POLITECNICO DI TORINO

Master's Degree in Artificial Intelligence and Data Analytics

Master's Degree Thesis

Color-Conditioned Abstract Image Generation with Diffusion Models

Supervisors

Prof. Tatiana TOMMASI

Prof. Giuseppe RIZZO

Ing. Angelica URBANELLI

Ing. Luca BARCO

Candidate

Christian BARDELLA

07 2023

Abstract

This thesis aims to investigate the promising Diffusion Models (DMs) technology by developing a prototype that meets the requirements for variability and quality of generated images.
The ultimate goal is to initiate the technology transition by exploring DMs in conjunction with the well-established StyleGAN. Through this work, I specifically examine the behavior of DMs in the "Color to Image" task and their ability to generate images based on color label conditioning with the final goal of producing 512x512 resolution images.

I adopt a step-by-step approach to gain a thorough understanding of this new technology, both practically and theoretically. Understanding how I can effectively condition a diffusion model to enable precise control over the generative process was an essential step. I implement a basic Diffusion-network, which uses a shallow vanilla U-net to grasp the functioning of the various components of the model and I successfully train this network on the "Letters font dataset"[1], focusing on conditional and unconditional generation at a resolution of 32x32.

The problem with this method is that the entire network works on a pixel level, meaning that the diffusion process is applied to the whole input image. For high-resolution image production, this approach immediately becomes impractical.

The Latent Diffusion Model by CompVis was a suitable solution, which applies the diffusion process to a reduced input representation. This model has demonstrated remarkable results in conditional image generation and has now been made open source. It comprises 400 million parameters two-stage architecture: an encoder-decoder network and a Diffusion network. Using a pre-trained encoder, I trained the diffusion network on a customized version of the Wiki-Art dataset. Still, the time and resources were insufficient for a complete state-of-the-art comparable training. This first working prototype is capable of producing well-conditioned images at a resolution of 256x256, showing that DM beat the previous StyleGAN in matching the required color and how, with more extended training, more time, and computational resources, I could achieve comparable performance in terms of FID.

Given these resource constraints, I have also adapted the Latent Diffusion code to run on a multi-GPU environment with limited resources exploiting a fully-sharded-data-parallel strategy.

Overall, this thesis offers a comprehensive exploration of diffusion technology, encompassing its mathematical foundations and relevant literature background. It

[1]External Link to the Kaggle Dataset

effectively highlights the strengths and limitations of this approach in the label-to-image task.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 The Era of Generative Model

In the last two years, we saw a significant rise of generative models, which show their incredible capabilities in understanding the meaning of a given context to generate ever more consistent and beautiful content such as images, entire videos, music but also 3D meshes, animations, and more. They gained immense popularity due to their remarkable versatility in effectively learning data distributions and capturing the underlying structure to efficiently extrapolate meaningful information and correlations to recombine it in an ever-seen manner. Their impact is not limited to the technological field but extended to the social sphere. For instance, notable examples like Chat-GPT, renowned for its language modeling capabilities, and MidJourney, a groundbreaking text-to-image generator, have sparked discussions on the relevance of certain human tasks and raised concerns about the potential risks posed by highly intelligent artificial systems in the modern society.

Access to the proper hardware is a challenge that often limits the research possibilities. Indeed the faster we go, the more we can experiment, the more memory we have, the bigger the models we will be able to train, and the larger the amount of data we can fit our model on. Every year GPUs and TPUs become faster and with more memory, while simultaneously, models become more and more memory-consuming. Nowadays, the big A.I. tech companies have started scaling out their models, discovering how, in general, the model's size can benefit their performance and precision. Indeed deeper models seem to understand better the underlying structures of the data giving them the ability to generalize well on specific tasks.

With generative models, we refer to an entire sub-field of Artificial intelligence in which neural networks, once trained on a dataset to satisfy a specific task, become capable of generating never seen data or, it would be better to say, they become capable of generating data that has never been seen during their training phase.

Intuitively, they are models able to generate entirely new synthetic data. This generation capability can also be helpful for other machine learning tasks, for example, data augmentation but also, if adequately trained, for transforming the style or domain of the data.

1.2 Context and Structure of proposed Work

I have to point out what was already being done on this project and explain the role of my thesis in the frame of this work.

This project is based on an existing operative pipeline built on a StyleGan v3. The purpose of this introductory section aims to enhance comprehension of how my work contributes positively and identify areas that require further research.

It is essential to provide an introduction to the general structure of the existing pipeline.

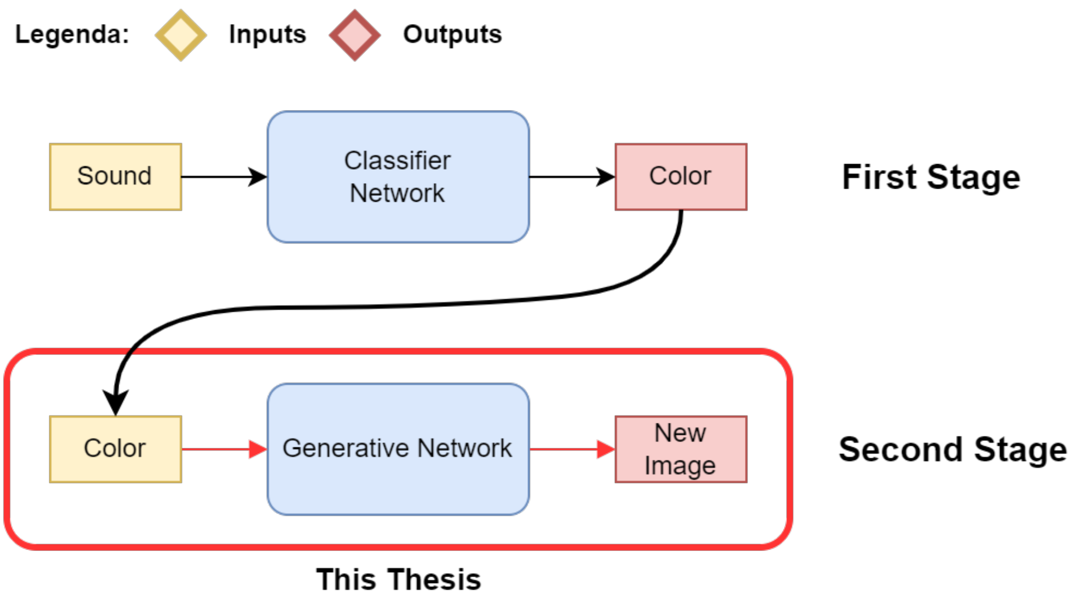


Figure 1.1: Complete Pipeline General Scheme.

The actual pipeline is composed of two main steps:

1. Given a Sound, it outputs a Color-label (Classifier);
2. Given a Color-label, it outputs the new Image (StyleGan).

In the first stage, a classifier is trained to predict a color given a sound as input. The primary focus of my work centers around the second stage, as shown in figure 1.1, of this pipeline, which, as previously mentioned, involves initiating the technological transition from StyleGAN to Diffusion Models.

During the course of this thesis, the field of generative A.I. has witnessed exponential advancement. While this growth has provided abundant learning resources and ample experimental examples, achieving competitive levels of precision and accuracy of the generated sample remains challenging due to several factors, including limited computational resources, lack of adequate comprehension of some concepts, and time constraints.

In the following chapter, I explain the mathematical theory behind generative models and all the other required building blocks, aiming to gain a comprehensive understanding of the statistical concepts behind the variational inference approximation that lead to the diffusion process's development and how it can be reversed.

The literature background chapter briefly overviews the previous generative models, serving two primary purposes. Firstly, understanding the limitations and capabilities of alternative technologies assists us in making decisions about which method to employ based on specific requirements. Secondly, recognizing the contributions of previous models is crucial as they shape the development and practical implementation of new models.

Subsequently, in the Materials and Methods 4 chapter, I explore the key components and intuitions that make the Latent Diffusion (LDM) approach preferable and more feasible than pixel-level Diffusion. I present different experiments and architectures employed during the prototyping phase. I also focus on the problems arising from hardware constraints and some solutions I use to solve them. I discuss some multi-GPU algorithms to spread a very weighty model on a node of N GPU with limited VRAM.

In the Results chapter 5, I thoroughly analyze the performance of state-of-the-art class conditional latent Diffusion applied to the Wiki-art dataset against the StyleGan v3, demonstrating whether these architectures can enhance the quality and diversity of image generation. I provide results and insights on where this technology outperforms its predecessors and where it does not. I also show how the diffusion model properly understands the conditioning on color-label, exposing it as the primary color on the generated images.

In chapter 6, I propose several potential improvements for the developed project and explore alternative approaches that can be explored by leveraging existing implementations. Lastly, I briefly introduce ideas inspired by state-of-the-art research in the Variational-Auto-Encoder (VAE) field on which I focus.

Chapter 2

Mathematical Foundations

In this section, we introduce and summarize some key mathematical concepts, which are the necessary building blocks to understand better how models work. It is crucial to remember where the math came from and how it is derived to remain focused and understand the design choices behind the new models.

2.1 Bayes Rule

Bayes Rule is a fundamental key concept in everything that concerns statistical analysis. It is a counter-intuitive concept that needs time to be understood and fully appreciated.

It describes the relationship between an event conditioned to the probability of another event.

Given X, Z representing two Random Variables:

$$P_\theta(Z|X) = \frac{P_\gamma(X|Z) \cdot P_\sigma(Z)}{P_\beta(X)} \tag{2.1}$$

Where:

- $P_\theta(Z|X)$ is the "Posterior Probability" (Conditioned probability of Z given X).

- $P_\gamma(X|Z)$ is the "Likelihood" (Conditioned probability of X given Z).

- $P_\sigma(Z)$ is the "Prior" (Marginal Probability of the event Z ignoring all the conditioning).

- $P_\beta(X)$ is the "Evidence" or "Marginal Probability" or "Normalization Term."

2.2 Expected Value

The expected value of a random variable X represents the average value obtained from its possible outcomes, where each outcome is weighted by its corresponding probability of occurrence. It is a weighted average defined by the formula in the discrete and continuous form, respectively:

$$E[X] = \sum_{i=1}^{\infty} x_i \cdot p_\theta(x_i) \qquad E[X] = \int_{-\infty}^{+\infty} x \cdot p_\theta(x)\, dx \tag{2.2}$$

We can easily extend this form to functions of Random Variables (R.V.). For example, defining $h(X)$ a function of R.V. we can rewrite the equations as:

$$E[h(X)] = \sum_{i=1}^{\infty} h(x_i) \cdot p_\theta(x_i) \qquad E[h(X)] = \int_{-\infty}^{+\infty} h(x) \cdot p_\theta(x)\, dx \tag{2.3}$$

2.3 Kullback-Leibler Divergence

Kullback-Leibler Divergence is a useful measure that tells us how two distributions are similar to each other and, in general, is defined with the equation below:

$$D_{KL}(q\|p) = E_x\left[\log\left(\frac{p_\theta(x)}{q_\phi(x)}\right)\right] = \int \log\left[\frac{q_\phi(x)}{p_\theta(x)}\right] q_\phi(x)\, dx \tag{2.4}$$

2.3.1 Derivation

Given an R.V. $X = \{x_1, x_2, ..., x_n\}$ and two distributions of our R.V. $p_\theta(X)$ and $q_\phi(X)$, we want to know how similarly they model the distribution over X. In this case, we can say that we want to take a realization of $x = x_1$ and know how well our two distributions $p(.)_\theta$ and $q_\phi(.)$ perform (where θ and ϕ are the parameters of the distribution).

We can easily compute the difference between the value of the two distributions plugging in a realization of X:

$$p_\theta(x_1) - q_\phi(x_1)$$

This term will be zero if the two values are equal. Now we can generalize it to all the realizations of X:

$$p_\theta(x_i) - q_\phi(x_i)$$

and then take the log values of the term in order to avoid numerical instability due to the fact that often these numbers are very close to zero:

$$\log(p_\theta(x_i)) - \log(q_\phi(x_i))$$

And for the logarithm properties, we can rewrite it as:

$$\log\left(\frac{p_\theta(x_i)}{q_\phi(x_i)}\right) = h(X)$$

We are interested in the average difference between all the realizations for our two distributions. Taking as reference what was said before about expected value, we can see the log-likelihood ratio as a function of Random Variables, and always from the definition of expected value 2.2 in the continuous form, we can write:

$$E_x[h(X)] = \int \log\left(\frac{p_\theta(x_i)}{q_\phi(x_i)}\right) p_\theta(x_i)\, dx \tag{2.5}$$

It is important to notice the fact that it is called divergence and not distance because it is not a symmetric measure. For a **metric** to be considered as such, it must satisfy certain properties:

1. **Non-negativity:** for any pair of points x, y the metric $d(x, y) \geq 0$;
2. **Identity:** for any point x the metric $d(x, x) = 0$;
3. **Symmetry:** for any pair of points x, y the metric $d(x, y) = d(y, x)$;
4. **Triangle inequality:** for any triplet of points x, y, z the metric should satisfy the inequality $d(x, z) \leq d(x, y) + d(y, z)$.

The KL divergence does not satisfy the symmetric properties. Is indeed true the following:

$$D_{KL}(p\|q) \neq D_{KL}(q\|p) \tag{2.6}$$

We have two distinct possibilities for computing the KL divergence from the equation above.
There is also another important property of the KL-Divergence:

$$D_{KL}(p\|q) \geq 0 \tag{2.7}$$

It is essential to bear in mind the above properties 2.7 since it will be very useful as we proceed to the following sections.

2.4 Variational Inference and Evidence Lower Bound (ELBO)

2.4.1 Variational Inference

Variational Inference is an important concept when we talk about probabilistic inference tasks, which are at the base of VAE, GAN, D.M., and other probabilistic models. It is an optimization procedure whose purpose is to approximate an unknown, complex distribution through another parameterized known one.

Let's define a random variable (R.V.) denoted as X, representing our observed data. Additionally, we introduce another R.V. named Z, which represents some other data characteristics but, in this case, their realizations remain unobserved (or latent; in fact, we refer to Z as a latent variable).

In our case is convenient to re-think this concept using a data-based example. We can think of X as the R.V. that represents all the data and $p(X)$ as the distribution which models his behavior, while Z is an unknown R.V. on which our known data depends. For example we can say that $X = \{x_1, x_2, ..., x_n\}$ represent any image and x_i represent a specific realization of X while Z represent important features.

We do not have a tractable posterior term $p_\theta(Z|X)$ because, as we can see from the 2.1, the divisor term, which is called **Evidence**, does not have a close form. We can write the evidence term as the marginalization integral:

$$p_\theta(X) = \int_Z p_\theta(X, Z)\, dZ \ = \int_Z p_\theta(X|Z) \cdot p_\theta(Z)\ dZ \tag{2.8}$$

This term is completely intractable for high dimensional latent spaces Z and in everyday use cases is always unmanageable.

Summarizing we want to model our posterior, which has an intractable evidence term, in order to make predictions. We will estimate the posterior probability introducing a new distribution q which will serve to approximate our "true" posterior so that:

$$p_\theta(Z|X) \simeq q_\phi(Z) \tag{2.9}$$

In practice, we want to minimize the differences between two distributions, the unknown and the one we choose. We can do this operation by exploiting the KL divergence measure: So from the equation 2.5:

$$\overset{*}{q}(z) := \text{argmin}_q\ KL(q_\phi(z)\ ||\ p_\theta(z \mid x)) \tag{2.10}$$

$$\begin{aligned} D_{KL}(q_\phi(z) \| p_\theta(z|x)) &= \int_{z_0} ... \int_{z_{d-1}} \log\left(\frac{q_\phi(z)}{p_\theta(z|x)}\right) q_\phi(z)\, dz_0 ... dz_{d-1} \\ &= \int_Z \log\left(\frac{q_\phi(z)}{p_\theta(z|x)}\right) q_\phi(z)\, dZ \end{aligned} \tag{2.11}$$

In this way, we rewrote our intractable problem into another intractable one. In-fact we still do not have access to the posterior $p_\theta(z|x)$ and the marginal $p_\theta(x)$ but we have the joint distribution $p_\theta(z, x)$.
Using the Bayes Rule 2.1 we know that:

$$p_\theta(z|x) = \frac{p_\theta(z, x)}{p_\theta(x)} \tag{2.12}$$

So we can rearrange the formula substituting 2.12 in the 2.11:

$$D_{KL}(q_\phi(z) \| p_\theta(z|x)) = \int_Z \log\left(\frac{q_\phi(z)\, p_\theta(x)}{p_\theta(z, x)}\right) q_\phi(z)\, dZ \tag{2.13}$$

2.4.2 ELBO or Variational Lower Bound Derivation

The **ELBO** is a very useful mathematical result [1]. It gives us a tractable lower bound over an intractable value, enabling Variational inference to train models. Starting from the equation 2.13, we can split the integral into two different integrals. One which has only tractable terms and another one that has intractable ones:

$$\begin{aligned} D_{KL}(q_\phi(z) \| p_\theta(z|x)) &= \int_Z \log\left(\frac{q_\phi(z)}{p_\theta(z, x)}\right) q_\phi(z)\, dZ + \int_Z \log\left(p_\theta(x)\right) q_\phi(z)\, dZ \\ &= E_{Z \sim q}\left[\log\left(\frac{q_\phi(z)}{p_\theta(z, x)}\right)\right] + E_Z\left[\log\left(p_\theta(x)\right)\right] \\ &= E_{Z \sim q}\left[\log\left(\frac{q_\phi(z)}{p_\theta(z, x)}\right)\right] + \log\left(p_\theta(x)\right) \end{aligned} \tag{2.14}$$

Since the second term does not depend on Z we can remove the expected value and write only the $\log\left(p_\theta(x)\right)$.
We then call the first term $L(q)$:

$$L(q) = E_{Z \sim q}\left[\log\left(q_\phi(z)\right)\right] - E_{Z \sim q}\left[\log\left(p_\theta(z, x)\right)\right] \tag{2.15}$$

Now we can rewrite everything in the reduced form:

$$D_{KL}(q_\phi(z)\|p_\theta(z|x)) = -L(q) + \log(p_\theta(x)) \tag{2.16}$$

It is useful to point out the following facts that help us to derive the final ELBO:

1. $D_{KL}(q_\phi(z)\|p_\theta(z|x)) \geq 0$ by definition (2.7) and as consequence we can write;
$$-L(q) + \log(p_\theta(x)) \geq 0 \tag{2.17}$$

2. Given the two previous statements, in order to verify the equation, the following must be true:
$$\log(p_\theta(x)) \geq L(q) \tag{2.18}$$

We can derive from the last equation that $L(q)$ **(tractable)** is the lower bound of the evidence term **(intractable)** and from this the name: **Evidence Lower Bound**.
So the **ELBO** is defined as:

$$\textbf{ELBO} := L(q) = E_{Z\sim q}[\log(p_\theta(z,x))] - E_{Z\sim q}[\log(q_\phi(z))] \tag{2.19}$$

From the equation 2.16 we can easily derive:

$$\begin{aligned} D_{KL}(q_\phi(z) \mid\mid p_\theta(z|x)) &= \log(p_\theta(x)) - ELBO \\ \log(p_\theta(x)) &= K \text{ (intractable but constant)} \\ D_{KL}(q_\phi(z) \mid\mid p_\theta(z|x)) &= \cancel{\log(p_\theta(x))} - ELBO \end{aligned} \tag{2.20}$$

We just derived the ELBO from the Variational Inference. Consequently, we can notice that minimizing the KL divergence between these two distributions is the same as maximizing the Evidence Lower Bound.

$$\overset{*}{q}(z) := \operatorname{argmax}_q ELBO_q = \operatorname{argmin}_q D_{KL}(q_\phi(z) \mid\mid p_\theta(z|x)) \tag{2.21}$$

Chapter 3

Literature Background

This section is focused on explaining some important building blocks that lead to the development of Diffusion Models. My purpose is to give a general understanding of everything that is needed summarizing all the concepts I have encountered during the investigation phase of the Thesis.

3.1 Auto-encoder (AE)

The idea of the Variational Auto-encoder came from the previous concept of Auto-Encoder. An Auto-encoder maps the data we give as input in points in a latent low-dimensional space finding the right mapping transformation. The model is trained using a simple reconstruction loss:

$$\text{Loss}(L^2) = ||x - \hat{x}||^2 = ||x - d(z)||^2$$

Where $\hat{x}$ is the reconstructed data and $d(z)$ is the decoder transformation applied on the latent variable z.

If we want to generate new data using this approach, we encounter some problems. This model's main issue relies on how it shapes the latent space. Indeed, when we sample the latent space in a zone that has not been modeled, the network output will be completely non-sense. To permit the sample, a latent space must satisfy two main properties:

1. **Continuity**: two close points in the latent space should give similar outputs;
2. **Completeness**: any point in the space should give meaningful outputs.

We can say that the Auto-Encoder does not respect both of these properties because it is not trained to have any structure in how it organizes the latent space.

Researchers aim to solve this problem by introducing a probabilistic approach to the Auto-Encoders: the Variational Auto-Encoders.

3.2 Variational Auto-encoder (VAE)

In contrast to what was done before, a variational Auto-Encoder is trained to model the latent space distribution over the data, postulating that the data we are trying to learn depends on a multidimensional latent (unknown) R.V. Z, which represents some data characteristics such as orientation, position, etc.

As we have already seen in the Variational Inference derivation, we want to model an unknown/intractable distribution $p_\theta(z|x)$ by employing a parameterized known distribution selected from a specific family. In this case, a Gaussian distribution was used.
The Objective can be derived by minimizing the K.L. divergence between the approximating $q_\phi(z|x)$ distribution and the true one $p_\theta(z|x)$.

$$D_{KL}(q_\phi(z|x) \| p_\theta(z|x)) = \int q_\phi(z|x) \log \left(\frac{q_\phi(z|x)}{p_\theta(z|x)} \right) dz$$

Following the same procedure used for the ELBO derivation, exposed in the Mathematical foundation section 2.4 we can write:

$$\log p_\theta(x) \geq \overbrace{\underbrace{E_{Z\sim q}\left[\log\left(p_\theta(x|z)\right)\right]}_{\text{Reconstruction Term}} - \underbrace{D_{KL}(q_\phi(z|x) \| p_\theta(z))}_{\text{Normalization Term}}}^{\text{ELBO, } L(x,\theta,\phi)}$$

$$\overset{*}{\theta},\ \overset{*}{\phi} = argmax_{\theta,\phi}\ L(x,\theta,\phi) \tag{3.1}$$

The K.L. divergence term is minimized and tells us how much the true prior $p_\theta(z)$ and the approximating posterior $q_\phi(z|x)$ differ from each other. The expected value of the likelihood is maximized and tells us how well the data is reconstructed.

We can also see the loss's K.L. divergence term as a regularization term for the modeled latent space, ensuring that the model will learn a well-structured continuous and complete (in section [3.1]) latent space. Thanks to that, if we want to sample an ever-seen point in the distribution, the generation does not produce meaningless data but can blend through the latent space modeled characteristics. In order to generate new data from a VAE, we can randomly sample a set of parameters θ, ϕ from the latent space to use then the trained decoder network which will produce new data.

3.2.1 VAE general Architecture

The Variational Auto-encoder is often implemented using two deep described in the paper [2], symmetrical neural networks structure composed of a convolutional encoder and a convolutional decoder. We have an encoder Network that works by modeling the posterior distribution $q_\phi(z|x)$ while the decoder network is trained to reconstruct the original data modeling the likelihood distribution $p_\theta(x|z)$.

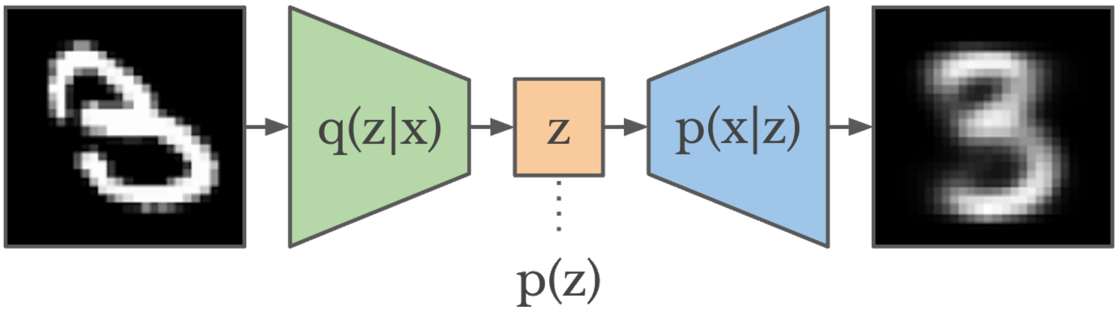


Figure 3.1: Vanilla Convolutional VAE architecture. Image from [3]

Since its discovery, numerous variations of this approach have been explored to enhance its performance on reconstruction capabilities. A VAE can also be considered as a data compressor or a feature reduction method that encodes input values into a numerical representation with lower dimensionality compared to the original data. This is achieved through the convolutional nature of the down-sampling structure, which extracts the most crucial features contributing to the image composition while disregarding irrelevant information. By doing so, the decoder can faithfully reconstruct the images. Furthermore, it is frequently utilized as an intermediate step for preprocessing data into a low-dimensional form, thereby enabling resource savings in more computationally intensive stages of complex pipelines.

For example, we can encode a 512x512x3 image in a low dimensional matrix (64x64x4) and then use this new compressed data to shrink down the information to the most important one and train other networks to preserve computational power.

Posterior Collapse Problem

One of the main problems while training a VAE is called posterior collapse [4] [5]. This occurs when the learned variational distribution becomes very similar to the prior distribution. As a result, the generative model loses its ability to utilize all the information in the latent dimensions. The latents are ignored when they are paired with a mighty auto-regressive decoder.

Blur Effect on generated Images

Another commonly encountered issue is the generation of images with a blurred effect on the reconstructed data. This problem arises due to the pixel-wise or element-wise reconstruction loss which promotes smoother and more averaged images rather than sharp and realistic ones. Furthermore, VAEs rely on the assumption that both the latent variables and the data conform to Gaussian distributions, which might fail to capture the true complexity and diversity present in the data, resulting in a loss of information during the encoding and decoding stages.
One way to address this issue can be done by using a "Perceptual Loss" instead of using a L^2 reconstruction loss.

3.2.2 Perceptual Loss

The underlying concept behind Perceptual Loss is easily comprehensible. Employing an element-wise reconstruction loss might fall short of capturing the true essence of image dissimilarity. In fact, it is possible for two entirely distinct images to exhibit similar pixel values, leading them to be erroneously perceived as similar, despite their significant dissimilarities.

As shown in 3.2, instead of using a straightforward L^2 Loss, which focuses on **pixel level** computing the element-wise difference between pixels, we use another network trained to extract high-level features of images, to then compute the L^2 loss on the **features level**. Usually, the VGG16 model is used. This process evaluates the difference between two images focusing on their general structure favoring the perceptual similarity (from here the name "Perceptual loss") over their pixel value similarity.

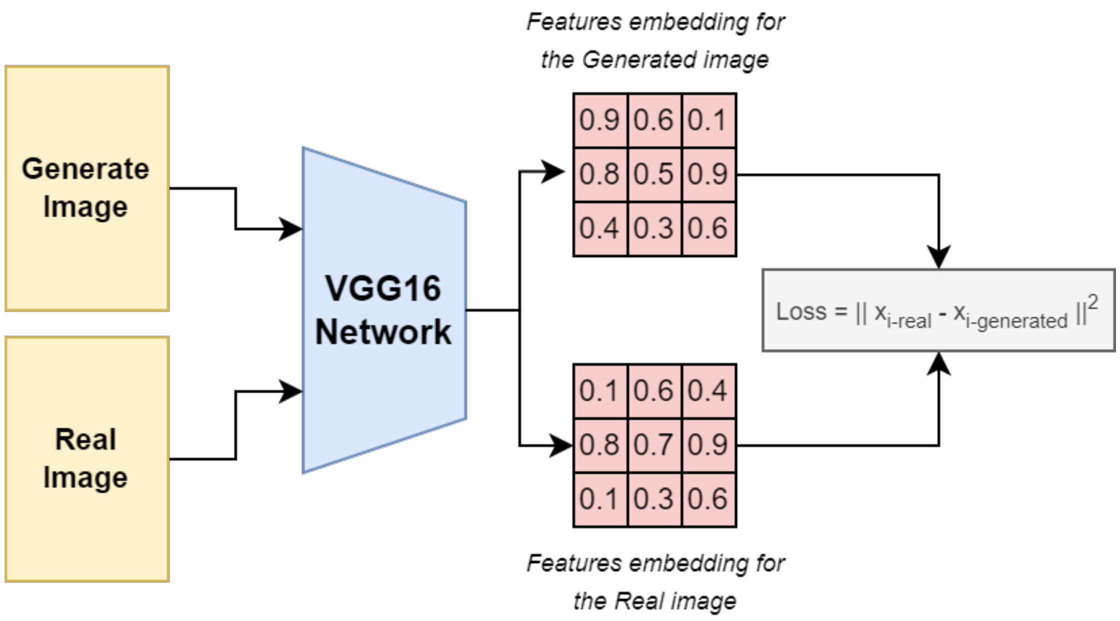

Features embedding for the Generated image
Generate Image
Real Image
VGG16 Network
0.9 0.6 0.1
0.8 0.5 0.9
0.4 0.3 0.6
Loss = || $x_{i\text{-real}}$ - $x_{i\text{-generated}}$ ||^2
0.1 0.6 0.4
0.8 0.7 0.9
0.1 0.3 0.6
Features embedding for the Real image

Figure 3.2: High-level functioning scheme of Perceptual Loss.

3.3 Generative Adversarial Networks (GAN)

In the landscape of the generative models, we can't avoid mentioning the Generative Adversarial Networks, otherwise called GANs. They remain the undefeated state-of-the-art generative model in terms of the quality of the synthetically generated data. Today they have widely used thanks to their generation process's extreme efficiency and velocity. In fact, they need only one pass through the network to produce new samples, which is in contrast with the DM technology analyzed in this thesis.

We have a Generator network that needs to improve at producing even more realistic images and a Discriminator network that has to improve its discrimination capabilities in trying to choose if a generated image is from the dataset (Real) or generated by the network (Fake). The two networks are jointly optimized using a min-max algorithm where, while a network **minimizes** its objective, the other network, in parallel, **maximizes** its loss. The GAN loss is composed of two main terms:

$$\text{Objective}_{GAN}(G, D) = E_{x\sim p(x)}\left[\log D(x)\right] + E_{z\sim p(z)}\left[\log(1 - D(G(x)))\right]$$
$$\text{Loss}_{GAN} = \underbrace{\min}_{G}\ \underbrace{\max}_{D}\ \text{Objective}_{GAN}(G, D) \tag{3.2}$$

Where:

- $D(x)$ is the Discriminator Output for the real Data taken from the dataset;
- $D(G(x))$ is the Discriminator Output when it is given synthetically generated data from our generator.

In a nutshell, the Generator has to fool the Discriminator.
Once trained, we can discard the discriminator remaining with the trained convolutional generator network only. The beauty of this model lies in its conceptual simplicity.

3.3.1 GANs Instability Problems

Usually, GANs suffer from instability during training, resulting in the model never converging. Moreover, the model can also collapse its representation capabilities, reducing the produced outputs' variability. These problems are usually challenging to control and prevent. Indeed GANs need a lot of tuning and time. This lack of controllability puts the need to find new generative models that can offer more control both during training and inference time.

3.4 Vector Quantized VAE (VQ-VAE): modeling a discrete latent space

The need to introduce the GAN and the VAE has the final purpose of giving the correct intuition and knowledge behind the next concept: VQ-VAE. The VQ-VAE introduced by Van Den Oord et al.[6], and further improved by the "Generating Diverse High-Fidelity Images with VQ-VAE-2" [7], is a method to learn a discrete latent space instead of a continuous one. According to the authors, this would help mitigate the posterior collapse problem, which, as previously explained, is one of the main issues of VAEs. Structurally is very similar to a vanilla VAE, where we have a symmetrical convolutional structure with an encoder and a decoder network but with an important difference.

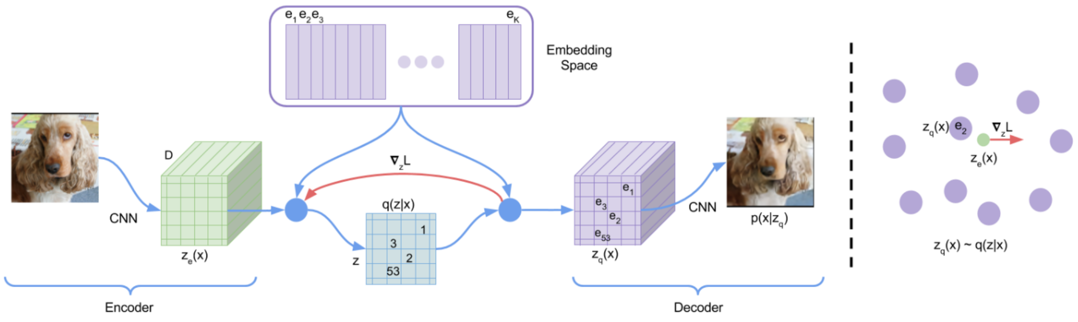


Figure 3.3: Scheme of VQ-VAE from the paper [6]

After the encoder forward step, the continuous latent space is transformed through a discretization step. This new representation is then given to the decoder network, which will learn how to decode the data starting from a discrete one.
One of the main issues is that since we are working with discrete space, during the backward pass, we have to back-propagate a piece of information that is not differentiable.

The Code-Book

To make this happen, we need to describe the quantization procedure and illustrate how learning a usable representation from data can be possible.

Firstly, we must define what a **Discrete Embedding Space** is. We can see this space as an ensemble of K vectors of dimension D which compose the Discrete latent space R of dimensionality KxD. This space is commonly referred to as the **Code-Book** in literature, characterized by a dimension of R^{KxD}.
The quantization algorithm can be described as follows:

1. Subdivide the continuous latent space into continuous sub-vectors;
2. For each continuous sub-vector find the nearest using $L^2 distance$ discrete vector in the Code-book;
3. Substitute the continuous sub-vector with the corresponding discrete vector in the Code-book

We have to denote that we can choose to use multiple discrete Code-Book vectors to represent each encoder output. This choice expands the range of possible combinations of discrete vectors available to the model, resulting in an increased capacity to encode a greater amount of information within the Code-Book maintaining a relatively small dimension for the Code-Book. Intuitively, the biggest the Code-Book, the more information can store.

Formally:

$$\begin{aligned} z_q(x) &= \text{argmin}\|z_e(x) - e_j\| \\ z_q(x) &\sim q_\phi(z|x) \end{aligned} \tag{3.3}$$

Where:

- $z_e(x)$ is the output of the encoder.
- $z_q(x)$ is the output of the quantizer.

- e_j is the j^{th} vector and $j \in R = \{0, 1, ..., R-1\}$. Each vector has dimension D;

In the long run $z_q(x)$ will be a discrete approximation of the continuous $q_\phi(z|x)$ which is the encoder approximating posterior.

The objective function should take into account the fact that the discretization operation is not differentiable, so the researcher decided to compute the Gradient of the objective

$$\text{Loss} = \underbrace{\log p(x|\ z_q(x))}_{\text{Reconstruction Loss}} + \underbrace{\|sg[z_e(x)] - e\|_2^2}_{\text{Code-book Loss}} + \underbrace{\beta\ \|z_e(x) - sg[e]\ \|_2^2}_{\text{Commitment Loss}} \tag{3.4}$$

The *sg* term stand for **Stop gradient** and mean that the gradient, on that term, will not be computed.
The decoder optimizes the reconstruction loss term only, the encoder optimizes the reconstruction and the Commitment loss terms, while the embeddings are instead optimized by the code-book loss term.

In section 3.2 of the paper [6] we can read:

> "The gradient of the loss $\nabla Loss$ will be passed unaltered to the encoder. Since ... the gradients contain useful information for how the encoder has to change its output to lower the reconstruction loss."

Below I reported a snippet of code that contains the actual implementation of the Code-book loss from the Quantizer class to demonstrate and visualize how to implement the Stop Gradient terms of the loss equation.

The code-book loss is used to move the embeddings toward the encoder outputs. The commitment loss is important **not to make** the loss diverge by encouraging the encoder to "commit" to a specific vector in the code book rather than mapping it to an arbitrary location. By penalizing deviations from the chosen code-book vectors, the commitment loss pushes the encoder to learn embeddings that are closer to the selected code-book vectors.

3.4.1 VQ-GAN

This model was well studied and analyzed in the paper "Taming Transformer" by Patrick Esser et al. [8]. VQ-GANs structure is very similar to the VQ-VAE. The two main differences between these two models are the addition of a patch-based discriminator and a perceptual loss. Obviously, the model, such as in the vanilla GAN case, is trained in an adversarial way.

```
 1  import torch, torch.nn as nn
 2
 3  class Quantizer(nn.Module):
 4      def __init__(self, n_e, e_dim, ...):
 5          ...
 6          # Define the codeBook space
 7          # n_e = K and e_dim = D
 8          self.embedding = nn.Embedding(self.n_e, self.e_dim)
 9          ...
10
11      def forward(self, z, ...):
12          z = rearrange(z, 'b c h w -> b h w c').contiguous()
13          z_flattened = z.view(-1, self.e_dim)
14          ...
15          # d = Distances from z to embeddings e_j
16          #   = (z - e)^2 = z^2 + e^2 - 2 e * z
17          min_encoding_indices = torch.argmin(d, dim=1)
18          z_q = self.embedding(min_encoding_indices).view(z.shape)
19
20          # Compute loss for embedding
21          loss = self.beta * \
22                 # Commitment Loss
23                 torch.mean((z_q.detach()-z)**2) +\
24                 # Code-book Loss
25                 torch.mean((z_q - z.detach()) ** 2)
26          ...
```

Figure 3.4: Codebook Loss computation. From line 20 in the code, we can read the actual implementation of the correspondent formula 3.4

First of all, they substitute the reconstruction loss term $\log p(x|\ z_q(x))$ with a perceptual loss which allows the model to not only learn pixel-level information but also helps focus on an image structure level increasing the sharpen and decreasing the blurriness of the generated images.

The Loss for the VQ-GAN is indeed composed of two main parts:

1. The already presented VQ-VAE loss 3.4;

2. The GAN loss is defined as:

$$\text{Loss}_d = \underbrace{\log D(x) + \log(1 - D(\hat{x}))}_{\text{Discriminator Loss}}$$

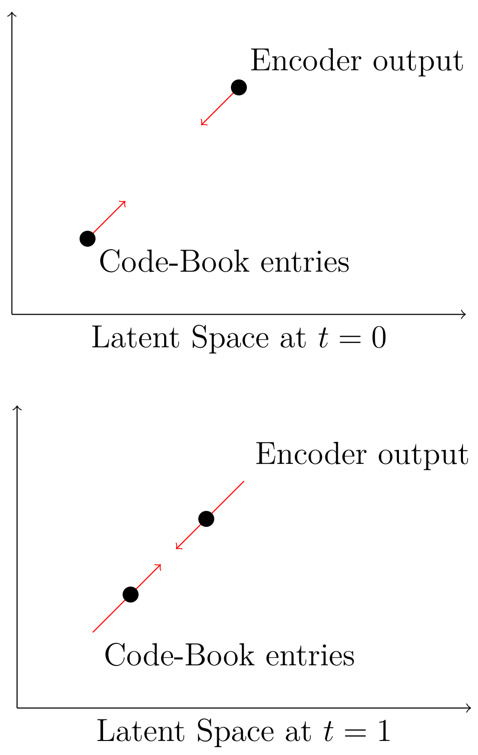


Figure 3.5: The commitment loss has the effect of moving the encoder outputs towards an embedding

Total VQ-GAN total Loss is the sum of the two:

$$\text{Loss}_{vqGan} = \text{Loss}_{vqVae} + \ \lambda \ \cdot \text{Loss}_d \tag{3.5}$$

Where λ is a weight on the Discriminator loss computed on each iteration- It is called "adaptive weight" and serves as a trade-off between the two losses.

3.5 Denoising Diffusion Probabilistic Models (DDPM)

Denoising Diffusion Probabilistic Models are the new state-of-the-art model in terms of output quality. They are now the current running model at the base of very famous architectures such as Stable-Diffusion, by Stability AI, Dall-E 2 by OpenAI, and MidJourney. DDPM shows a groundbreaking capability in text-to-image tasks because they are very good at modeling the latent space where to sample from,

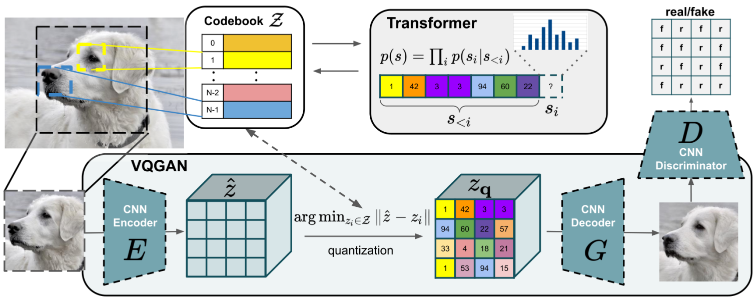


Figure 3.6: VQ-GAN structure presented by the paper [8].

given a large amount of data. A diffusion model takes inspiration from the physics concept of diffusion. It can be seen as a Markov chain of events where we go, step by step, from one distribution to another. In a Markov chain, each event at time-step t depends only on the event at time-step $t-1$.

3.5.1 Forward Process

We want to corrupt our initial data with Gaussian noise to have pure isotropic Gaussian noise at the end of the procedure. [9] [10] [11] Isotropic Gaussian noise is a Gaussian that has the Covariance matrix in the form: $\Sigma = \sigma^2\mathcal{I}$. So at each time step, we add Gaussian noise to the distribution creating a new distribution from which we will sample from the next time-step.

Given an R.V. X, representing our input data, we call $q(X)$ the distribution that models our data.
We denote also x_t a sample from the distribution $q(x_t)$ such as:

$$x_0 \sim q(x_0) \text{ and in general } x_t \sim q(x_t)$$

where t is the i^{th} time-step in the interval $[0, T]$.

Here we can note that the actual sampling operation depends on the previous time step through the previous distribution state.

If we say that $q(.)$ is on the family of the Gaussian distribution, we can write:

$$q(x) \sim \mathcal{N}(x; \mu, \Sigma) \tag{3.6}$$

We define the μ and β depending from the current time-step:

$$\begin{aligned} \mu_t &= \sqrt{1-\beta_t x_t} \\ \Sigma_t &= \beta_t I \end{aligned} \tag{3.7}$$

The variance β lies in the interval [0,1] and, by coupling it with the identity matrix, we can extend the variance on a multi-dimensional model which has the same variance on each feature. By so, the Covariance Matrix is defined as $\Sigma = \beta I$. Varying the variance we can control the amount of noise we add at each time step. In this way, we can schedule the quantity of noise we add at each time step.

So given the Markov chain of events, we can define the sampling operation as follows:

$$\text{Sampling: } x_t \sim q(x_t|x_{t-1})$$

Let's connect the above consideration in one equation. We can write:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t;\ \sqrt{1-\beta_t}x_{t-1},\ \beta_t I) \tag{3.8}$$

The above equation represents one step of the Diffusion Process where we modify the data distribution by injecting at each time-step Gaussian Noise.

3.5.2 Reparametrization Trick in Diffusion Models

We can exploit the fact that a sampling in an R.V. modeled by a Gaussian can be written as a function of a deterministic term and a stochastic one that can not be learned. To obtain a sample at a random time-step t, we can generally write:

$$\begin{aligned} x &= \mu + \Sigma \cdot \xi \\ x_t &= \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t} \cdot \xi \\ &= \sqrt{(1-\beta_t)(1-\beta_{t-1})}x_{t-2} + \sqrt{\beta_t \beta_{t-1}} \cdot \xi \\ &= ... \\ &= \sqrt{(1-\beta_t)...(1-\beta_1)}x_0 + \sqrt{\beta_t...\beta_1} \cdot \xi \end{aligned}$$

We can do a smart variable renaming calling:

$$\begin{aligned} \alpha_t &= 1 - \beta_t \\ \bar{\alpha_t} &= \prod_{s=0}^{t} \alpha_s \end{aligned}$$

The term $\bar{\alpha_t}$ is the cumulative product from 0 to t. Denoting that ξ does not depend from time ($\xi = \xi_t = \xi_{t-1} = ... = \xi_0$) we can elegantly rewrite everything as:

$$x_t = \sqrt{\bar{\alpha_t}}x_0 + \sqrt{1-\bar{\alpha_t}} \cdot \xi$$

And we can derive the sampling equation in the new form:

$$q(x_t|x_0) = \mathcal{N}(x_t;\ \sqrt{\bar{\alpha}_t}x_0,\ (1-\bar{\alpha}_t)I) \tag{3.9}$$

The advantages of this reparametrization are that we can pre-compute all the $\bar{\alpha}$ terms and, by doing so, we can sample the distribution in whatever time-step we want, enabling training the network using random time-step as input. The other important characteristic is that the sample at time t always depends on the initial data x_0 and not on the sample at the previous time-step x_{t-1}.

The 3.9 formula can be summarized in one equation which takes into account every time-steps and this final equation is called: trajectory.
The name trajectory derives from the fact it gives us a step-by-step view of the way the distribution changes over time into another distribution.

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{3.10}$$

It can also be visualized as trajectories of the values of the random variable X because, through time, it draws a path, a specific route, of R.V. realizations that characterize the final value of that X

There is a problem though that if we want our distribution $q(x_t)$ at a random time-step in this format we are obliged to start from time-step 0 until we reach our chosen t. This operation is time-consuming and computationally expensive. Moreover, the linearity in the training can not allow us to generalize well on the time dimension during the training phase.

The solution is to re-parameterize the forward process in a way that will depend only on the initial data x_0 and the current time step t.

3.5.3 Backward Process

With the forward process, we learned a way to create a close-form trajectory equation that samples into a data distribution that changes over time into pure Gaussian distribution. But how we can reverse this process and how we can make the network learn from it?

We have seen how $q(x_t|x_{t-1})$ is the sampling procedure in the forward process. We can think of it as sampling from the posterior distribution defined by the following Bayes Rule:

$$q(x_t|x_{t-1}) = \frac{q(x_{t-1}|x_t)q(x_t)}{q(x_{t-1})}$$

So we can write the equation to extract the reverse process posterior.

$$q(x_{t-1}|x_t) = \frac{q(x_t|x_{t-1})q(x_{t-1})}{q(x_t)} \tag{3.11}$$

The problem that arises from that equation is that we do not have a close-form for the posterior of the reverse process and for the normalization term as they would require to know the distribution over our data input, which we do not have. In particular:

- $q(x_{t-1}|x_t)$ is our need reverse process posterior;
- $q(x_t|x_{t-1})$ this term is defined by us as a Gaussian trajectory from $t = [0, T]$;
- $q(x_{t-1})$ and $q(x_t)$ should be the distribution that models our data at time step t. The problem is that it $q(x_t)$ which depends by construction on our x_0 which is the initial data and we cannot have it at time-step=T.

We need a closed form of 3.11 equation to optimize and make the optimization process available. Is indeed proved that if we condition the model directly on the initial sample at time-step $t = 0$ (x_0) we can obtain a close form for the reverse diffusion Process. We can write the forward equation dependent from x_t and from x_0:

$$\begin{aligned} q(x_t|x_{t-1}) &= q(x_t|x_{t-1}, x_0) \\ q(x_t|x_{t-1}, x_0) &= \frac{q(x_{t-1}|x_t, x_0)q(x_t|x_0)}{q(x_{t-1}|x_0)} \end{aligned}$$

And then, using the Bayes rule, extract the reverse posterior:

$$\begin{aligned} q(x_t|x_{t-1}, x_0) &= q(x_t|x_{t-1}) \\ q(\mathbf{x}_{t-1}|x_t, x_0) &= \frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)} \\ &= q(x_t|x_{t-1})\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)} \end{aligned} \tag{3.12}$$

In this form, the 3.11 is now tractable since we know all the terms:

1. $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is the forward posteriors 3.8;
2. $q(\mathbf{x}_{t-1}|\mathbf{x}_0)$ and $q(\mathbf{x}_t|\mathbf{x}_0)$ as we have already defined them in 3.9.

It can also be seen as the forward posterior probability scaled by a factor $\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}$.

I recall some important equations:

1. $\mathcal{N}(x, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})}$
2. $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \xi$

3. $q(x_t|x_{t-1}) = \mathcal{N}(x_t;\ \sqrt{1-\beta_t}x_{t-1},\ \beta_t I) = \frac{1}{\sqrt{2\pi\beta_t}} e^{-\frac{1}{2}\left(\frac{x_t-\sqrt{1-\beta_t}x_{t_1}}{\sqrt{\beta_t}}\right)^2}$

4. $q(x_t|x_0) = \mathcal{N}(x_t;\ \sqrt{\bar{\alpha}_t}x_0,\ (1-\bar{\alpha}_t)I) = \frac{1}{\sqrt{2\pi(1-\bar{\alpha}_t)}} e^{-\frac{1}{2}\left(\frac{x_t-\sqrt{\bar{\alpha}_t}x_0}{\sqrt{1-\bar{\alpha}_t}}\right)^2}$

Substituting everything in 3.12 and simplifying we obtain:

$$\begin{aligned}
\tilde{\beta}_t &= \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \\
\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 \\
q(x_{t-1}|x_t, x_0) &= \mathcal{N}(x_{t-1}; \tilde{\boldsymbol{\mu}}(x_t, x_0), \tilde{\beta}_t I)
\end{aligned}$$

Unfortunately, one problem still remains: we lack access to x_0 at any time-step t (where x_0 represents the data without being corrupted by any amount of noise).

To understand better why this is an issue, let's consider this question from a different perspective. Our intention is to generate a new data sample by starting with pure Gaussian noise. However, from the formula, we see that this process necessitates the availability of the data that we want to generate from time-step $t = T$. Drawing an analogy to the task of image synthesis, we encounter the paradoxical requirement of having the target image itself as a prerequisite for its generation, which is simply not logical.

Nevertheless, it still makes sense. By employing this inverse trajectory in conjunction with the target image, it becomes possible to faithfully reconstruct the target from a state of pure Gaussian noise, provided that the process is conditioned upon the target. Intuitively, this involves the selection of the optimal denoising trajectory among countless possibilities, as we possess knowledge regarding the precise extent and location for noise removal (i.e., where to proceed along the trajectory) at each time step.

From here, the necessity to approximate the backward process with a new distribution which in the literature is called $p(x)$.

We define the approximated backward process using a Normal distribution as done for the forward. The formulation for the reverse process as a function of p is:

$$p(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \boldsymbol{\mu}_\theta(x_t, t), \boldsymbol{\Sigma}_\theta(x_t, t))$$

Or in trajectory form:

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

3.5.4 Training Objective Function

Now we have the mathematical basis behind the forward and backward concept, which leads to the making of the diffusion process. In the next step, we need to formalize how we can obtain and optimize a loss function in order to fit a Neural Network on our data. Then we will use the NN to the backward posterior and generate new images starting from pure Gaussian noise.

We start with the two trajectories equation in mind. As in the VAE objective derivation, we can minimize the KL divergence between our true distribution and another distribution that tries to approximate it, particularly since we are dealing with trajectories, the intractable backward trajectory, and our approximating backward trajectory.

$$D_{\mathrm{KL}}(q(\mathbf{x}_{1:T}|\mathbf{x}_0)\|p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)) = \int \log\left(\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right) q(\mathbf{x}_{1:T}|\mathbf{x}_0) d\mathbf{x}_{1:T} \tag{3.13}$$

Derivation:

$$\begin{aligned}
&= \int \log\left(\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right) q(\mathbf{x}_{1:T}|\mathbf{x}_0)\, d\mathbf{x}_{1:T} \\
&= \int \log\left(\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)p_\theta(\mathbf{x}_0)}{p_\theta(\mathbf{x}_{1:T},\, \mathbf{x}_0)}\right) q(\mathbf{x}_{1:T}|\mathbf{x}_0)\, d\mathbf{x}_{1:T} \\
&= \int \left[\log\left(\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{1:T},\, \mathbf{x}_0)}\right) + \log p_\theta(\mathbf{x}_0)\right] q(\mathbf{x}_{1:T}|\mathbf{x}_0)\, d\mathbf{x}_{1:T} \quad ; p_\theta(\mathbf{x}_{1:T},\, \mathbf{x}_0) = p_\theta(\mathbf{x}_{0:T}) \\
&= \log p_\theta(\mathbf{x}_0) + \int \log\left(\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right) q(\mathbf{x}_{1:T}|\mathbf{x}_0)\, d\mathbf{x}_{1:T} \\
&= \log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \qquad ; \log p_\theta(\mathbf{x}_0) - ELBO
\end{aligned}$$

If we recall the variational Inference equation 2.20 we can use the same considerations made previously together with the ELBO notion to derive the objective function.
We know that the KL-Divergence must be ≥ 0 so we can finally write:

$$\begin{aligned}
\log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \geq 0 \\
\log p_\theta(\mathbf{x}_0) \geq -\mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]
\end{aligned} \tag{3.14}$$

We have again found an optimizable objective using the negative log-likelihood as a lower bound. Now we need an analytically computable form for the above equations. I skip the passages and I'll go straight to the point here:

$$\begin{aligned}\log p_\theta(\mathbf{x}_0) \geq & \,\mathbb{E}_q[\underbrace{\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}] \\ & - \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T}] \\ & - \sum_{t=2}^{T} \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}}]\end{aligned} \tag{3.15}$$

We want to have an analytical form for the term L_t. After replacing and simplifying, we obtain:

$$L_t = \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right]$$

After all these steps, we are finally able to define this simple analytical equation which is the L^2 distance between the predicted noise and the true one. It is differentiable so we can compute the gradient and optimize a Neural Network over it. Neural networks are mathematical models very good at approximating complex functions. We exploit their high approximation capabilities to fit the backward trajectory through data.

3.5.5 U-Net Architecture

It is essential to understand the math behind the Diffusion model, but learning more about how to implement it is equally important. It is important to discuss what kind of neural network we should use to predict the amount of noise to be removed. The most used one is the U-Net model structure which showed his high capability on image segmentation tasks as shown in the original paper of "U-Net: Convolutional Networks for Biomedical Image Segmentation" by Olaf Ronneberger et al.[12].

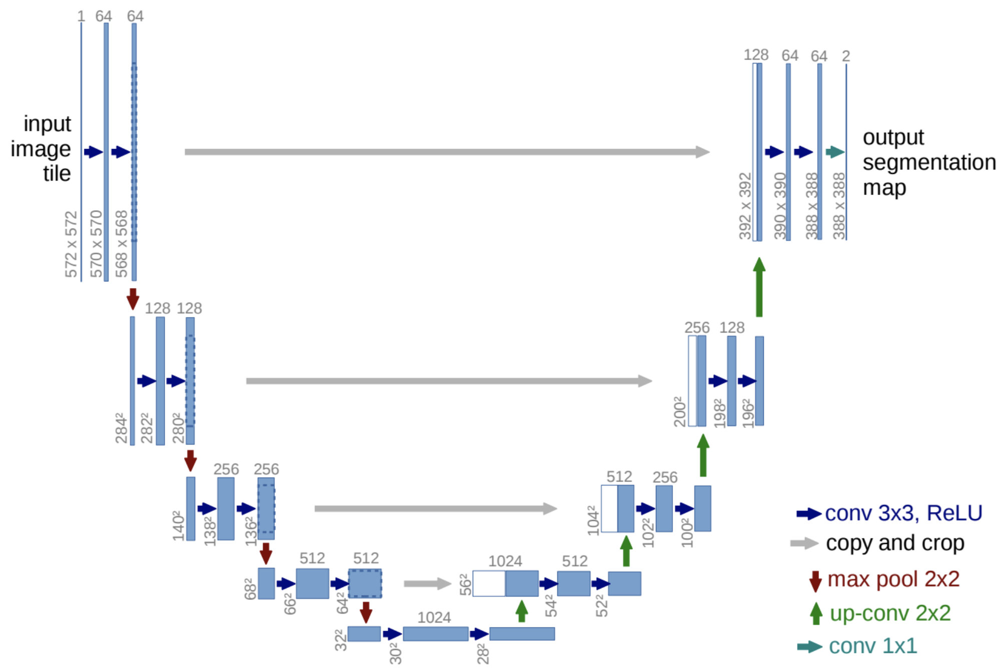


Figure 3.7: Vanilla U-Net architecture from the "U-Net: Convolutional Networks for Biomedical Image Segmentation" [12] paper. Each downsample step doubles the size of channels and halves the feature's dimension.

Its structure is very similar to the VAE, i.e., that has a symmetrical structure composed of a down-samples (encoder) and a up-samples (decoder) network having at the bottom a bottleneck convolution. It is implemented with some addition, such as the residual connection between down-sample and up-sample layers of the same level to help the gradient flow during the backpropagation. This network characteristic helps to avoid or mitigate the gradient vanishing problem. Briefly, the **vanishing gradient problem** happens when the gradient of the loss collapses and becomes closer to zero step by step, preventing the model from learning. In the context of Diffusion Models, they are implemented with self-attention modules, some layers which are useful for the network to give more importance to some discriminating features.

As explained in the paper [13], improvement can also be made in the way we decide to schedule the operation of adding noise to the network.

3.6 Latent Diffusion Models (LDM)

The major problem with DM is that they are very demanding to train due to their linear pipe-lined structure over T time-steps. The sampling procedure from the trained model takes a lot of time compared with the previous SOTA method, such as GANs.

Let's consider this term: to generate a new image, we need to go through T denoising steps, which involve calling the forward function of our model T times. Subsequently, we obtain an image with progressively reduced noise compared to the previous iteration.

To generate our final, completely denoised image, we have to generate a total of T intermediate images, which is very computationally expensive. Moreover, if we also think that the computation required scale in a quadratic way to the resolution of the generated image, we can easily understand how the problem scale out very quickly, leaving us with few optimization options.

3.6.1 Combining Latent Representation with DDPM

The researcher's idea was to take advantage of the excellent compression capability of the Auto-Encoder to reduce the initial size of our data to a smaller and light one and then apply the diffusion process to the reduced data.

As already seen in the previous section, when discussing VAE, we can encode a data source in a low-dimensional version of itself in a way that preserves its most important features. This gives us a representation that holds the very useful characteristic of the original data leaving out all the high frequency and noisy features, which are well-known to be hard to be appropriately modeled.

The LDM approach is indeed composed of two independently trained stages:

1. An Encoder-Decoder network which can be any VAE VQ-VAE, VQ-GAN, KL-VAE, etc..: whose main purpose is to reduce the dimensionality of the input;

2. The Diffusion process: which is implemented using a U-Net applied on the low dimensional feature latent space produced by the Encoder.

Once the model is trained, we can exclude the encoder component. We initialize the denoising U-net with low-dimensional pure Gaussian noise and iterate through the denoising process T times. The output of the denoising U-net is then passed to the Decoder Network, which aims to reconstruct the original-sized image.

Despite reducing the input size significantly, the diffusion operation remains computationally and time-intensive. This challenge becomes particularly critical during the evaluation phase. Evaluating a generative model requires a substantial number of samples to accurately estimate the similarity between the generated and real data distributions, and this process can be time-consuming.

Chapter 4

Materials and Methods

After the previous general treatment of the necessary concept, we start speaking about the work that has been done to implement a first working prototype. Here I present all the details of the used dataset together with the implementation behind the different experiments that led to the development of the final model. The Experiment section is indeed organized into two main parts where in the first, I present the results of how I used a simple model to understand the functioning at the core of DM, and then go on to explain the very final implementation and all the resources problem we encountered to make it work.

4.1 Resources

Available Hardware

To conduct the experiments, I had access to two primary hardware resources. Firstly, I utilized a Google Colab Pro account, which provided access to an **NVIDIA A100 GPU** with 40GB of memory. This powerful resource allowed me to carry out extensive computations.
Additionally, Links Foundation granted me access to a Cluster equipped with **NVIDIA 4 GTX 1080Ti GPUs**, each with 12GB of memory. While primarily used for preliminary testing, this resource was crucial in preparing for the final training phase, which demanded more memory.

Used Software

The code was developed exclusively in PyTorch, serving as the core library for the project. To ensure compatibility, I used PyTorch Lightning as a development framework. This Machine Learning Tool offers a convenient wrapper for PyTorch, simplifying the management of the model during the training and evaluation phases.

It also automatically implements and handles some useful multi-GPU strategies making development much easier.
For dataset preprocessing, I leveraged Albumentations, a powerful library that outperforms TorchVision in terms of speed.

4.2 Dataset

During the experimenting phase, I used two main datasets. During the initial experiments I used the "Letters Font" dataset as it can give more distinguishable results than an abstract image to then, in the second part, make use of my benchmark dataset which is a custom Wiki-Art.

4.2.1 Letters Font Dataset

With my initial simple model, I have used the "Letters font"[1] dataset, which is composed of 390k grey-scale images subdivided in 26 folders (A-Z) of rendered grayscale alphabet characters using over 14900 fonts, each with a size of 32×32 pixels. The motivation behind this choice is that, at first, I want data to have two main characteristics: data gas to be highly recognizable, and the resolution of the input must be low in order to permit fast experimentation without worrying about memory resources during this phase. Considering a grayscale dataset simplifies all the training because an image can be represented with only one channel of values in the range of [0, 255]. This can potentially significantly reduce memory requirements and speed up the training. In opposition to that, I decided anyway to treat the images as a 3-channel RGB grayscale value to observe how well the model behaves in a dataset setting that overall is similar to the final one.

4.2.2 Wiki-Art Dataset

The second phase of the experiment is characterized by trying to scale out the model with a complete, fully functional DM and training it on the benchmark dataset to accomplish our final target of comparing our results with the StyleGAN ones. This dataset is indeed the one on which we train the StyleGAN. The dataset I have is a custom version of the Wiki-Art dataset. It is composed of 19k abstract painting images with a resolution of 512x512 already cropped in the center to maintain the spacial details coherent and not introduce distortions. Some images were discarded because they may portray only one plain color (a completely Blue

[1]External Link to the Kaggle Dataset

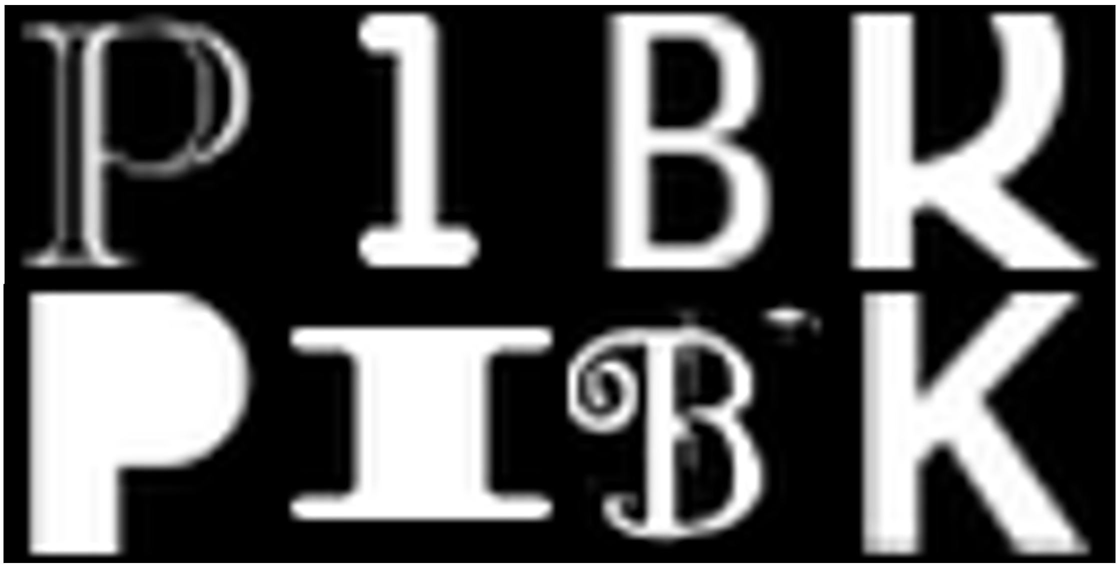


Figure 4.1: Images of Letters font dataset

image, for example), and that kind of sample is not well representative of the dataset with the risk of introducing instability in the training phase.

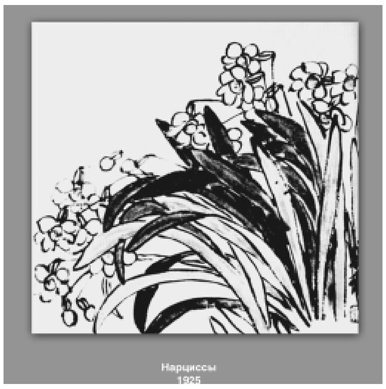
The images are subdivided into 8 different color categories as shown in Table 4.1.

Label	Color	N of samples	Label	Color	N of samples
0	Orange	2586	4	Yellow	142
1	Green	2493	5	Red	647
2	Blue	388	6	Black	3531
3	Purple	4098	7	White	5889

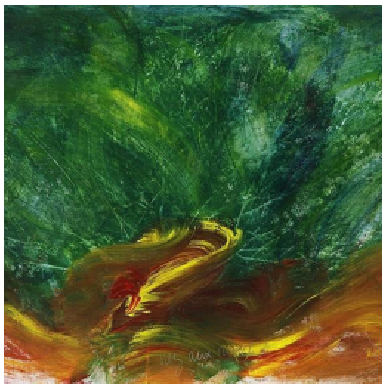
Table 4.1: The table reports how many images are in each category together with the corresponding discrete label.

Wiki-Art Dataset Issues

The dataset we use suffers from an un-balancing problem. Indeed the distribution of the images inside the respective classes is, as shown by 4.3, not uniform. It is important to highlight this issue because the way we organize data highly influences the behavior and the performance of the model training. As shown by Figure 4.2, another important problem is that not all the images in the specific category perfectly match the color they belong to. That leads to the generation phase to obtain images based on a totally wrong color.



(a) It should be labeled as White or Black



(b) It should be labeled as Green

Figure 4.2: Both images are labeled as Purple but clearly, they are wrongly matched.

The subdivision in classes is made by taking an image, computing its average color, and then taking the Euclidean distance of that value from all the color categories in the RGB space. The class is assigned to take the color category with the small distance value. In other words, we assign an image to its nearest color category.

The problem with the RGB color space is that spatially near colors are often not perceptually similar. It is true that the contrast between colors plays an important role in the way we perceive them in an image. For example, darker colors are accentuated by lighter colors and other factors.
I find a suitable solution to this problem by considering computing the color category of an image using another color space which can also consider the way humans perceive colors.
The **CIE-LAB** color space is perfect to solve this problem. It considers 3 different parameters from the RGB: the perceptual lightness L, A, and B stand for the four unique hues of human vision. (Unique hue is defined as a color that an observer perceives as pure, without any admixture of the other colors).
A trivial attempt I made was to re-cluster the dataset using CIE-LAB color space. I have applied the following algorithm:

1. Read the image RGB values and remap them to CIE-LAB color space;
2. Compute the mean on each color axis to obtain the average color of the image;
3. Compute the distance from the average color of the image and all the label colors.

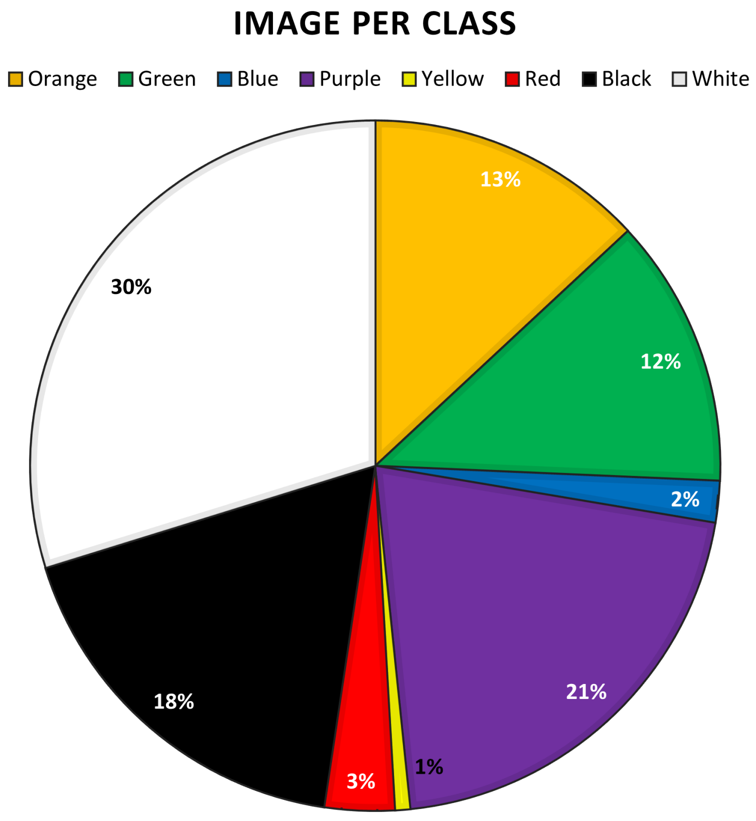


Figure 4.3: The chart shows how the images are subdivided in the corresponding color category color by percentage. It is clear how highly unbalanced the dataset is, where 30% of the images are under the white label, but only 1% of the images are in the yellow one.

4. Take the Color labels that have the minimum distance from our average color.

After this re-clustering, I noticed how the re-distribution of images has deeply changed but the number of samples in each category remains highly unbalanced.

4.2.3 Pre-Processing

All images from all datasets are pre-processed using the Albumentations Python library. In the pre-processing pipeline, the images are first cropped and then resized to match the base resolution of the experiment. The cropping step is performed prior to resizing to ensure the preservation of the spatial distance of the details in the images, avoiding any deformation. Introducing additional deformations during the process could lead to the model learning from distorted images, which may

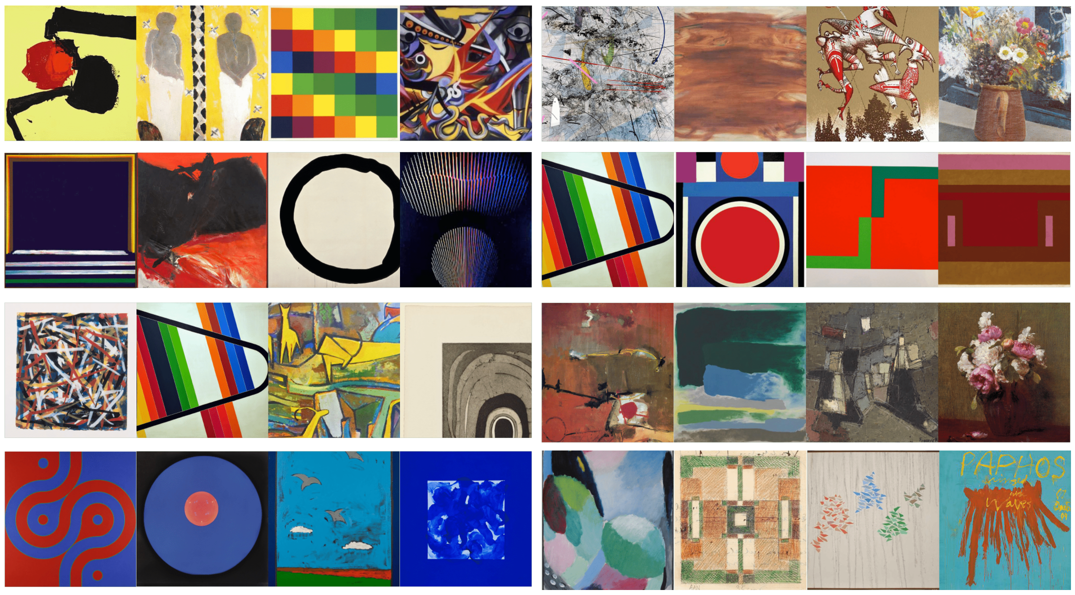


Figure 4.4: Images of Custom Wiki-Art Dataset. The figure shows four images per color label. From left to right, top to bottom, we have Yellow, Orange, Black, Red, White, Green, Blue, and Purple.

result in producing distorted outputs.

Secondly, I re-scale the range of values from RGB integer between $[0, 255]$ to a float range that goes $[-1, 1]$, which is a more suitable range for the networks to work with.

4.3 Experiments Decription

4.3.1 Conditioning the Network

From the start of this project, our goal has been to provide immediate answers to two key aspects: how to condition the DM to a label and how to tailor its behavior to align with our desired tasks.

Out in the wild, there are a lot of existing implementation of DM pre-trained pipeline but everyone focus their attention on text-to-image or other similar text-based tasks because, at the moment, the research convolves around this topic of interest. To have an idea, a text embedding is a Numerical representation of the variable length of some piece of information to make it computer readable. In the case of text embeddings, the embedding captures the meaning of a word but also, in the case of a sentence, encapsulates the meaning of a word with respect to the other ones providing a good amount of context to the model, which will be guided through them. Because of the Diffusion of text-to-image tasks, a very important amount of the literature, examples, and pre-trained models are DM-guided using the information provided by a text label.

Instead, very few explicit materials teach how to condition the diffusion model on a simple discrete label properly.

In this work, I experimented with two principal methods to condition the U-net:

- Concatenating the label-embedding immediately after the positional encoding;
- Pass the label-embedding directly through the cross-attention mechanism

First, I present some initial experiments based on the concatenation method. In the second phase of experiments, I then expose the advantages of using a cross-attention mechanism.

4.3.2 Use positional Encoding for time conditioning the network

Previously we explained how the DM is Time-dependent in the sense that at any time T, I have to remove a certain amount of noise from the data. As a consequence, the model needs some kind of mechanism capable of properly supplying the time information. This will make the model aware of what the time is in order to decide how much noise we have to subtract from the data. The way we insert this information inside the model training is very clever and it was introduced by the "Attention Is All You Need" [14] paper, previously used to encapsulate the positional meaning for the transformer structure. This is needed because the transformers architecture takes a bunch of embeddings altogether, which is where they differ from the LSTM models.

The idea is to use the same approach used to encapsulate positional meaning for the time dependency, which is, in a sense, a kind of positional information.
The procedure involves using sine and cosine functions of increasing frequency so that, by sampling them at different positions, we can ensure near embeddings maintain similar values, also for high frequency, and tend to differ when very far apart.

$$\mathrm{PE}_{(pos,\ 2i+1)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$
$$\mathrm{PE}_{(pos,\ 2i)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

The above formulas are used to compute positional embedding, where:

- *pos* is the position of the embeddings in the series;

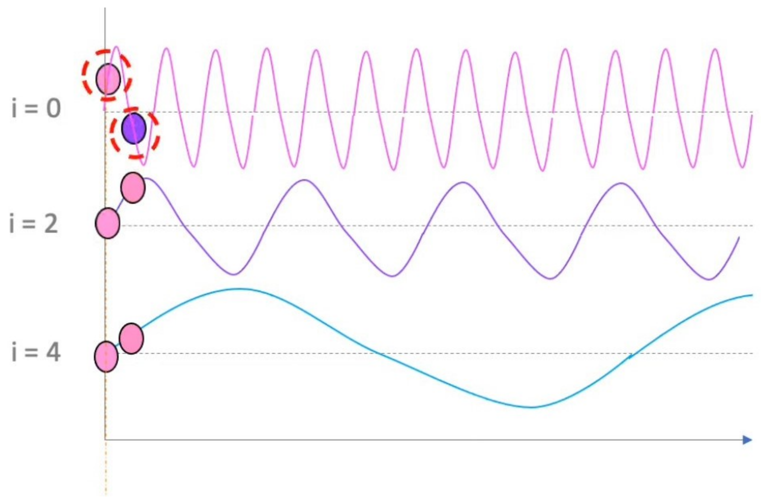


Figure 4.5: Time dependency in positional embeddings. Temporally near embeddings will have similar values because spatially near sampling in the sine/cosine function results in small changes in both low and high frequencies.

- i is the i^{th} value of the single embedding, which also represents the frequency of the sine function we are considering;

- d is the dimension of the embedding. Taking the transformer case as an example, this dimension is taken as the same size as the single-word embeddings. **In fact, it is shown that it is enough to sum together the positional embedding and the word embedding to condition the network properly**. The values start to repeat when the sequence of embeddings length is greater than 10001.

In the example presented in figure 4.5, we can see how two positional embeddings that lie spatially near samples the sinusoidal functions in points that are similar to each other. The fact that we use the sine and cosine functions is because they are periodic, limited in range, and the differences between adjacent values in the positional embeddings follow a smooth pattern. In the case of sine/cosine positional embeddings, the values are symmetrically distributed around zero, meaning that the positive and negative values alternate. This symmetry helps capture the relative positions of tokens effectively. Also, the differences between values decrease or diminish as the distance between the positions of the tokens increases. This property is desirable because it allows the model to capture the notion of sequential order and relative distances between tokens in a meaningful way.

However, other functions with similar properties can be used as substitutes.

```
    UNET_CLASS(nn.Module):

        def timestep_embedding(self, time, dim):
            half_dim = dim // 2
            embeddings = math.log(10000) / (half_dim - 1)
            embeddings = torch.exp(half_dim * -embeddings)
            embeddings = time[:, None] * embeddings[None, :]
            embeddings = torch.cat((embeddings.sin(), embeddings.cos
()))
            return embeddings

        def __init__(self, num_classes, t_emb_dim):
            ...
            self.label_emb = nn.Embedding(num_classes, t_emb_dim)
            self.time_embed = timestep_embedding(timesteps)
            ...

        def forward(x, timesteps, label):
            t_emb = self.time_embed(timesteps)
            # Embedding the time to the position by adding
            # together their values
            emb = self.label_emb(label) + t_emb
            ...


```

Figure 4.6: Pseudocode that shows how time embeddings are added together with the label embedding.

4.4 Control the Diffusion Process

4.4.1 Guide the Diffusion

The Diffusion can be guided, to some degree, using another piece of data, which helps guide the final data generation. With the word "Guide," we mean that we give the network additional direction information to steer the sampling trajectory toward a distribution section where it is more probable to find more consistent results with the provided label. In other terms, we can see this operation as increasing the probability of sampling in a zone of the modeled distribution that better represents our label.

This operation can be done mainly in two ways:

1. Guide the Diffusion during the generation phase using the gradient of an

additional pre-trained Classifier;

2. Guide the Diffusion, using the model itself during the sampling, training it using an additional label that we will call "**Zero-label**.".

4.4.2 Classifier-Guided Diffusion

This method relies on another, separately trained classifier defined as $f(y|x,t)$. This classifier is a discriminative model which directly estimates the posterior probability $f(y|x)$. We can train it to predict a label given a noisy image, at time-step t, as input. The guidance is performed by summing the gradient log posterior of the classifier to the mean of the Gaussian in the backward process during the sampling phase.

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \boldsymbol{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) + s \cdot \nabla \log f(y|x_t) \tag{4.1}$$

Where s is a guidance scale and ∇ is the gradient of the classifier posterior.
This method has a drawback: we heavily rely on an external classifier, but by doing it, we introduce a dependency of our DM on the classifier's performance.

4.4.3 Classifier-Free Diffusion

This method was introduced and described in the [15, Classifier-Free Diffusion Guidance]. We can guide the diffusion sampling using the model itself. In order to obtain good results, the model must be properly trained.

The intuition is that we train at the same time a label-conditioned version of the model and an un-conditioned one. This operation can be done using a "Zero-label" and putting some of the train-batch labels to the "Zero-label" label with some probability. The "Zero-label" can be chosen from any number but must exceed our labels' range. For example, given a batch size of ten data samples, each paired with a discrete label in the range $[0,7]$, the discrete label can be "Zero-label"$= 7+1 = 8$, or similarly, we can shift our range to the left by 1 position $[1,8]$ and choose "Zero-label"$= 0$:

$$\text{Labels} = \Big[4,7,2,3,0,3,5,2,2,3\Big]$$

We set "Zero-label" $= 8$ and we change the labels with some probability p.

$$\Big\Downarrow\ p = 20\%$$

$$\text{Labels} \ = \Big[4,\underline{8},2,\underline{8},0,3,\underline{8},2,\underline{8},3\Big]$$

In the paper [15], researchers show results on training fixing the diffusion model architecture but changing the unconditional-training probabilities $p = 0.1, 0.2, 0.5$. In paragraph 4.2, the authors explain that using $p = 0.1$ or $p = 0.2$ produce quite the same results in terms of sampling generation quality. Using $p = 0.5$ is instead shown to worsen the quality. This finding demonstrates that it is **not necessary** to train a lot of the model around the unconditional label. To quote the paper[15]:

> "..., we conclude that only a relatively small portion of the model capacity of the diffusion model needs to be dedicated to the unconditional generation task in order to produce classifier-free guided scores effective for sample quality."

```
    def training_step(self, batch):
        # SET the 20% of the label to the unconditional class using a random strategy
        batch_size = batch["class_label"].shape[0]
        label = batch["class_label"]
        # I set the probability of True boolean mask to 20%
        uncond_boolean_mask = np.random.choice([False, True], size=batch_size, p=[0.8, 0.2])
        label[uncond_boolean_mask] = batch["unconditional_label"][0]


```

Figure 4.7: Snippet of code of "Zero Label" assignment.

4.4.4 Cross Attention Mechanism

The concept of the attention mechanism originates from the seminal paper "Attention Is All You Need" by Vaswani et al. [14], serving as the foundation for the powerful Transformers architecture. This mechanism possesses a unique ability to amplify the importance of regions that are most likely to contain valuable information relevant to the task at hand. Consequently, the model dedicates greater "attention" to these specific regions within the input data, leading to enhanced performance.

The cross-attention mechanism further empowers the model to focus on particular regions of the input based on an external embedding, which can represent various forms of information [16].

The true strength of the cross-attention mechanism lies in its versatility, allowing us to condition the network on any chosen piece of data. This could range from text embeddings and semantic masks to future experiments involving sound embeddings directly, expanding the horizons of potential conditioning for generation tasks.

To effectively condition a deep model, this approach yields superior results compared to the previous method of concatenating the label embedding after the input data. In the case of the Latent Diffusion Unet, the label is seamlessly integrated into the deeper layers using this mechanism.

4.4.5 First experimental settings

In order to understand the model and its building blocks, in the initial phase, I decided to use a simpler model composed of a shallow convolutional U-Net with 4 down and up-sampling layers. It also implements a linear attention ("Linear Attention Mechanism: An Efficient Attention for Semantic Segmentation" [17]) module on each level right before the down-sampling and up-sampling block.

The goal of this phase was to understand if my acquired knowledge and understanding of the theory was correct at that point of the project.

This first iteration works at pixel level on the original input resolution of the images without an encoding phase. The loss function used was a simple reconstruction L^2 loss The training was done using a batch size of 512 images. In order to produce acceptable results in terms of training was

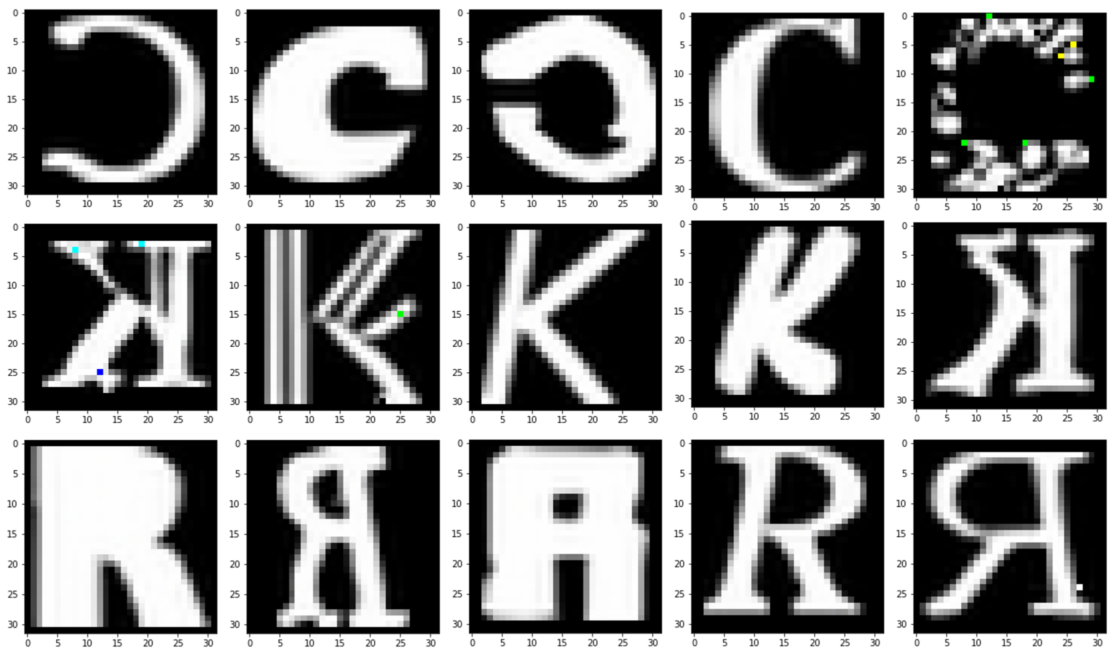


Figure 4.8: Generated Letters. Interestingly, the model is trained using a random Flip transformation on the y-axis and some letters are, in fact, correctly mirrored.

4.4.6 Second experimental settings

In the second phase, the attention is focused on reaching the target goal of producing 512x512 resolution images conditioned to a discrete color label. The code is heavily based on the official repository of Latent Diffusion by CompVis[2] which is the official implementation of the correspondent paper "High-Resolution Image Synthesis with Latent Diffusion Models"[18].

4.4.7 Latent Diffusion

Latent diffusion is an idea that makes use of a pre-diffusion phase where the input dimensionality is reduced by another network which usually is a VAE. This is mainly done for two principal reasons. Reducing the dimensionality decreases the overall size of the network and the computation needed to train the model. The second motivation is that a compression phase summarizes the data to contain only useful and discriminant information ignoring useless data and favoring more concise and meaningful outputs.

Two stage Model

Latent diffusion is structured into a two-phase model where, during the first phase. To properly have an effective diffusion process, we need to have a fully functional first stage. In the original paper[18], they make use of different first-stage approaches considering KL-VAE, VQ-VAE, and VQ-GAN. I decided to use the VQ-GAN because they were built with the purpose of solving some problems that concern the vanilla VAE. There are two more parameters to take into consideration when we spoke about VQ models and they concern the dimensionality of the used code-book. The code-book dimension is a non-trivial hyper-parameter choice and, if not properly chosen, can negatively impact both the performance of the final generation capabilities and the computation required during the training. In this case, we choose a code-book with a dimensionality of 16384x4 which can offer a good amount of encoding possibilities while keeping small the dimension of each code-entries.

Once chosen what model to use for the first stage, we have three options to proceed with:

1. Train the VQ-GAN from scratch on our custom Wiki-Art Dataset;
2. Preform a fine-tuning of a pre-trained model on our Custom Dataset;
3. Use a pre-trained model;

[2]https://github.com/CompVis/latent-diffusion

Train the VQ-GAN from Scratch

This operation may offer the best performance, but reaching a good quality output by training this model from scratch takes an unavailable amount of time and computational resources. I have started training anyway from scratch only as a proof of concept 4.9.

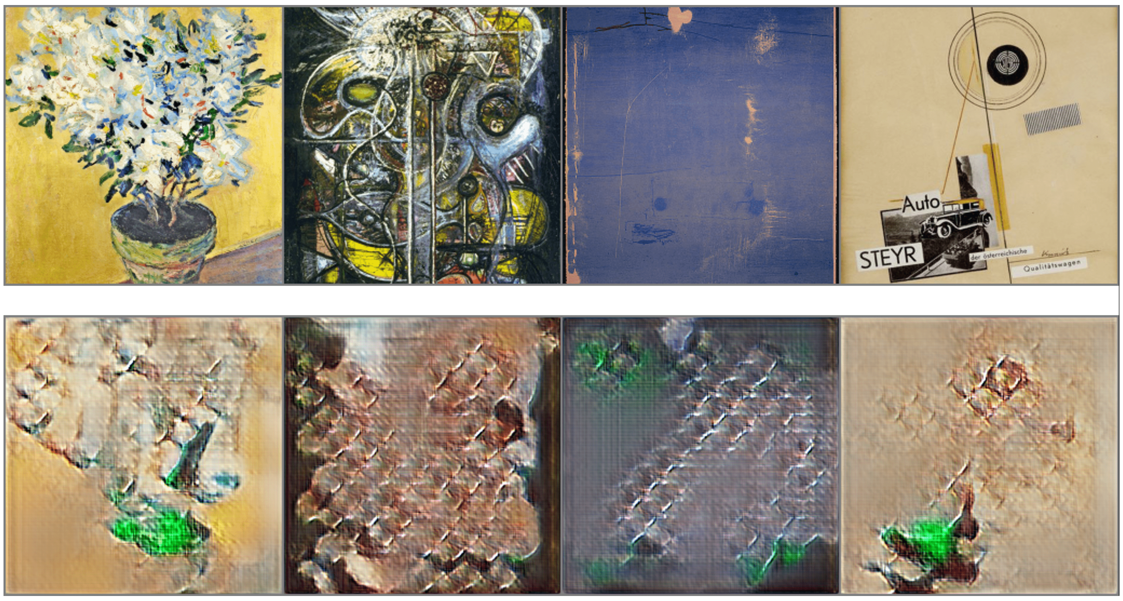


Figure 4.9: Qualitative results of VQ-GAN trained from scratch

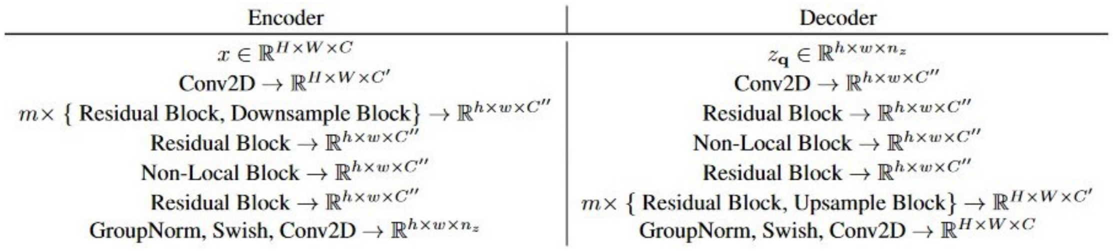
Preform a fine-tuning of a pre-trained model on our Custom Dataset

A good compromise could be taking the model and fine-tuning it on our dataset. This option combines the lack of time with the good performance that we can potentially reach using fine-tuning. In reality, performing the fine-tuning starts to degrade the reconstruction capabilities of the VQ-GAN model and for those problems, I stick with using a pre-trained model, which is available from the repository of "Taming Transformers" always by CompVis [8]. May a longer fine-tuning can lead to better results, but I leave this task for future work.

Use a pre-trained model

I have adopted this solution because unify the quality of reconstruction to our lack of time. From the reconstruction analysis performed on our custom dataset, the pre-trained model reached very good results in terms of FID on our dataset, in opposition to the previous results obtained with the fine-tuning. The other main reason is that VQ-GAN tends to overfit if not properly trained or any an early stop is applied. The Negative Log Likelihood (NLL) is a good indicator of overfitting.

In detail, the pre-trained model is trained on the OpenImages datasets and has a reduction factor of 8. This means that, for example, if the input has a resolution of 256x256, the Diffusion process will receive a 32x32 resolution latent space ($\frac{256}{8} = 32$). The choice of the reduction factor is made taking into account the consideration made in the paper [8] where it is explained that the reduction factor $f = 8$ is the best trade-off in terms of reconstruction precision and NLL. For the VQ-GAN model details, please refers to the official GitHub Repository[3]. The structure of the model is well explained in the official paper "Taming Transformer" [8].



Encoder	Decoder
$x \in \mathbb{R}^{H\times W\times C}$	$z_{\mathbf{q}} \in \mathbb{R}^{h\times w\times n_z}$
Conv2D $\rightarrow \mathbb{R}^{H\times W\times C'}$	Conv2D $\rightarrow \mathbb{R}^{h\times w\times C''}$
$m\times$ { Residual Block, Downsample Block} $\rightarrow \mathbb{R}^{h\times w\times C''}$	Residual Block $\rightarrow \mathbb{R}^{h\times w\times C''}$
Residual Block $\rightarrow \mathbb{R}^{h\times w\times C''}$	Non-Local Block $\rightarrow \mathbb{R}^{h\times w\times C''}$
Non-Local Block $\rightarrow \mathbb{R}^{h\times w\times C''}$	Residual Block $\rightarrow \mathbb{R}^{h\times w\times C''}$
Residual Block $\rightarrow \mathbb{R}^{h\times w\times C''}$	$m\times$ { Residual Block, Upsample Block} $\rightarrow \mathbb{R}^{H\times W\times C'}$
GroupNorm, Swish, Conv2D $\rightarrow \mathbb{R}^{h\times w\times n_z}$	GroupNorm, Swish, Conv2D $\rightarrow \mathbb{R}^{H\times W\times C}$

Figure 4.10: Structure of the used VQ-GAN [8].

As shown by the figure 4.10 this model implements an Encoder-Decoder structure stacked up together as shown in figure 4.10. Moreover, to compute the perceptual loss, utilize a perceptual network implemented using a pre-trained version of the VGG16 model.

4.5 Denoising network

The Second stage of the LDM is composed of the U-Net, which performs the denoising operations. The code for the U-net is taken from the official implementation of OpenAI. The U-net class offers different settings, such as the possibility to choose from a number of heads in the attention mechanism, the context dimensionality, and the number of channels multiplier to create an arbitrarily deep U-net structure.

4.5.1 Model Structure

The model comprises a total of 400 million trainable parameters and 89 million non-trainable parameters, with the VQ-GAN accounting for the latter. The U-Net

[3]https://github.com/CompVis/taming-transformers

architecture consists of 3 down-sample stages and 3 up-sample stages. It takes as input a tensor of dimensionality $BATCH - DIM$x4x32x32, which is the resulting latent space images extracted by the previous VQ-GAN stage. In each stage, the channel size of the input doubles while the resolution halves. Additionally, two residual connections are incorporated at each layer.

It is important to note that conditioning in this model is achieved through the utilization of the Cross attention mechanism, which propagates the label embedding across all stages of the U-Net. The label provided to the model is passed as a 512-dimensional embedding vector known as the context.

To enhance the model's capabilities, an attention block is introduced after each Residual Block, and this behavior can be customized. Furthermore, at each downsample layer, the number of attention heads doubles, enabling the network to extract meaningful information from lower-resolution inputs.

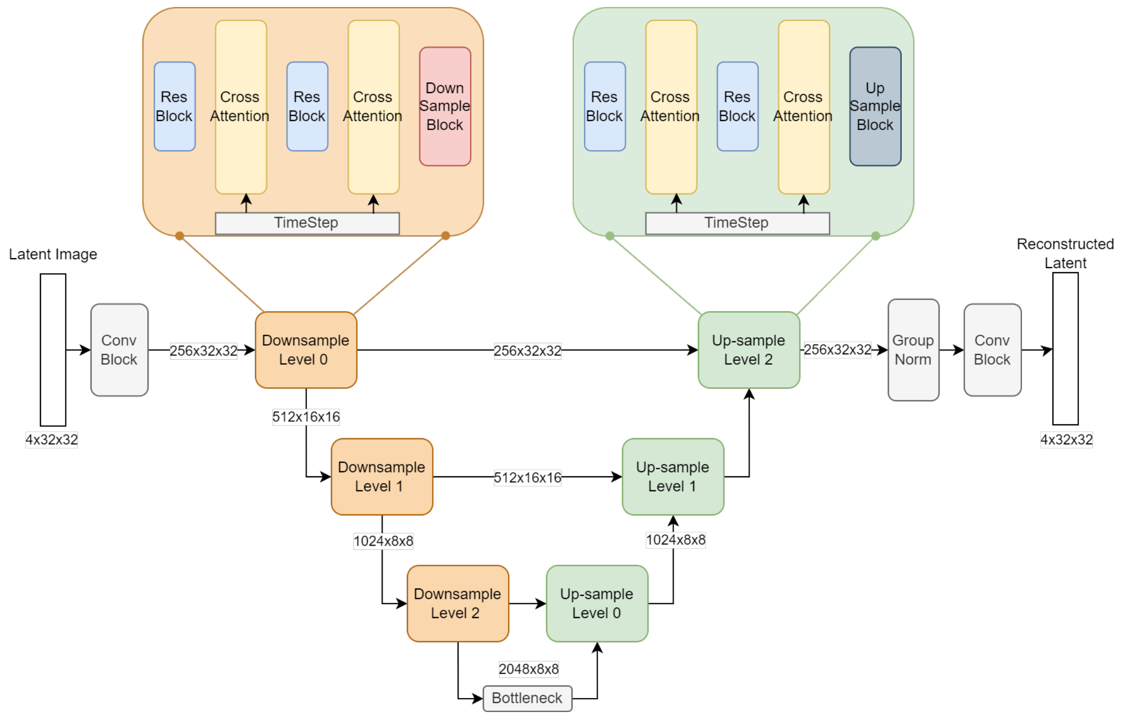


Figure 4.11: Simplified Structure of the used model

4.6 Speed up the sampling

4.6.1 DDIM and PLSM

Generating samples from a Denoising Model (DM) can be time-consuming due to the iterative nature of the denoising process. Training the DM to remove noise from an image using 1000 timesteps necessitates the same number of timesteps to generate a new sample. Consequently, generating a new batch of images can take several seconds. For instance, with our hardware configuration, generating 15 images with 1000 denoising timesteps typically requires around 4 minutes.

Fortunately, various methodologies have been devised to address this issue, aiming to develop more efficient denoisers that effectively eliminate noise. The paper "Denoising Diffusion Implicit Models" by Song et al. [19] presents a method to reduce the number of denoising steps required during the generation process. By utilizing the Denoising Implicit Model (DDIM) sampler, the number of denoising steps can be decreased from 1000 to 200, significantly accelerating the generation process.

Another denoiser, outlined in the paper "Pseudo Numerical Methods for Diffusion Models on Manifolds" by Liu et al. [20] (PLMS), can reduce the number of samples from 1000 to 50 while maintaining satisfactory result quality.

Due to resource and time limitations, I have chosen to employ the PLMS denoiser during the generation phase to expedite the process. By utilizing PLMS as the primary denoiser, I can generate a batch of 15 images in just 40 seconds on a 1080 Ti GPU.

4.7 Multi GPU computing

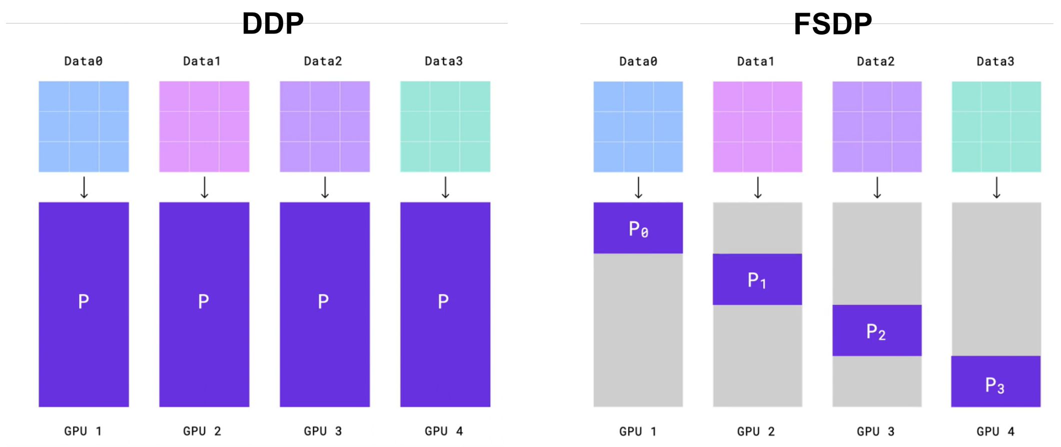
As previously said, the model was trained using Google Colab-Pro. It provides us with a 40 GB Nvidia A100 which is enough to run the model with a good batch size. The problem is with the time we can make the training run. Colab-Pro makes 500 Computational units available that, at maximum GPU load, are about 30 hours of training. In this amount of time, I was able to train the model for 300 epochs with a batch size of 64 latent images but it is clearly not enough.

To deal with this problem, a possible solution was to make the model train on the other available hardware resources: the cluster.

Pytorch Lightning implements various strategies for multi-GPU training. For my experiments, I choose two main strategies to try with. The first strategy is called Distributed Data-Parallel (DDP), and the second one is called Fuly-Sharded-Data-Parallel (FSDP).

DDP

Distributed Data-Parallel, as we can see from the image 4.12, sends a copy of the model ‘P’ to all the available GPUs on the system.



(a) The model "P" is entirely copied in each GPU

(b) The model "P" is spread through the available GPUs

Figure 4.12: Images taken from video explanation by Lightning.ai. Link to video.

Each GPU is fed with a different batch of data, and the loss value is independently computed by every GPU. As a result of this process, at the end of a single training step, we will have different gradient values on each GPU.

To address this, a synchronization mechanism is employed prior to the back-propagation step. This strategy ensures all GPUs possess identical gradients before performing back-propagation, aiming to maintain uniformity across the model on every GPU. However, a challenge arises in this implementation. Once loaded, our model is too big to fit a single 12GB GPU and, since this strategy sends a copy of the model to all GPUs, any of them can accommodate the complete model.

This strategy **is not** a suitable solution for our memory problems.

FSDP

The idea is we always give different batches to each GPU but we can also subdivide the model itself.

For example, each GPU computes the gradient for the ‘P0’ on their relative loss results. Then all the gradients are sent to GPU 0, which is in charge of managing the ’P0’ part of the model. The other GPUs can immediately discard their ’P0’ gradient right after computing it because only GPU 0 needs to memorize it. In this clever way, memory usage is reduced. In this way, we are able to train a relatively big model on a modest resources machine, sacrificing on the batch size. We can also use a sub-strategy called CPU-offloading which moves from the GPU memory to the optimizer parameters, the gradients, and activation weight to the CPU memory (RAM). When needed, it moves back on the GPU (VRAM). The downside of this methodology is the **GPU communication overhead**.

Chapter 5

Results

5.1 Evaluation metrics

5.1.1 Fréchet Inception Distance (FID)

To evaluate the Model generation quality, we make use of the standard metric Fréchet Inception Distance (FID), also used to assess the previous StyleGan implementation. FID is a commonly used metric in the image generation field that compares two sets of images. It measures the similarity between the real pictures and the generated images, considering both the real dataset features distribution and the visual appearance of the images. It uses an external pre-trained network called Inception Network which is trained to extract features from images. This network is used to extract features from both real and generated images. Then statistics are computed, particularly the covariance matrix and mean of the Gaussian distribution, which models the extracted features. The Fréchet Distance is then applied to measure the dissimilarity between two multivariate Gaussian distributions defined by the mean and covariance matrix.
A lower FID score indicates a closer similarity between the real and generated images, implying better image generation quality.

When it comes to generating abstract images, the FID metric may not be the most suitable choice since it fails to consider the essence of abstraction. If we aim to create abstract content, relying on a metric that primarily evaluates similarity to a dataset becomes irrelevant. FID is more applicable when assessing the quality of specific objects like faces or well-defined subjects such as cars, where its ability to compare against a dataset can be useful.

FID Resources problems

To ensure optimal FID statistics, a minimum of 10,000 generated images compared to 10,000 real samples is required. In state-of-the-art papers, typically, 50,000 samples are generated for each FID computation.

Given our hardware resources, generating 10,000 images for a single color category using the fastest sampler (PLMS) on a single GPU takes approximately 7 hours. Consequently, it would take a total of 56 hours to generate 50,000 samples for one label. Considering we have eight color classes and utilizing two GPUs, a comprehensive analysis using 50,000 samples would require approximately 4.5 days. However, it is unfeasible to freeze the training after a certain number of epochs to compute the FID due to the significant increase in training time, which is not accessible in our current hardware conditions.

As a solution, I conducted a partial analysis focusing on the FID with 10,000 samples from each color category only after completing the full training process.

Other Used Indicators

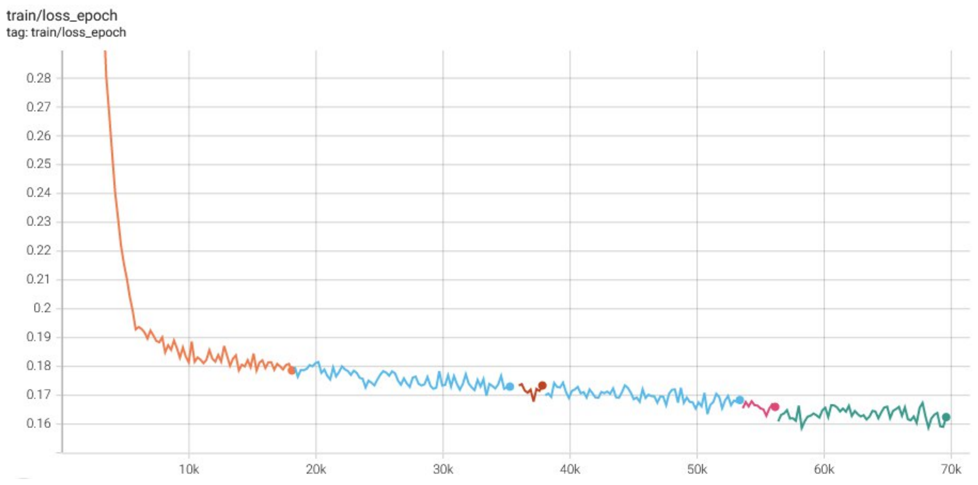
Since one of our goals was to generate images based on a color label, some results will be evaluated basing the considerations exclusively on the coherence of the produced color with the given color category. I provide some results considering each image as a point in the three-dimensional space using both RGB and CIE-LAB color space. The evaluation metric of the produced output will be the distance of the image from its corresponding color label.

5.2 LDM Training Results

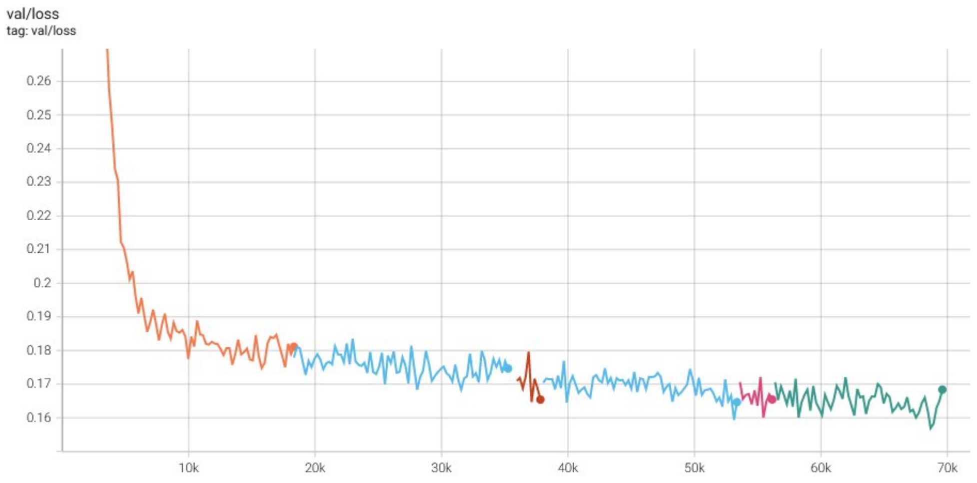
The presented LDM 4.5.1 was trained for a total of 34 hours with a batch size of 64 on an NVIDIA A100 occupying 32 out of 40GB on Colab-Pro. Due to time limitations on GPU usage, the training was conducted in multiple attempts. The objective of the training was to reach a minimum of 300 epochs, and we successfully completed the training for the intended duration.

The weight update was done by computing the gradient of the L^2 Loss between the noise we add at a certain timestep T and the predicted one. The model was also further trained using a copy of the weight to exploit the EMA update, which has the purpose of reducing the weight variability throughout the training period. The charts below show the trend of the training during the 300 epochs and I will show the results at evaluation time together with the EMA weight. The gaps in the charts are due to the different times the training was interrupted.

The behavior of the model during the is quite stable. During the first 10K steps, the learning procedure proceeds very fast to then slows down for the remaining



(a) LDM training Loss



(b) LDM validation Loss

time. Generally, the model continues to decrease its loss value, suggesting not to have reached the final plateau of the learning curve. The interesting fact about the loss is that even if the loss decrease by a relatively small amount, the produced outputs continue to improve in quality and sharpness.

5.3 Generated Images

In this section, I present the LDM (Latent Diffusion Model) generated samples to showcase its ability to produce images based on provided labels.

Figure 5.1 exhibits the generated images at a resolution of 256x256 pixels. The layout comprises nine rows, each representing a specific color label in the following order: Orange, Green, Blue, Purple, Yellow, Red, Black, White, and "Zero-Label." For more details about the "Zero-Label" and its assignment to the training samples, please refer to Section 4.4.3.

The first image demonstrates how training without the classifier-free Diffusion approach can capture the input color label. However, it occasionally falls short in producing images with strong color bias. Additionally, the "Zero-Label" (last row) generates images that lack meaning, uniformly colored with a dark blue hue.

In contrast, the model trained using the classifier-free guidance, with a scale factor of one (indicating no application of the classifier-free technique), produces similar results to the one without classifier-free guidance. However, in the "Zero-Label" row, the images exhibit more variation in terms of subjects and colors. We anticipate the "Zero-Label" row to generate images that could belong to any other color category. This label is trained using images that are indiscriminately sourced from all the other labels.

5.3.1 Increase the Scale Value

By increasing the scale value in the free guidance formula 4.1, the generation process becomes more inclined towards the provided label, intensifying its influence. This operation effectively reduces the variability of the generated outputs, resulting in greater consistency with the label. This effect can be likened to truncating the latent sampling in GANs generation.

As the scale factor increases, 5.3 we observe the emergence of vibrant colors, indicative of the model's comprehension of our conditioning. To explore the impact of scale variation, I intentionally exceeded the recommended quantity and analyzed its effect on generation. In Figure 5.4, it is evident that the dominant color prevails while the image variance is minimized. The model behaves as anticipated.

Considering our objective to produce images with a significant dominant color indicated by the label while maintaining a desirable level of variability, I opted to adopt a scale factor of 2. This choice strikes a balance between the color hint provided by the label and the desired variability in the generated images.

Qualitative Results of FSDP Training

As explained in the 4.7 I have modified the Official Latent diffusion to be able to use the FSDP strategy. I have trained using the model on two out of four GPUs to test if the code can run and produce meaningful outputs comparable with its twin model on Colab-Pro.

The figure shows the same model frozen on the 14^{th} epoch. On the left, the model was trained on Colab-Pro with a batch size (BS) of 64. On the right, the same model was trained on two GTX 1080-Ti with a BS of 6 for each GPU. This means that the model at each training step receives a total BS of 12 (number of GPUs * BS). We notice an interesting behavior. It seems that smaller batch sizes and, consequently, more frequent optimizer updates make the training procedure faster.

5.4 FID Results

The comparison with the StyleGAN should be done by directly comparing the relative FIDs values. For the LDM, I have computed the different FID values for our final output at a resolution of 256x256. For each color category, I generated at least 10000 images using the PLMS sampler, ensuring the best generation speed performances. Then I computed the FID against their relative training images in the dataset. Due to the fact that the dataset is highly unbalanced, some color labels, for example, with the yellow label, FID is computed with 10K generated samples against 147 real Images.

5.4.1 Diffusion Model and StyleGAN Comparison

The StyleGAN was fine-tuned for 100 epochs with a batch size of 4 images, enabling it to generate high-resolution images at 1024x1024 pixels. To ensure a fair comparison with the LDM, I generated 10,000 images using StyleGAN v3 at the same resolution (1024x1024) for each color category. These generated images were then downsampled to our target resolution of 256x256 pixels. The FID for StyleGAN was computed following the same methodology as described for the LDM to permit a direct comparison between FID values.

We utilized FID values along with two additional results to compare the two models.

In figure 5.5, I show the results in terms of FIDs values I obtained by computing it in two ways.

1. All the generated images of a color category against all the dataset images of that category only;

2. All the generated images of a specific color category against all the unified dataset.

From the results, we can clearly see that StyleGAN outperforms our LDM by a big margin. One of the reasons could be the lack of training time or also some errors in the implementation. The fundamental thing to have in mind is the difficulty of computing those values for only a partial analysis. In order to have complete results, we should compute the FID at training time with a good frequency and maintain control over the training process and its trend of results.

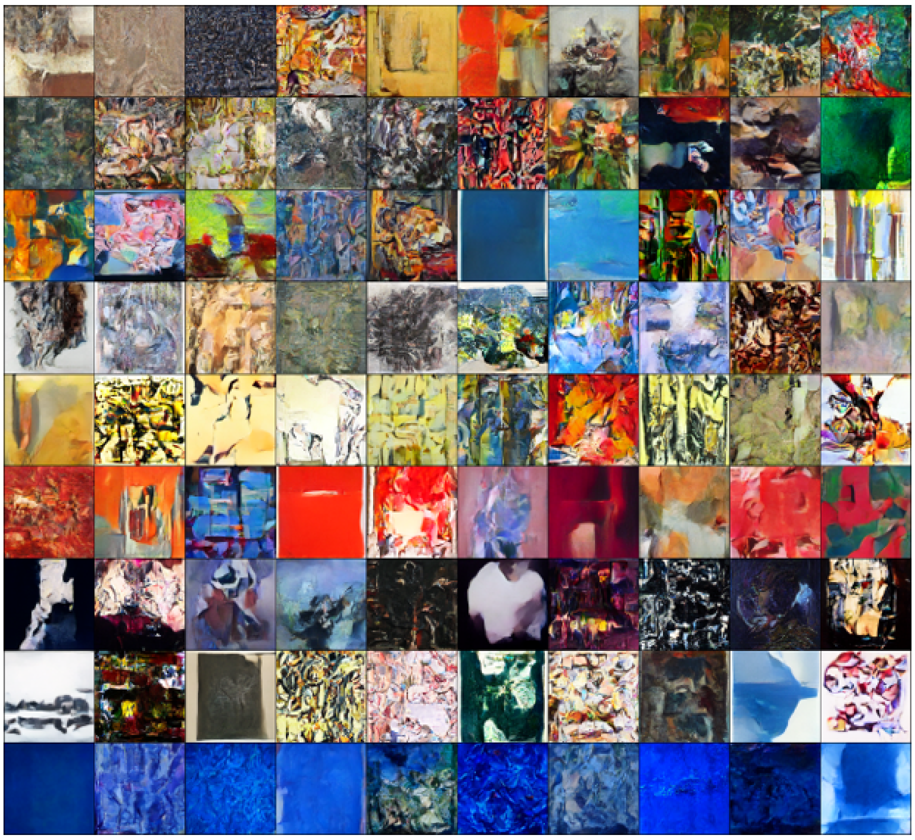
To compute the shown values in figure 5.6, I calculate the distance between each pixel in the image and its corresponding color category. This process was performed for all pixels in the image, and the average color distance was calculated. This computation was repeated for each image and then averaged across the entire dataset to obtain a single value.

The results demonstrate that the differences between the LDM and StyleGAN are comparable, and in some cases, the LDM yields superior results. Overall, we can conclude that both models achieve a similar level of accuracy.
Lower distance results are obtained with the CIE-LAB color space, which also considers the perceptual difference between colors.

Figure 5.7 presents results that exhibit similar behavior, even though computed using a slightly different approach. In this case, the computation involves averaging the colors of each image within a specific category and subsequently calculating the distance from the true color label.

Structural complexity in generated images

Figure 5.8 provides a visual comparison between the Latent Diffusion Model (LDM) and StyleGAN, highlighting the distinctive characteristics of each approach. The LDM demonstrates its capability to generate well-structured images, showcasing the potential benefits of longer training. However, it is evident from the coherence and the number of details in some of the created images that StyleGAN currently excels in producing high-quality results, surpassing the current LDM implementation.


(a) Training without classifier-free method

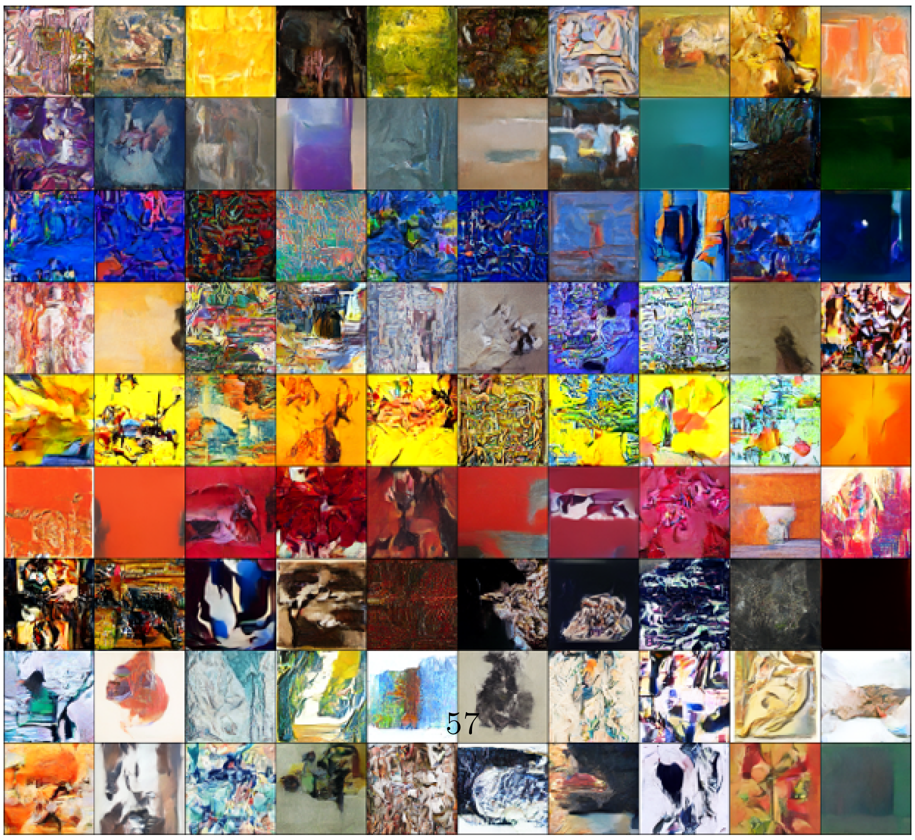

(b) Training with classifier-free method

Figure 5.1: Each row corresponds to a single color category. From top to bottom, we have: Orange, Green, Blue, Purple, Yellow, Red, Black, White, and "Zero-Label".

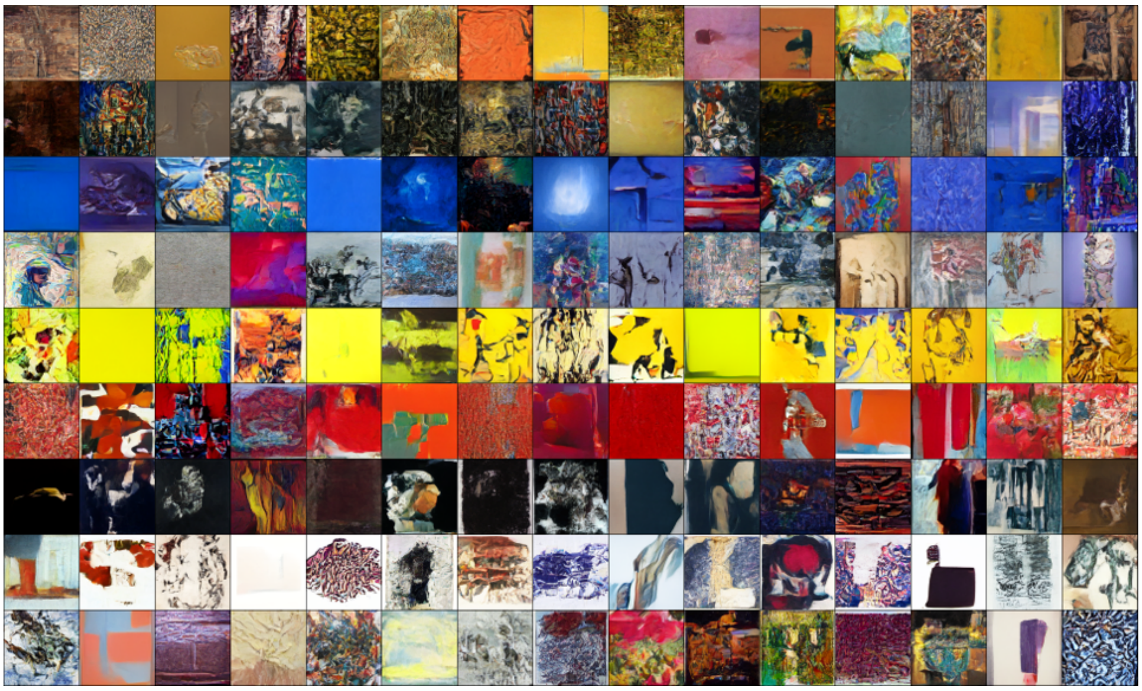


Figure 5.2: Generated images with scale factor $s = 2$

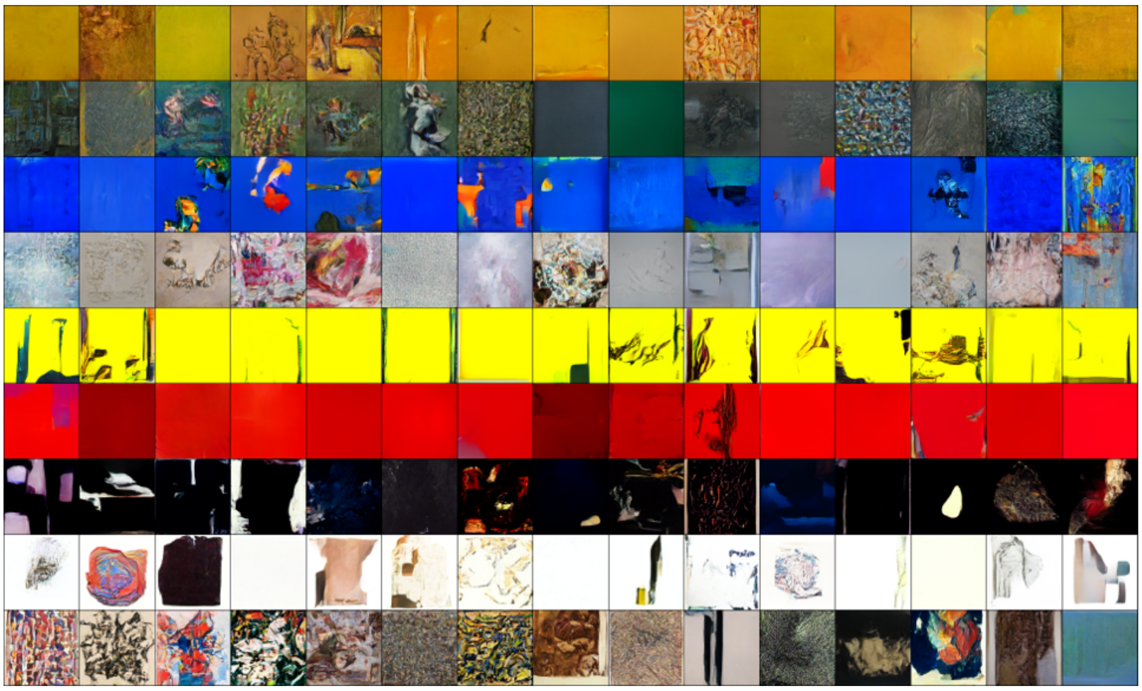


Figure 5.3: Generated images with scale factor $s = 5$

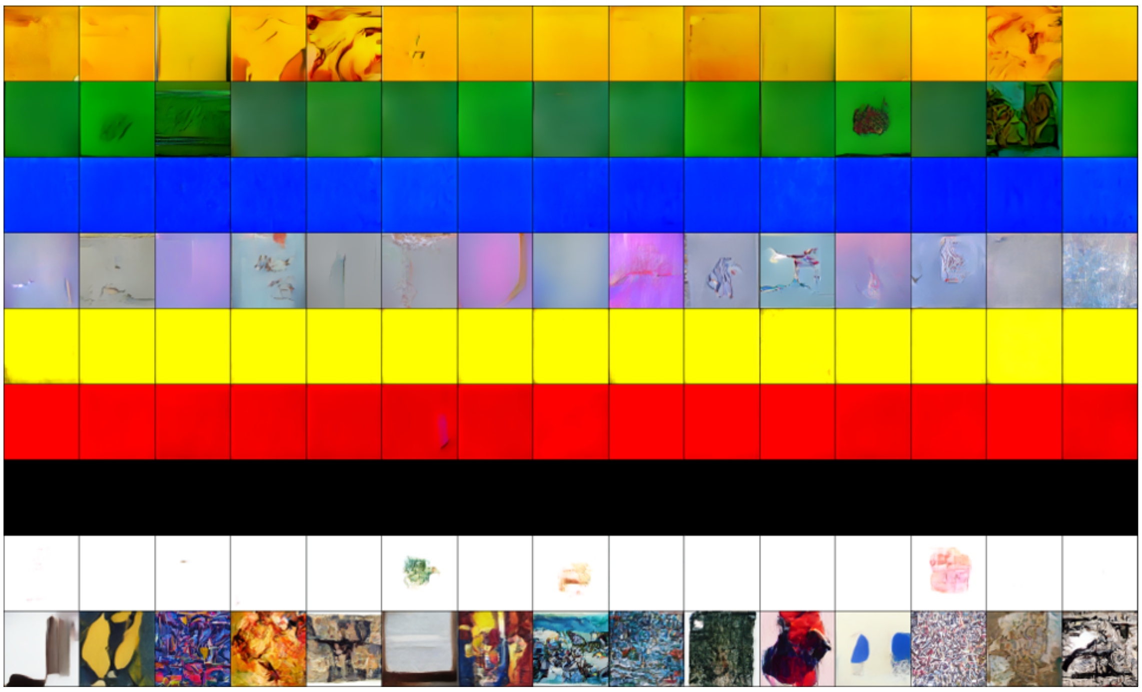
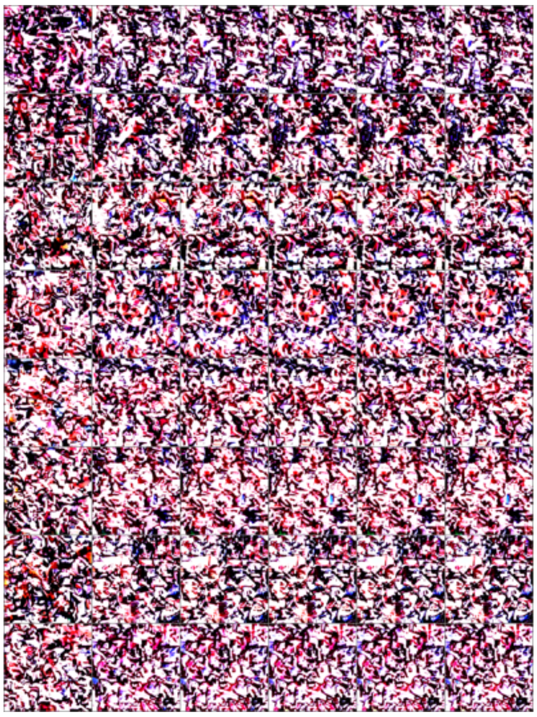
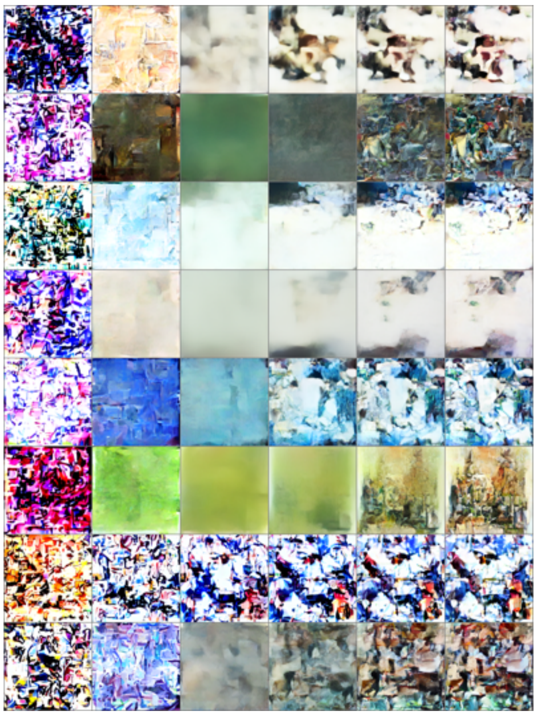


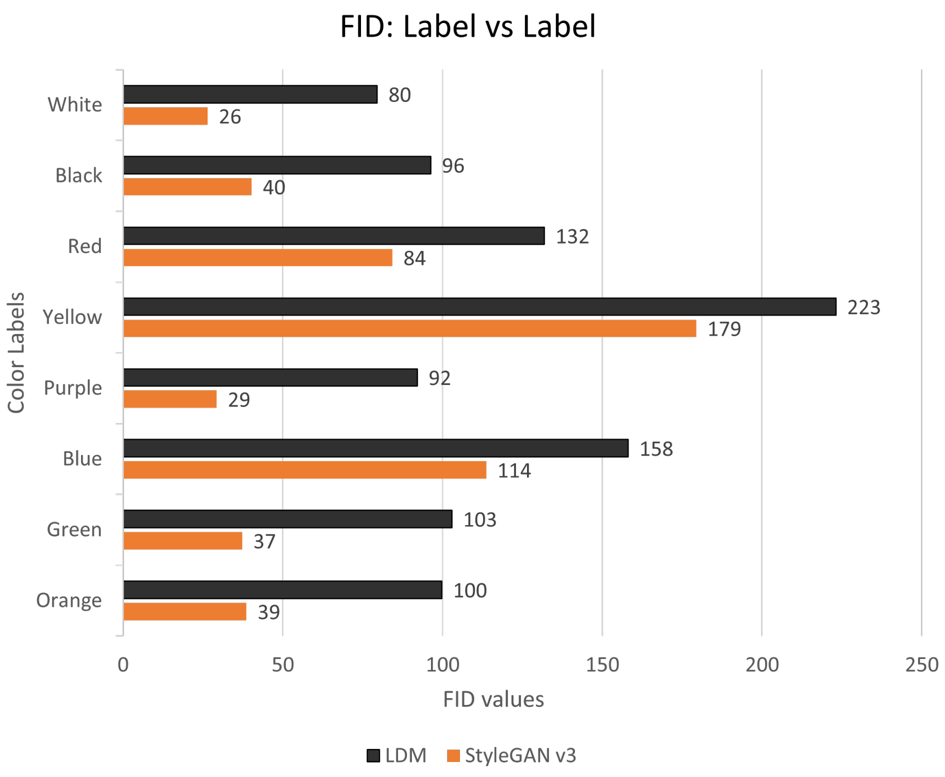
Figure 5.4: Generated images with scale factor $s = 15$



(a) Denoising procedure captured at epoch 14 on A100 on Colab -Pro



(b) Denoising procedure captured at epoch 14 on two 1080-ti on the local cluster



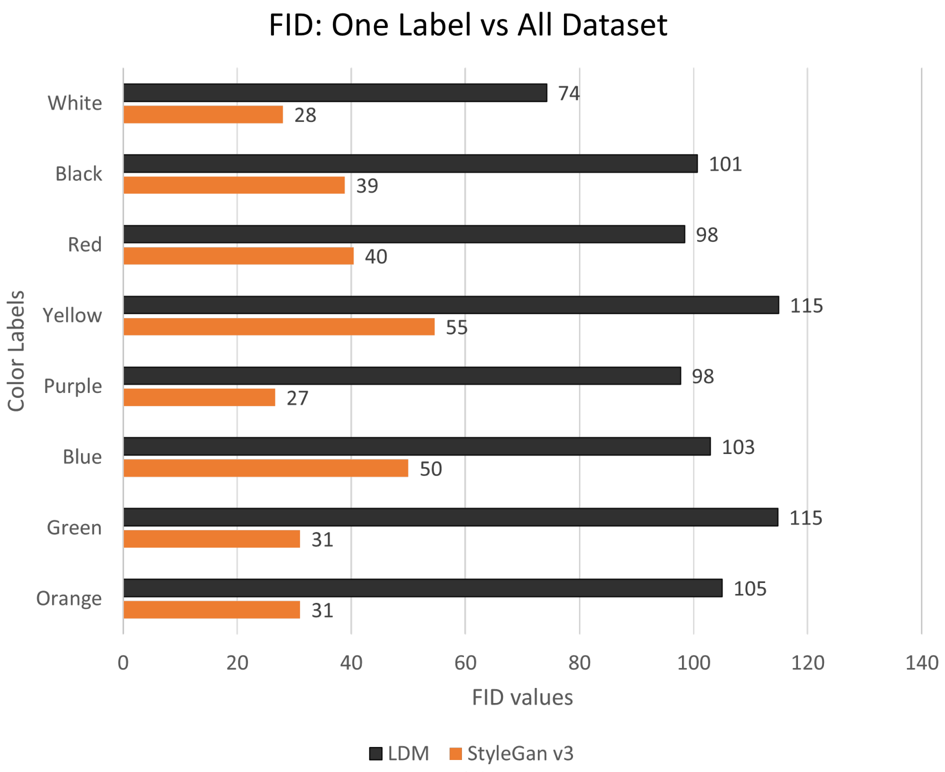


Figure 5.5: FID values

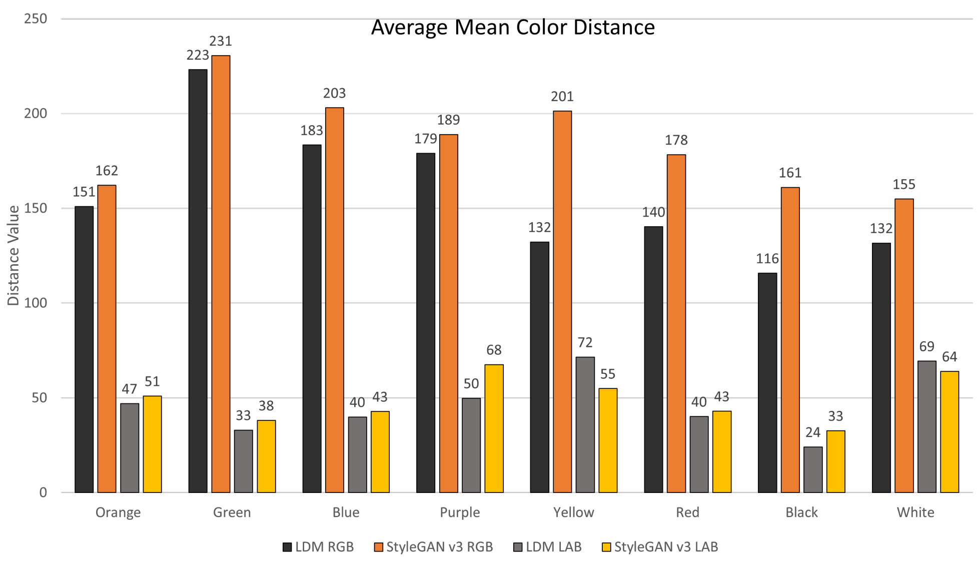


Figure 5.6: Average mean distance of colors in RGB and LAB color spaces.

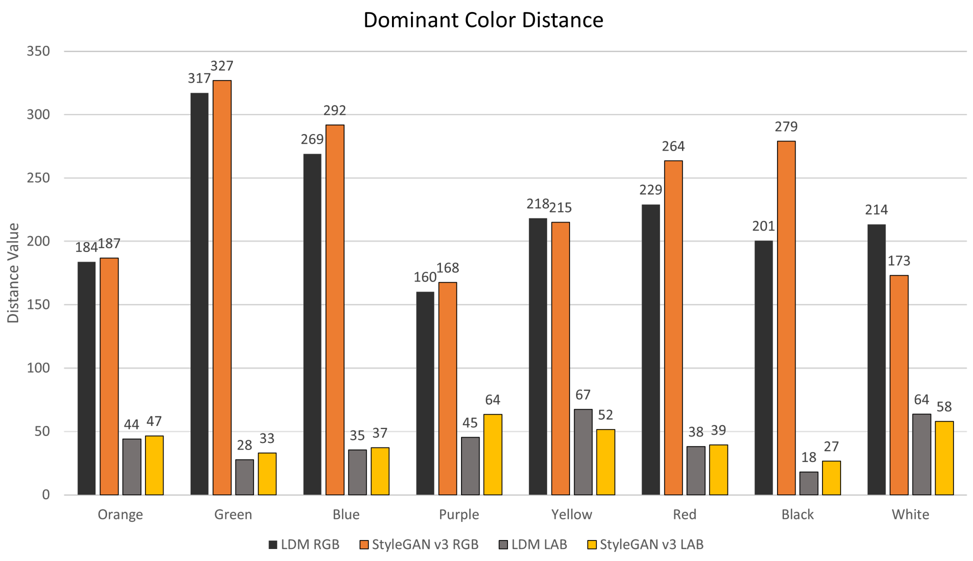


Figure 5.7: Average colors distance in RGB and LAB color spaces.

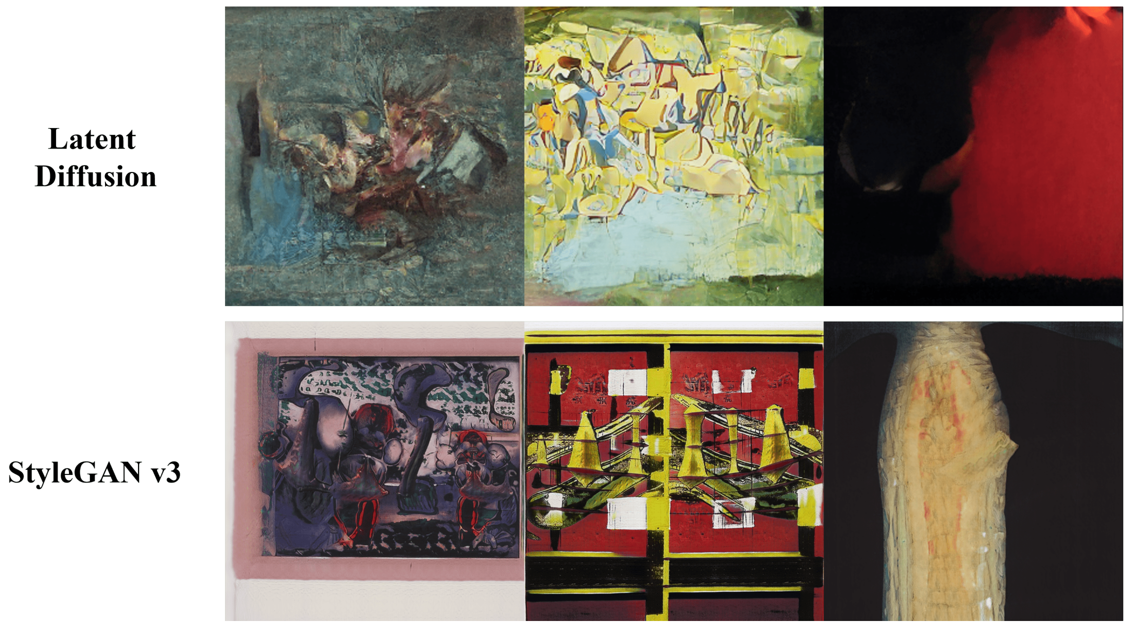


Figure 5.8: Top row LDM structured images. Bottom row StyleGAN generated images.

Chapter 6

Conclusions

6.1 Conclusion

In conclusion, this master thesis project aimed to explore the potential of leveraging Diffusion Models (DM) in comparison to the previously implemented StyleGAN v3. The results revealed the high flexibility of DM in accommodating the provided conditioning requirements. A customized prototype based on Latent Diffusion was successfully developed and trained using the available Custom Wiki-Art Dataset, enabling the color-conditioned generation of images at a resolution of 256x256 pixels.

The experiments showcased the model's ability to accurately capture the given color conditioning while also offering the option to fine-tune the degree of influence exerted by the conditioning on the generated images. It is worth noting that, at the present stage, the StyleGAN still exhibits better performance in terms of Fréchet Inception Distance (FID) values. However, it is important to highlight that further improvements in training time and resource allocation have the potential to impact the performance of the Latent Diffusion Model positively.

The findings also indicate that a longer training duration can lead to enhancements in terms of FID scores and the overall structure of the generated images. This suggests that with additional training resources and time allocation, the Latent Diffusion Model has the capacity to narrow the performance gap with the StyleGAN and potentially match its performance in terms of FID values and image quality.

Overall, this thesis project has demonstrated the potential of Diffusion Models as a viable alternative to StyleGAN, showcasing their flexibility in accommodating conditioning requirements and paving the way for further exploration and improvement in the field of generative image modeling.

6.2 Future Works

The limitations imposed by limited resources and time have significantly restricted my ability to achieve optimal results.

One major drawback of our obtained results is the high FID value, which correlates with the quality of structures within individual images, especially when compared to the impressive capability of StyleGAN in generating intricate and complex abstract structures. To address this, increasing the training time of our model is likely to result in a substantial improvement in terms of FID and, consequently, the production of better-structured abstract images. Another potential avenue for improvement is to explore alternative metrics that are better suited for evaluating the generation of abstract images, given the inherent challenges associated with this task.

Another prominent issue arises from the highly unbalanced dataset. The inherent imbalance introduces errors and disequilibrium during the image generation process. A thorough analysis of the dataset's properties may lead to alternative class subdivisions, significantly enhancing the overall results.

Also, the dimensionality of the dataset plays a crucial role in determining the quality of the generated results. A more diverse and expansive dataset provides the model with a greater range of examples to learn from, thereby increasing the variability of the final generated images.

Another aspect worthy of exploration pertains to the first stage of our model, namely the VQ-GAN. Training a VQ model from scratch holds the potential for achieving excellent results, given its specialization in feature extraction for our specific dataset. Alternatively, employing smaller and more efficient models can help reduce the system's memory requirements.

Taking a different perspective and focusing on altering the overall structure, we could consider eliminating the current first stage of the pipeline (Figure 1.1). Instead, we could explore the integration of sound within the Diffusion Pipeline. Inspired by LLM and BERT, an initial idea involves training a separate model to generate sound embeddings, specifically music embeddings, that encapsulate meaningful information about images. This can be accomplished by employing a dataset that associates musical tracks with corresponding images, such as film scenes accompanied by their background soundtracks or songs matched with their corresponding CD cover images.

Bibliography

[1] Matthew N. Bernstein. *The evidence lower bound (ELBO)*. 2020. URL: `https://mbernste.github.io/posts/elbo/` (visited on 05/25/2020) (cit. on p. 9).

[2] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: `1312.6114 [stat.ML]` (cit. on p. 13).

[3] Danijar Hafner. *Building Variational Auto-Encoders in TensorFlow*. Blog post. 2018. URL: `https://danijar.com/building-variational-auto-encoders-in-tensorflow/` (cit. on p. 13).

[4] James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. *Understanding Posterior Collapse in Generative Latent Variable Models*. 2019. URL: `https://openreview.net/forum?id=r1xaVLUYuE` (cit. on p. 13).

[5] Yixin Wang, David M. Blei, and John P. Cunningham. *Posterior Collapse and Latent Variable Non-identifiability*. 2023. arXiv: `2301.00537 [stat.ML]` (cit. on p. 13).

[6] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. *Neural Discrete Representation Learning*. 2018. arXiv: `1711.00937 [cs.LG]` (cit. on pp. 16, 17, 19).

[7] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. *Generating Diverse High-Fidelity Images with VQ-VAE-2*. 2019. arXiv: `1906.00446 [cs.LG]` (cit. on p. 16).

[8] Patrick Esser, Robin Rombach, and Björn Ommer. *Taming Transformers for High-Resolution Image Synthesis*. 2021. arXiv: `2012.09841 [cs.CV]` (cit. on pp. 19, 22, 45, 46).

[9] Angus Turner. *Diffusion Models as a kind of VAE*. 2021. URL: `https://angusturner.github.io/generative_models/2021/06/29/diffusion-probabilistic-models-I.html` (visited on 06/19/2021) (cit. on p. 22).

[10] Nikolas Adaloglouon Sergios Karagiannakos. *How diffusion models work: the math from scratch*. 2022. URL: `https://theaisummer.com/diffusion-models/#classifier-free-guidance` (visited on 09/29/2022) (cit. on p. 22).

[11] Kashif Rasul Niels Rogge. *The Annotated Diffusion Model*. 2022. URL: `https://huggingface.co/blog/annotated-diffusion` (visited on 09/07/2022) (cit. on p. 22).

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: `1505.04597 [cs.CV]` (cit. on pp. 28, 29).

[13] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 2021. arXiv: `2102.09672 [cs.LG]` (cit. on p. 29).

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: `1706.03762 [cs.CL]` (cit. on pp. 38, 42).

[15] Jonathan Ho and Tim Salimans. *Classifier-Free Diffusion Guidance*. 2022. arXiv: `2207.12598 [cs.LG]` (cit. on pp. 41, 42).

[16] Hezheng Lin, Xing Cheng, Xiangyu Wu, Fan Yang, Dong Shen, Zhongyuan Wang, Qing Song, and Wei Yuan. *CAT: Cross Attention in Vision Transformer*. 2021. arXiv: `2106.05786 [cs.CV]` (cit. on p. 42).

[17] Rui Li, Jianlin Su, Chenxi Duan, and Shunyi Zheng. *Linear Attention Mechanism: An Efficient Attention for Semantic Segmentation*. 2020. arXiv: `2007.14902 [cs.CV]` (cit. on p. 43).

[18] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2021. arXiv: `2112.10752 [cs.CV]` (cit. on p. 44).

[19] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. 2022. arXiv: `2010.02502 [cs.LG]` (cit. on p. 48).

[20] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. *Pseudo Numerical Methods for Diffusion Models on Manifolds*. 2022. arXiv: `2202.09778 [cs.CV]` (cit. on p. 48).