable compression ratios never seen before and allow downstream technologies to be more accessible to more people, as training neural networks in a highly compressed latent space can be realized more easily on consumer hardware, while datasets could be stored on commodity hardware as well.

# A Appendix

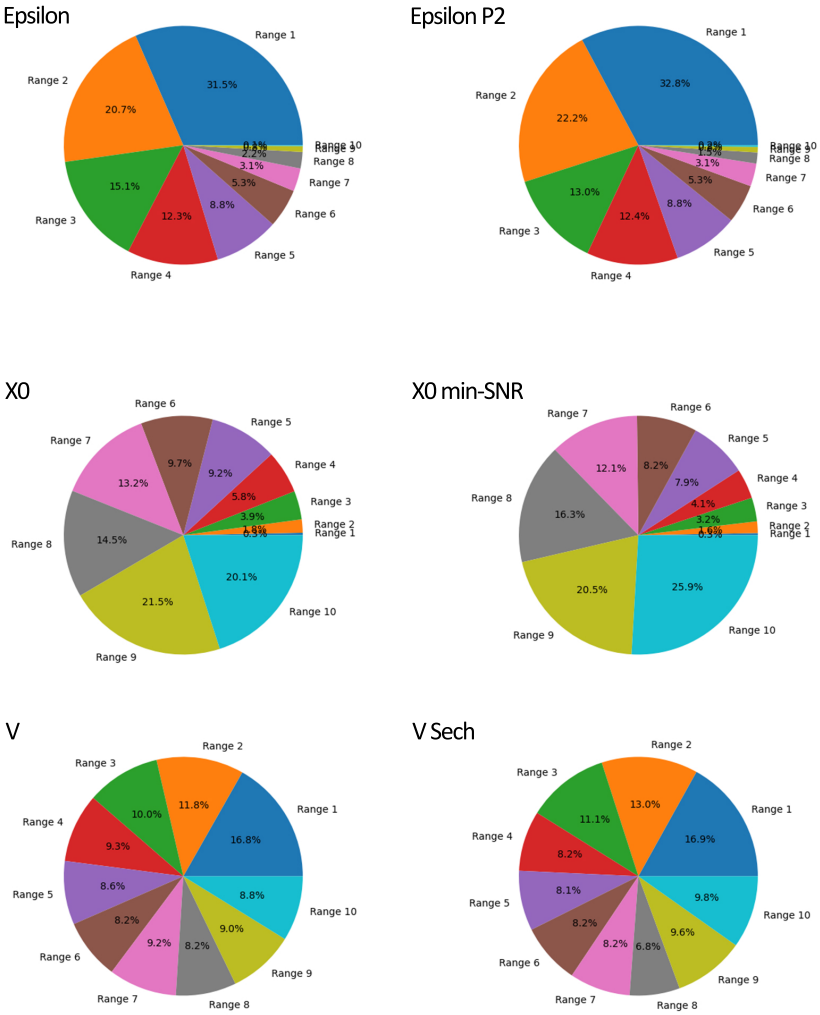


Figure 6: Loss distributions using pie charts for each training experiment regarding the target and loss weight. A higher area in one part refers to a higher loss in that timestep range. Depicted here are 10 timestep ranges from $t = 0.0$ (Range 1) to $t = 1.0$ (Range 10) in 0.1 increments. For instance, $\epsilon$-objective has the highest loss in low-noise ranges, close to $t = 0.0$.


Figure 7: Visual comparison between diffusion autoencoders of varying model sizes (300M, 700M, 3B). Note, the 700M and 3B were trained for longer and trained at $1024 \times 1024$, while the 300M model only trained at $512 \times 512$.

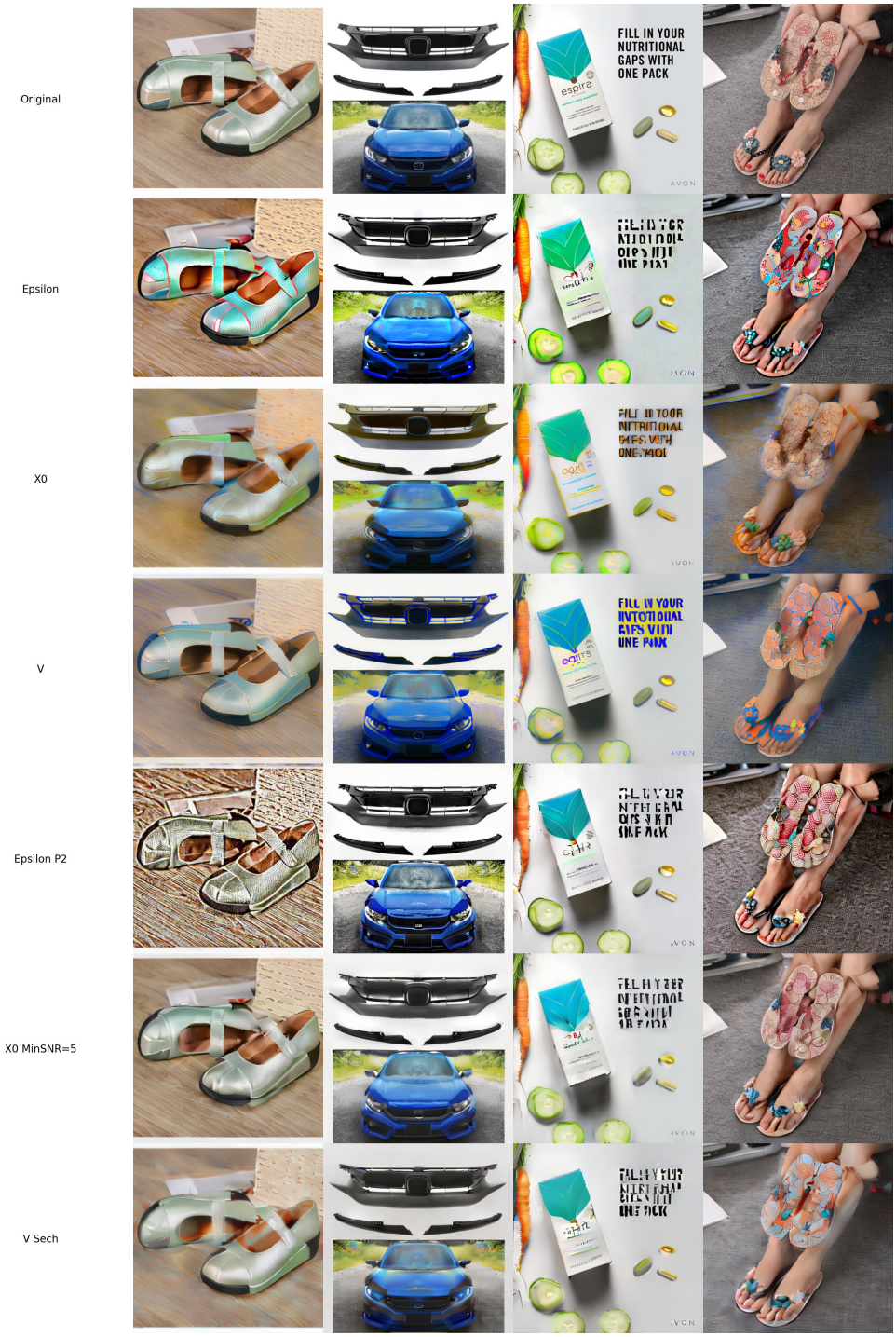


Figure 8: Visual comparison between diffusion autoencoders trained with different targets and loss weightings. All models are trained and sampled at an image resolution of 512×512.

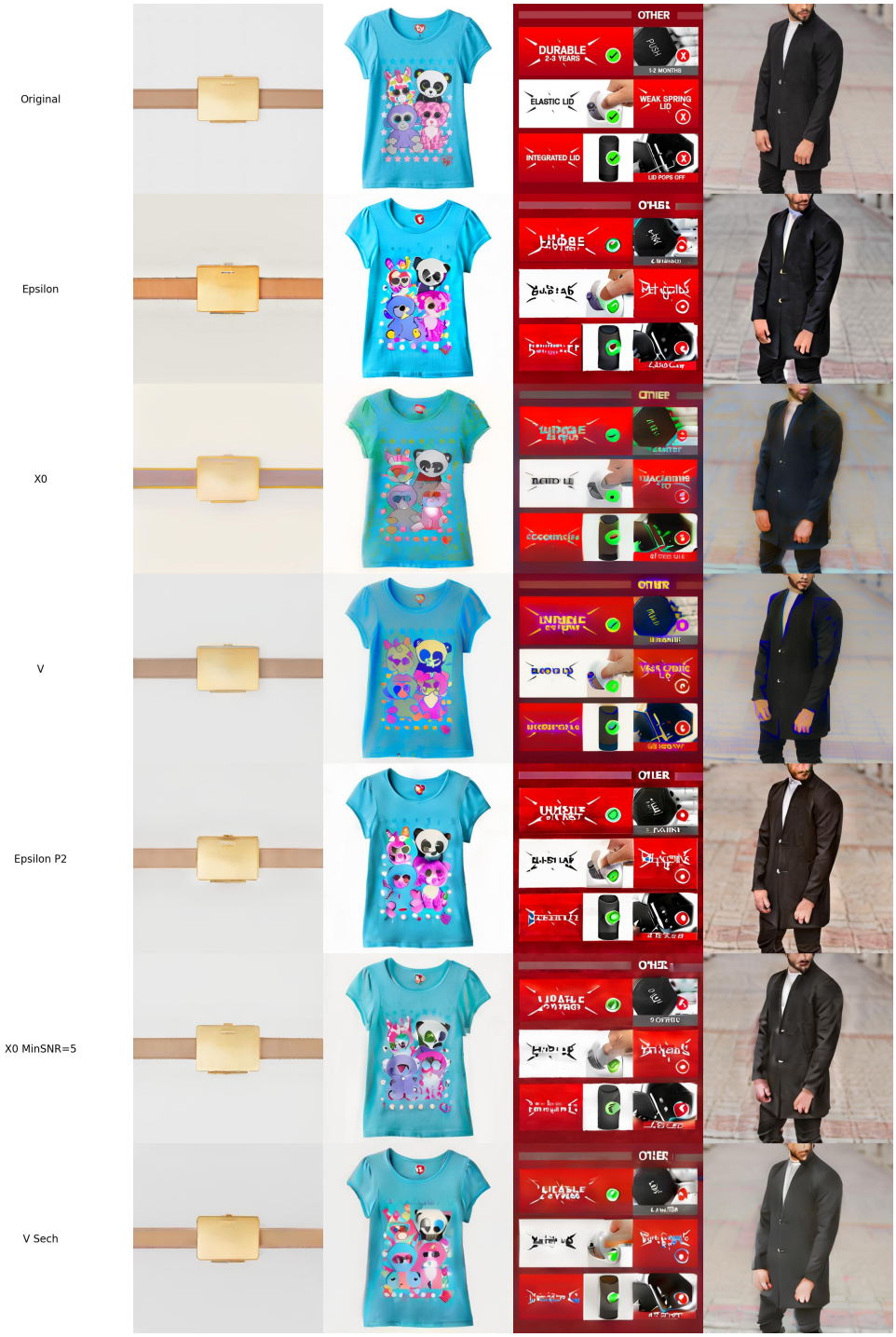


Figure 9: Visual comparison between diffusion autoencoders trained with different targets and loss weightings. All models are trained and sampled at an image resolution of 512×512.

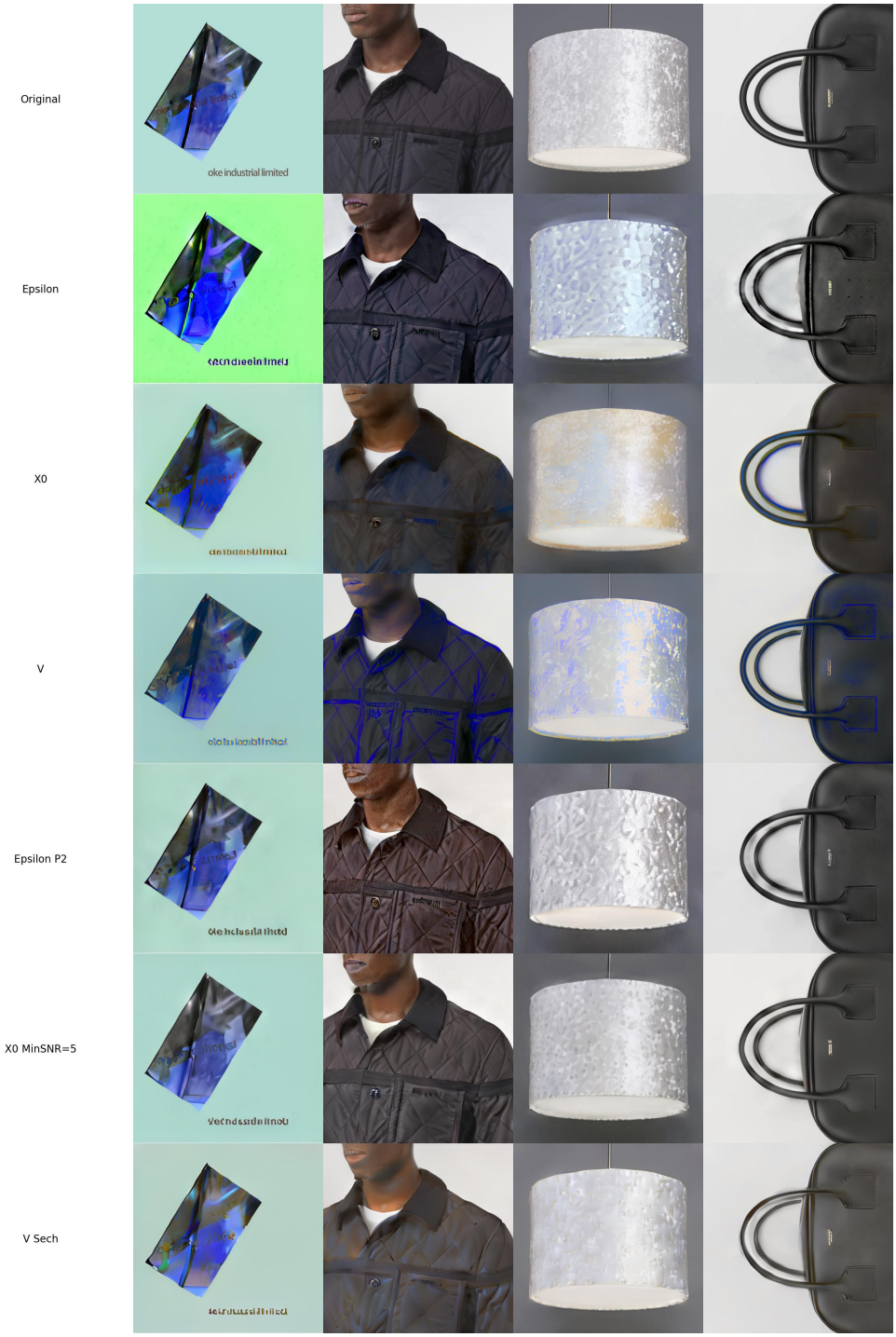


Figure 10: Visual comparison between diffusion autoencoders trained with different targets and loss weightings. All models are trained and sampled at an image resolution of 512×512.


Figure 11: Visual comparison between diffusion autoencoders trained with different targets and loss weightings. All models are trained and sampled at an image resolution of 512×512.


Figure 12: Visual comparison between a VQGAN (f32) a comparable diffusion autoencoder with different number of inference steps ($N \in [1, 10, 30]$). All models are trained and sampled at an image resolution of $512 \times 512$.

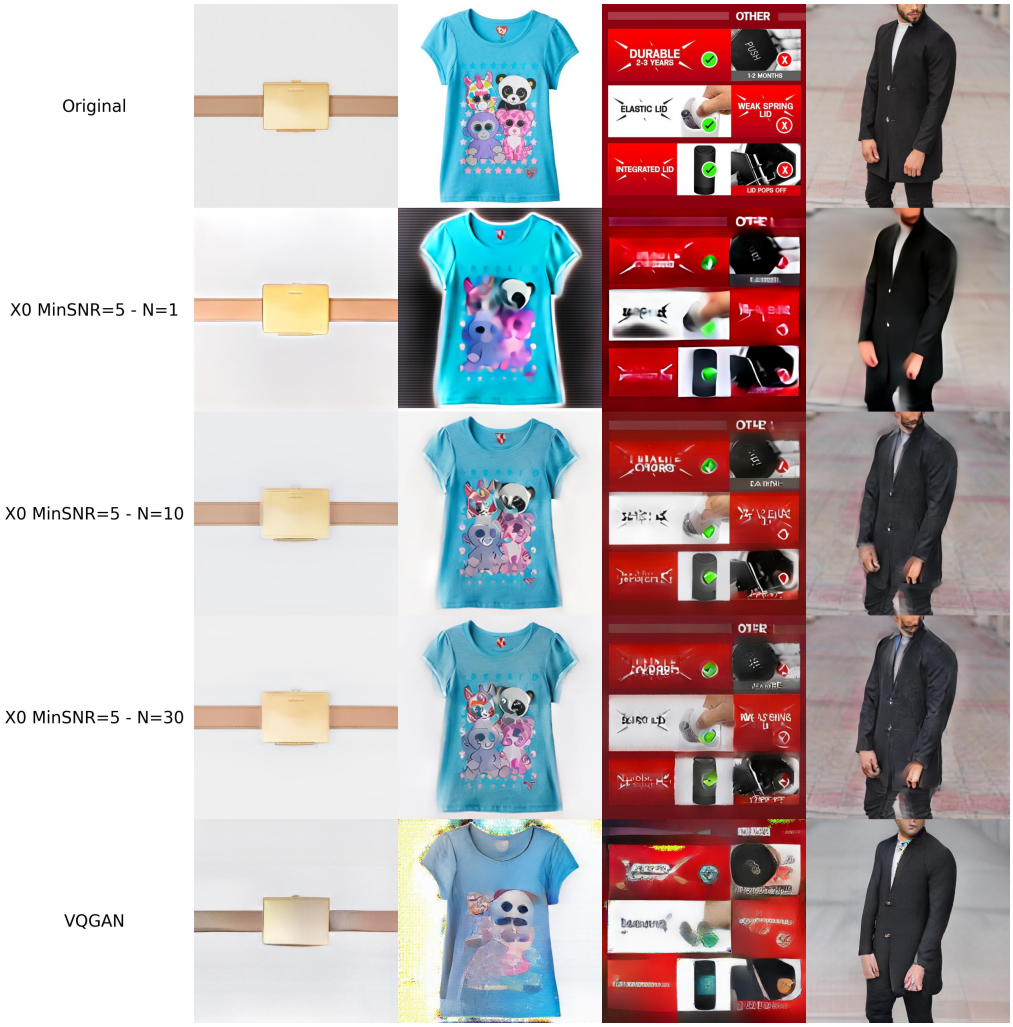


Figure 13: Visual comparison between a VQGAN (f32) a comparable diffusion autoencoder with different number of inference steps ($N \in [1, 10, 30]$). All models are trained and sampled at an image resolution of $512 \times 512$.

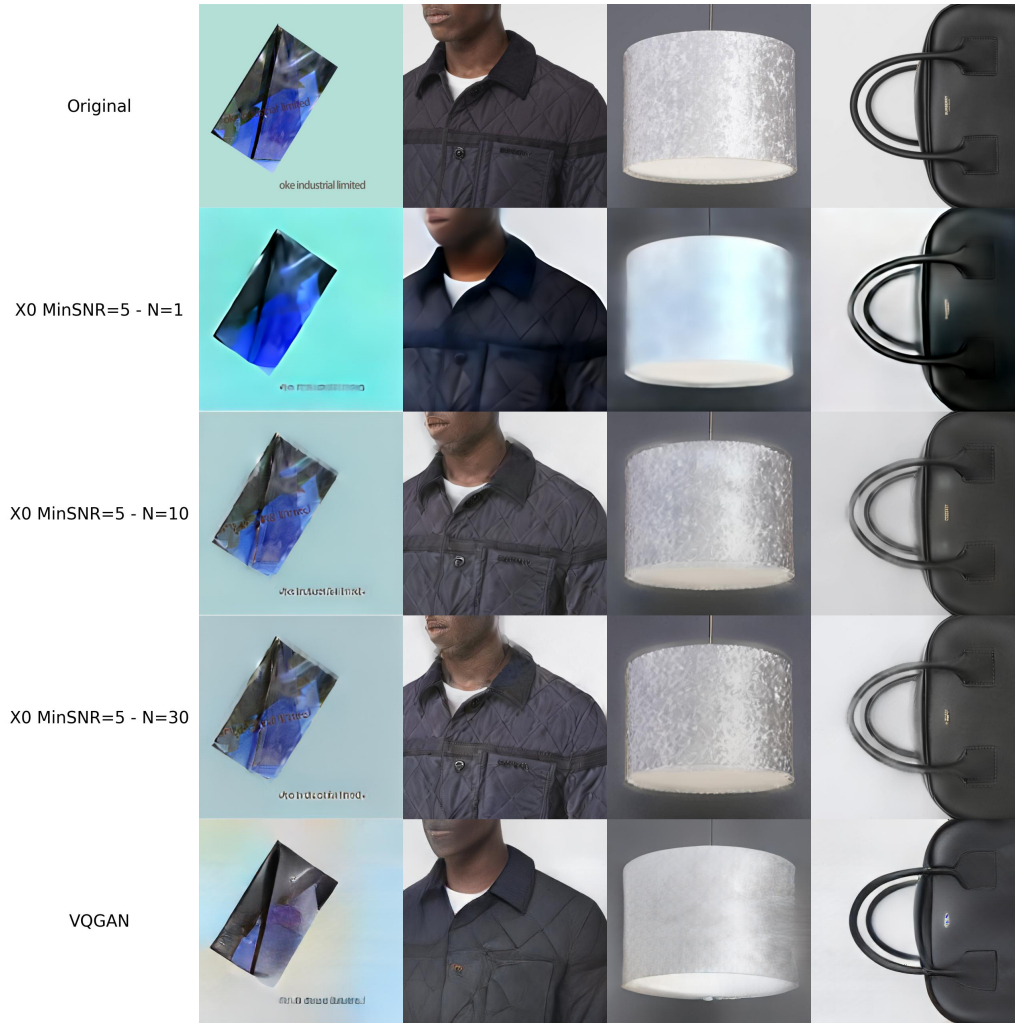


Figure 14: Visual comparison between a VQGAN (f32) a comparable diffusion autoencoder with different number of inference steps ($N \in [1, 10, 30]$). All models are trained and sampled at an image resolution of $512 \times 512$.

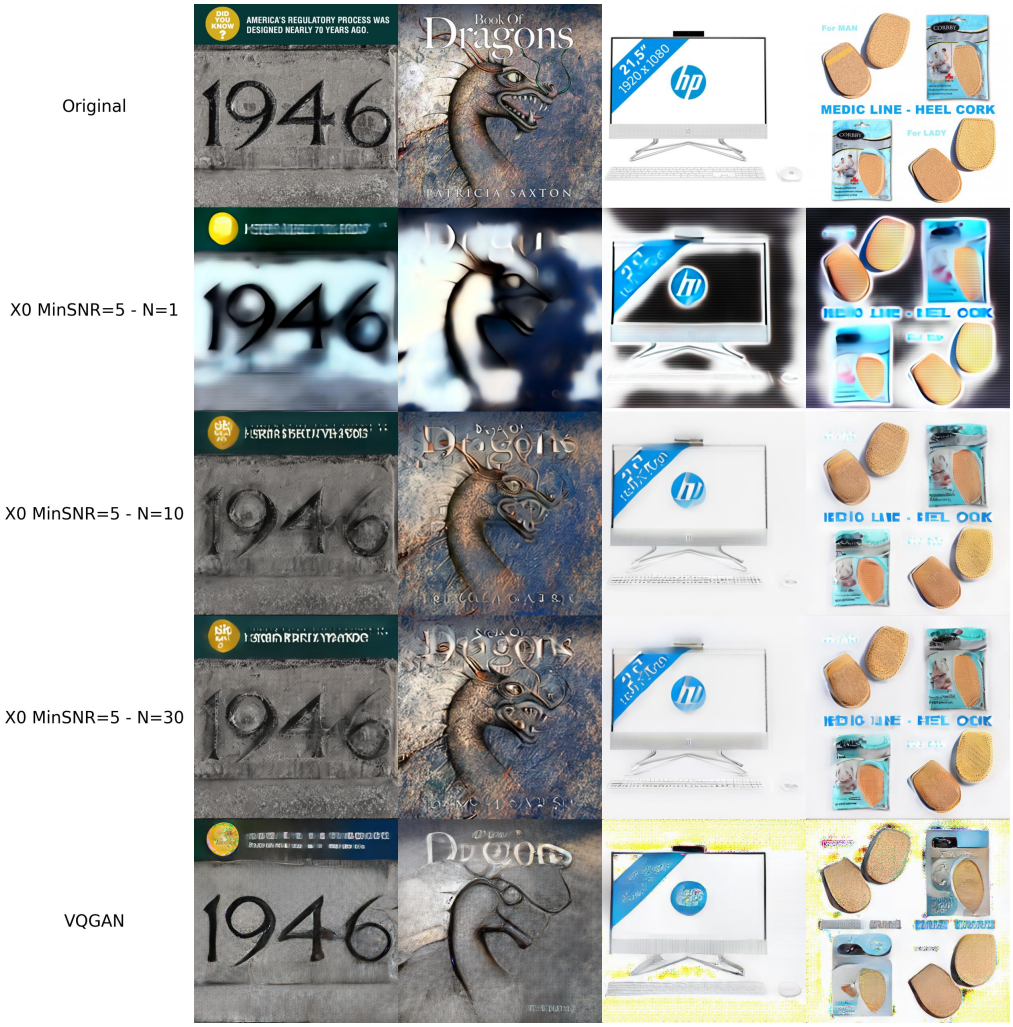


Figure 15: Visual comparison between a VQGAN (f32) a comparable diffusion autoencoder with different number of inference steps ($N \in [1, 10, 30]$). All models are trained and sampled at an image resolution of $512 \times 512$.

# References


[1] F. Liang, W. Yu, D. An, Q. Yang, X. Fu, and W. Zhao, "A survey on big data market: Pricing, trading and protection," *Ieee Access*, vol. 6, pp. 15 132–15 154, 2018.

[2] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[3] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.

[4] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8780–8794, 2021.

[5] T. Karras, M. Aittala, T. Aila, and S. Laine, "Elucidating the design space of diffusion-based generative models," 2022.

[6] R. Berns, *Billmeyer and Saltzman's Principles of Color Technology*. Wiley, 2019. [Online]. Available: https://books.google.de/books?id=vEyMDwAAQBAJ

[7] Y. Patel, S. Appalaraju, and R. Manmatha, "Human perceptual evaluations for image compression," *arXiv preprint arXiv:1908.04187*, 2019.

[8] G. K. Wallace, "The jpeg still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[9] W. Weber, "Differential encoding for multiple amplitude and phase shift keying systems," *IEEE Transactions on Communications*, vol. 26, no. 3, pp. 385–391, 1978.

[10] P. N. Graphics, "Portable network graphics," *Overview. html*.

[11] J. Eppink, "A brief history of the gif (so far)," *Journal of visual culture*, vol. 13, no. 3, pp. 298–306, 2014.

[12] A. J. Hussain, A. Al-Fayadh, and N. Radi, "Image compression techniques: A survey in lossless and lossy algorithms," *Neurocomputing*, vol. 300, pp. 44–69, 2018.

[13] R. Gray, "Vector quantization," *IEEE Assp Magazine*, vol. 1, no. 2, pp. 4–29, 1984.

[14] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[15] "Video developer report 2018." [Online]. Available: https://go.bitmovin.com/hubfs/Bitmovin-Video-Developer-Report-2018.pdf

[16] S. Soatto, R. Frezza, and P. Perona, "Motion estimation via dynamic vision," *IEEE Transactions on Automatic Control*, vol. 41, no. 3, pp. 393–413, 1996.

[17] G. H. Granlund, "In search of a general picture processing operator," *Computer Graphics and Image Processing*, vol. 8, no. 2, pp. 155–173, 1978. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0146664X78900473

[18] D. Gabor, "Theory of communication. part 1: The analysis of information," *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, vol. 93, pp. 429–441(12), November 1946. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/ji-3-2.1946.0074

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[20] O. Loyola-González, "Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view," *IEEE Access*, vol. 7, pp. 154 096–154 113, 2019.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[22] M. M. Petrou and C. Petrou, *Image processing: the fundamentals*. John Wiley & Sons, 2010.

[23] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner,

S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[24] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes *et al.*, "Photorealistic text-to-image diffusion models with deep language understanding," *arXiv:2205.11487*, 2022.

[25] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with CLIP latents," *arXiv:2204.06125*, 2022.

[26] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," *arXiv:2112.10741*, 2021.

[27] G. E. Hinton and R. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Advances in Neural Information Processing Systems*, J. Cowan, G. Tesauro, and J. Alspector, Eds., vol. 6. Morgan-Kaufmann, 1993. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1993/file/9e3cfc48eccf81a0d57663e129aef3cb-Paper.pdf

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[30] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2528–2535.

[31] D. P. Kingma, M. Welling *et al.*, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[32] J. Rocca, "Understanding variational autoencoders (vaes)," 2019. [Online]. Available: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

[33] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[34] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 12 873–12 883.

[35] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International Conference on Machine Learning*. PMLR, 2015, pp. 2256–2265.

[36] "General diffusion framework." [Online]. Available: https://github.com/WARP-AI/gdf

[37] T. Salimans and J. Ho, "Progressive distillation for fast sampling of diffusion models," *CoRR*, vol. abs/2202.00512, 2022. [Online]. Available: https://arxiv.org/abs/2202.00512

[38] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8162–8171.

[39] J. Choi, J. Lee, C. Shin, S. Kim, H. Kim, and S. Yoon, "Perception prioritized training of diffusion models," 2022.

[40] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," *arXiv preprint arXiv:2011.13456*, 2020.

[41] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 681–688.

[42] P. Mörters and Y. Peres, *Brownian motion*. Cambridge University Press, 2010, vol. 30.

[43] Y. Song, "Generative modeling by estimating gradients of the data distribution," 2021. [Online]. Available: https://yang-song.net/blog/2021/score/

[44] B. D. Anderson, "Reverse-time diffusion equation models," *Stochastic Processes and their Applications*, vol. 12, no. 3, pp. 313–326, 1982.

[45] R. Po, W. Yifan, V. Golyanik, K. Aberman, J. T. Barron, A. H. Bermano, E. R. Chan, T. Dekel, A. Holynski, A. Kanazawa *et al.*, "State of the art on diffusion models for visual computing," *arXiv preprint arXiv:2310.07204*, 2023.

[46] T. Hang, S. Gu, C. Li, J. Bao, D. Chen, H. Hu, X. Geng, and B. Guo, "Efficient diffusion training via min-snr weighting strategy," *arXiv preprint arXiv:2303.09556*, 2023.

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[48] R. L. Burden, *Numerical analysis*. Brooks/Cole Cengage Learning, 2011.

[49] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, "Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models," *arXiv preprint arXiv:2211.01095*, 2022.

[50] G. T. Fechner, "Elements of psychophysics, 1860." 1948.

[51] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[52] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[53] T. Unterthiner, S. van Steenkiste, K. Kurach, R. Marinier, M. Michalski, and S. Gelly, "Fvd: A new metric for video generation," 2019.

[54] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6299–6308.

[55] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 684–10 695.

[56] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 096–10 106.

[57] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[58] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[59] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, 2015, pp. 234–241.

[60] P. Pernias, D. Rampas, M. L. Richter, C. J. Pal, and M. Aubreville, "Wuerstchen: An efficient architecture for large-scale text-to-image diffusion models," 2023.

[61] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1874–1883.

[62] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[63] "Stability ai." [Online]. Available: https://stability.ai/

[64] J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Li, L. Ouyang, J. Zhuang, J. Lee, Y. Guo *et al.*, "Improving image generation with better captions," *Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf*, 2023.

[65] M. Byeon, B. Park, H. Kim, S. Lee, W. Baek, and S. Kim, "Coyo-700m: Image-text pair dataset," https://github.com/kakaobrain/coyo-dataset, 2022.

[66] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman *et al.*, "Laion-5b: An open large-scale dataset for training next generation image-text models," *arXiv:2210.08402*, 2022.

[67] M. Bain, A. Nagrani, G. Varol, and A. Zisserman, "Frozen in time: A joint video and image encoder for end-to-end retrieval," in *IEEE International Conference on Computer Vision*, 2021.

[68] S. Lee, J. Chung, Y. Yu, G. Kim, T. Breuel, G. Chechik, and Y. Song, "Acav100m: Automatic curation of large-scale datasets for audio-visual video representation learning," in *ICCV*, 2021.

[69] H. Xue, T. Hang, Y. Zeng, Y. Sun, B. Liu, H. Yang, J. Fu, and B. Guo, "Advancing high-resolution video-language representation with large-scale video transcriptions," in *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[70] J. Ho and T. Salimans, "Classifier-free diffusion guidance," *arXiv:2207.12598*, 2022.

[71] D. P. Kingma and R. Gao, "Understanding diffusion objectives as the elbo with simple data augmentation," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[72] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[73] A. Sauer, D. Lorenz, A. Blattmann, and R. Rombach, "Adversarial diffusion distillation," *arXiv preprint arXiv:2311.17042*, 2023.

[74] X. Liu, X. Zhang, J. Ma, J. Peng, and Q. Liu, "Instaflow: One step is enough for high-quality diffusion-based text-to-image generation," *arXiv preprint arXiv:2309.06380*, 2023.

[75] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever, "Consistency models," 2023.

[76] Y. Kirstain, A. Polyak, U. Singer, S. Matiana, J. Penna, and O. Levy, "Pick-a-pic: An open dataset of user preferences for text-to-image generation," 2023.

[77] L. Yu, J. Lezama, N. B. Gundavarapu, L. Versari, K. Sohn, D. Minnen, Y. Cheng, A. Gupta, X. Gu, A. G. Hauptmann *et al.*, "Language model beats diffusion–tokenizer is key to visual generation," *arXiv preprint arXiv:2310.05737*, 2023.