Data compression through a diffusion based model

Honors project

Isidora Stojadinovic
Angelos-Ermis Mangos
Šimon Sukup
Austin Roose

Supervisor
Pratik Gajane

Eindhoven, May 2023

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Contents

1 Introduction

In this project we have been experimenting with the diffusion models used in image compression, restoration and generation. Our main objective was training the model to find an optimal way of compressing an image and then learning how to return it to form as close as possible to the original one in a given number of steps.

The problem of image compression can be defined as the task of reducing the amount of data required to represent an image, while minimizing the loss of visual quality. This is achieved by removing redundant or irrelevant information from the image data, without compromising the essential details or features that make the image recognizable.

Image compression, reconstruction and generation using the diffusion models is motivated by the diffusion in physics and is based on the idea of simulating a diffusion that spreads information across the image. In diffusion-based compression algorithms, the image is represented as a set of pixel values, which are modified over time using a set of diffusion equations. The resulting modified pixel values are then used to reconstruct a compressed image that closely approximates the original image (to be maximized).

The key challenge in diffusion-based compression is to design diffusion equations (ex. neural net models) that effectively capture the important features of the image, while also achieving high compression ratios. This requires a careful balance between the amount of diffusion applied to the image data, and the preservation of important details and structures in the image. Another challenge in diffusion-based compression is to optimize the compression performance for different types of images and compression requirements. This includes considerations such as the image resolution, color depth, and compression ratio, as well as factors such as computational efficiency, storage requirements, and perceptual image quality.

Hence, the main problem of image compression using diffusion models involves designing and implementing a diffusion-based compression algorithm that can effectively compress images while maintaining a high level of image quality and fidelity.

More concretely we have been experimenting with the Lossy MNIST[1], Lossy CIFAR[2] and Lossless model. For the lossy models we can assume that when compressing the input (ground truth) can be corrupted (increasing compressive capabilities) as they can infer the semantic details and keep the perceptual quality high. For the lossless model on the other hand, the input is compressed as is. The results obtained are contained in Chapter 4.

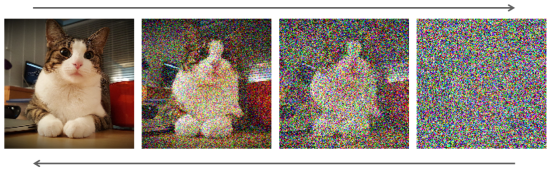
[1] MNIST database is a database of handwritten digits that is used for training various image processing systems
[2] CIFAR-10 dataset is a collection of 60000 images used to train machine learning and computer vision algorithms

2 Basics

2.1 What is diffusion model?

Simply put, diffusion models are generative models used for the generation of images (what we focused on in our project), sound etc. In other words, diffusion models, as well as any other generative model, are used for generating data similar to the one that they are trained on.

So, let us start from the beginning. Diffusion models are inspired by non-equilibrium thermodynamics [4] that deals with physical systems which are not in thermodynamic equilibrium but can be described in terms of macroscopic quantities. For instance, if we drop a tiny bit of blue paint (of the approximately same density as water) in a glass of water, the whole system (paint and water) will tend to its equilibrium that we can think of bluish water that has the same colour in all of its part. However, starting from the initial stage when the paint is concentrated in one spot of glass at its surface, it continues *diffusing* in water and eventually reaches (or at least gets closer to) equilibrium. Moreover, due to the physical and chemical properties predicting the level of diffusion and behaviour of paint is possible to describe. On the contrary, once equilibrium is reached or approached, reversing the diffusion process is not possible, i.e. once paint is deluded enough in water it is not possible to say and describe the diffusion process and tell where paint has been initially dropped. This process motivates the idea of diffusion models in image synthesis. Namely, given an image we would like to add some random noise to it in the form of kernels (which will be described later) which we can think of as diffusion and then train a model to reverse the process and reconstruct the image. Then, once a model is trained with a certain precision it will be able to generate completely new images and reconstruct the unfamiliar one. We provide an illustration in the following figure.



As we can see in the example, noise is gradually added to the image and then removed. Note that diffusion models consist of many more steps/ stages and that image quality is lost much more gradually i.e. the process is much smoother. The noise-adding process can be modelled using the Markov chains (a stochastic model that describes the sequences of possible events in which the probability of each event depends only on the state attained in the previous event). Hence, given a model in which noise is added gradually throughout $n \in \mathbb{N}$ steps, the colour of the pixel in j^{th} time step is determined purely based on the colour of pixels in $(j-1)^{th}$ time step and no other. We will come to this later and further explain possible ways of noise adding and removal process.

On the other hand, the reverse diffusion process takes as input the partially destroyed image (output of the forward diffusion process) and learns how to predict the mean of the noise at each time step noise of the image [3].

2.2 Why diffusion model?

Diffusion models play a substantial role in image (and sound) generation and reconstitution. One of the implementations of diffusion models that most of us have tried at least once is the text-to-image generation (mostly used is DALL-E-2 model), but there are many more of them [1]. Furthermore, image synthesis based on the provided information is of great importance when it is not possible to obtain an image in a standard way, as for instance in the imaging of complex molecules. Besides, this widespread use in anomaly detection, medical image reconstruction, and waveform signal processing show the significance and presence of diffusion models in science.

In addition to these applications, diffusion models find relevance in various other domains. Alongside image generation, they are employed in image reconstitution after compression for more cost-effective sharing and storage. This aspect becomes particularly appealing to the wider population as it allows for efficient sharing of images without sacrificing quality. Similarly, these models can also be employed in the reconstruction of images (and sound) from crime scenes and surveillance cameras. By utilizing the available data and applying diffusion models, investigators can enhance and restore valuable information from these visual and auditory records, aiding in criminal investigations and overall security.

The potential applications of diffusion models extend beyond the realms of image and sound generation. These models can also be applied in fields such as natural language processing, where they can assist in generating coherent and contextually relevant text based on given prompts. Additionally, diffusion models have shown promise in areas like data augmentation, where they can be used to generate synthetic data to augment training sets and improve the performance of machine learning algorithms.

Overall, the versatility and effectiveness of diffusion models make them a valuable tool in various domains. Their ability to generate, reconstruct, and augment data has significant implications for image synthesis, anomaly detection, medical imaging, crime investigation, and many other scientific and technological applications.

Exploring code implementations and enhancing the model architecture for specific purposes, such as image compression, is a pivotal area of current research and also focus of this project.

3 Models

As one can imagine, diffusion models are heavily reliant on stochastics, probability theory and math in general. Hence, in order to explain the model, let us firstly introduce some notation. When we talk about the $n \times m$ pixel image at any stage of diffusion process, we refer to its encoding of colour of each pixel as an $n \cdot m$ dimensional vector. We use the following notation

- x_0 initial (input) image
- x_n image encoding in n^{th} time step, where $n \in \{1, 2, ..T\}$
- $q(x_t \mid x_{t-1})$ is function (probability distribution) that takes image at time step $t-1$ and returns image at next time step t (related to forward process)
- $p(x_{t-1} \mid x_t)$ is function (probability distribution) that takes image at times step t and returns the mostly likely encoding of the image at time step $t-1$ (related to reverse process)
- I identity matrix

3.1 Variational diffusion model (VDM)

The variational diffusion model is the most standard one and the easiest to understand. Understanding the idea behind this model can be a good starting point for understanding other models and as so we start with it.

As we have already mentioned, given a collection of images (data), the first thing to be defined is the so-called forward diffusion process. During this process, Gaussian noise is gradually added to the input image x_0 step by step producing the sequence of image encoding $x_0, ...x_T$. The number of steps $T \in \mathbb{N}$ needs to be predetermined and should be large enough so that training of the backward process is beneficial but not too large since for $T \to \infty$ we will obtain the complete noise (image whose colour of each pixel can be assumed to be chosen at random). In other words, given an image encoding x_i for $i \in \{0, 1, 2...T-1\}$ we obtain x_{i+1} by changing the colour of each pixel (each entry in vector x_i) based on Gaussian distribution

$$q(x_t \mid x_{t+1}) \sim \mathcal{N}(\sqrt{1-\beta_t}x_t, \beta_t I) \qquad \beta_t \in [0,1] \tag{1}$$

where $\beta_t \in [0, 1]$ so that variance of the normal distribution is bounded (colour of any pixel does not vary to much from one to the next time step) and so that buildup of this drastic changes is avoided. Notice, that for reach time step we apply the Gaussian with different (potentially) variance. According to [3] this leads to more stable model, since it can be chosen such that the variance is slowly decreasing and that way that adding noise in later time steps is less prominent. The intuitive way to think about the issue with the constant variance, is that linear noise adding significantly speeds up increasing the loss of information needed for proper training of the model [5]. The effect can be also understood from the following example.

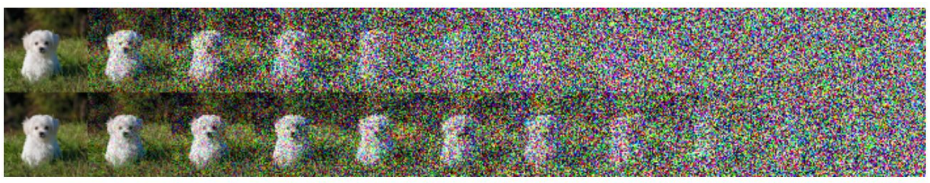


Figure 1: First line:linear noise adding, Second line: cosine noise adding

Now in order to find the closed formula for x_i we note that eq.(1) can be equivalently written as $x_{i+1} = \sqrt{1-\beta_t}x_i + \beta_t\mathcal{N}(0,1)$. That is, to avoiding applying the same formula k times and obtain x_k we can do the following. Let $\alpha_t = 1 - \beta_t$ and let $\overline{\alpha_t} = \prod_{k=1}^{t} \alpha_k$, then one rewrites eq.(1) as

$$\begin{aligned} q(x_t \mid x_{t+1}) &= \sqrt{1-\beta_t}x_t + \sqrt{\beta_t} \cdot \mathcal{N}(0, I) \\ &= \sqrt{\alpha_t}x_t + \sqrt{1-\alpha_t} \cdot \mathcal{N}(0, I) \\ &= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}x_{t-1} + \sqrt{1-\alpha_{t-1}} \cdot \mathcal{N}(0, I)) + \sqrt{1-\alpha_t} \cdot \mathcal{N}(0, I) \\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-1} + \sqrt{1-\alpha_t\alpha_{t-1}} \cdot \mathcal{N}(0, I) \end{aligned}$$

Where we used the property of normal distribution $\mathcal{N}(\mu_1, \sigma_1^2) + \mathcal{N}(\mu_2, \sigma_2^2) = \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. Hence, continuing this way, we obtain the following closed formula for x_t

$$\begin{aligned} q(x_t \mid x_0) &= \sqrt{\alpha_t\alpha_{t-1}...\alpha_0}x_{t-1} + \sqrt{1-\alpha_t\alpha_{t-1}...\alpha_0} \cdot \mathcal{N}(0, I) \\ &= \sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}} \cdot \mathcal{N}(0, I) \end{aligned}$$

Similarly can can define the reverse diffusion process function $p(x_t \mid x_T)$. We write

$$p(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

where μ_θ and Σ_θ are the neural networks that parameterise the Gaussian distribution and θ is the set of parameters (mean and variance) that is to be found.

Now, to find the function that predicts the output noise in image between two times steps we need some additional machinery. We define the *loss function* to be $-\log(p_\theta(x_0))$ (negative log-likelihood, defined as a sum over the log-probabilities of the observed data points, where $p_\theta(x_0)$ is the probability of a particular input) which is often used as the measurement of a loss for machine learning models, telling us how bad it's performing (the lower values of negative log-likelihood[3] imply the higher likelihood[4]). Now, notice that $p_\theta(x_0)$ depends on all of the element of the finite sequence $x_0, x_1, ...x_T$ so finding the value of x_0 for which p_θ reaches the minimum would require knowledge about the whole sequence and is not possible in practice [2].

To solve this problem, we introduce the concept of KL (Kullback-Leibler) divergence that is a measure of the difference between two probability distributions. In particular, KL divergence measures the amount of information lost when approximating one distribution with another. It is defined as the expected difference between the logarithms of the two distributions, weighted by the probability of each outcome:

$$D_{KL}(q(x_i \mid x_0) \mid\mid p_\theta(x_i \mid x_0)) = \sum_i q(x_i \mid x_0) \log(\frac{q(x_i \mid x_0)}{p_\theta(x_i \mid x_0)})$$

where as we have already defined $q(x_i \mid x_0)$ is the true distribution, $p_\theta(x_i \mid x_0)$ is its approximation and $i \in \{1, 2, ...T\}$.
Now note, that $D_{KL} \geq 0$, which can be easily proven using the fact that $-\log(x) \geq 1 - x$ for all $x > 0$. In particular (using the simpler notation),

$$\sum_i p(x_i) \log(\frac{p(x_i)}{q(x_i)}) = \sum_i -p(x_i) \log(\frac{q(x_i)}{p(x_i)}) \geq \sum_i p(x_i)(1 - \frac{q(x_i)}{p(x_i)})$$

[3] Equivalently, higher values of log-likelihood
[4] Chance of some calculated parameters producing some known data ($\neq$ probability).

$$= \sum_i p(x_i) - \sum_i q(x_i)$$
$$\geq 1 - \sum_i q(x_i) \geq 0$$

Furthermore, it is worth noticing that $D_{KL} = 0$ if p_θ is a perfect approximation of q (as $\log(1) = 0$) and $D_{KL} \geq 0$. Thus we obtain the following expression

$$\begin{aligned}
-\log(p_\theta(x_0))) &\leq -\log(p_\theta(x_0))) + D_{KL} \\
&= -\log(p_\theta(x_0))) + \sum_i q(x_i \mid x_0) \log(\frac{q(x_i \mid x_0)}{p_\theta(x_i \mid x_0)}) \\
&\leq -\log(p_\theta(x_0))) + \sum_i \log(\frac{q(x_i \mid x_0)}{p_\theta(x_i \mid x_0)}) \\
&= -\log(p_\theta(x_0))) + \sum_i \log(\frac{q(x_i \mid x_0)}{\frac{p_\theta(x_0|x_i)p_\theta(x_i)}{p_\theta(x_0)}}) \\
&= -\log(p_\theta(x_0))) + \sum_i \log(\frac{q(x_i \mid x_0)}{\frac{p_\theta(x_i)}{p_\theta(x_0)}}) \\
&= -\log(p_\theta(x_0))) + \sum_i \big(\log(p_\theta(x_0)) + \log(\frac{q(x_i \mid x_0)}{p_\theta(x_i)})\big) \\
&= \sum_i \log(\frac{q(x_i \mid x_0)}{p_\theta(x_i)})
\end{aligned}$$

However, using the conditioning we know that $p_\theta(x_i) = p(x_T) \prod_{i=1}^T p_\theta(x_{i-1} \mid x_i)$, while $q(x_i \mid x_0) = \prod_{i=1}^T (x_i \mid x_i - 1)$. Thus, we can get even nicer expression for the upper bound on $-\log(p_\theta(x_0))$ as follows using the $\log$ rules

$$\begin{aligned}
-\log(p_\theta(x_0))) = \sum_i \log(\frac{q(x_i \mid x_0)}{p_\theta(x_i)}) &= \sum_i \log(\frac{\prod_{i=1}^T (x_i \mid x_i - 1)}{p(x_T) \prod_{i=1}^T p_\theta(x_{i-1} \mid x_i)}) \\
&= -\log(p(x_0)) + \sum_{i=1}^T \log(\frac{q(x_t \mid x_{i-1})}{p_\theta(x_{t-1} \mid x_t)}) \\
&= -\log(p(x_0)) + \log(\frac{q(x_1 \mid x_0)}{p_\theta(x_0 \mid x_1)}) + \sum_{i=2}^T \log(\frac{q(x_t \mid x_{i-1})}{p_\theta(x_{t-1} \mid x_t)})
\end{aligned}$$

Now, using the Bayes' formula we can write $q(x_t \mid x_{i-1})$ as $\frac{q(x_{i-1}|x_i)q(x_i)}{q(x_{i-1})}$. However, since variance of all of the terms in $\frac{q(x_{i-1}|x_i)q(x_i)}{q(x_{i-1})}$ is to expected to be quite high [2][3] add an extra conditioning on the input image x_0 i.e. modify the expression to get the form $q(x_t \mid x_{i-1}) = \frac{q(x_{i-1}|x_i,x_0)q(x_i|x_0)}{q(x_{i-1}|x_0)}$. To understand why this is suitable choice of regulation and why it might lead to even better performance of the algorithm, take an image at time stem i. Then, given an information about x_0, we are more certain about the image at time step $i-1$ compared to the case we do not have this information (as it can be seen in following example). Basically, especially in the case when i is close to T probability of image at time step $i-1$ being any vector x_{i-1} is really low and set of possible outputs x_{i-1} is highly dispersed (i.e. variance is high). However, if initial image is provided probability distribution $p_\theta(xi-1 \mid x_i, x_0)$ brings more information and variability in the likely outcomes decreases.

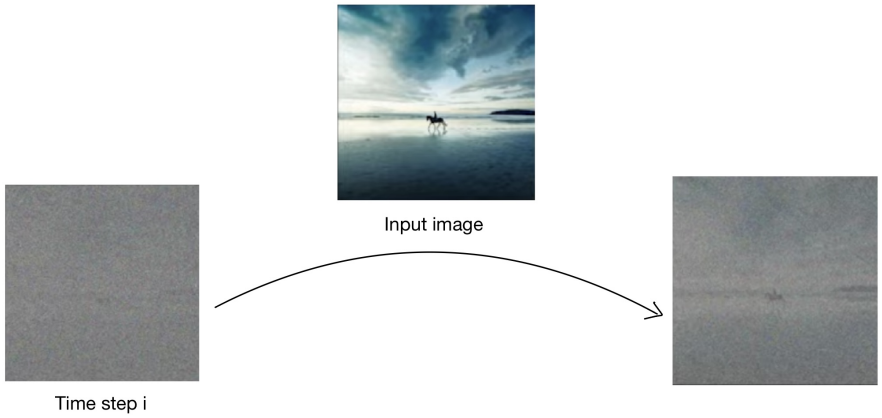


Figure 2: Conditioning on x_0 yields much more stable model

On the other, as one can imagine, conditioning can also be achieved through the use of prior information about the image. This can take the form of statistical priors, such as assuming that the image has a certain distribution, or geometric priors, such as assuming that the image contains specific structures or features.

We leave the demonstration and results of our implementation for later and now briefly, summarize the steps conducted by the training (learning) of a Variational diffusion model. The Variational Diffusion Model (VDM) learns through a two-step process: training and inference:

- *Training*: During the training phase, the VDM learns the underlying dynamics of the system using observed data. The training data consists of input variables and their corresponding outputs or targets, while VDM aims to capture the relationship between these variables and learn the parameters of the model that best represent the data.
  To achieve this, the VDM employs a combination of variational inference and optimization techniques. Variational inference involves approximating complex probability (posterior) distributions over the latent variables that govern the dynamical system with simpler distributions. This is achieved by formulating an optimization problem that minimizes the difference between the true data distribution and the estimated distribution (where we use KL divergence to measure the performance).
  During training, the VDM adjusts its parameters iteratively by optimizing a chosen objective function. This process involves computing gradients of the objective function with respect to the model parameters and updating them using gradient-based optimization algorithms such as stochastic gradient descent. The training data is typically presented to the model in batches, and the parameters are updated after processing each batch. This iterative process continues until the model converges or reaches a stopping criterion (imposed by us).
- *Inference*: Once the VDM is trained, it can be used for inference tasks, such as making predictions or estimating unobserved variables. Inference involves using the trained model to compute the posterior distribution over the latent variables given new input data. This is done by applying Bayes' rule and leveraging the learned parameters of the model.
  During inference, the VDM utilizes the trained variational approximation to estimate the posterior distribution. This approximation enables efficient and scalable computation, allowing the model to handle complex and high-dimensional data. The posterior distribution can then be used to generate predictions, perform simulations, or analyze the system dynamics.

It's important to note that the VDM's ability to learn and generalize depends on the quality and representativeness of the training data. Sufficient and diverse training data help the model capture the underlying patterns and dynamics of the system, enabling it to make accurate predictions and inferences on unseen data.

In conclusion, the Variational Diffusion Model learns by iteratively optimizing its parameters using variational inference and optimization techniques during the training phase. Once trained, it can perform inference tasks by estimating posterior distributions, leveraging the learned parameters to make predictions and analyze system behavior and even generate completely new images (voice).

3.2 Latent space diffusion model

The Latent Space Diffusion Model (LSDM) is a machine learning algorithm that models the evolution of a set of observations over time, by learning a low-dimensional latent space representation of the data, and modeling the dynamics of the latent variables over time. The basic idea behind the LSDM is to assume that the observed data points are generated from a set of latent variables, which evolve over time according to a diffusion process.

Latent space diffusion model is trained using a maximum likelihood estimation approach (which is actually based on maximizing the log-likelihood, recall higher values the higher values imply a more accurate model), where the goal is to find the set of parameters that maximize the likelihood of the observed data, given the model. To optimize the likelihood function, the LSDM uses a stochastic gradient descent approach, where the gradients are computed using a technique called the score function estimator (also known as likelihood-ratio estimator). This technique allows the gradients to be computed efficiently, even in the presence of high-dimensional latent variables.

Now, let us formulate LSDM using the following set equations. The diffusion process is modeled as

$$dZ(t) = \mu(Z(t), t)dt + \sigma(Z(t), t)dW(t)$$

where $Z(t)$ is the state of the latent variables at time t, $\mu(Z(t), t)$ is the drift term that describes the time-varying mean of the process, $\sigma(Z(t), t)$ is the diffusion term that describes the time-varying variance of the process, $dW(t)$ is Brownian motion (i.e. Wiener process), and dt is an infinitesimal time step. We will not go in a lot of details with the Brownian $W(t)$ motion here, as the only important thing to be noted is that it is a real-valued continuous-time stochastic process that satisfies certain set of properties. One and the most important is that increments $dW(t) = W(t + dt) - W(t)$ are normally distributed for any $dt > 0$.

Furthermore, we have the so called *observation model*:

$$X(t) = f(Z(t), t) + \varepsilon(t)$$

where $X(t)$ is the observed data at time t, $f(Z(t), t)$ is the observation function that maps the latent variables to the observed data, and $\varepsilon(t)$ is the observation noise. As already mentioned, LSDM relies heavily on the likelihood function of the observed data $X = \{X(1), X(2), ..., X(T)\}$ given the parameters $\theta = \{\mu, \sigma, f, \varepsilon\}$ and is given by:

$$L(X; \theta) = \prod [p(X(t)|Z(t), \theta) \cdot p(Z(t)|Z(t-1), \theta)]$$

where $p(X(t)|Z(t), \theta)$ is the conditional probability of observing $X(t)$ given $Z(t)$ and the parameters θ, and $p(Z(t)|Z(t-1), \theta)$ is the conditional probability of the latent variables at time t given the latent variables at time $t-1$ and the parameters θ. The product is taken over all time steps $t = \{1, 2, ..., T\}$.

Then we defined the log-likelihood function as

$$\log L(X;\theta) = \sum_{t=1}^{T} [\log p(X(t)|Z(t),\theta) + \log p(Z(t)|Z(t-1),\theta)]. \tag{2}$$

The whole objective is finding the parameters $\theta = \{\mu, \sigma, f, \varepsilon\}$ for which (2) attains maximum, which can be done using numerical optimization techniques, such as gradient descent or the expectation-maximization algorithm. Furthermore, the most common technique used for computing the gradient of the log-likelihood function with respect to the parameters is the score function estimator. By using the score function estimator, the diffusion model can efficiently estimate the gradients of the objective function without requiring explicit calculations of the likelihood function or solving complex differential equations.

In conclusion, by combining latent variables and diffusion processes, the LSDM captures the underlying structure and dynamics, enabling accurate modeling, analysis, and generation. It also offers a few benefits over the previously introduced Variational Diffusion Model.
Firstly, the incorporation of latent variables allows LSDM to capture the underlying structure and dependencies in the data. This allows for a more compact, flexible and expressive representation of the data, enabling better modeling and analysis (in VDM the ability to capture complex data dynamics and relationships is on contrary very limited).
Secondly, LSDM excels in generating new samples from the learned latent space. By leveraging the diffusion processes within the latent space, the LSDM can generate diverse and high-quality samples that capture the underlying data distribution. This makes it particularly useful for tasks such as data augmentation, synthesis, and simulation. The VDM, on the other hand, may face challenges in generating new samples without an explicit latent space representation.
Lastly, LSDM shows big improvements in the anomaly detection. By learning the normal data distribution in the latent space, the model can identify deviations or anomalies in the data by measuring their distance from the learned distribution. This can be beneficial in various applications, such as fraud detection, fault diagnosis, and outlier identification. The VDM, without a latent space representation, may have limited capabilities in anomaly detection.

3.3 Score-based generative modeling through stochastic differential equations (SDE)

Now we turn to the score-based generative modeling based on the stochastic differential equations (SDEs) which is a powerful method for modeling more complex distributions than Gaussian. The score-based generative model has as an objective to learn a probability distribution over a set of variables by specifying a set of differential equations that describe the evolution of the variables over time. Let, us describe the most important aspects of this model as well as present the idea behind 'learning'. For this purposes we introduce the notion of the score function, which is in our case the gradient of the log-likelihood of the data with respect to the data itself. The score function provides information about how the data is changing and can be used to learn a generative model. The next important ingredient for Score-based generative model are the the drift[5] and diffusion functions. these are the components of the SDEs that describe the evolution of the variables over time. The drift function describes the deterministic evolution of the system, while the diffusion function describes the random fluctuations in the system. Having these define, 'learning' goes as follows:

1. Estimate the drift and diffusion functions: The drift and diffusion functions can be estimated using the score function. This is done by using the score function to compute the gradient of

[5] One can think of it as a drift of any random process, for instance a stock price process

the drift and diffusion functions with respect to the parameters of the model. The gradient can then be used to update the parameters using an optimization algorithm.

2. Simulate the SDEs: Once the drift and diffusion functions have been estimated, they can be used to simulate the SDEs. This involves solving the differential equations numerically to obtain a sample from the probability distribution.
3. Train the model: The model can be trained by iteratively estimating the drift and diffusion functions and simulating the SDEs to obtain samples from the probability distribution. The samples can then be used to compute the log-likelihood of the data and update the parameters of the model using an optimization algorithm.

The special thing about this model is that we model the diffusion process by a prescribed stochastic differential equation (SDE) that does not depend on the data and has *no* trainable parameters. forward SDE:

$$dx = f(x,t)dt + g(t)dW.$$

Then, by reversing the process, we can generate new samples (images). To do this, we need to reverse the diffusion process. The SDE was chosen to have a corresponding reverse SDE in closed form [8]:

$$dx = [f(x,t)g^2(t)\nabla_x \log p_t(x)]dt + g(t)dW. \tag{3}$$

Hence as it can be seen from eq.3 in order to compute the reverse SDE, the score function (that is $\nabla_x p_t(X)$) needs to be estimated and for that purposed score-based model and Langevin dynamics are used.

4 Our models

We created 3 models during this research project for compressing and decompressing images with different levels of attributes as well as restoring these images with different levels of quality. First model that we created was based on MNIST dataset that is open-source database containing 60000 hand-written digits in training dataset [5]. When first model would be implement on only 1 black-white color channel, then second model would be trained using Cifar-10 dataset that contains classified images of dimensions 32 by 32 and these images have depth of 3, red, green and blue color scale. With final model we want to aim for lossless compression using efficient state-of-the-art bits-back coding algorithm [10]. Thus, it can be seen that in general our goal is to increase complexity of models over the project. We chose to use MNIST and Cifar-10 datasets for image data as these are easily accessible big datasets as well as there isn't any license attached to these sources that we should be aware of.

4.1 Convolutional Neural Network

All the models share the same task to compress and decompress images. This can be achieved with Convolutional Neural Network (CNN) that is abbreviation of Artificial Neural Network (ANN) that is better suited for image data type [11]. Neural networks in general are computational systems that include interconnected computational nodes that learn the input and optimize output. In our case we are interested in reaching score function as the output of this network based on images. Therefore, multiple steps would be necessary to extract different features of input image. In our case we use deep autoencoder for the main task of generating from original image data lower dimensional feature vector $\mathbf{z}$. It works by iteratively scaling down image.

4.2 Model for MNIST

Our first model was a simple VDM (which has been described above) alongside an encoder-decoder architecture to compress MNIST digits and decompress them. We started by going through the Variational diffusion models paper and implementing the theory described there alongside looking at other open sourced attempts to apply it. An important part of our implementation was to preserve the variance of the latent variables as they were diffused, as this was a integral part of the model (part of guiding the denoising process [6]).

To be more precise, the MNIST dataset consists of handwritten digits in black and white (meaning 1 channel), in a 28x28 width, height dimension. As such our model's encoder should take input $(B, 1, 28, 28)$ with B being the size of the image batch and output (B, Z) with Z being the latent dimension, which in our case equals to 32. The latent variables are then diffused but with the varianced preserved, using in this model a linear noise schedule (to keep it simple).

After the latent variables have undergone diffusion while preserving their variance, they are passed through the score network for denoising purposes. The score network is responsible for estimating the score, which represents the gradient of the log-density of the latent variables. By utilizing this gradient information, the model can iteratively update the latent variables and guide the denoising process effectively.

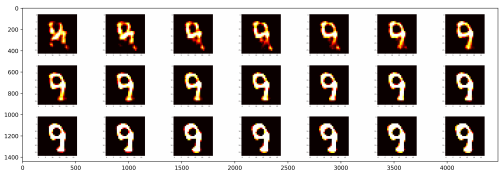
Once the latent variables have been denoised, they are fed into the decoder of our model. The decoder takes the denoised latent variables as input and generates a reconstructed image. In our implementation, the decoder maps the latent variables from the denoised space to the original data space by employing a series of transposed convolutions, which upsample the latent representation.

This process allows the decoder to generate a reconstructed image that closely resembles the original input.

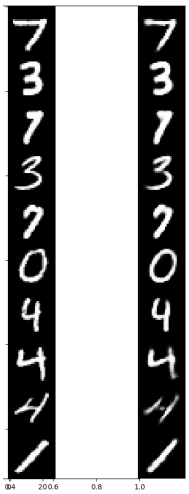
The output of the decoder is modeled as a Bernoulli distribution, representing the probability of each pixel being activated or deactivated in the reconstructed image. This probabilistic formulation enables the model to capture the uncertainty and variability present in the data. By sampling from this distribution, we obtain a reconstructed image that takes into account the inherent stochastic nature of the generative process.

Overall, our model combines the power of Variational Diffusion Models (VDM) and an encoder-decoder architecture to compress and decompress MNIST digits. Through the diffusion process, guided by the score network, the model effectively denoises the latent variables. Subsequently, the decoder reconstructs the image by transforming the denoised latent representation into a Bernoulli distribution, capturing the probabilistic nature of the data. This integrated approach enables our model to generate high-quality reconstructed images while preserving the variability and uncertainty inherent in the MNIST dataset. The loss of the whole model is modelled as explained in the "Variational diffusion model" section.

The results of the model were very satisfying, as we were able to do almost perfect reconstructions with about 95% percent reduction in size (0.04bpp). The diffusion also worked well albeit for small number of steps, meaning that when we used a lot of steps (max 1000) the results where bad, with many decompression artifacts.



(a) 21 chosen examples from the 1000 steps of diffusion during the reconstruction process of one of the MNIST digits.



(b) Example of part of batch used for compression and decompresison during testing. The original is on the left, the reconstructed pictures are on the right

Figure 3

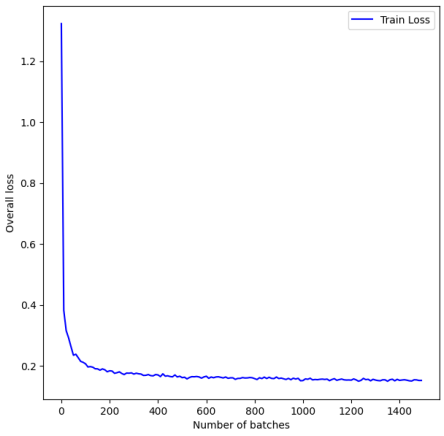


Figure 4: Loss graph during training of the linear-layer-based MNIST model.

4.3 CNN-based model for CIFAR-10

In order to improve the performance of the original linear layer-based model for the MNIST dataset, a number of changes to the architecture were implemented. The linear layers were employed for several reasons, particularly because of their computational efficiency and relatively good performance on small single-channel (single-colour) images. Most models in the field of computer vision, on the other hand, adopt an architecture based on convolutional neural networks. This is due to the fact that with linear operations on flattened images, information about the relationships of pixels over several columns or rows are lost. This can be prevented by using convolutional kernels instead of linear layers of neurons. PyTorch, unfortunately, requires considerably more computational resources.

To implement a CNN-based architecture, the model's encoder and decoder were first rewritten to include ResNet blocks, each comprising of convolutional layers, batch normalization for faster and more stable training, and max-pooling for downsampling in the encoder. The batch normalization also helped to prevent the vanishing gradient and exploding gradient problems. The encoder was implemented in such a way to enable several latent space dimensions, particularly a latent space of total sizes 7X7 and 14X14. This way, enough space at the bottleneck is given for further processing in the latent dimension by the ScoreNet.

The decoder had a symmetrical architecture, nearly identical to the encoder but with transpose convolutional layers for upscaling. The decoder was adjusted to upscale from the 7X7 and 14X14 back to a 32X32 latent space.

ScoreNet was also rewritten using convolutional layers; otherwise, the decoder would have to work with linear vectors which would prevent the implementation of convolutional layers. The architecture of ScoreNet was first done using U-Net. U-Net based architectures have their own bottleneck. It later turned out that the bottleneck caused by downsampling by the encoder and the U-Net bottleneck led to too large a loss of information on such small pictures. The approach was therefore changed and ScoreNet was rebuilt to keep the dimension constant during the entire propagation through the latent space. For this purpose, the architecture of the middle block of ResNet 18 with attention was implemented.

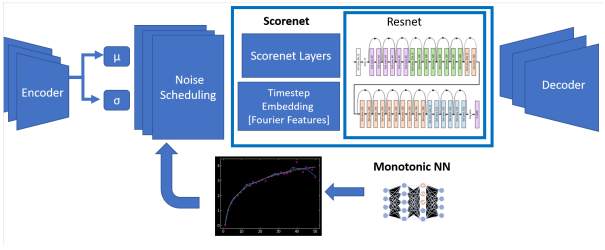
The loss functions and diffusion process structure based on the Variational Diffusion Models remained unchanged from the mathematical point of view, but several adjustments and code rewrites were

necessary. Unlike in the previous model with linear layers, additional operations on the 4-dimensional tensors were necessary to ensure that all concatenations of conditioning and embedding would work properly in PyTorch. Particularly during the first implementation, it turned out that some parts of the model did not learn due to the fact that the gradients did not propagate through the entire architecture during back-propagation. This was caused by the use of NumPy functions that discard the gradient that is saved along with the PyTorch tensor. It was therefore necessary to rewrite all functions that use NumPy and replace them with adequate PyTorch built-in operators. The correct gradient propagation was checked by looking at weights and bias coefficients of all layers of each part of the model.

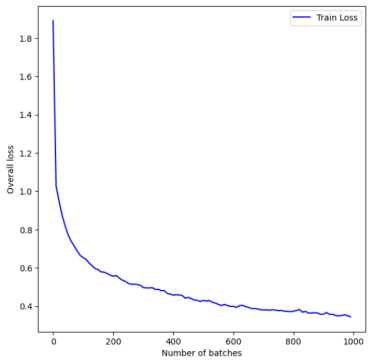
Subsequently, the model was trained using the AdamW optimizer, while a grid of parameters was tested to prevent overfitting and gradient explosion. Further, as was mentioned in the VDM publication, the model incorporated adaptive scheduling. So far, the models we deployed used linear noise scheduling that simply increases in magnitude with increasing time steps during diffusion. It was, however, shown that each dataset is specific in its features and linear scheduling might cause underperformance. The ability of the model to also learn when to induce what intensity of noise helps to learn to reconstruct certain features. For this purpose, it is necessary to implement a separate neural network that learns the noise scheduling function alongside the main model training.

The final improvement was the change of distribution for mapping to the pixel space in the final stage of the decoder. The original linear layer model used the Bernoulli distribution, which is typically used in variational autoencoder models. The distribution was changed to normal distribution. Additionally, the original model with linear layers included rounding in the decoder. This was used for the purpose of fast convergence during the diffusion process because rounding pixel values meant converging to a final shape with all pixels of the number of MNIST being bright. On the other hand, this led to highly contrasted and often white pictures on the CIFAR-10 dataset that has 3 channels. It turned out that discarding this function led to more natural-looking reconstructed photos.

These improvements and changes led to improved performance on the CIFAR-10 dataset as expected (we achieved 4.5bpp trained on 10 epochs or up to about 50% reduction in size with a non-corrupted input), but for a further quality improvement, much longer learning times would be needed.



(a)



(b)

Figure 5: The model architecture in (a) and the training loss curve for CIFAR-10 (b).

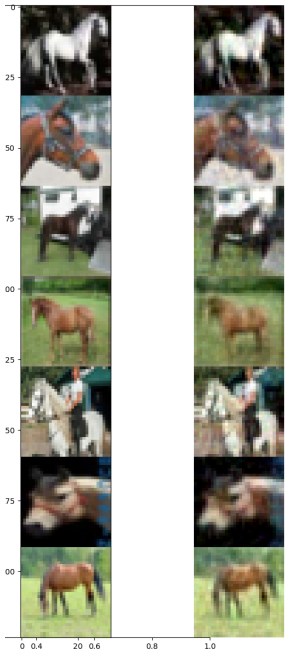


Figure 6: Subset of images from CIFAR-10 and their reconstructed version after compression and decompression using the model

4.4 Bits back encoding ANS model

With previous models we could only estimate through approximating probabilities, the distribution of data $p(x)$. We created neural network to optimize score function $\nabla log\ p(x_t)$ [12] of distribution $p(x)$. However, it would be much more preferred if we could somehow exactly encode this probability that we used to diffuse data. We also need to take into account that sometimes lossy compression is undesirable for a number of applications and some lossless implementation[6] of our model can potentially provide state-of-the-art results (if not practical).

We start with the initial message length that would contain compressed data x with probability p_x that is $-log\ p(x)$ [13], but as we need to express distribution p_x with some other distribution, then we would again need to compress this into the bitstream, increasing the size of the bitstream [14]. As mentioned earlier [look at section 3.1 of VAEs], we can express one distribution in terms of other. In that case will rely on Evidence Lower Bound (ELBO), which helps to calculate $p(x)$ knowing model $p(x,z)$ with posterior $q(z|x)$ in the formula $\mathbb{E}_{q(z|x)}[log\frac{p(x,z)}{q(z|x)}]$ [12]. If we, then appproximate $p(z|x)$ with distribution $q(z|x)$, then we can find that

$$\begin{aligned}
\mathbb{E}_{q(z|x)}[-\log p(x|z) - \log p(z) + \log q(z|x)] &= \mathbb{E}_{q(z|x)}[-\log p(x,z) + \log q(z|x)] \\
&= \mathbb{E}_{q(z|x)}[-\log p((z|x)p(x)) + \log q(z|x)] \\
&= \mathbb{E}_{q(z|x)}[-\log p(z|x) - \log p(x) + \log q(z|x)] \\
&= \mathbb{E}_{q(z|x)}[-\log p(x) + \frac{\log q(z|x)}{\log p(z|x)}] \\
&= \mathbb{E}_{q(z|x)}[-\log p(x)] + D_{KL}(q(z|x)\|p(z|x))[14]
\end{aligned}$$

If we now move $p(y)$ to the left, we get that

$$\begin{aligned}
\log p(x) &= -\mathbb{E}_{q(z|x)}[log\frac{p(x,z)}{q(z|x)}] + -\mathbb{E}_{q(z|x)}[log\frac{q(x,z)}{p(z|x)}] \\
&= -ELBO + D_{KL}(q(z|x)\|p(z|x))[14]
\end{aligned}$$

Therefore, we can reduce the length of the message by decoding ELBO part from bits without addressing KL part.

Bits-back algorithm works on bitstream and we have both sender and receiver side. Exact steps for bits-back algorithm is that by given initial bitstream of N_{init} bits, we decode z using $q_\theta(z|x)$. Then as we know latent variable z, then next by using z we can encode original data x with $p_\theta(x|z)$ and finally we would encode z to bitstream using $p(z)$ [15]. This implies total bitstream of length

$$N_{tot} = N_{init} + logq_\theta(z|x) - logp_\theta(x|z) - logp(z)[15]$$

Decoder would perform these steps in the reverse order. Relation to ELBO can be made as $N_{tot} - N_{init}$ is equal to negative ELBO. This also means that in order for sender to successfully decode z it needs

[6] The below theory of BB-ANS is short version of the cited papers, as we focus more on the artificial intelligence side of things and a bit less on lossless compression based on entropy coding.

to have initial bitstream of

$$N_{init} \geq -logq_\theta(z|x)$$

In our case what makes bits-back model more complicated is that it needs to utilize hierarchical latent variables z_t that are generated at various timesteps of diffusion, see 3.1. Therefore in iterative order, we decode from z_t z_{t+1}, encode z_t from $z_t + 1$ and finally for last layer L of latent variable model we encode z_L with $p(z_L)$. This all implies that for our implemented BB-ANS model required initial bits is equal to:

$$N_{init}^{BB-ANS} = -logq_\theta(z_{1:L}|x) = -\sum_{i=0}^{L-1} logq_\theta(z_{i+1}|z_i)[15]$$

We know that a variational diffusion model with a discrete (and fixed) number of timesteps can act as a hierarchical latent model [12] and as such, be used for the purposes of BB-ANS. Initially we simply improved upon the CNN-based VDM model, turning into a model without an initial encoder/decoder so that it operates on image space (which for BB-ANS lossless compression, is more than fine). Instead we made major changes on the score net part of the model (which predicts the noise). We used a deep ResNet module paired with an attention module to capture the intricacies of the input images and thus, model the score function better.

We our hierarchical model done, it was now time to build the BB-ANS module of the lossless model. This was modelled from available open sources and theory as well. The module used the latent model's prior and posterior probability functions to encode the picture (that we want to compress and send) into a bitstream. As such the size of the bitstream is directly related to the efficacy of the probability functions of the latent model. This means that the lossless model will have varying results depending on the dataset which the latent submodel is trained on.

The results of the lossless compression model, integrating Bits back coding (BB-ANS) with a variational diffusion model (VDM), were highly promising. After training the model for 10 epochs on the MNIST dataset, we achieved a compression rate of 1.411 bits per dimension (bpd). This result highlights the potential of our approach in significantly reducing the storage requirements for images while ensuring faithful reconstruction. Further evaluation and comparison with existing state-of-the-art compression algorithms would provide deeper insights into the model's performance and its potential for real-world applications. Of course though, the speed of the model is still an issue to be used all over the world (in the internet). This would be an interesting future avenue of research. The implementation of the BB-ANS module should also have been clearer in code so that its more idiomatic of the inner workings.

5 Individual parts

5.1 Isidora Stoajdinović

I am Isidora Stojadinovic, a third-year student of Applied Mathematics at Eindhoven University of Technology. In this section of the appendix, I will describe my contribution to the team process and team deliverables of our project.

I joined the group in November after the choice of topic and the general planning of the work to be done were completed. During the first few weeks, I caught up with the work of the other group members. Afterwards, I mainly focused on the theoretical part behind the generative (and in particular diffusion) models. To better understand how kernels are used in image processing and generation, I started by obtaining basic knowledge of image processing. Later on, I shifted my attention to diffusion models and learnt a great deal about many probabilistic and stochastic approaches used in image generation and reconstruction (decompression).

During my study of diffusion models, I have gained a deeper understanding of a powerful class of machine learning models, and the theory behind them and understood various techniques used in image generation. As a natural, follow-up to my studies, I have learnt many different topics and applications of various mathematical techniques. The main and most interesting is the Score-based generative model due to the fact that it relies on stochastic differential equations. By studying these models, I have learnt about the Brownian motion and Ito Integration, which were of particular interest due to their wide application out of the scope of Generative Models themselves.

Even though I did not participate in the development of the models, the work of other group members and group meetings provided me with practical examples and a glimpse of how the implementation of a few models might work. Furthermore, by trying to understand the issues that have been arising in the implementation processes as well as keeping track of the results I have obtained an insight into the practical part of the topic and got an idea of how theoretical ideas can be translated into the product that is nowadays used by all of us.

When it comes to group work, I believe that work division enabled each member of the group to learn about things that he is interested in, while not keeping the quality of the end deliverable at a really high level. Additionally, in my opinion, and when I reflect on the group organization and efficacy compared to the one that I had in my project group last year I notice a big improvement. Firstly, the size of the group has been halved, which allowed for better communication. Secondly, with an average of $8-10$ hours invested on this project, I have learnt significantly more compared to the last year. One of the reasons for this is the better organization inside the group, better organization of the track, but most importantly also the motivational topic itself. Lastly, I believe that group did a great job and that contribution of each member has led to the successful completion of this year's project.

5.2 Šimon Sukup

In this project, I primarily made contributions towards the implementation and improvement of the Variational Diffusion Model (VDM) for image compression. Firstly, I developed the first working autoencoder for the group to see the whole process of compression by a simple architecture in action using a convolutional architecture. We consequently worked on the first implementation of the VDM for the MNIST dataset. Here, I implemented one of the loss functions in the first VDM and also worked on the debugging of the three losses. This was an essential component of the VDM, and I reused the implemented autoencoder as the first and last sections of the VDM.

Additionally, I rewrote the VDM's encoder-decoder architecture to be linear, which turned out to be necessary to prevent the long training time (at that time without the Google Colab accounts). I also made considerable contributions towards the optimization of the model to see if it was presentable by adjusting the depth, activation functions, optimizers, learning rate, and hyperparameters to achieve the best possible results with the current architecture. I prepared materials such as the model structure and photos of the input, output, and embedding for the midterm presentation, which were also reused in this report.

I then worked on improvements of the model while other members of the team worked on the bits-back coding algorithm implementation with the old model. Particularly, I worked on reading publications and implementing the monotone neural network for the noise scheduling.

Further, I improved the model architecture for the CIFAR-10 dataset by making a new version of our VDM and rewriting the whole model to work with convolutional layers. Initially in this phase, I designed the model comparison of convolutional encoder and decoder, which themselves downsampled and upsampled the images, and the ScoreNet in the latent space that comprised of a U-Net-based reconstruction network. However, this approach was infeasible because of the double downsampling, which resulted in high loss of information and vanishing gradient. To prevent this, I replaced the U-Net-based ScoreNet with a ResNet-based ScoreNet without downsampling, which kept the dimensions the same over all layers of the ScoreNet.

These steps took a considerable amount of time and effort because I had to check if gradients propagate, if the loss functions work correctly, if all dimensions match (CNN engineering), if all model sections' weights update, and if the model does not suffer from gradient explosion or vanishing gradient. I also worked on rewriting the model to propagate the gradient correctly in CUDA to be able to train them faster in Google Colab. Finally, I tested, implemented, and acquired data on the new improved model for the report and wrote the report section on this model version.

Throughout the project, I spent an average of 11-13 hours a week, including studying and reading the literature, publications, and learning to code in PyTorch. I also participated in the Honors Academy as a representative, gave feedback to the Honors council, helped with the meeting organization, and took part in council meetings. In terms of the teamwork, I am sure that the team did a great job discussing the steps, tackling all technical and organizational challenges, and preparing for the presentations and meetings. Without the team and their insights into the model architecture and the technical skills, the project would not have given such results.

5.3 Angelos-Ermis Mangos

During the course of this project, I first worked on compiling a file of research dedicated to diffusion models and its potential applications in compression data (in particular, images). To begin, I thoroughly researched variational diffusion models and latent diffusion models, reading academic papers, existing literature, and cutting-edge techniques in the field. This research phase allowed me to gain a deep understanding of the theoretical foundations and mathematical frameworks of these models. I also focused deeply on understanding the framework we were going to use (torch) and how to appropriately use it to build custom models, which took a significant amount of time to learn correctly (that is because I looked into competing frameworks such as TensorFlow and jax).

Next, I translated my research into practical implementation by developing a lossy compression model tailored specifically for the MNIST dataset. A key area of my focus was the score net, which plays a crucial role in estimating the score function required for diffusion processes. I dedicated significant effort to designing the architecture of the score net, selecting appropriate neural network structures

(such as attention modules, resnet, etc), and optimizing its hyperparameters (slightly as this wasn't the focus of the project).

During weeks before the midterm presentations, I made significant contributions in debugging the MNIST model code so that we finally have a working compression scheme to show and talk about at the presentations. The bugs mainly stemmed from our then lack of understanding of the framework used (pytorch) and the intricacies of the loss function we developed for the MNIST dataset (in particular the diffusion loss part and how that should be calculated based on literature).

Furthermore, I delved into the mathematical formulation of the loss functions used in the model (and understanding how to use them). I ensured that the encoder/decoder (latent diffusion models) and score net outputs were configured correctly to align with these loss functions. By fine-tuning these components, I aimed to minimize information loss while achieving significant compression ratios.

I also participated significantly in extending the lossy compression model by incorporating a lossless version using Bits back coding (BB-ANS) and asymmetric numerical systems (ANS). This involved integrating the BB-ANS method with a variational diffusion model (VDM) as a hierarchical latent model (when it has discrete timesteps). To accomplish this, I conducted extensive research on the BB-ANS method, its theoretical foundations, and its practical implementations found online, while also exploring similar projects that utilized variational autoencoders (VAEs) and BB-ANS as its latent model.

My initial focus was on understanding the BB-ANS method and its underlying principles. I gained insights into the integration of BB-ANS with latent variable models and the potential benefits it offers for lossless compression. This newfound knowledge was used to produce the VDM+BB-ANS lossless compression model.

In total I spent an average of about 10-12 hours per week to work on the project. This includes organizing with the team, having meetings to talk how to proceed with the project and also reflect on what we want to do (splitting work, working on what we want to learn and also on our strengths). In general, I think the project was very good for all of us and we produced some very interesting and practical results (with the team contributing mostly fairly across the board).

5.4 Austin Roose

During this project I mainly was investigating relations between different models that we were developing in order to understand technically how we can gain performance for image compression with neural networks. In terms of code, I didn't unfortunately write any significant parts of code for the final model, but I was working on testing if dimensions for encoding and decoding the image data are correct for our autoencoder. For this I created unit tests that would check the dimensions of encoded and decoded data at timestamps and assert these values to expected values. I also worked on some sampling code that was required in our case to sample the data using our score network. In order to fit generators that would sample values from specific distribution into our model seed needed to be generated appropriately and different parameters set to specific values. For report work, I wrote the introduction to models and datasets that we will use and mathematical formulation for bits-back model.

As more valuable insight, I got familiar how powerful neural networks are for optimizing distributions and losses in order to capture relations between different datapoints. This enables to combine many mathematical formulas into neural networks to achieve processes such as diffusion, which can be used in many different fields. I got better understanding of types of different neural networks such as convolutioonal neural network and how forward process and back-propagation work. I learned how

to structure these neural networks into specific entities using pytorch framework and how to connect different units such as sampling and loss function code into coherent packages.

I definitely overestimated the amount of work needed to be contributed to this project and as our models that we worked with were more mathematically complex, then I didn't have enough time to focus into the mathematical theory part to fully understand the concepts and so I couldn't contribute that much to the final result of the project. Mostly what knowledge I gained from this project is that neural networks are very powerful for optimizing different distributions and combining neural networks with encoder-decoder models can lead to insightful data. In average, I worked for the project around 4 hours per week.

References

[1] Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Zhang, W., Cui, B., Yang, M.-H. (2022). Diffusion Models: A Comprehensive Survey of Methods and Applications.https://arxiv.org/abs/2209.00796

[2] Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., Ganguli, S. (2015). Deep Unsupervised Learning using Nonequilibrium Thermodynamics. https://proceedings.mlr.press/v37/sohl-dickstein15.html

[3] Ho, J., Jain, A., Abbeel, P. (2020). Denoising Diffusion Probabilistic Model. https://arxiv.org/pdf/2006.11239.pdf

[4] Weng, L. What are Diffusion Models? Retrieved May 8, 2023, from https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

[5] Nichol, A., Dhariwal, P. (2021). Improved Denoising Diffusion Probabilistic Models. https://arxiv.org/abs/2102.09672

[6] Kingma, D. P., Salimans, T., Poole, B., Ho, J. (2023). Variational diffusion models.https://arxiv.org/abs/2107.00630

[7] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., ; Ommer, B. (2022, April 13). High-resolution image synthesis with Latent Diffusion Models. arXiv.org. Retrieved April 23, 2023, from https://arxiv.org/abs/2112.10752

[8] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., ; Ommer, B.Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole (2021, February 10). Score-Based Generative Modeling through Stochastic Differential Equations.

[9] LeCun, Y. (n.d.). THE MNIST DATABASE of handwritten digits. Available at: http://yann.lecun.com/exdb/mnist/ (Accessed: 28 May 2023).

[10] Keng, B. J. L. (n.d.). Lossless Compression with Latent Variable Models using Bits-Back Coding. Available at: https://bjlkeng.github.io/posts/lossless-compression-with-latent-variable-models-using-bits-back-coding/ (Accessed: 28 May 2023).

[11] O'Shea, K. and Nash, R. (2015) An introduction to Convolutional Neural Networks, arXiv.org. Available at: https://arxiv.org/abs/1511.08458 (Accessed: 28 May 2023).

[12] Luo, C. (2022) Understanding diffusion models: A unified perspective, arXiv.org. Available at: https://arxiv.org/abs/2208.11970 (Accessed: 28 May 2023).

[13] Townsend, J., Bird, T. and Barber, D. (2018) Practical lossless compression with latent variables using bits..., OpenReview. Available at: https://openreview.net/forum?id=ryE98iR5tm (Accessed: 28 May 2023).

[14] Deep Render. (n.d.). Bits-Back Coding. Available at: https://deeprender.ai/blog/BitsBackCoding (Accessed: 28 May 2023).

[15] Kingma, F. H., Abbeel, P., Ho, J. (2019). Bit-Swap: Recursive Bits-Back Coding for Lossless Compression with Hierarchical Latent Variables. In: Advances in Neural Information Processing Systems. Available at: https://arxiv.org/pdf/1905.06845.pdf (Accessed: 28 May 2023), p.22.

6 Appendices

6.1 Source code

The source code of the models can be found in this github link:

https://github.com/RubyBit/CDM

In the model directory you can find cdm.py and advanced_cdm.py which represent the building blocks of our original 2 models (MNIST and Cifar10). The cdm.py file contains the original architecture of the MNIST model while the advanced_cdm.py contain the model for Cifar10 (to use the models you have to go look into the training notebooks as some more high level classes are defined there, while also using components from the lossless model). In the lossless directory you can find the the utils.py file containing the building blocks for the BB-ANS module (specific implementation details can be found at [13]), lossless_cdm.py containing the model used as a hierarchical latent model and compress.py, a file showcasing the compression.