eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA

FACULTAD DE INFORMÁTICA

Bachelor Thesis

Bachelor Degree in Computer Engineering

Computer Science

# tfVAE: a Python library for the study of the Variational Auto-Encoder - a use case

*Iker Pintado García*

**Advisors**

Unai Garciarena Hualde

June 24, 2023

# Acknowledgments

Thanks to all my friends and family that have accompanied me through all my academic years. Thank you for all the support you gave me and for being there in every up and down.

I also want to thank all my coworkers. You all have been by my side almost every day since I started this project and you have helped me to cope with all the stress with your wonderful sense of humor and company.

Last but not least, special thanks to Unai Garciarena Hualde for being such a great advisor. Thank you for giving me all your trust to carry on this project and giving me this incredible chance. Thank you for all the support and for being there during all the project, for being there every week in our meetings on Tuesday. Thank you for all your patience and having such an active role as the advisor of this thesis. And congratulations for being such a good advisor, this thesis would not have been possible without your role!

# Abstract

In recent years, the Artificial Intelligence (AI) has experienced a significant upswing, more particularly in the field of Deep Learning (DL). The Variational Auto-Encoder (VAE) is a deep generative model that maps the features of the input data to a latent prior distribution, and reconstructs data from latent vectors. To map and reconstruct the data, the model uses a encoder and decoder Deep Neural Networks (DNN) respectively, and the latent prior is usually a Gaussian distribution.

Although the VAE is a functional generative model, its training method is conflicting because its cost function is composed of two values: the reconstruction error and the Kullback-Leibler (KL) divergence. For this reason, we experimented on this training process and on different variants of the state of the art of the model that propose improvements on this aspect.

For the experimentation, a lack of tools to work with the VAE in a simple, agile and flexible way was observed. For this reason, the creation of a library called **tfVAE** was proposed, which offers an ordered, structured and scalable solution to the given problem. This not only allows the study or experimentation of the model, but also facilitates the implementation of new variants of it. This library unites concepts from Computer Science with the use of neural models, and from Software Engineering, such as the use of an architectural pattern based on Domain Driven Design (DDD).

For the experimentation, preliminary experiments were first carried out using the MNIST database, which would serve as a reference point for the rest of the experiments, where the MNIST and Fashion-MNIST databases would be used. The main experiments were carried out with the model architectures of the preliminary experiments and applied to the state-of-the-art models. In this stage of the experimentation, only the generative capability of the models was tested using the Frechet Inception Distance (FID) computed using our own inception models as a metric. It was mainly observed that the FID obtained from a custom model can obtain consistent results, that reducing the magnitude of the KL divergence leads to better generations, and that scaling the latent vectors can lead to better image generation.

# Laburpena

Azken urtetan, Adimen Artifizialak gorakada handia izan du, Ikaskuntza Sakonaren arloan, hain zuzen ere. *Variational Auto-Encoder*-a (VAE) eredu sortzaile sakon bat da, aldez aurretik ezkutuko banaketa batera sarrera datuen ezaugarriak mapatzen dituena, eta datuak ezkutuko bektoreetatik abiatuta berreraikitzen dituena. Datuak mapatzeko eta berreraikitzeko, ereduak erabiltzen ditu kodetzailea eta deskodetzailea Sare Neuronal Sakonak, hurrenez hurren, eta a priori banaketa latentea Gaussiana banaketa da normalean.

VAE eredu sortzaile funtzionala den arren, bere entrenamendu-metodoa gatazkatsua da, kostu-funtzioa bi baliok osatzen baitute: berreraikitze-errorea eta Kullback-Leibler (KL) dibergentziak. Horregatik, esperimentatu dugu entrenamendu-prozesu horri buruz, eta ereduaren egoeraren bariante desberdinak esperimentatu nahi izan dira, alderdi horretan hobekuntzak proposatzen dituztenak.

Esperimentaziorako, VAErekin modu sinple, azkar eta malguan lan egiteko tresnarik ez zegoela ikusi zen. Horregatik, **tfVAE** izeneko liburutegia sortzea planteatu zen, emandako arazoari irtenbide ordenatua, egituratua eta eskalagarria emateko. Horrek, ereduaren azterketa edo esperimentazioa ahalbidetzeaz gain, haren bariante berrien inplementazioa ere errazten du. Liburutegi honek Konputazio Zientzien eredu neuronalen eta Software Ingeniaritzaren kontzeptuak batzen ditu, hala nola *Domain Driven Design*-en oinarritutako arkitektura-patroiaren erabilera.

Esperimentaziorako, lehenik eta behin, MNIST datu-basea erabiliz egindako aurretiazko esperimentu batzuk egin ziren, gainerako esperimentuetarako erreferentzia-puntu gisa balioko zutenak, non MNIST eta *Fashion*-MNIST datu-baseak erabiliko ziren. Esperimentu nagusiak aurretiazko esperimentuen ereduen arkitekturekin egin ziren, artearen egoeraren ereduei aplikatuz. Esperimentazio-etapa honetan, modeloen sortze-gaitasuna soilik egiaztatu zen, *Frechet Inception Distance* (FID) erabiliz, betiere *inception* eredu propioen baldintzapean, metrika gisa. Batez ere ikusi zen eredu pertsonalizatu batetik lortutako FIDak emaitza sendoak lor ditzakeela, KL dibergentziaren magnitudea murrizteak sorkuntza hobeak dakartzala, eta ezkutuko bektoreen eskalatzeak irudi-sorkuntza hobea ekar dezakeela.

# Resumen

En los últimos años, la Inteligencia Artificial ha experimentado un importante repunte, más concretamente, en el campo del Aprendizaje Profundo. El *Variational Auto-Encoder* (VAE) es un modelo generativo profundo que mapea las características de los datos de entrada a una distribución previa latente, y reconstruye los datos a partir de vectores latentes. Para mapear y reconstruir los datos, el modelo utiliza una Red Neuronal Profunda codificadora y otra decodificadora, respectivamente, y la distribución latente a priori suele ser una distribución Gaussiana.

A pesar de que el VAE es un modelo generativo funcional, su método de entrenamiento resulta conflictivo debido a que su función de coste se compone de dos valores: el error de reconstrucción y la divergencia Kullback-Leibler (KL). Por esto mismo, se ha experimentado sobre este proceso de entrenamiento y sobe distintas variantes del estado del arte del modelo que proponen mejoras sobre este aspecto.

Para la experimentación, se observó una ausencia de herramientas para trabajar con el VAE de una manera simple, agil y flexible. Por ello, se planteó la creación de una librería llamada **tfVAE** que plantea una solución ordenada, estructurada y escalable al problema dado. Ésta no solo permite el estudio o la experimentación del modelo, sino que también facilita la implementación de nuevas variantes del mismo. Esta librería une conceptos de las Ciencias de la Computación con el uso de modelos neuronales, y de la Ingeniería del Software, como el uso de un patrón de arquitectura basado en *Domain Driven Design*.

Para la experimentación, se realizaron primero unos experimentos preliminares hechos utilizando la base de datos MNIST, y que servirían como punto de referencia para el resto de experimentos, dónde se utilizarían las bases de datos MNIST y *Fashion*-MNIST. Los experimentos principales se realizaron con las arquitecturas de los modelos de los experimentos preliminares aplicándolas a los modelos del estado del arte. En esta etapa de experimentación se comprobó únicamente la capacidad generativa de los modelos, utilizando el *Frechet Inception Distance* (FID) utilizando modelos *inception* propios como métrica. Se observó principalmente que el FID obtenido de un modelo personalizado puede obtener resultados consistentes, que la reducción de magnitud de la divergencia KL lleva a mejores generaciones, y que el escalado de los vectores latentes puede desembocar en una mejor generación de imágenes.

# Contents

# Glossary

**abstract function** In the context of software development, an abstract function is a function that is declared in a class or interface but does not have an implementation. Instead, the implementation of the function has to be provided by a subclass or implementing class. 28

**activation function** In the context of neural networks, it is a mathematical function that is applied to the output of a neuron to introduce non-linearity into the model. 6–8, 39, 40, 57

**architectural pattern** In the context of software development, it is a reusable solution or template that provides a structured approach for designing and organizing software systems. It offers a high-level blueprint or framework that helps developers address common design challenges and create robust, scalable, and maintainable applications . 25, 26

**architecture** For neural networks, the design and configuration of a model, including the number and type of layers, the activation functions used, and the connections between the layers. xv, 3, 6–8, 13–15, 33, 34, 36, 37, 39, 48, 56

**backward-propagation** Also known as back-propagation, in the context of machine learning, it is the process of calculating the gradient of the loss function with respect to the weights of a neural network. 8, 16

**bias** In a neural network, a parameter that represents the intercept term in a linear equation. 5, 8

**convolutional layer** A type of layer in a neural network that performs a convolution operation on the input data. Convolutional layers are commonly used to extract features from images. 9, 11, 13, 35

**dense layer** A type of layer in a neural network where all the neurons in one layer are connected to all the neurons in the previous layer. Also known as a fully connected layer. 7, 13, 35, 39

**differentiable** In mathematics and optimization, it refers to a property of a function that allows it to have well-defined derivatives at every point within its domain. Differentiable functions are used extensively in optimization algorithms, such as gradient descent, to train and update the parameters of neural networks . 16, 17

**discriminator** In the context of an Generative Adversarial Model, it is the model that evaluates the generated samples and determines whether they are real or fake. 2

**epoch** In the context of neural networks, an epoch refers to a complete pass through the entire training dataset or the current batch during the training of a model. 8, 9, 17, 35–37, 40–43

**feature map** Output of a convolution between a kernel and the input, which maps the features of the input to a series of features. 9, 11

**forward-propagation** The process of feeding input data through a neural network to produce an output. 8

**generative model** A model that can generate new data that is similar to the data it was trained on, by capturing its underlying structure. 1, 2

**generator** In the context of generative models, a model that generates new data samples that are similar to the training data. 2

**gradient** In the context of neural networks, the gradient is a vector that contains the partial derivatives of a loss function with respect to the weights of a neural network. The gradient provides information about the direction of steepest ascent of the loss function, which is used during the optimization process to update the weights of the network. 9, 16

**hyperparameter** In machine learning, a hyperparameter is a parameter on which the model is highly dependent, and whose value is set before the learning process begins . 3, 8, 34, 36

**kernel** In the context of Convolutional Neural Networks, it is a small matrix that is used to transform the input to a feature map in the convolutional layers. xv, 9–12

**latent representation** The latent representation is the representation of an input in a lower-dimensional space mapped by the model. 13, 14, 16

**latent space** In the context of generative models, the latent space is typically a lower-dimensional representation of the data, where similar data samples are mapped to points close to each other in the space. 3, 13, 19, 39, 56, 57, 59

**layer** A component of a neural network that performs a specific computation on its input data. 6–9, 11–13, 36, 57, 59

**learning rate** The learning rate is a hyperparameter that controls the step size at which the weights of a neural network are updated during training when using a gradient descent technique. xv, 8, 9

**lint** In the context of software development, lint refers to a tool or program that analyzes source code for potential errors, bugs, and stylistic or formatting issues. Linting is often performed as a part of a development workflow to ensure code quality and consistency, catch common errors, and identify potential security vulnerabilities. 24, 25

**loss function** A mathematical function that calculates the difference between the predicted output of a machine learning model and the actual output. 2, 8, 9, 16, 19, 20, 28, 37

**multi-objective optimization** Process that aims to find a set of optimal solutions that represents the trade-offs between multiple potentially conflicting objectives. Its best known approach is the pareto optimization or pareto-based optimization, which aims to explore the trade-off space and identify a set of solutions that are non-dominated, meaning that no other solution in the set is better in all objectives. This set of non-dominated solutions is known as the pareto set . xv, 35, 36

**pooling layer** A type of layer that downsamples the output of a convolutional layer by summarizing groups of data in the feature maps into a single value. 12

**reinforcement learning** A branch of machine learning that deals with how an agent can learn to make sequential decisions in an environment to maximize a cumulative reward. It involves an agent interacting with an environment and learning from feedback in the form of rewards or punishments. 1

**stochastic** In the context of machine learning and optimization, it refers to a process or algorithm that involves randomness or probabilistic elements . 16

**stride** In a convolutional layer, the stride is the step size used to move the receptive field across the input data during convolution. The stride determines the amount by which the receptive field is shifted at each convolution operation. xv, 9–11, 39, 40

**supervised learning** A subcategory of machine learning that uses labeled datasets to train algorithms to accurately classify data or predict outcomes. 1

**trace linear algebra operation** In linear algebra, the trace of a square matrix is a scalar value that represents the sum of the diagonal elements of the matrix . 35

**transposed convolutional layer** A type of layer in a neural network that is used to upsample feature maps. Transposed convolutional layers, also known as deconvolutional layers or fractionally-strided convolutions, use a fractionally larger stride value than the corresponding convolutional layer to perform upsampling. 11, 39

**unsupervised learning** A subcategory of machine learning where the algorithm learns patterns and structures from unlabeled data without explicit guidance or labels provided by a human. 1

**weight** In a neural network, a parameter that determines the strength of the connection between two neurons. The value of the weights determines the contribution of the corresponding input to the output of the neuron. 5, 8, 9

# List of Figures

# List of Tables

# List of Acronyms

**AE** Auto-Encoder. xv, 13, 14

**AI** Artificial Inteligence. 1

**CNN** Convolutional Neural Network. xvi, 9, 28, 33–35, 39, 49, 52, 55, 62, 80

**CVAE** Convolutional Variational Auto-Encoder. xv–xvii, 28, 33, 34, 36, 43, 44, 56, 57, 60, 73, 75–77, 79, 80

**DDD** Domain Driven Design. xv, 25–28

**DirVAE** Dirichlet Variational Auto-Encoder. 21

**DL** Deep Learning. 1

**DNN** Deep Neural Network. 2, 9

**Fashion-MNIST** Fashion Modified National Institute of Standards and Technology. xv–xvii, 34, 35, 37, 41–49, 51, 53, 55, 56, 74, 76, 78, 80

**FID** Frechet Inception Distance. xv–xvii, 35–38, 40–48, 55–57, 79, 80

**GAN** Generative Adversarial Network. 2, 21

**IDE** Integrated Development Environment. 27

**KL** Kullback-Leibler. xv, xvi, 2, 15, 16, 20, 37, 39–48, 77, 78

**ML** Machine Learning. 1, 23

**MLP** MultiLayer Perceptron. xv, xvi, 6–8, 28, 33, 34, 39, 40, 42, 43, 48–51, 53, 55, 62, 80

**MNIST** Modified National Institute of Standards and Technology. xv–xvii, 33–37, 40–47, 49, 50, 52, 54–56, 73, 75, 77, 80

**MSE** Mean-Squared Error. xv, 2, 16, 37, 39–48

**MVP** Minimum Viable Product. 61

**NN** Neural Network. 1, 3, 5, 6, 8, 14, 16, 21, 33, 56

**PR** Pull Request. 24

**RBM** Restricted Boltzmann Machine. 2

**RNN** Recurrent Neural Network. 59

**RVAE** Recurrent Variational Auto-Encoder. 59

**SB-CVAE** Stick-Breaking Convolutional Variational Auto-Encoder. xvi, xvii, 28, 34, 44, 46–49

**SB-VAE** Stick-Breaking Variational Auto-Encoder. xvi, xvii, 19–21, 28, 34, 44–46, 48, 59

**SGD** Stochastic Gradient Descent. 8, 16

**TVD** Total Variation Distance. 35–37

**VAE** Variational Auto-Encoder. xv–xvii, 2, 3, 13–16, 19–21, 23, 25, 27, 28, 33, 34, 40–44, 48, 55–57, 59, 60, 62, 73, 74, 77, 79, 80

CHAPTER 1

# Introduction

## 1.1 Motivation and Background

In recent years, Artificial Inteligence (AI) has undergone exponential growth, completely transforming both the current industry and people's daily lives. One sub-field of AI, Machine Learning (ML), is particularly responsible for this popularity increment. This area encompasses algorithms that mimic the learning from data and make predictions or decisions in an informed way. This learning can be reinforcement learning, unsupervised learning, or supervised learning.

In turn, within the ML field, there is a particularly high performing kind of model, the Neural Networks (NNs), that are traditionally associated with supervised learning and reinforcement learning. These models are capable of performing what is called Deep Learning (DL), which takes ML to a whole new level, making it possible to make choices or forecasts of much higher quality. In the last decade, improved solutions to problems such as natural language processing (e.g., sentiment analysis [1] or machine translation [2]), image recognition (e.g., face recognition [3] or image segmentation [4]), speech recognition (e.g., spoken language recognition [5] or speech transcription [6]), among others, can be found.

Although it may seem that the field of AI has been extensively explored. The truth is that this area of research is constantly expanded with new problems that can find their solution with ML and there are a lot of techniques, and learning models awaiting to be discovered.

The first important step that would lead to DL, and that, as a consequence, would give rise to the entire revolution that is being experienced in these times, was the proposal of the perceptron [7], a little more than half a century ago. The boom of this topic happened approximately ten years ago along with the uprising of some NN models [8]. This only means that the potential of ML is huge if we take a look at how advanced it is after these years. Proof of this is the constant advances in the field, which, moreover, are becoming more and more frequent. One example could be the OpenAI organization [9], which is continually advancing the state of the art with brand new models that push the limits of what is possible to obtain through AI [10, 11].

In this thesis, we will work over generative models to explore the automatic generation of data using Deep Neural Networks (DNNs). Some well-known generative models could be the Restricted Boltzmann Machine (RBM) [12], that learns a probability distribution over the input data without any labeled output data; the VAE [13], that encodes the input data to some latent distributions that are sampled and reconstructed to its original version; and the Generative Adversarial Network (GAN) [14], where a generator and a discriminator are confronted pushing each other to learn.

More specifically, we will work over the VAE. This model assumes that there is a prior latent statistical distribution in the data, and, once trained, it is capable of generating new data by receiving random samples of the mentioned distribution.

This model has some inherent drawbacks, like tending to generate blurred images or slow convergence of the gradient, but it can still achieve impressive results and learn complex distributions, making the model still a good choice of generative model.

The goal is to study the VAE in depth, to develop an environment that facilitates its study and work, and propose some variations that could ideally lead to some improvements in its performance.

## 1.2 Objectives and Scope

Just like mentioned in the previous section, the main objectives of this thesis are to create an easy-to-use library using TensorFlow [15], Keras [16] and Numpy [17], and experiment by using it as a tool for applying some variations to several elements of the VAE while documenting how the performance of the model reacts under these applied changes. The library would set up a whole environment to work with the model and explore the capabilities of it, making both of the objectives highly linked. This work does not exclusively aim to be conclusive, but also aspires to contribute with quality tools and lines of research that can be resumed in future studies.

As the main objectives are so wide, and cannot be completely addressed in a work of the magnitude of a bachelor's thesis, we need to establish a clear scope which will help us to partially accomplish the proposed objectives successfully, and in an orderly way. This does not mean that the objectives will be left aside, but that they will be studied under other constraints to make the work feasible.

First of all, we will limit the scope of the VAE. Moreover, we will limit the type of data that is used to train the model, and, consequently, the type of data that is generated. The implementation of the VAE will be specifically created to be trained to generate images. This way, we set a consensus where every time that data is mentioned under the context of this thesis, it will refer to images. In any case, the implementation will be structured to leave room for future implementations of the VAE that accept other specific kinds of data that are not just images, or even a single implementation that accepts any type of data as input.

The main task under the scope of the thesis will be the study of the role played by the loss function of the VAE. The loss function of the model is computed through combination of the Kullback-Leibler (KL) divergence [18] and any metric for measuring the difference between the input and the output data, such as the Mean-Squared Error (MSE). This loss function, results in a defective training process. It happens because the KL divergence and the other function that compounds the loss function usually return values of different magnitudes making one of them less relevant over the other, and creating plain spaces that will obstruct the learning algorithm [19].

Highly related to the point mentioned right above, another research direction will be trying different distributions for the latent space of the model. Commonly, the normal distribution is used, but there are other distributions have been proposed, and reported interesting results [20, 21, 22]. Thus, we will explore different combinations of the different dependencies of the model.

Finally, as the neural models are known to be highly dependent on their architecture, a hyperparameter selection will be performed among the different experiments in search of the optimal configuration for the model. These hyperparameters go from the typically seen in every problem related to NNs, to the ones that only make sense under the context of the VAE. This results in a considerable number of decisions to be made when designing a VAE.

CHAPTER 2

# Neural Networks: A Brief Overview

## 2.1 Perceptron

Artificial NNs find their roots in the percetron. It was introduced in the year 1958 and its algorithm can be used for binary classification tasks, where the goal is to separate the data into decision boundaries in a high-dimensional space. An example of a decision boundary learned by a perceptron on a set of two-dimensional samples can be found in Figure 2.1

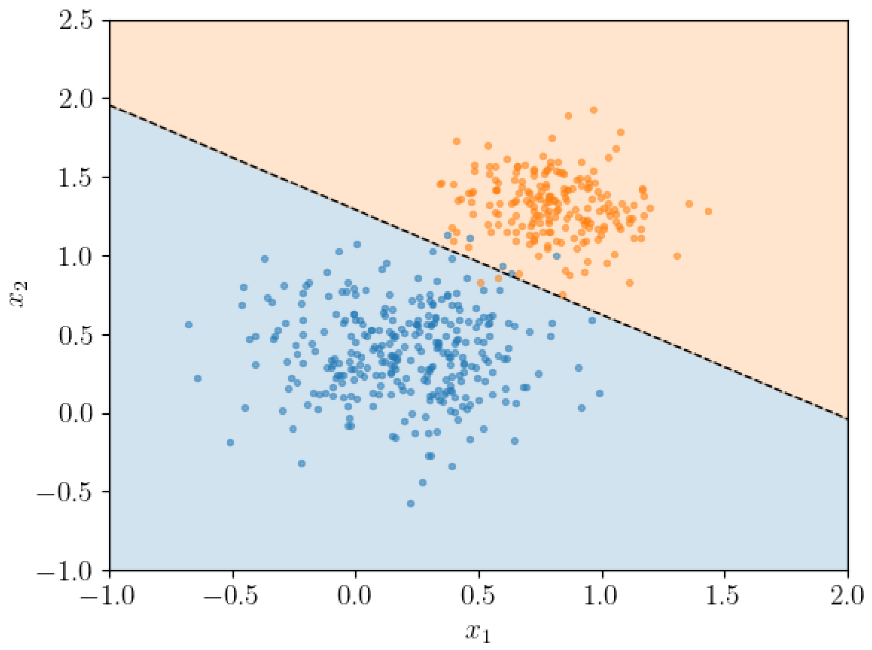

**Figure 2.1:** Decision boundary of a perceptron in a two-dimensional space

A single perceptron consists of a series of weights and a bias. There is always one single bias per neuron and as many weights as input values the model has. An example of this architecture can be seen in the following Figure 2.2. The algorithm for classifying consists on a weighted sum of the input data plus the bias, and piping that output to an activation function. Traditionally, the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ is used because it maps any input value to a value between $0$ and $1$, which makes it convenient for modeling binary classification problems. A plot of this function is showed in Figure 2.3.

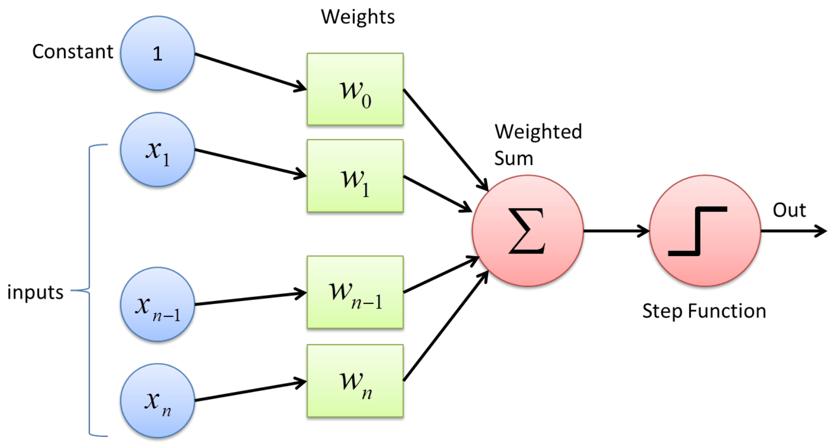

**Figure 2.2:** Illustration of a single perceptron

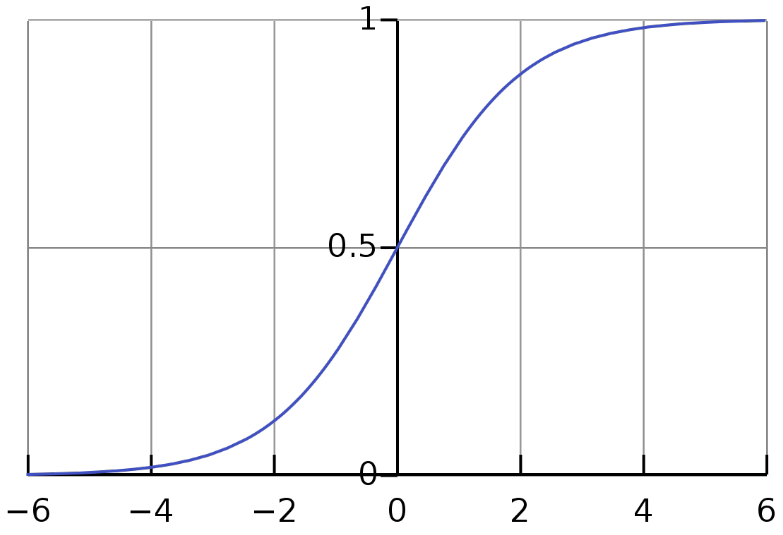

**Figure 2.3:** Sigmoid function

## 2.2 Multi-layer Perceptron

The MLP [23] is a type of artificial NN that consists of multiple layers. Every MLP's layer has at least one single perceptron. An example of a typical MLP is shown in Figure 2.4.

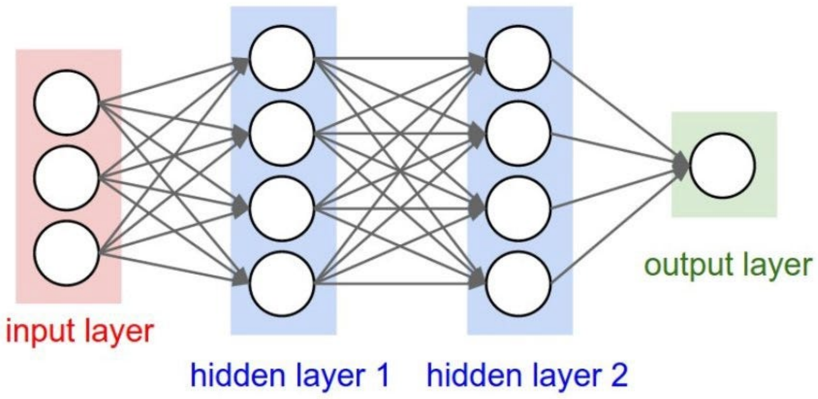

**Figure 2.4:** Architecture of a MLP

The MLP always has at least one hidden layer between the input and output layers. The hidden layers allow the network to learn much more complex patterns from the input data, like the one shown in Figure 2.5, through the usage of non-linear transformations [24]. The most basic layers are the dense layers.

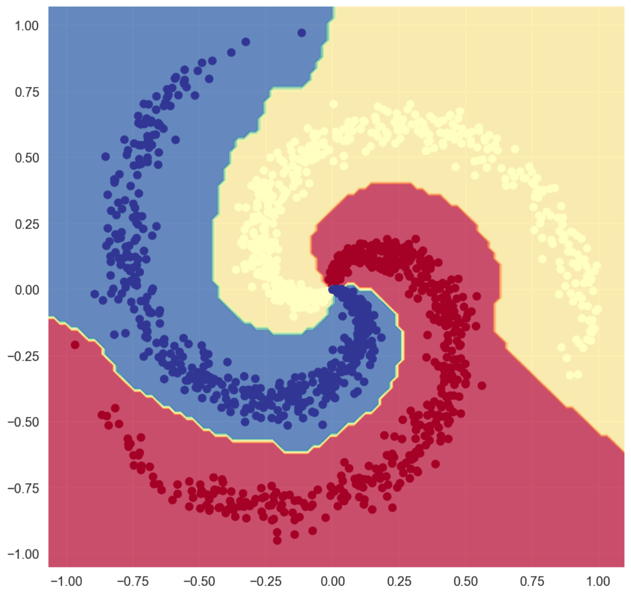

**Figure 2.5:** Decision boundaries of a MLP in ternary classification

For the output layer, the activation function is dependent on the type of problem that the MLP is bearing. In some occasions, the output layer of a model does not have an activation function.

The input and hidden layers, as a common choice, have the same activation function, the ReLU [25] function, which has been proven to be effective and its cost is extremely low, allowing the model to be trained rapidly and with high performance.

$$\mathrm{ReLU}(x) = \max(0, x)$$

The way a MLP works is not very different from the perceptron. It can have different architectures, but the most common one is making all its layers dense. So, the input data is passed to all the neurons of the first layer, then, the output of the first layer serves as the input of the second, and so on until the output layer is reached. So to say, if we call $h_p$, $w_p$ and $b_p$ to the $p$-th layer's output, weights and bias respectively, for any layer $l$ after the input layer, $h_l = \sigma_l(w_l \cdot h_{l-1} + b_l)$.

The most common training learning algorithm uses Stochastic Gradient Descent (SGD). It has two different phases: forward-propagation and backward-propagation [26]. Expressed in mathematical language, an instance is defined as $(x, y)$, where $x \in \mathbb{R}^n$ is a $n$-dimensional vector of features of the instance that we want the model to be able to correctly classify in the future, and $y \in \{0, 1\}$ is the actual label. We have $m$ instances for training: $\{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ and we will use $X \in \mathbb{R}^{n \times m}$ to refer to the training set, and $Y \in \mathbb{R}^m$ for the label set.

The forward-propagation is used for all the different phases of a model. Each perceptron of the model having a set of weights $W \in \mathbb{R}^{n+1} \mid W = [w_0, w_1, ..., w_n]$, where $w_0$ is the value of the bias. For each perceptron of the model, we will perform the scalar vector multiplication between any input data $x_t$ and the weights as: $[1, x_{t,1}, x_{t,2}, ..., x_{t,n}] \times W = r$, notice that the 1 appended at the beginning of the array is a fixture to match the bias. Once we have $r$, we apply the activation function $g$. The output of $g$, which in this context is the considered as the output of the NN, can be taken as the probability that $x_t$ belongs to the class 1. For that reason, the output of $g$ is usually rounded to the closest value to know the most probable classification of the data, $g(r) \approx 1 \vee g(r) \approx 0$, making the classification binary.

The backward-propagation is only used in the training phase and it is the part of the algorithm that grants the model the capability of learning. Once the forward-propagation has been performed, and therefore $f(x_i) = \hat{y}$, which is the prediction of the model over the feature vector $x_i$, it is measured how far $\hat{y}$ is form $y$. For that task, many different loss functions can be used, for example, for binary classification, the binary cross-entropy is commonly used:

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

After calculating the loss value for all the training samples of one epoch, the mean of those values is calculated, which is called the average loss. This value is used to update the weights of the model through the SGD technique [27] after every epochs, expecting to reduce the error value for the next iteration. Thus, if the average loss is expected to get smaller through the epoch, the model is expected to learn and fit to the problem at the same time.

For the learning process, the learning rate hyperparameter ($\alpha$) is critical because a small $\alpha$ can make our model to take too long to learn, and a big $\alpha$ can make the network not to be precise enough to reduce the loss. For this end, different optimizer strategies can be followed.

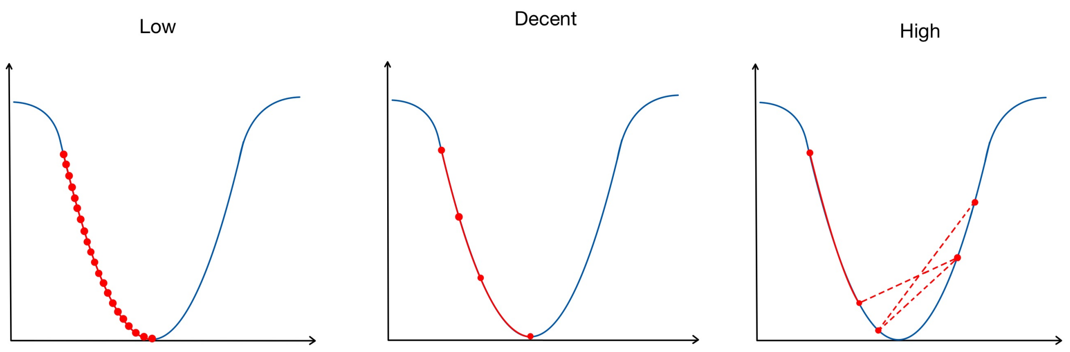

**Figure 2.6:** Importance of the learning rate

In Figure 2.6, we can see three different gradient descent scenarios with three different values of $\alpha$. Where in the $x$ axis we have the value of the loss, and in the $y$ axis we have the value of the loss function, and that each dot represents a gradient descent step. In the low $\alpha$ scenario, the learning takes too many epochs making the learning longer and, as a consequence, much more costly. In the high $\alpha$ scenario what happens is that the learning rate is so big that, at some point, the error value is increased. But, in the decent $\alpha$ scenario, the value approaches securely to the lowest point making the process fast and the error follows a decreasing tendency.

## 2.3 Convolutional Neural Networks

A CNN [27, 28] is a kind of DNN commonly used in computer vision applications, such as image recognition, image generation, video recognition; speech recognition; time series. Due to its most common applications, the input data is usually a set of images that are represented by a three-dimensional matrix (length $\times$ width $\times$ channels). The third dimension of the images comes from the color channels of the image. Images usually have three channels (RGB) or just one (greyscale).

The CNNs have some special types of layers, convolutional layers, that allow the model to have a better way of processing data types whose internal features are highly related if they are in close proximity to each other, such as images. Usually, most of the layers of the network are convolutional, and at the end of the network, the data is flattened for the last layers, that are commonly dense, to process the data, and retrieve the desired output from the network.

### 2.3.1 Convolutional Layers

The purpose of this type of layer is to extract features from the input data. To perform this task, the kernels of the convolutional layer are displayed among all the dimensions of the input data and feature maps are generated.

The kernel contains weights, and strides across the input applying the dot product to different slices of the input data, reducing its dimensionality and obtaining features from it. An example of a kernel sliding over a feature map is shown in Figure 2.7.

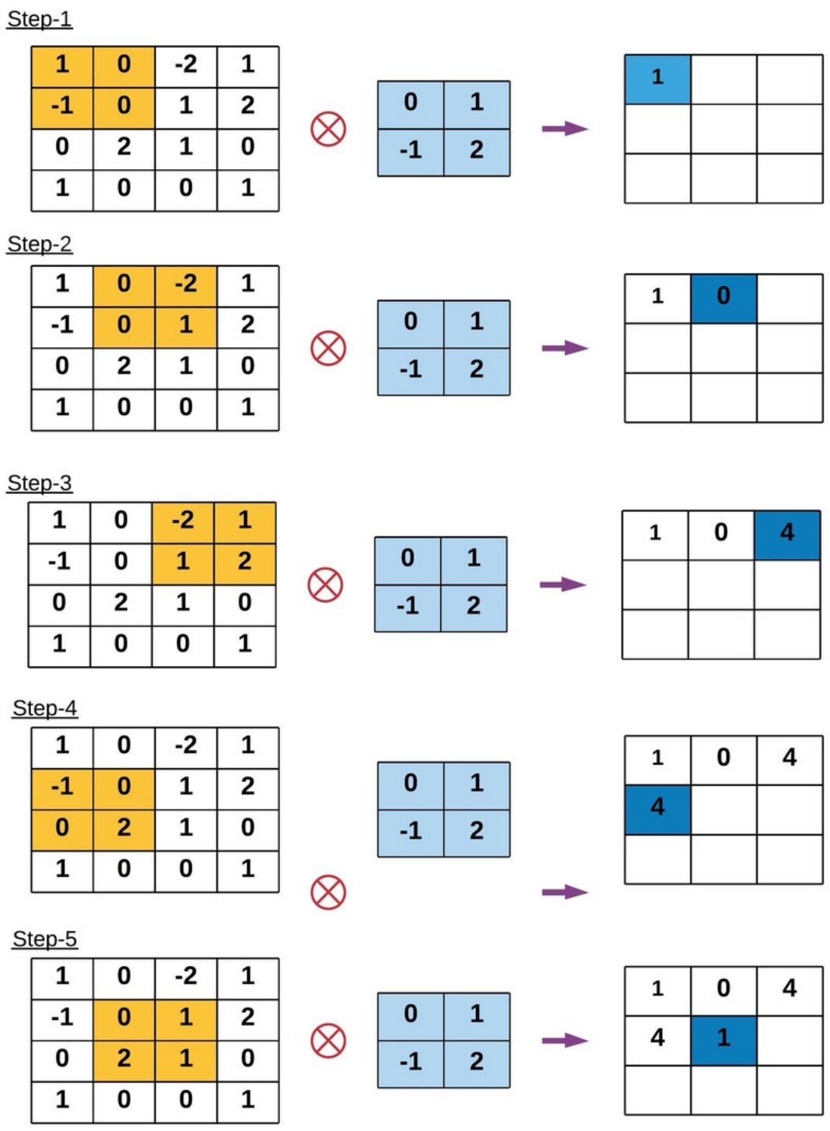

**Figure 2.7:** Kernel performing steps with a stride of 1 over a feature map

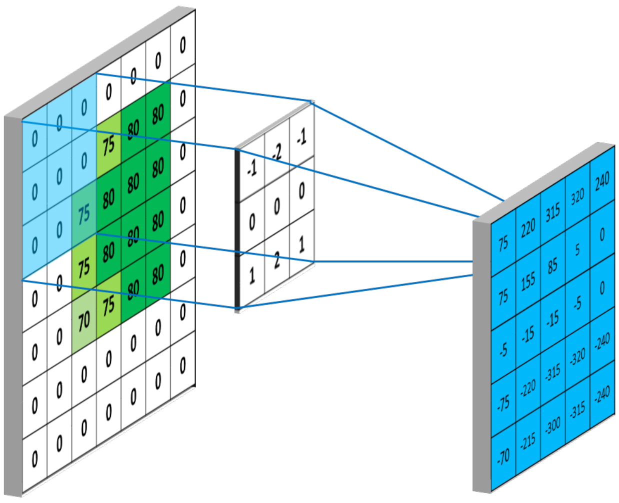

**Figure 2.8:** 3 × 3 kernel convoluting with a 7 × 7 input data

In the convolutional layers, the feature maps are expected to have their size reduced. In Figure 2.8 we can see one feature map being reduced through a kernel of a convolutional layer.

### 2.3.2 Transposed Convolutional Layers

The transposed convolutional layers [29] perform the opposite operation of the convolutional layers, the feature maps are expected to have their size incremented. The transposed convolutional layers take feature maps as input data and transform them to bigger feature maps or, sometimes, to similar data types, such as images. In essence, they are used for upsampling the input data to a higher resolution.

This kind of layers work by reversing the forward pass of a common convolutional layer. A transposed convolutional layer strides a kernel over the output feature map and computes the dot product between the kernel and the output at each location, as shown in Figure 2.9.

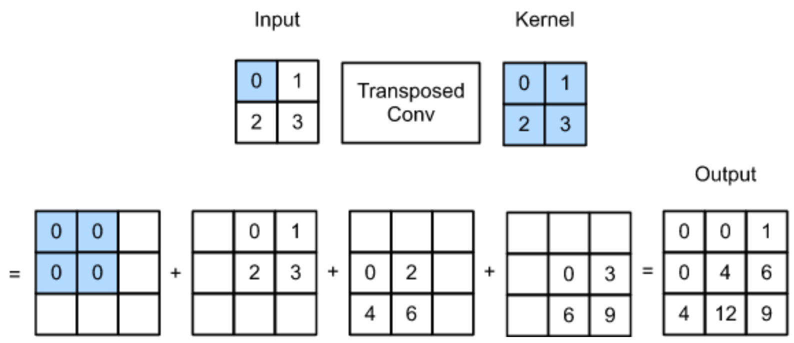

**Figure 2.9:** $2 \times 2$ kernel deconvoluting with a $2 \times 2$ input data

### 2.3.3 Pooling Layers

The pooling layers [30] are used to reduce the dimensionality of the data and, as a consequence, reduce the computational complexity of the model, making it faster to train and evaluate. To perform this action, a kernel that applies a simple operation to reduce dimensionality (average or maximum) is used. The operation is performed among the different data matrices that serve as input for the layer as depicted Figure 2.10.

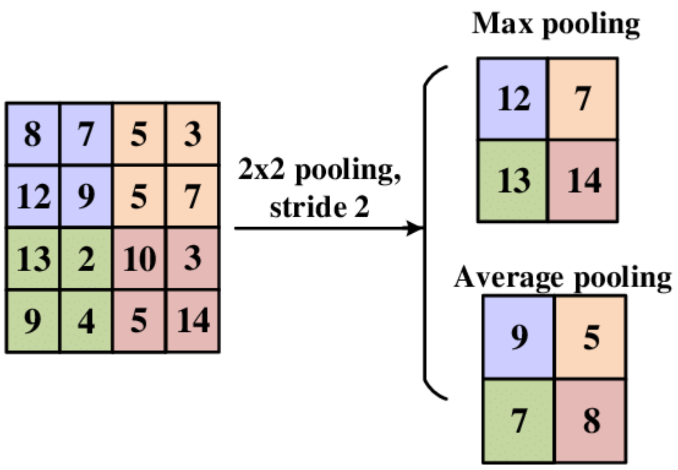

**Figure 2.10:** Average and Max-pooling with a $2 \times 2$ kernel size and a stride of 2

CHAPTER 3

# Variational Auto-Encoder

## 3.1 Auto-Encoder

The Auto-Encoder (AE) is a model whose architecture is very similar to the VAE. Indeed, the VAE inherits its name from the AE because of it. For that reason, before talking about the VAE, it is important to talk about the model from which inherits its name.

### 3.1.1 Concept

The main parts of the AEs [31] are an encoder, a latent space, and a decoder. These three parts are the main parts of the AE's architecture.

The latent space of an AE, which is calculated through the encoder, should ideally meet more characteristics than only being low-dimensional. A perfect latent space would also be continuous, disentangled, informative, and robust. This means that small changes in the latent space should lead to small changes in the output data, the different latent variables should correspond to different interpretable features of the input data, and the the latent space should learn a meaningful and continuous representation of the input [28].

The AE can be used for reducing the noise of the input data. Compressing the input data allows the model to only focus on relevant areas of the input and, when reconstructing it, the input data is denoised [32].

### 3.1.2 Architecture

The encoder consists of one or more layers of neurons, which are often dense layers or convolutional layers, that apply a series of transformations to the input data to obtain a reduced representation in the latent space.

The decoder is usually a mirror of the encoder, although asymmetric AEs exist [33], consisting of one or more layers that take the latent representation generated by the decoder and transform it back to the original input space. The final output is a reconstructed version of the input data that should be as close as possible to the input data. An example of an AE is shown in Figure 3.1.

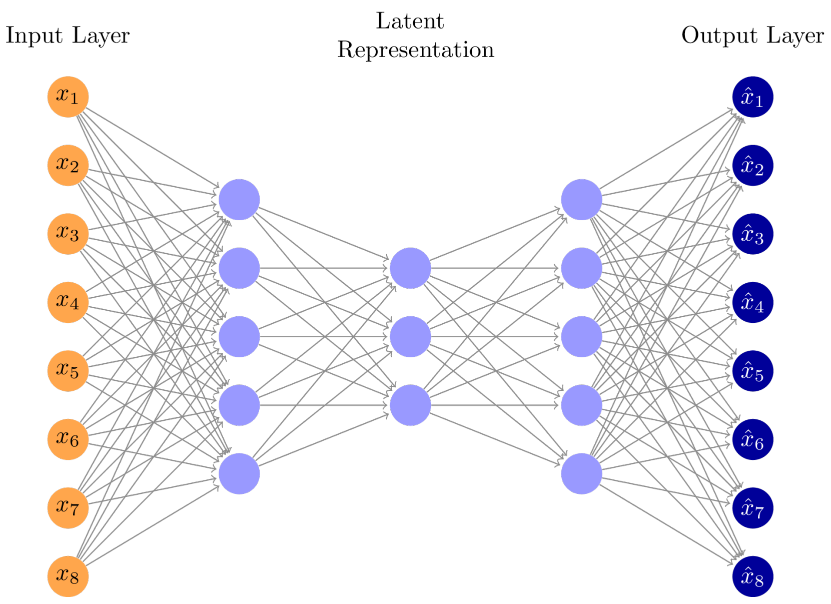

**Figure 3.1:** Example of an AE neural network

## 3.2 Architecture of the VAE

The VAE has an encoder NN that takes the input data and reduces it to a latent representation which is interpreted as series of means and standard deviations that define a series of statistical distributions. Once the input data has been transformed to a series of distributions, commonly just one sample of each is taken as the input of the decoder NN that reconstructs the input data as the output. We can see the architecture schema in Figure 3.2.

In further detail, given an input $x$, it is processed by the encoder through a series of transformations to get $\mu_x$ and $\sigma_x$, which are vectors that contain the means and standard deviations of the latent representation respectively. Then, $z$ is calculated, which is a sample of the distributions that those means and deviations represent, $z \sim N(\mu_x, \sigma_x)$. At the end of the process, $z$ serves as the input for the decoder, resulting on the reconstructed input $x'$, which is expected to be as similar as possible to the input, $x' \sim x$.

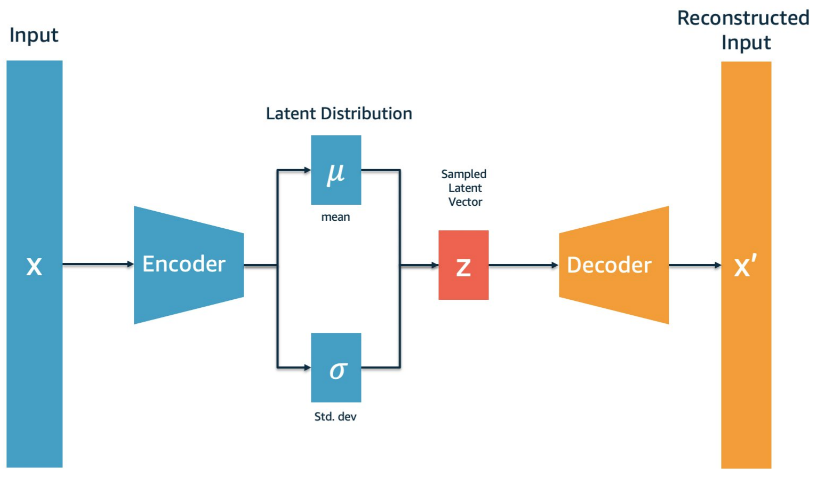

**Figure 3.2:** Architecture of the VAE

As the idea of the VAE is to generate brand new data, it is not only trained to reconstruct the output as accurately as possible, but it is also trained to generate latent distributions similar to well-known distributions, such as the Gaussian distribution [34]. This target distribution is commonly used because it makes the calculation of the KL divergence fairly simple as a closed version of the KL exists when assuming that one of the distributions is $\mathcal{N}(0, 1)$.

The full architecture of the VAE is only used in the training phase. After the training, when inferring data, random samples of the target distribution are used as the input of the decoder, which will generate new data within the distribution of the training set. This is why the target distribution has to be known, so it is possible to sample it later. In this phase, that can be seen in Figure 3.3, the encoder is no longer used unless the model starts a new training phase.

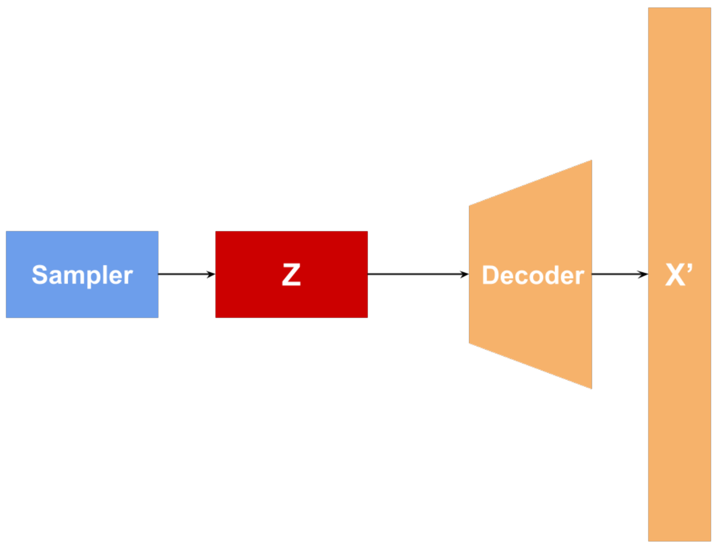

**Figure 3.3:** Generation of new data through the VAE

## 3.3 Training

The basis of the training phase are very similar to other models' training. A SGD algorithm is applied, such as Adam [35], a loss function and backward-propagation.

The loss function is always a key for the learning of a NN, but in the case of the VAE, it is also critical because there are two learning processes happening at the same time, and one single loss function has to measure both of the errors. These learning processes are, as mentioned above, learning to recreate the data and learning to encode the input as a specific distribution.

For the reconstruction, any loss function that measures the difference between two pieces of data of the same type would fit, for example, the MSE. And, for the distribution, as defined in [13], the KL divergence is used.

For more detail, if we consider a dataset, as previously defined, $X = \{x^i\}_{i=1}^N$. And we have the probabilistic encoder model $q_\phi(z|x^i)$, that takes any sample of $X$ as input and transforms it to the latent vector of samples $z$. In a similar way we have the probabilistic decoder model $p_\theta(x^i|z)$, that takes $z$ as an input and produces $x^i$. The loss function would have this form:

$$\mathcal{L}(\theta, \phi, x^i) = -D_{KL}(q_\phi(z|x^i)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x^i)}[\log p_\theta(x^i|z)]$$

Where $D_{KL}(q_\phi(z|x^i)||p_\theta(z))$ is the function for measuring the difference between the target distribution that the model should seek $p_\theta(z)$ and the distribution we obtain encoding the input data $q_\phi(z|x^i)$. And, $\mathbb{E}_{q_\phi(z|x^i)}[\log p_\theta(x^i|z)]$ defines the expectancy that, from the latent distribution $q_\phi(z|x^i)$, the model obtains $x^i$ completely well reconstructed, $\log p_\theta(x^i|z)$, so to say, the reconstruction loss value.

### 3.3.1 Reparameterization Trick

There is a problem with the training of the VAE if the backward-propagation is used within the training of the model, and it is that, as there is a sampling process in the training process to obtain the latent representation, there is a stochastic component that makes that section of the model non-differentiable. This means that it is impossible to train a model like this, because the gradients cannot be computed.

As a solution to that problem, the stochastic component is isolated in a vector $\epsilon$. So, instead of directly calculating the samples using the means and standard deviations, the values of $z$ are obtained using the reparameterization trick: $z = \epsilon \odot \sigma + \mu$. This way, the stochastic component can be isolated, allowing the training of the rest of the model.

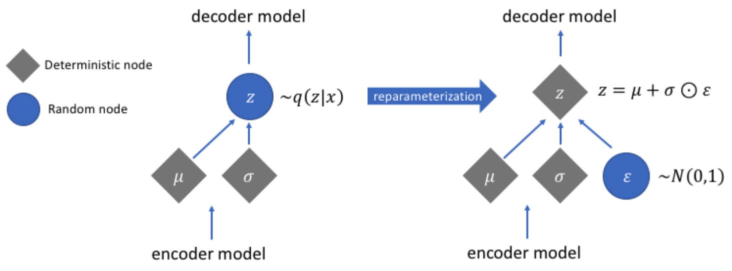

**Figure 3.4:** VAE schema before and after the reparameterization trick

In Figure 3.4, whose left end is a zoom of the latent distribution shown in Figure 3.2, we can see that the random node is, at first, between the encoder and decoder models. That is why it is not possible to back-propagate. When the random node is isolated in $\epsilon$, as in the right end of the figure, we let a differentiable path to back-propagate and update all the gradients every epoch.

CHAPTER 4

# State of the Art

The VAE was originally proposed in 2013 by Kingma and Weiling. Since that moment, the model has been used for generating automatically different types of data like music [36] or images [37] with remarkable outcomes.

The VAE tries to maximize the lower bound of the log-likelihood of the data, resulting in a loss of detail of the generated data. The latent space of the VAE, most of the times, is not directly interpretable, which makes it difficult to understand the features that are being mapped to the latent space. At the same time, the original model produced blurry images because it models a probabilistic encoding of the data instead of directly modeling the data distribution. Because of this, the loss function is a composition of two different loss functions with two completely different magnitudes.

To improve this aspect of the original model, variations of it have been proposed over time in search of improving its performance. Most of these variations consist of changes on how the latent space is modeled, which is an essential part of the VAE. Some changes suggest the usage of different prior distributions for the model or adding extra features to the model.

Eric Nailsnick and Padhraic Smyth, in 2017, described how to use the VAE's learning algorithm for posterior inference for the weights of stick-breaking processes. As the Beta distribution, which would be the normal choice for an approximate posterior, does not have the features that the learning algorithm requires, the Kumaraswamy distribution [38] was used. Using the Kumaraswamy as an approximate posterior, the Stick-Breaking Variational Auto-Encoder (SB-VAE) [21] was proposed, which is a Bayesian non-parametric version of the original VAE.

The SB-VAE draws the latent variables form a GEM distribution, making the hidden representation an infinite sequence of stick-breaking weights. The generative process of the model is very similar to the process of the original VAE. In mathematical terms, where $\pi_i$ is the vector of stick-breaking weights and $\alpha_0$ is the concentration parameter of the GEM distribution from where the latent variables are drawn.

$$\pi_i \, GEM(\alpha_0)$$

The inference process of the SB-VAE requires modification from the standard VAE's in order to sample for the stick-breaking reconstruction. An inference network computes the parameters of $K$ fraction distributions and samples values $v_{i,k}$ according to one parameterization. Next, an operation composes the stick segments from the sampled fractions.

$$\pi_i = (\pi_{i,1}, \pi_{i,2}, ..., \pi_{i,K}) = \left( v_{i,1}, v_{i,2}(1 - v_{i,1}), ..., \prod_{j=1}^{K-1} (1 - v_{i,j}) \right)$$

The optimization of the SB-VAE is the same as the one of the original model. The KL divergence term can be closely approximated for the Kumaraswamy distribution.

In the paper where the SB-VAE was proposed, the ability of the model for recreating the data and preserving the class structure was tested and compared to other models. For the experimentation, the original VAE had a twenty-five dimensional latent distribution, so, the SB-VAE had set its truncation level to the same number, $K = 25$.

After the experimentation with the SB-VAE, some conclusions were obtained. Firstly, that the only extra computational cost of the proposed model compared to the previous ones resides in assembling the stick segments, which is a linear operation on the order of $K$. And finally, that although the original VAE converges to a better likelihood, the SB-VAE outperforms the original model making its latent space capture the class structure.

The same year the SB-VAE was proposed, another important variant for the original model was proposed. The motivation for this new variant was to find a model capable of learning a disentangled posterior distribution over the underlying generative factors of sensory data without a prior knowledge of the nature of these factors. Under this context, the $\beta$-VAE [22] was proposed.

The way this disentangling task is achieved is by adding a single hyperparameter $\beta$ to the model. This new hyperparameter modulates the model's learning constraints that impose a limit on the capacity of the latent information channel and control the emphasis on learning independent latent factors. $\beta$ is applied to the $\beta$-VAE's loss function being multiplied to the KL divergence.

$$\mathcal{L}(\theta, \phi; \beta, x, z) = -\beta D_{KL}(q_\phi(z|x^i)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x^i)}[\log p_\theta(x^i|z)]$$

The $\beta$-VAE with $\beta = 1$ would match with the original VAE model. With $\beta > 1$, according to the paper where the $\beta$-VAEis proposed, the model is pushed to learn a more efficient latent representation, which is disentangled if the underlying structure of the data allows it.

To measure the disentanglement and interpret the results, a new measure is proposed in the paper. This new metric measures the independence and interpretability of the latent spaces. To apply it, inference of a number of data examples is run by fixing the value of the data generative factor while randomly generating all others. If the independence and interpretability properties hold for the inferred representations, there will be less variance in the inferred latents that correspond to the fixed generative factor. Then, a low capacity linear classifier is used to identify this factor and its accuracy is taken as the disentanglement metric score. A linear classifier is used to ensure that this model is not able to learn non-linear disentangling by itself.

The experimentation of this new variation using the self-created metric showed that, overall, $\beta$-VAE tends to consistently and robustly discover more latent factors and learn cleaner disentangled representations than the other models the $\beta$-VAE was compared with. Another good point of the $\beta$-VAE compared to other generative models is that, as it was shown, the $\beta$-VAE requires no assumptions about the data, and is very stable to train.

The $\beta$-VAE, when compared to the original VAE, is able to consistently learn significantly more disentangled latent representations. Also, it was observed that increasing $\beta$ often leads to better disentanglement, but its cost is to get blurrier reconstruction and losing representations for some factors, particularly those that correspond to minor details of the data.

More recently, in 2019, the Dirichlet Variational Auto-Encoder (DirVAE) [20] was introduced. This approach uses a Dirichlet [39] prior distribution. The DirVAE is able to model the multi-modal distribution that was not possible with other approaches.

The Dirichlet distribution is a composition of the Gamma [40] random variables, so the DirVAE approximates the Gamma cumulative distribution function with the asymptotic approximation.

The DirVAE is a result of the investigation of the component collapsing of the VAE. This issue was already palliated by the SB-VAE, but this model has a latent value collapsing issue due to many resulting near-zero values on its latent representation. The proposed DirVAE shows neither the component collapsing nor the latent value collapsing issues. This happens because of the multi-modal prior of the DirVAE.

The model proved to have interpretable latent dimensions in most of the cases, and not suffering of any of the previously mentioned collapsing issues. The DirVAE showed to utilize the full spectrum of latent dimensions, having a really good capability of learning in the decoder network.

The examples of improvement proposals given above are not the only ones that have been registered since 2013, but some examples of how the state-of-the-art related to the VAE has evolved through the time. Other good examples of improved VAE's variations could be [41] or [42]. Besides, during its existence, the VAE has not only had improvement proposals, but also has been proposed for improving other models. For example, using it as the generator NN for a GAN model [43].

All of the research behind the VAE and its usages until this date, show how much potential this variational model has. One of the most impressive recent usages for it, has been utilizing a variation of the VAE to research the transformation of mice neural waves to videos that represent what the animals were seeing by that moment [44].

As it can be noticed among the different variations, the prior distribution of the model is one of the most-frequently chosen variations for the original approach. This is nothing else but a reason to develop a coding library to work over these variations and to provide tools to enhance the creation of more variations and expand the idea of the model.

CHAPTER 5

# Implementation

Any formal project that requires an implementation, should follow a well-defined structure and some development rules that promote good programming practices. Although the addressed problem in this project is not strictly a development problem but an investigation about the VAE, a quality code-development makes the work more accessible to everyone and leaves the door wide open to future contributions to the project from external researchers or developers.

A code development only focused on being functional is a widely scattered practice in the world of ML research. This type of code is not easily scalable and, usually, is not very useful for future investigations due to the lack of flexibility that that type of code provides. It is a fact that quality code takes more time to be developed, and many times that extra amount of time is not well-received by researchers that make the decision of developing code that *just works*, which is better by short-term means and will only benefit their current research, instead of taking time to develop code that not only works, but is also well-built and could be helpful for future researches.

It might be a way too ambitious objective, but, what this structured and exhaustive development aims to become a library that is used in future researches or for educational purposes about the VAE in the future, and not to become another forgotten development of a ML research project.

## 5.1 Version Control System

Version control systems are a very powerful tool for developing any programming project. It allows the developers to produce code in an ordered way, making the task of group-programming much more agile, and to easily deploy open-source libraries. One of the most used version control systems is Git [45]. It is also highly recommended to use these kind of systems even if the development is individual because they allow a very ordered and documented development that is always helpful in any project.

In Git, each developer has a local copy of the repository they are working on, and all the changes that they perform can be committed and pushed to a remote repository. This technology uses snapshots of the code called commits to track changes and create a history of the project. Branches can be used to work on separate features or versions of the project, and merging allows developers to incorporate those features or versions from one branch to another. Git also allows developers to collaborate easily to request merging the changes from one branch into another branch, with what are called Pull Requests (PRs) and with code reviews.

For the development of the code related to this bachelor's thesis, GitHub [1] has been used. It is a web-based platform for projects that use Git version control system.

### 5.1.1 GitHub Setup

For ensuring the quality of the developed code, any GitHub repository needs a suitable setup. The most basic part of the setup process is adding a readme file to grant some documentation and a license to the repository to define how the project can be used by third parties. This project has been granted with an MIT license, which is defined in Table 5.1.

| Permissions | Limitations | Conditions |
|---|---|---|
| Commercial use | Liability | License and copyright notice |
| Modification | Warranty | - |
| Distribution | - | - |
| Private use | - | - |

**Table 5.1:** MIT License features

After the creation of the repository, the main branch, also called master, was protected by the usage of basic rules. Those rules state that for merging into master, it is necessary to create a PR from another branch. Also a dependabot was activated to inform of any dependency issue.

For ensuring a unique formatting of the code, a GitHub workflow was created that is executed on every pull request to check if the code follows the Black [46] formatting. This lint check is executed as a GitHub action on every pull request and works as a format checker. If the lint fails, the corrections should be pushed before merging the branch of the PR.

### 5.1.2 Workflow

Every new feature for the project should follow a strict process before being incorporated into master. This way, the task of programming different features in parallel becomes structured and ordered, allowing the project to reach the scoped objectives with ease and making it highly scalable.

For incorporating any feature, one or several branches are created with descriptive names. These branches usually are created from master, unless the branch is creating a sub-feature for some other branch. After the implementation of the features, one PR per branch is created to merge the changes into, eventually, master. Once a PR is created, the lint checker previously mentioned is run. If it fails, the Black formatting must be passed

[1]Link to the GitHub repository of the implementation

through the files that do not follow the formatting. After having a successful lint run, for the lack of more developers, a self-review of the code must be performed before merging the changes. This strict process helps to minimize the errors of the code and adds an extra layer of reflection to the implementation that is highly helpful for complex projects.

## 5.2 Code Structure and Organization

### 5.2.1 Architectural Pattern

Any high quality coding project should follow an architectural pattern. Following a good software architectural pattern helps to define the performance, quality, scalability, maintainability, manageability and usability of one project. This way, the software is flexible, extensible and can evolve as needed.

Although this thesis is not strictly about the development of quality code, it is still a essential part of the project. All the experimentation is built over the foundations of the code implemented. If that code is not well-structured and clean, it will cause the experiments to be more difficult to replicate. Also, if anyone wants to take this work as a basis for future development, developing quality code that follows design patterns and architectural patterns, would facilitate that task, contributing to the success of future investigations with a higher level of complexity.

There is precedent for data science projects that have developed quality code that has served as basis for further investigations. One very good example is Keras [16], which is at the same time, built over TensorFlow [15]. This project is going to add another layer to this schema, and be built over Keras and TensorFlow, going a bit further on the development of VAEs.

As the development work for this bachelor's thesis has been a library, which will be used for creating the experiments, it is not needed a really complex or strict architectural pattern because there will not be external factors like developing endpoints, accessing databases, etc. Thus, the chosen architectural pattern has been the Domain Driven Design (DDD) [47] but slightly adapted to the task that is being faced.

The DDD is a software development approach that focuses the development on programming a domain model that has a rich understanding of the processes and rules of a domain. In software development, domains refers to business and is related to business logic. As shown in Figure 5.1, the DDD architectural pattern has three main layers: domain, application and infrastructure.

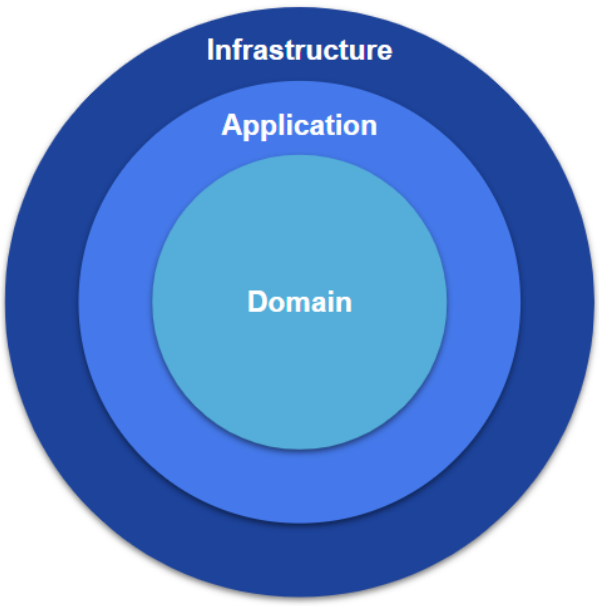

**Figure 5.1:** Layers of the DDD

The domain represents all the programming concepts, information and rules. The domain should be planned in a way that it persists different changes to the implementations. It should define the relationship between domain entities and the operations that are performed. Includes aggregates, entities, value objects, services and interfaces.

The application is responsible for interacting with the user or other applications. It is the gateway of any project. Every interaction with an application should be made through the application layer.

Finally, the infrastructure is the layer responsible of persisting the data, if needed. It also includes the implementation of the interfaces of the domain.

The reasoning behind separating an application in these three layers is to make them relatively independent of each other. The domain defines all the classes and relations of the project, as well as the operations that are possible from each domain object. The domain should not import anything from external projects, the application or the infrastructure, basically, should not import anything that defines the implementation. Then, the infrastructure should be the layer importing from external projects, to define a precise implementation, and from domain, to either use the resources defined in the domain or to implement the interfaces. Finally, the application cannot implement anything related to the logic of the project, it should only use the code inside the domain and infrastructure to allow the communication of the application with external actors.

For this project, the DDD has been taken as the basis, but it has not been strictly applied. The DDD architectural pattern has been properly adapted and customized to the particularities of the project. As the project's scope is reduced to a TensorFlow and Keras implementation, references to both of those projects can be found in the domain layer, which, according to a pure DDD architectural pattern should not appear. Also, the domain is limited to exceptions and interfaces, and following the same logic, the infrastructure is limited just to implementations of the domain's interfaces. The application layer, due to the nature of the project, is not needed in most of the cases, because the project does not aim to implement a functionality that can be used on user level.

### 5.2.2 Structure and Implementation Details

The code has been implemented using Python 3.9 [48] with typing. Typing has been added to the project to add clarity and order to the implementation. In Python, the typing is not strong because of the interpreted nature of the language. It is only used to help in big or scalable projects to keep type-coherency, if some variable is incorrectly typed, the Integrated Development Environment (IDE) will show a warning.

The whole programming project, according to DDD, has two different applications: Project and Utils. In DDD we call application to a set of domain, infrastructure and application. The application called Project is the core of the project itself, that is the reason of such an imperative name, inside this application the domain and infrastructure of the different VAE model implementations can be found.

Each utility implemented in the Utils application has its own sub-application itself. The sub-applications inside Utils are three: Batches, Epsilons and Losses.

The Batches sub-application is a single application for the batch managing of a model. A batch is a sample of training examples utilized in one iteration. The different batch strategy implementations can be seen in the class diagram of Figure 5.2.

The common batch divides the whole dataset in different slices of the same size but the last one, which is a remainder, just like the random batch. The strict batch does the same but discards the remainder, the same happens with the strict random batch. The difference between these batches and their random versions is the way the slices are selected. The common and strict batches return the corresponding slices in order until they run out and then, they stop. The random and random strict batches always give a random slice and never stop. Then, the cyclic batch calculates the slice when it is queried. To do that, it transforms the dataset to a circular array, so there will always be a next slice, making it never stop returning slices.

The Epsilons sub-application is where the epsilon for a VAE model is generated. A VAE needs to generate an epsilon in order to perform the reparameterization trick, and this epsilon can follow different strategies for being generated. Each of these strategies are expected to affect to the training process of the model. As it is visible in the class diagram of Figure 5.3, six different strategies have been implemented, and we can see a clear definition in Table 5.2, where the name of the class is given within a brief definition of the strategy and a flag that marks if a fixed random seed is used in the strategy or not. Using a random seed means that, among different executions, the strategy will always have the same "random" results.

| **Name** | **Definition** | **Uses random seed** |
|---|---|---|
| AllRandomEachTime-EpsilonGenerator | Every epsilon applied to every sample of the batch will be random each time the epsilon set for the batch is retrieved. | No. |
| AlwaysSameEpsilon-Generator | It will generate an epsilon set with the same epsilon in each row for each batch sample. This epsilon is always the same when retrieved. | Yes. |
| AlwaysSameEpsilonSet-Generator | It will generate an epsilon set where each epsilon for each batch sample is random, but it will always retrieve the same set. | Yes. |
| OneEachTimeEpsilon-Generator | It will return an epsilon set with the same epsilon for each batch sample, but each time the set is retrieved, the epsilon is different. | No. |
| SameEpsilonGenerator | It will generate an epsilon set with the same epsilon in each row for each batch sample. This epsilon is the same every time it is retrieved. | No. |
| SameEpsilonSetGenerator | It will generate an epsilon set where each epsilon for each batch sample is random, but it will retrieve the same set every time. | No. |

**Table 5.2:** Different epsilon strategies

To end with the Utils application, we have the Losses sub-application, which is a very small sub-application that has neither domain nor infrastructure, it just has an application package where the *ImageLossFunctionSelector* is located. As the loss functions do not need any kind of domain, they can just be implemented in the code of the selector itself.

By implementing new loss functions in the Losses sub-application, and adding the to the *ImageLossFunctionSelector*, new loss functions are completely added and integrated to the application. This way, new learning strategies or new models can be implemented efficiently and without any changes anywhere else in the code.

The Project application has all the models implemented. This application only has the domain and infrastructure packages because the application package of DDD is not needed. In the domain, the *VAEModel* is implemented with most of its functions being abstract functions and only a few extremely generic ones are implemented. Then, in the infrastructure package, the implementation of the *ImageVAE* can be found. This implementation is a middle point between the implementation of the functional VAE models for images and a generic interface. This is not a decision based on the DDD, but on the principle of not repeating code and generalization. The two main functional implementations are the common VAE, implemented with MLPs; and the Convolutional Variational Auto-Encoder (CVAE), implemented with CNNs. The Stick-Breaking Convolutional Variational Auto-Encoder (SB-CVAE) and SB-VAE inherit directly from the previously mentioned main functional implementations and add some restrictions to the original models in order to easily implement these new ones. The class diagram of this application in Figure 5.4 gives a full overview of how it is structured.

As it is visible in the diagram, every parameter of each model can be accessed via a getter method. This pattern will allow to analyze to the maximum the characteristics of the models by not compromising the integrity of the parameters. Also, all the parameters are protected. This is because, from the design point of view, it is a very good practice that allows a model to be scalable and use it freely while making other models inherit from it. At the same time, this practice does not allow to freely modify the parameters of the model outside it, granting some security to the implementation.

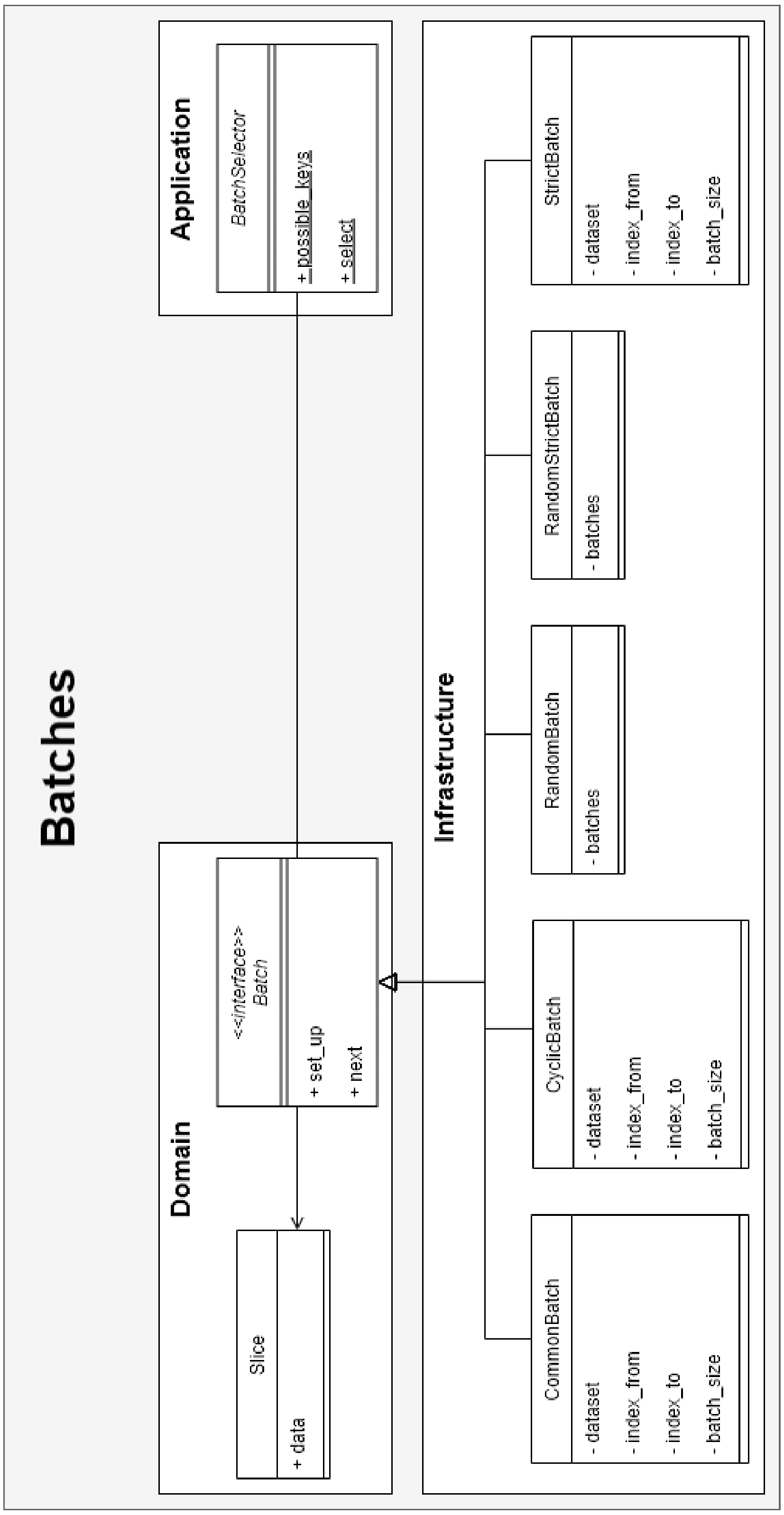

**Figure 5.2:** Class diagram of the Batches application

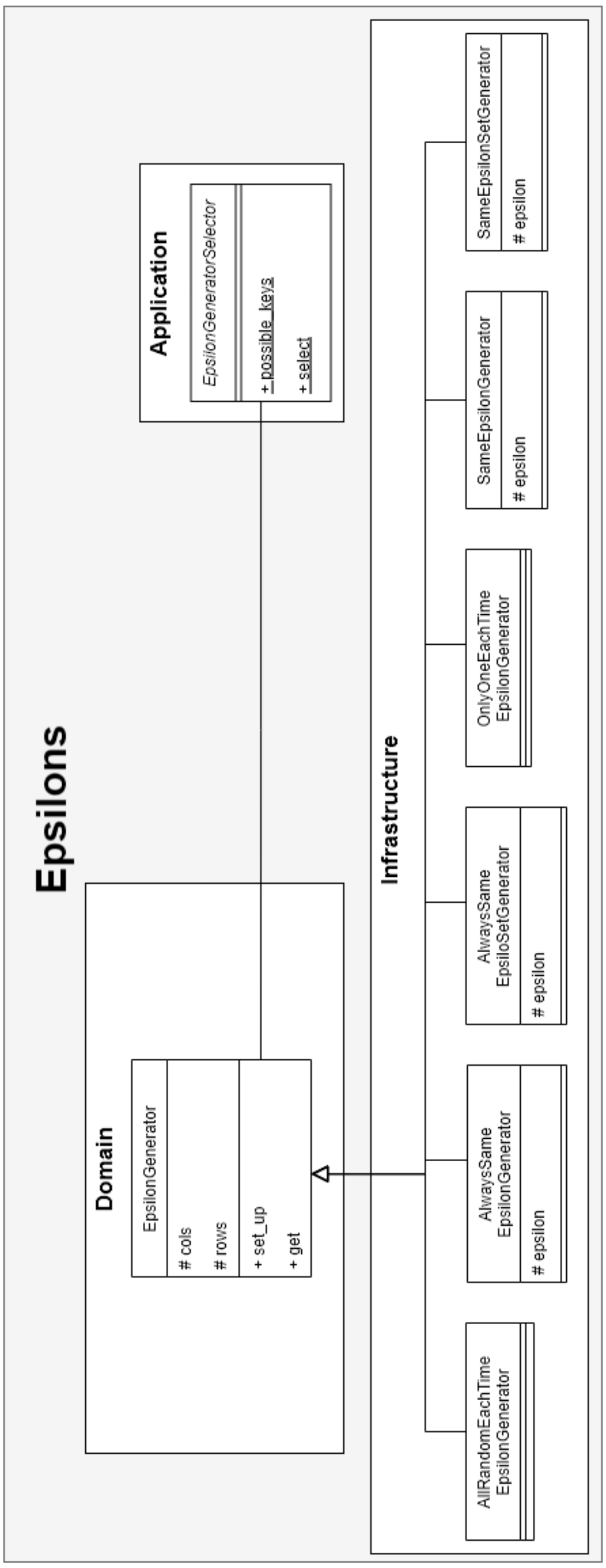

**Figure 5.3:** Class diagram of the Epsilons application

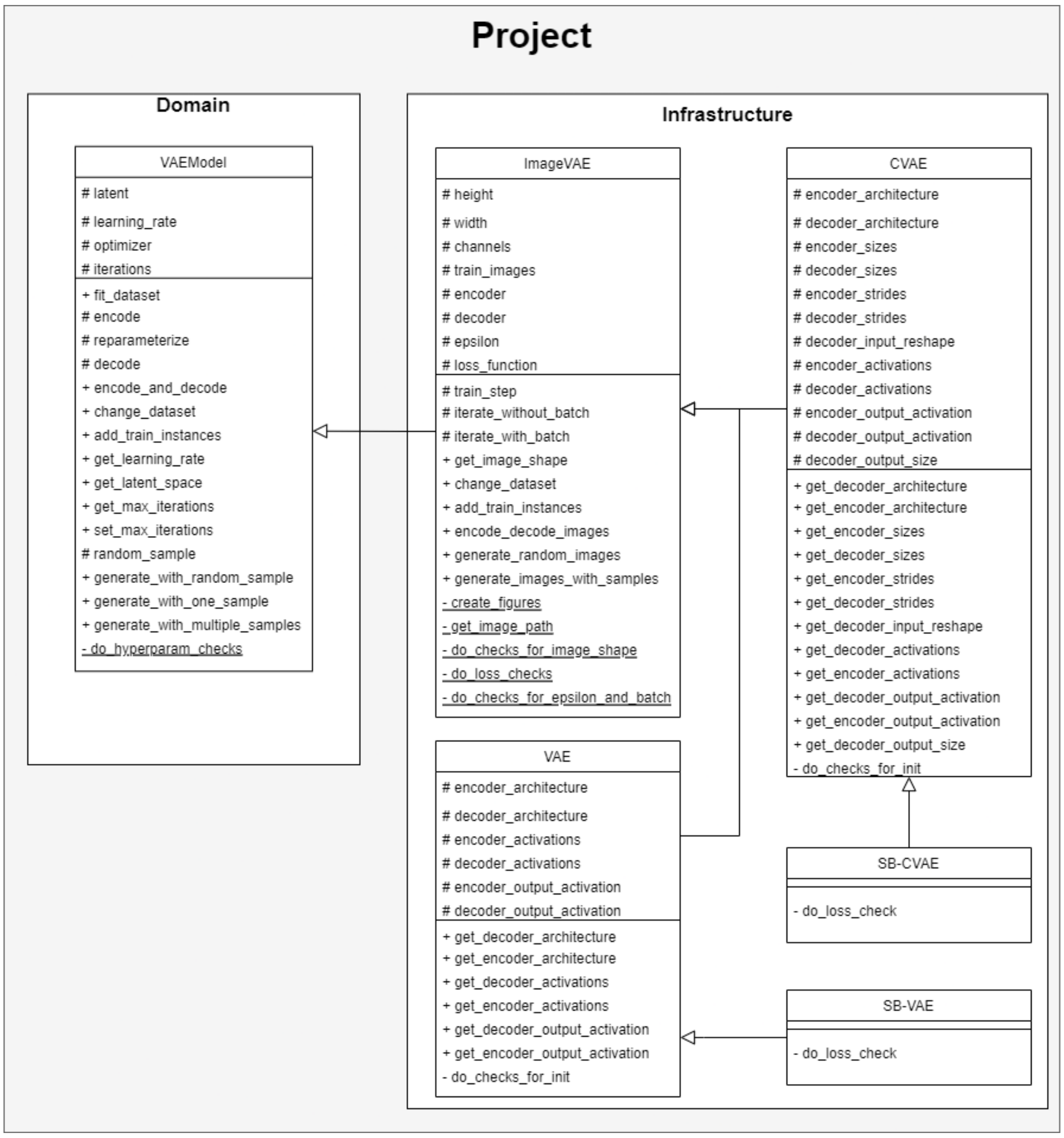

**Figure 5.4:** Class diagram of the Project application

CHAPTER 6

# Experiments and Results

## 6.1 Experiments

As we have observed in the state of the art, the training method of the VAE is certainly conflicting, so it would be ideal to find a way of making it simpler. The next logical step to reach that goal is to make experiments over the model for a better understanding of it, and its variations. To enable us to conduct the experiments successfully and without further complications, the usage of a library that could enable us to work easily and with flexibility with the model would be ideal. For the lack of libraries with the mentioned features, **tfVAE** was created within this bachelor's thesis and used for the experiments [1].

### 6.1.1 Methodology

The model has two main variations: the VAE (uses MLPs) and the CVAE (uses CNNs), and every experiment detailed in this section has been performed symmetrically for both main variations. Due to the limitations of the work, each experiment was executed once.

We know that any NN model is highly dependent of its architecture, and so is the case of the VAE. To discover one architecture for each variation of the model that works correctly and has certain guarantees, some preliminary experiments have been carried out.

These preliminary experiments were performed using the Modified National Institute of Standards and Technology (MNIST) dataset [49]. This dataset consists of $60,000$ greyscale images of dimension $28 \times 28$ that represent handwritten digits (ten different categories), as depicted in Figure 6.1. It is widely used in for training image processing systems due to its size, allowing to quickly check and prototype models.

[1]Link to the experiments in GitHub

**Figure 6.1:** MNIST dataset preview

After the preliminary experiments, from which we selected two different architectures for the MLP and CNN variations. Two main experiments have been conducted for each variation. The first experiment consisted of adding the $\beta$ hyperparameter ($\beta$-VAE and $\beta$-CVAE) to the models, and the second experiment consisted of using the stick-breaking processes to draw the latent variables (SB-VAE and SB-CVAE).

To test the generalization capabilities of the selected architectures, the main experiments after the preliminary ones will not only be performed over the MNIST, but also the Fashion-MNIST dataset [50]. It is a database that follows the same structure of the original MNIST ($28 \times 28$ greyscale images with ten categories), some instances are shown in Figure 6.2. It represents different articles of clothing, and was initially designed to be a direct drop-in replacement for the original MNIST.

**Figure 6.2:** Fashion-MNIST dataset preview

### 6.1.2 Evaluation

The different models in the preliminary experiments were evaluated using two different metrics. The first one used was the Total Variation Distance (TVD) [51]. This metric is a fundamental notion of distance between probability distributions. It is the highest difference between the probabilities that two probability distributions that are being compared can assign to the same event. Explained in mathematical terms, we have a dataset $W$ with ten different categories $D \in \{0,1,2,3,4,5,6,7,8,9\}$ and a dataset $V$ generated from training a generative model with the previous one, so its category group can also be taken as $D$. The probability measures for both datasets are $P$ and $Q$ respectively. Then, the TVD would be the following:

$$TVD(P,Q) = \max_{E \in D} \|P(E) - Q(E)\|$$

The second metric used was the Frechet Inception Distance (FID) [52]. This metric is a metric typically used to assess the quality of images created by a generative model. It measures the distance between the feature vectors calculated for images from the dataset and generated images. The feature vectors of an image are calculated by intercepting the values of the neurons of a hidden layer of the inception v3 model for image classification [53] when the image has been given as the input. Once that the feature vectors have been extracted, for example, from all the images of the previously mentioned datasets, $\mu_W$, $\mu_V$, $C_W$ and $C_V$ are calculated from the feature vectors of the datasets, that are the feature-wise mean and covariance matrix of both sets. Knowing that the operation $Tr(M)$ refers to the trace linear algebra operation over the matrix $M$, the FID is calculated as follows:

$$FID(P,Q) = \|\mu_W - \mu_V\|^2 + Tr(C_W + C_V - 2 \cdot \sqrt{C_W \cdot C_V})$$

Deciding the best-performing model with two different metrics is not a trivial task. For that, a multi-objective optimization methodology was used as a first approach. After that, one of the metrics was discriminated because of the superiority of the other in consistency terms between the training epochs and the reduction of the value of the metric. During all the experimentation process, greater value has been placed on the generative capability of the model than on the hypothetical quality of the latent distribution.

Nevertheless, due to the high computational cost of the inception v3 model for image classification and its difficult integration with the used datasets, the FID metric was implemented extracting smaller feature vectors from a custom inception-like classification CNN of lower complexity. This model was trained over the dataset the metric was going to be applied on, so when the metric was calculated for images generated from the MNIST, the inception-like model was created and trained only to classify images from this dataset. The same logic was applied with the Fashion-MNIST. The model was also always intercepted in the same point during all the experiments, the layer number six, which is the first dense layer after flattening the output of the last convolutional layer of the model.

### 6.1.3 Preliminary Experimentation

In this section, we will present the preliminary experimentation. To reproduce the whole process, the following steps are given:

1. Train over the MNIST an inception-like model to compute the FID.
2. Conduct a grid search for the model. The grid search is performed over the configuration of the neurons in the layers, because adding more hyperparameters to the grid search leads to an increase of the possible models that is not feasible for the magnitude of this work. During the grid search, compute the TVD and FID of the models, and store the models and architectures with the metrics.
3. Build a Pareto front [54] with the trained models during the grid search. The models inside the Pareto set are in red and the ones outside it are blue. See an example in Figure 6.3.
4. Perform another training but only over the models inside the Pareto set and compute the TVD and FID during the training epochs to observe their evolution.
5. To take the decision of which metric should be used to define the fitness of a model, create a graphic with one $x$-axis which represents the number of different training periods performed and two distinct $y$-axes, one colored in red for the TVD and the other colored in blue for the FID.

   If one metric shows consistency over the training periods and the other does not, the consistent one will be used; if both metrics show consistency, no metric will be discarded; and, if none of the metrics show consistency, the average loss value will be used. See an example of comparison graphic in Figure 6.4.
6. Once decided what kind of metric will be used for now on, train the models in the Pareto set for a very wide number of iterations and store the selected fitness metric or metrics.
7. In case only one metric is being used at this point, select the architecture of the best model as the one that will be used from now on. If both metrics are still being used, select the architecture of the model or models inside the new Pareto set to be used from now on.

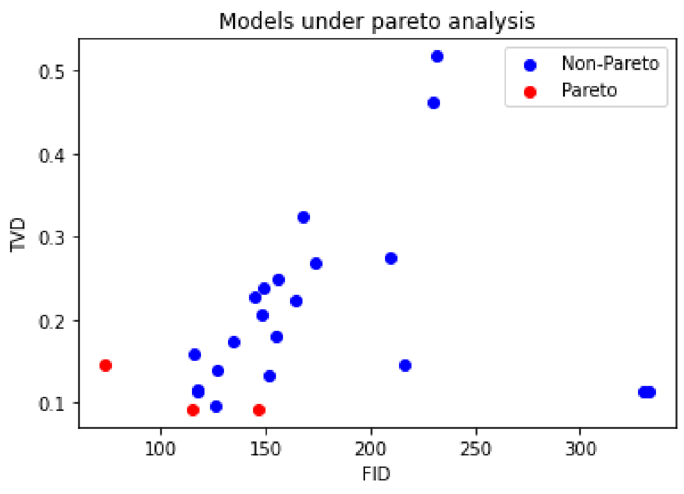

**Figure 6.3:** Example of result of the multi-objective optimization methodology in the preliminary experiment over the CVAE

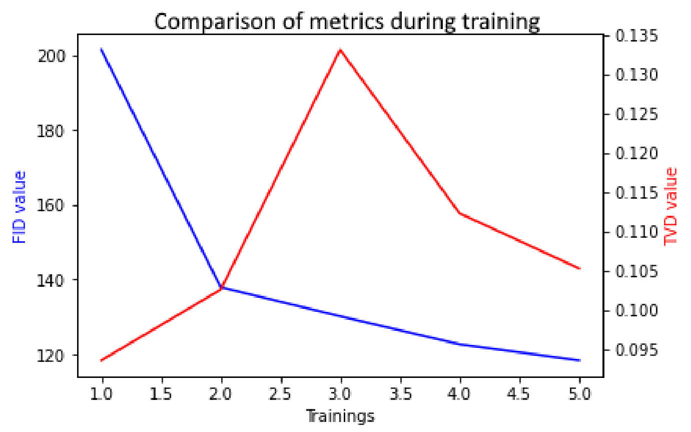

**Figure 6.4:** Example of result of the metric comparison of consistency of both metrics over training periods

### 6.1.4 Main experimentation

During our preliminary experimentation, the FID showed a better performance and consistency than the TVD. For this reason, the FID was selected as the metric for evaluating the models. So the steps followed in the experimentation will be conditioned by that election.

1. Prepare the configurations that will be used for each model of the experiment [2].

2. Train the inception-like model with the current dataset to calculate the FID.

3. Train all the models of the experiment over the dataset using the architectures chosen from the preliminary experimentation results. Compute the value of the FID and extract the loss function values (total, MSE and KL divergence) along the training epochs.

4. After the training of each model, plot the different extracted metrics during the training. See an examples for the FID, loss function, MSE, and KL divergence in Figure 6.5, Figure 6.6, Figure 6.7, Figure 6.8 respectively.

5. In case the experiment has more than one model being trained at the same time over the same dataset, compare all the models over the FID and loss values. Take as the best model the one with the lowest FID.

[2]From now on, all the steps will have to be done for each dataset (MNIST and Fashion-MNIST).

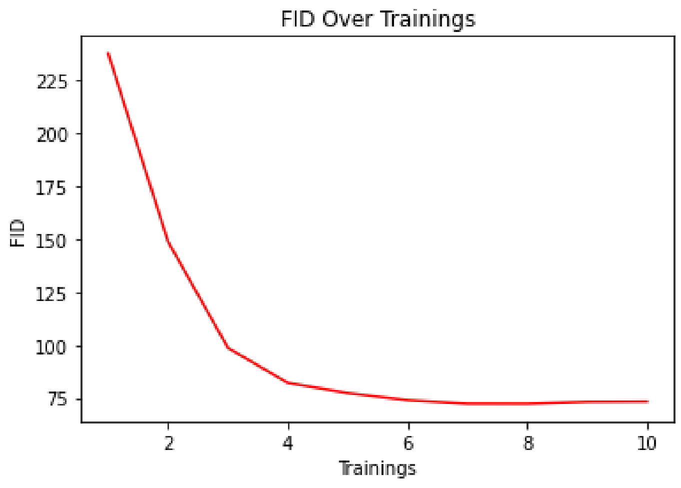

**Figure 6.5:** Example of progress over the iterations of the FID of a model

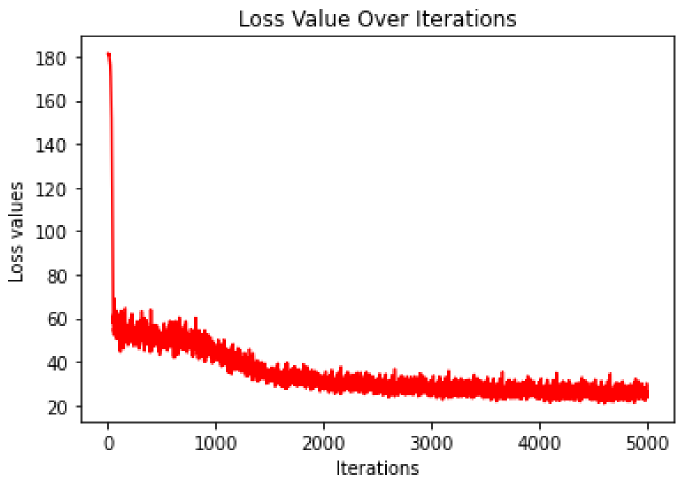

**Figure 6.6:** Example of progress over the iterations of the loss of a model

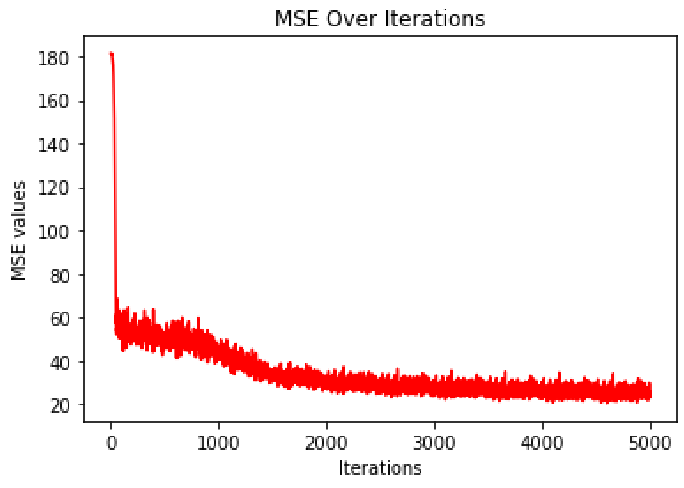

**Figure 6.7:** Example of progress over the iterations of the MSE loss of a mode

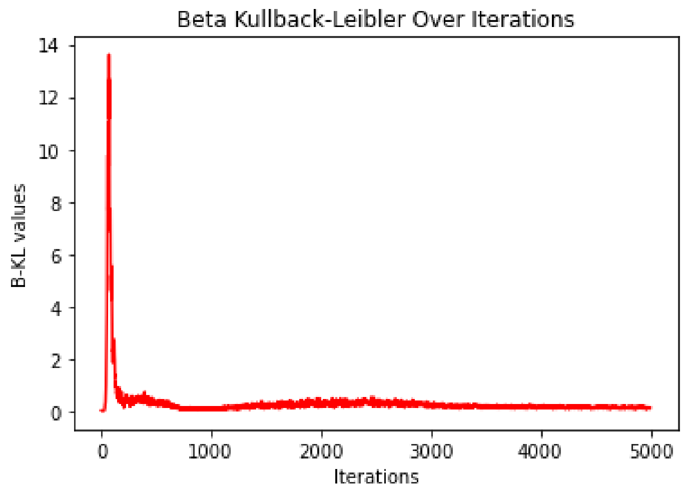

**Figure 6.8:** Example of progress over the iterations of the KL divergence loss of a mode

## 6.2 Experimentation Results

### 6.2.1 Preliminary Experimentation Results

The architecture for the MLP models will consist of an encoder with seven dense layers with 175, 150, 125, 100, 75, 50 and 25 neurons respectively, a latent space of five dimensions and a decoder that is the inverse of the encoder. Every layer of the model has a ReLU activation function, except the output layer of the decoder, which has a sigmoid activation function. For training, the learning rate is 0.0001.

The CNN models follow a different architecture. The encoder has three layers with 16, 32 and 63 filters respectively. All the filters of the encoder have $3 \times 3$ size, and a stride of two except for the filters of the first layer, that only have a stride of one. The latent space consist of five dimensions. The decoder has four layers with 60, 48, 32 and 16 filters respectively, and an initial reshape of the input of the initial transposed convolutional layers of the $(28, 28, 32)$ shape. All the filters of the decoder are of $3 \times 3$ size and have a stride of one. The following aspects are the same as the MLP models, every layer has a ReLU activation function except for the output layer, that employs sigmoid; and the learning rate is $0.0001$.

### 6.2.2 Beta Variational Auto-Encoder

In this experiment, four different models were analyzed at the same time. Two of them with $\beta > 1$, more precisely $\beta = 2 \wedge \beta = 5$, and the remaining two with $\beta < 1, \beta = 0.5 \wedge \beta = 0.2$. To refer to these kind of models, the $\beta$ can be replaced in the name, e.g. 2-VAE. The $\beta = 1$ models were excluded of this experiment because that value of $\beta$ stands for the common versions of the VAE that were used for the perliminary experiments.

After every sub-experiment, one model among the four displayed will be tagged as the best-performing one, which, in this context, it is a synonym of telling which model has a lower FID at the end of its training process.

#### 6.2.2.1 Multi-Layer Perceptron Models

When trained over the MNIST, all the models showed a constant decrease of their loss values and FID across their training process. In all the models, the loss value decreased considerably in the first five-hundred epochs. After that drop, the MSE of all the models would still tend to a moderated reduction after the epochs, while the KL divergence value tends to be constant for all the models with no exception. It is important to remark that the KL divergence starts with such a low value and in the first epochs it raises dramatically, and then is when it starts its lowering tendency.

The FID behaves differently, though. This metric also tends to decrease across the training process but more consistently over time. The metric, in the case of the 2-VAE and the $0.2$-VAE follows almost a linear decreasing, while the 5-VAE and the $0.5$-VAE follow a decreasing pattern that is stair-shaped, as depicted in Figure 6.9.

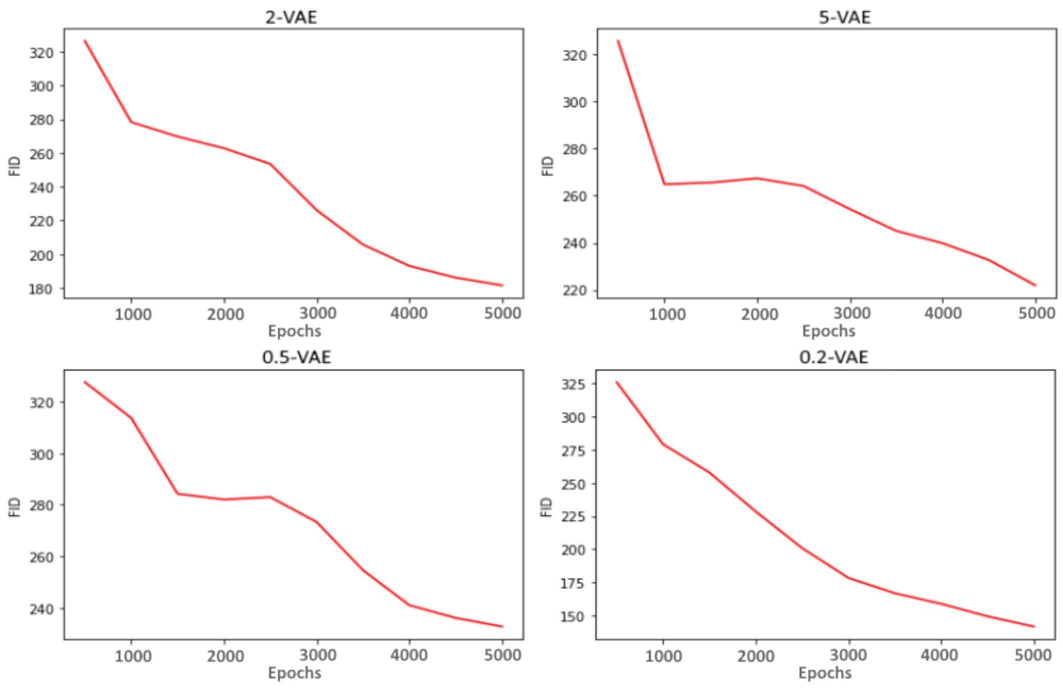

**Figure 6.9:** The FID value across the the training for the models conducted by the $\beta$-VAE experiment over the MNIST dataset.

When comparing the metrics of the different model we can see in Table 6.1 that the model with the best performance is the 0.2-VAE, and that no model with $\beta > 1$ has the lowest value in any of the metrics.

| **Metric** | **Lowest value** | **Model** |
|---|---|---|
| Loss | 33.9284 | 0.2-VAE |
| KL divergence | 0.1469 | 0.5-VAE |
| MSE | 33.7476 | 0.2-VAE |
| FID | 141.7186 | 0.2-VAE |

**Table 6.1:** Results of the $\beta$-VAE experiment with the MNIST dataset

For the Fashion-MNIST, the behavior of the results of the loss values across the training did not change in a remarkable way. The values show almost the same evolution through the training process.

For the FID, the scenario is completely different. First of all, the values are lower as a baseline, with no exception. The other observation is that, for this dataset, all the models follow the same decreasing tendency, and it is different compared to the tendencies observed when the MNIST dataset was used for training. The FID decreasing starts being very fast and tends to slow down across the epochs, drawing a curve. This behavior can be observed in Figure 6.10

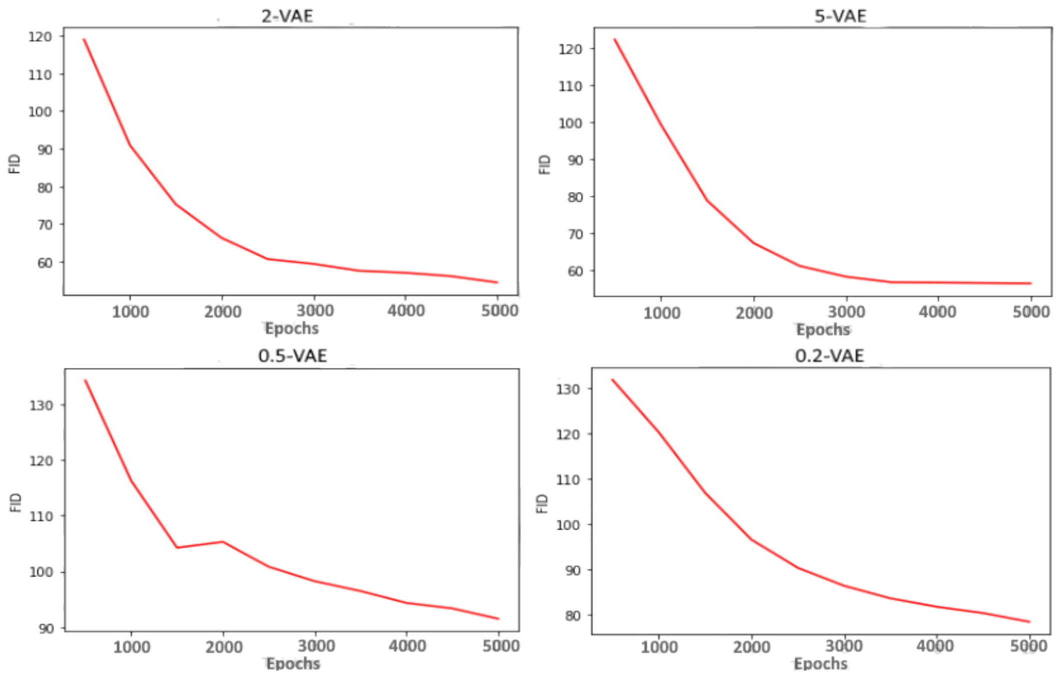

**Figure 6.10:** The FID value across the the training for the models conducted by the $\beta$-VAE experiment over the Fashion-MNIST dataset.

This time, when comparing the metrics of the different models we can see in Table 6.2 that some models with $\beta > 1$ appear to have the lowest value in some metrics. As a matter of fact, the model with the best performance according to our metric, the FID, is the 2-VAE.

| Metric | Lowest value | Model |
|---|---|---|
| Loss | 21.8268 | 0.2-VAE |
| KL divergence | 0.2041 | 5-VAE |
| MSE | 21.5517 | 0.2-VAE |
| FID | 54.5181 | 2-VAE |

**Table 6.2:** Results of the $\beta$-VAE experiment with the Fashion-MNIST dataset

#### 6.2.2.2 Convolutional Models

Trained with the MNIST, all the models showed a constant decrease of their loss values and FID, like with the MLP models. In this experiment, the loss values are lower on average compared to the ones of the previous experiment.

The loss values follow the same behavior across the training process compared to the loss values of the MLP models, with the exception of the KL divergence. In this case, the value of the KL divergence does not get very high at first, and takes more epochs to reach a constant range of values. The evolution of the KL divergence during the training can be seen in Appendix B.

Regarding the FID of the models, this metric behaves the same way as it did with the MLP models trained with the Fashion-MNIST dataset, see Figure 6.11.

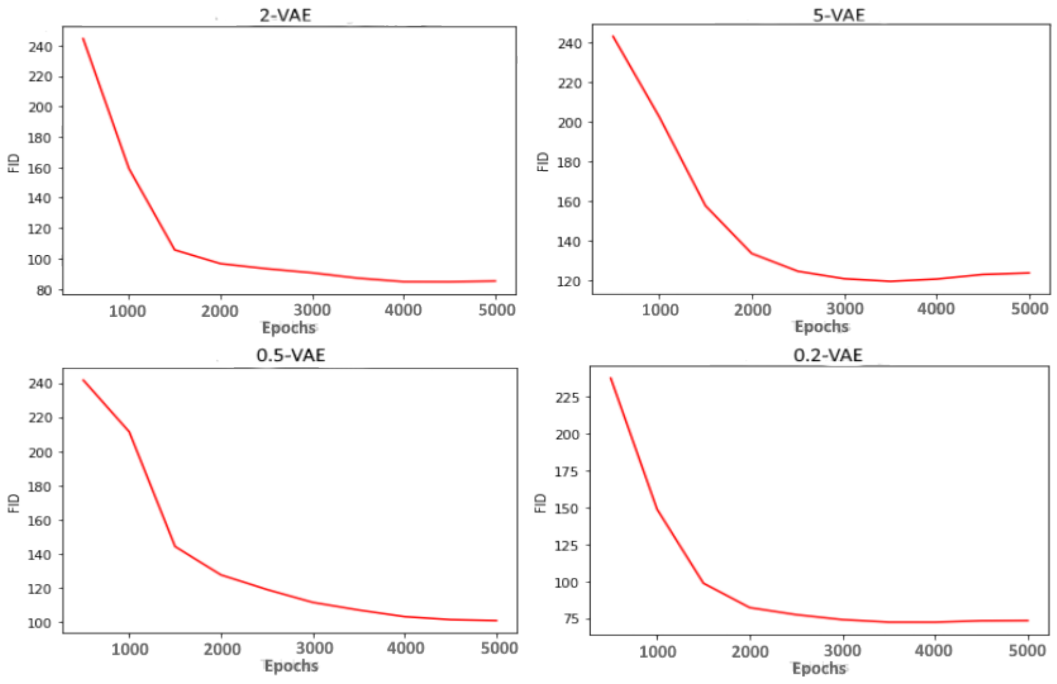

**Figure 6.11:** The FID value across the the training for the models conducted by the $\beta$-CVAE experiment over the MNIST dataset.

Among all the models, the 0.2-CVAE showed to be the best-performing model obtaining the best results when trained with the MNIST dataset. The model obtained the lower score for all the metrics, as it can be seen in Table 6.3.

| **Metric** | **Lowest value** | **Model** |
|---|---|---|
| Loss | 25.4636 | 0.2-VAE |
| KL divergence | 0.1980 | 0.2-VAE |
| MSE | 25.2656 | 0.2-VAE |
| FID | 73.3358 | 0.2-VAE |

**Table 6.3:** Results of the $\beta$-CVAE experiment with the MNIST dataset

The results of the models trained with the Fashion-MNIST have been slightly different from the others. The loss value and MSE start with a pronounced descent of their magnitude, but, in both metrics, the pronounced descent becomes more moderated for some epochs. After few epochs, the decreasing retakes the original rhythm until reaching a range of values that keeps constant for the rest of the training process. The described case can be applied to all the models.

The KL divergence behaves similarly as with the MNIST dataset. The difference is that this time, at the beginning of the training, the value gets higher than with the other dataset, but not as high as with the MLP models.

The FID behaves completely different if it is compared with any other case. This time, as shown in Figure 6.12, this value tends to get higher across the training. It gets higher in almost a linear way in all cases but one. The 0.2-CVAE has a FID value during the training that can be taken as constant due to its low variation.

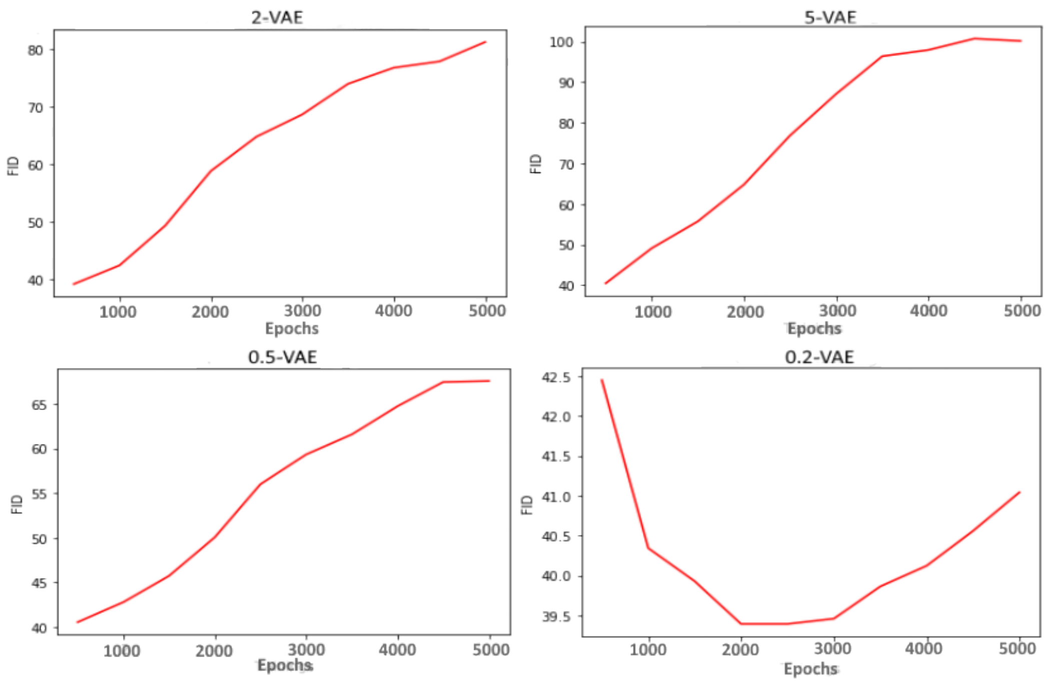

**Figure 6.12:** The FID value across the the training for the models conducted by the $\beta$-CVAE experiment over the Fashion-MNIST dataset.

The same way as with the MNIST, the 0.2-CVAE is the best-performing model. It has had the lowest score among all the models, as registered in Table 6.4

| **Metric** | **Lowest value** | **Model** |
|---|---|---|
| Loss | 17.7361 | 0.2-VAE |
| KL divergence | 0.1987 | 0.2-VAE |
| MSE | 17.5374 | 0.2-VAE |
| FID | 41.0419 | 0.2-VAE |

**Table 6.4:** Results of the $\beta$-CVAE experiment with the Fashion-MNIST dataset

### 6.2.3 Stick-Breaking Variational Auto-Encoder

In this experiment only one model is analyzed at a time, the stick-breaking version for each version of the VAE (SB-VAE and SB-CVAE).

#### 6.2.3.1 Multi-Layer Perceptron Model

For the MNIST dataset, the loss function and MSE values fall dramatically in the very beginning of the training process, and then stay in a constant range for the rest of the process. The KL divergence for the stick-breaking process remains strictly constant during the whole training process.

The FID gets increased at the beginning, but tends to decrease for the rest of the training process. Although, the range of this metric is so narrow that its value can be seen as constant through the training. See all the metrics evolution through the training process in Figure 6.13, and their final value in Table 6.5

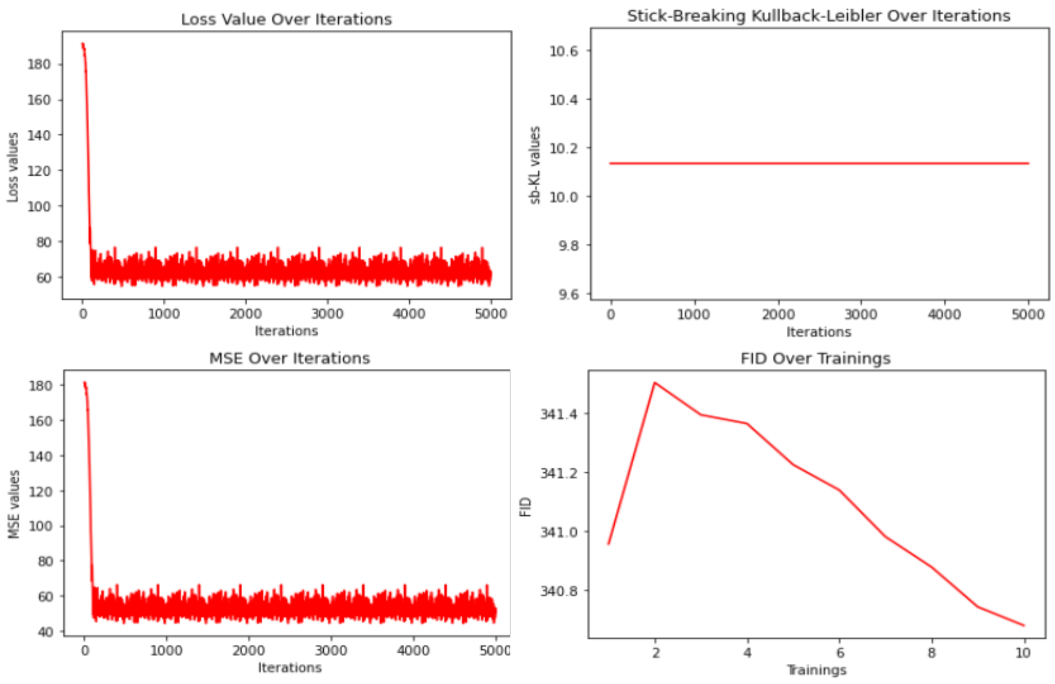

**Figure 6.13:** The values of the metrics across the the training for the models conducted by the SB-VAE experiment over the MNIST dataset.

| **Metric** | **Value** |
|---|---|
| Loss | 62.9052 |
| KL divergence | 10.1346 |
| MSE | 52.7706 |
| FID | 340.6806 |

**Table 6.5:** Results of the SB-VAE experiment with the MNIST dataset

The behavior of the loss values when training the model with the Fashion-MNIST is the same as with the MNIST. In this case, the FID shows a downward trend even though there are some parts where it gets increased. But, as in the case of the MNIST dataset, the value of the metric can be considered as constant. See the described behavior in Figure 6.14, and the final value of the metrics of the model in Table 6.6

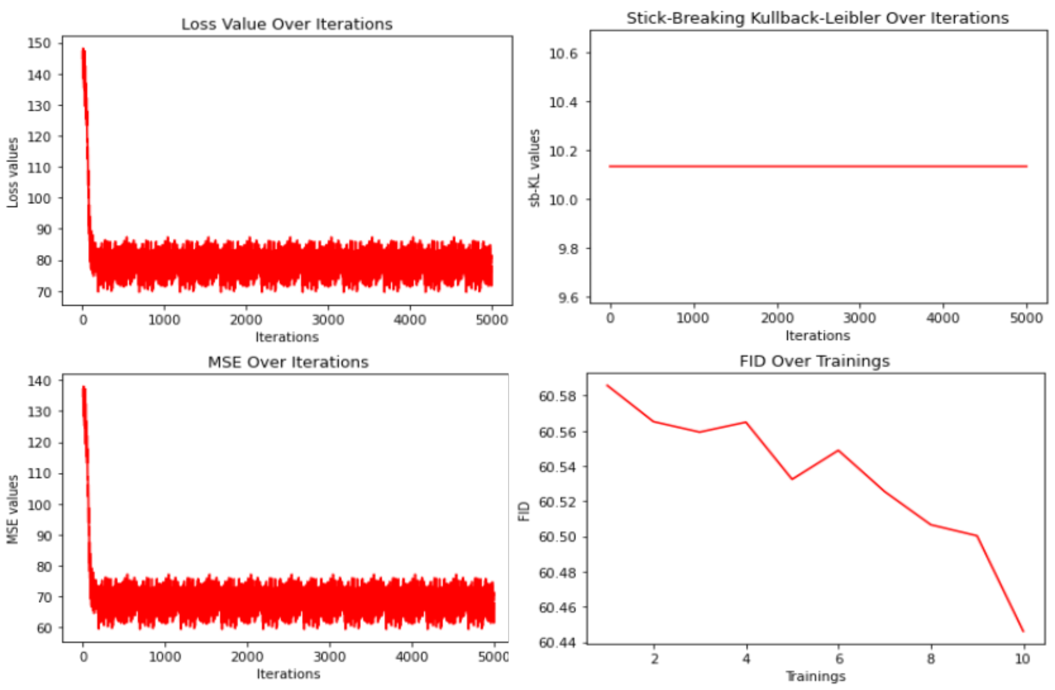

**Figure 6.14:** The values of the metrics across the the training for the models conducted by the SB-VAE experiment over the Fashion-MNIST dataset.

| **Metric** | **Value** |
|---|---|
| Loss | 76.6647 |
| KL divergence | 10.1346 |
| MSE | 66.5301 |
| FID | 60.4460 |

**Table 6.6:** Results of the SB-VAE experiment with the Fashion-MNIST dataset

### 6.2.3.2 Convolutional Model

When the SB-CVAE is trained with the MNIST, the results are almost the same as with the SB-VAE. The only difference is that the FID follows a shape similar to a parabola through the training, but the range of the values is still very narrow, see Figure 6.15 to observe the behavior of the metrics, and Table 6.7 for the values of the metrics at the end of the training.

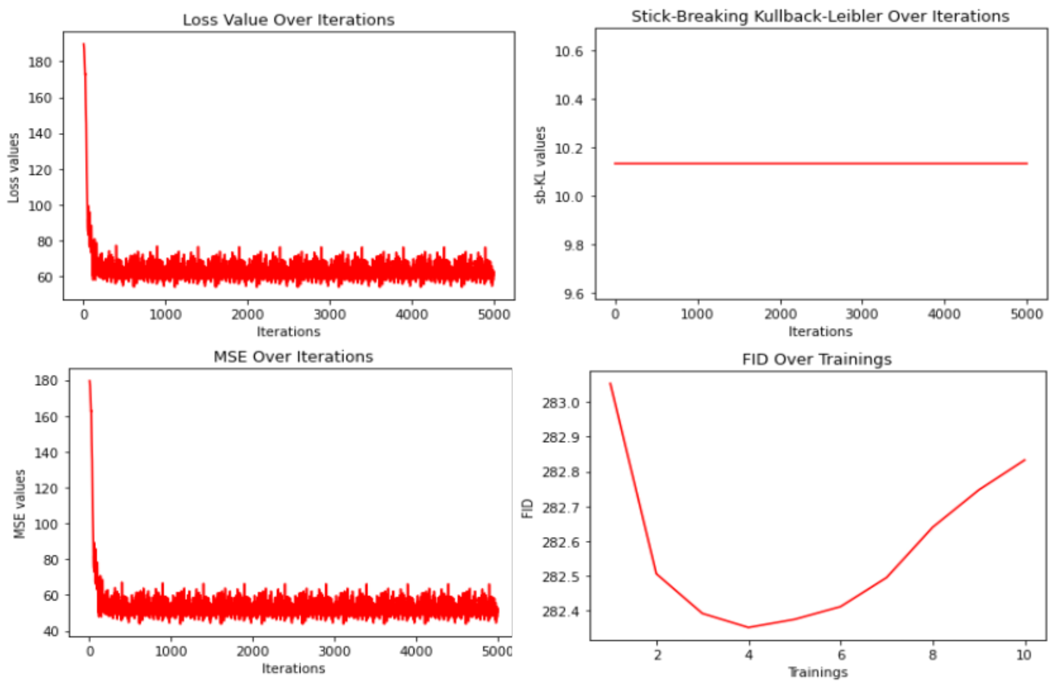

**Figure 6.15:** The values of the metrics across the the training for the models conducted by the SB-CVAE experiment over the MNIST dataset.

| **Metric** | **Value** |
|---|---|
| Loss | 62.7951 |
| KL divergence | 10.1346 |
| MSE | 52.6605 |
| FID | 282.8331 |

**Table 6.7:** Results of the SB-CVAE experiment with the MNIST dataset

When training the model with the Fashion-MNIST dataset, the results of evolution of the loss values are the same, and the FID is increased at first, and later on follows a decreasing trend, but the difference among the lowest and highest FID across the training is less than 0.1. See the behavior of the metrics through the training in Figure 6.16 and their final values in Table 6.8.

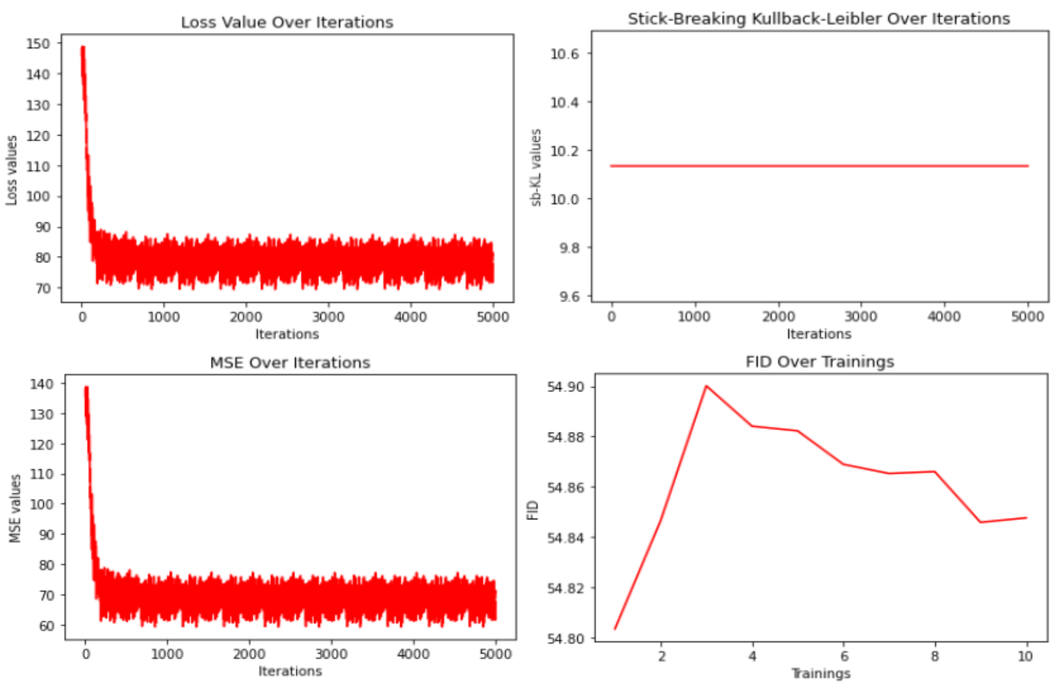

**Figure 6.16:** The values of the metrics across the the training for the models conducted by the SB-CVAE experiment over the Fashion-MNIST dataset.

| **Metric** | **Value** |
|---|---|
| Loss | 76.6795 |
| KL divergence | 10.1346 |
| MSE | 66.5449 |
| FID | 54.8477 |

**Table 6.8:** Results of the SB-CVAE experiment with the Fashion-MNIST dataset

### 6.2.4 Results of Generation

To visualize the generation capacity of the models, manifolds have been created using the models. To create the manifolds, the latent dimensions were reduced to two, and the models were trained from scratch, so the latent value combinations could be computed in just two dimensions, making it possible to plot the generations of those combinations. Nevertheless, the latent dimension reduction could lead to a loss of quality of the models because two latent dimensions are too few, and also the architectures of the models were optimized assuming a latent space of five dimensions. Anyway, it is still a good way to have an idea of what a model can generate. Some generation examples with the models with five latent dimensions can be depicted in Appendix A.

For the MLP models, eight manifolds have been created, two for each model, one per dataset. For the models, it was decided that, as minimum, three different models should create a manifold. One with $\beta < 1$, another one with $\beta = 1$ (the common VAE), and the last one with $\beta > 1$; so the models 0.2-VAE, VAE, and 2-VAE were selected. Finally, to visualize the odd results of the SB-VAE, a manifold is generated by this model as well.

The case of the CNN models is exactly the same. The convolutional models with the same values of $\beta$ were selected and for both datasets too. Then, the SB-CVAE was selected to generate a manifold to be able to visualize its generations.

Every manifold that was created has its own alternate version, where the values of the latent distribution that would serve as input for the decoder are multiplied by one-hundred. This anomalous generation task was performed because the generations with the sampling input multiplied by a big number returned interesting results.

The resulting sixteen manifolds have been grouped in four different groups. The first one stands for all the MLP models trained with the MNIST, see Figure 6.17. The second one stands for the same models as in the first group, but trained with the Fashion-MNIST, as depicted in 6.18. The third and fourth groups follow the same logic but applied to the CNN models, where the third one stands for the models trained with the MNIST, ilustrated in Figure 6.19, and the fourth one for the models trained with the Fashion-MNIST, as shown in Figure 6.20

Taking into account that the distribution learned by the models is not exactly the normal distribution because there is always some error, we tried to generate the manifolds adding or subtracting constant numeric values to see how the generation behaved. With this method, by trial and error with different constant values, we saw that for some models, the manifold would improve by adding two as a constant. This resulted in less blurred and more varied images, see Figure 6.21 and Figure 6.22.

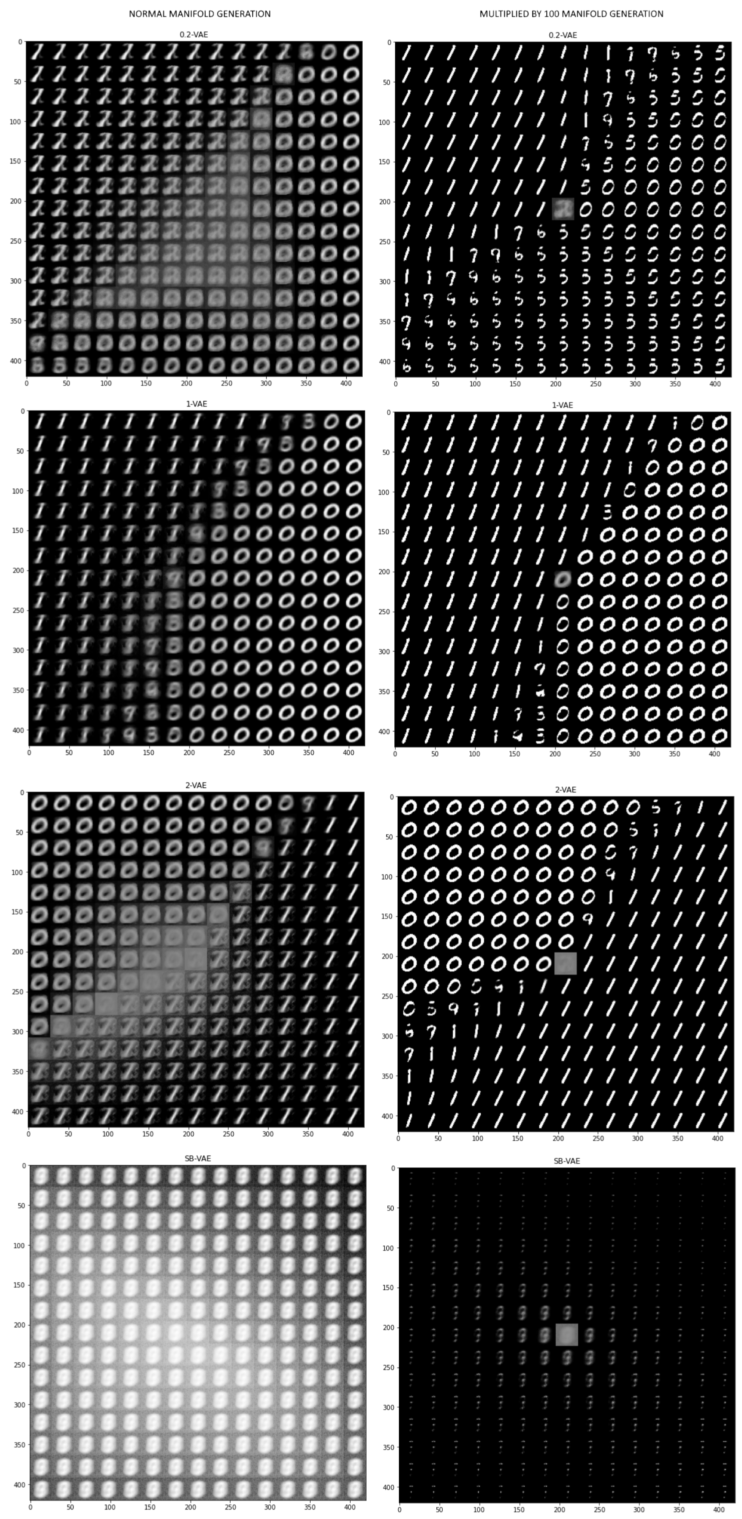

**Figure 6.17:** Manifolds generated with the MLP models trained using the MNIST where each column represents the way the manifolds were generated.

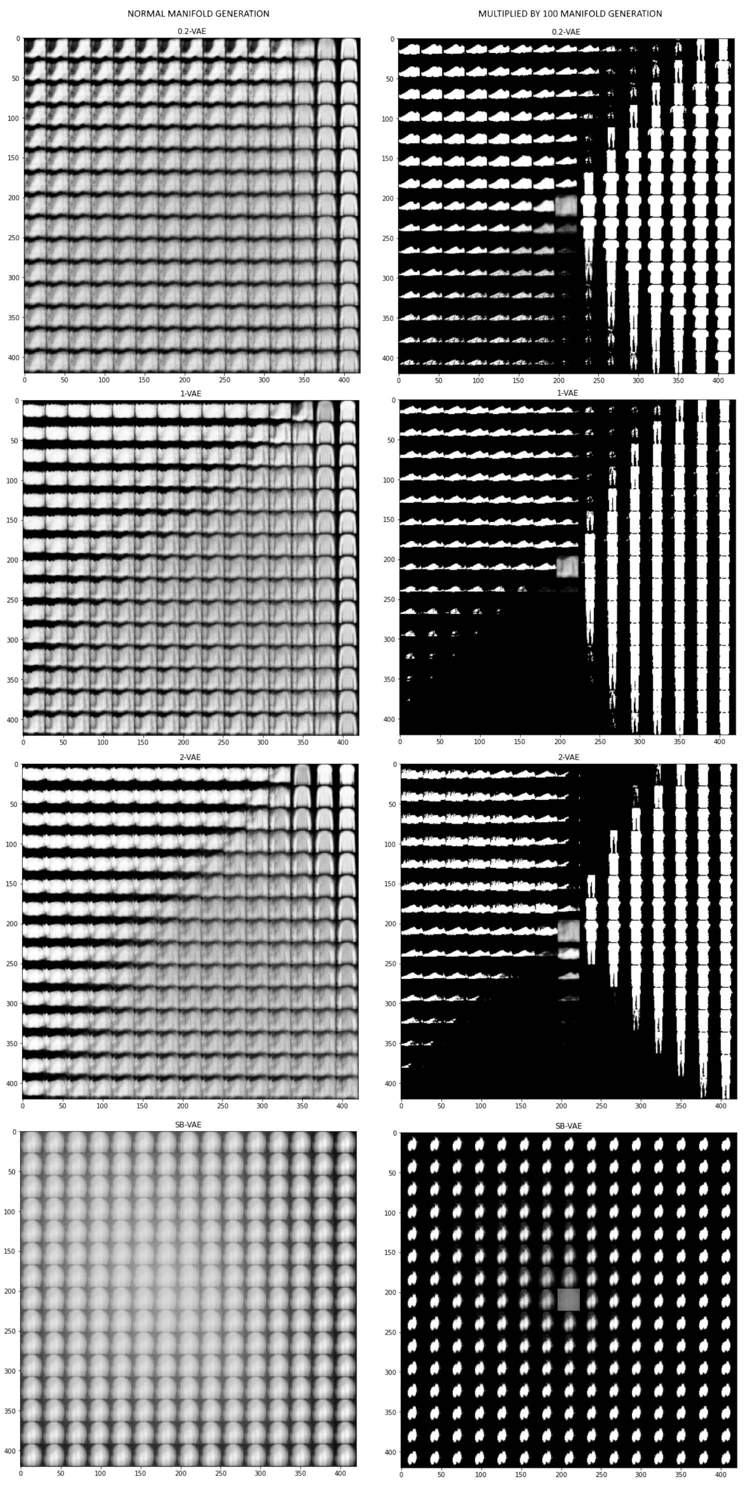

**Figure 6.18:** Manifolds generated with the MLP models trained using the Fashion-MNIST where each column represents the way the manifolds were generated.

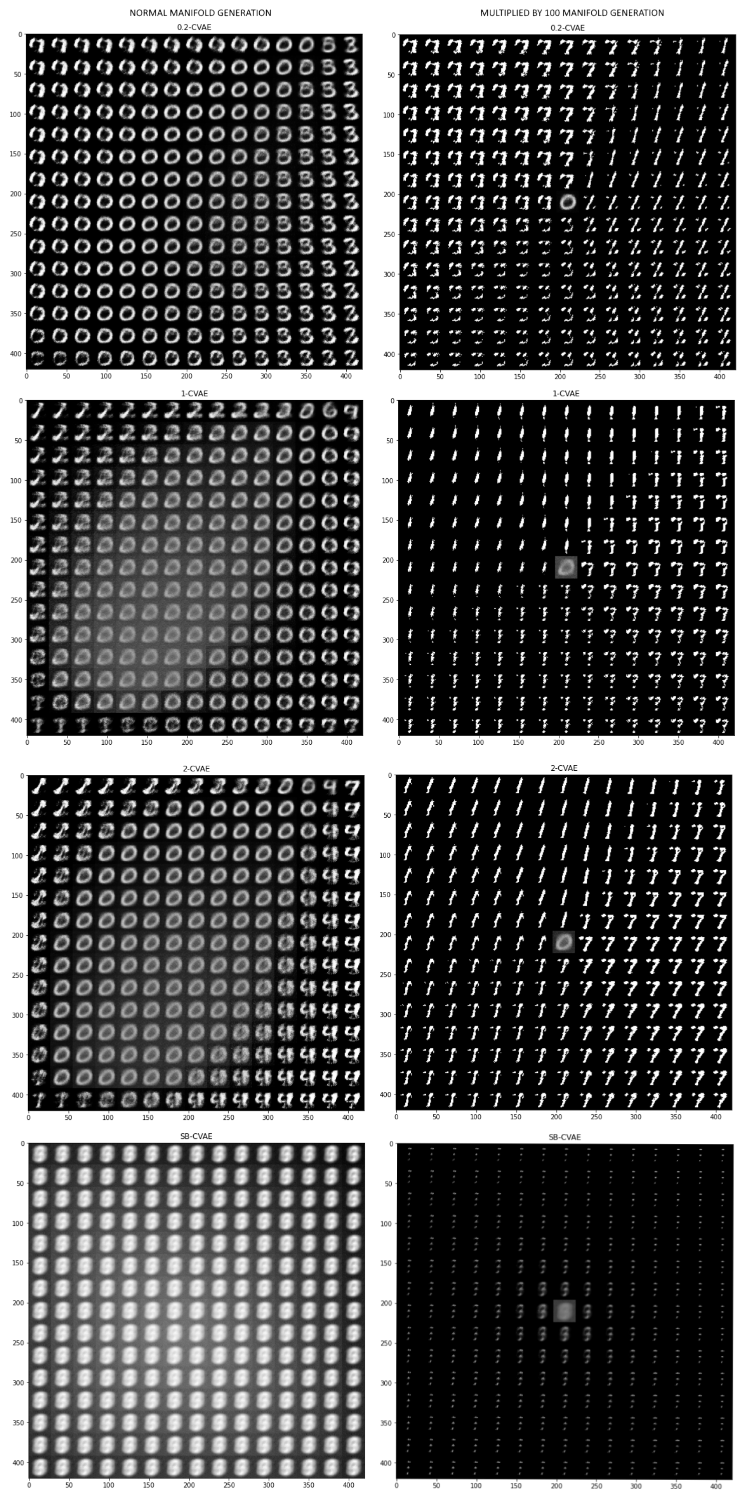

**Figure 6.19:** Manifolds generated with the CNN models trained using the MNIST where each column represents the way the manifolds were generated.

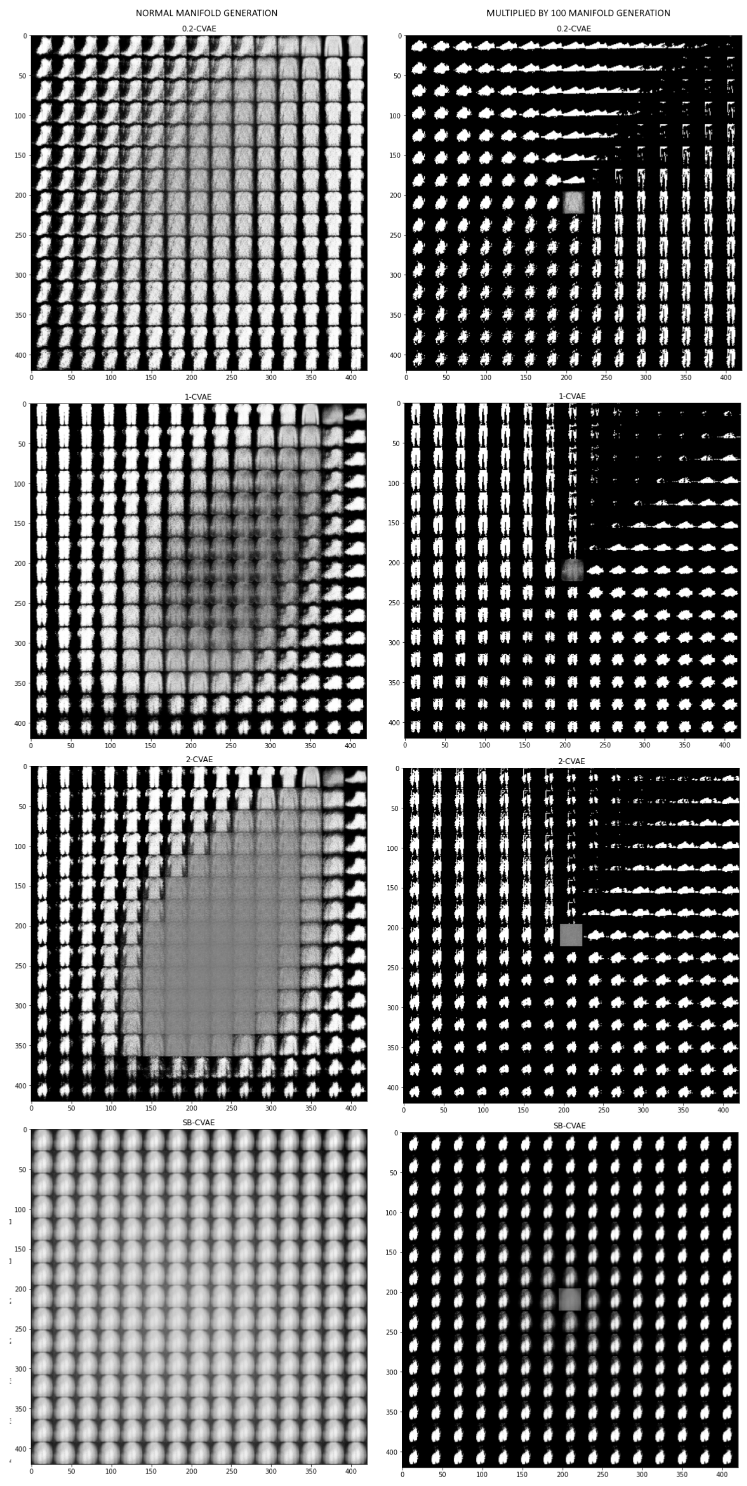

**Figure 6.20:** Manifolds generated with the MLP models trained using the Fashion-MNIST where each column represents the way the manifolds were generated.

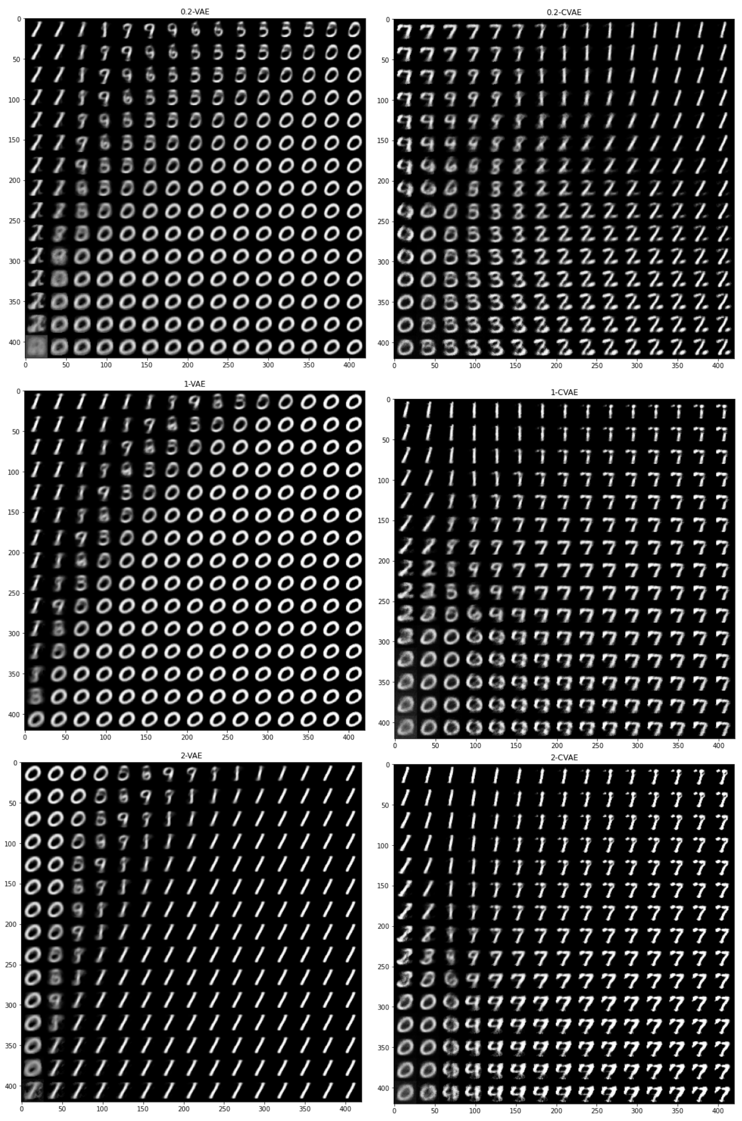

**Figure 6.21:** Manifolds generated by adding two as a constant to the input samples with the models trained using the MNIST.

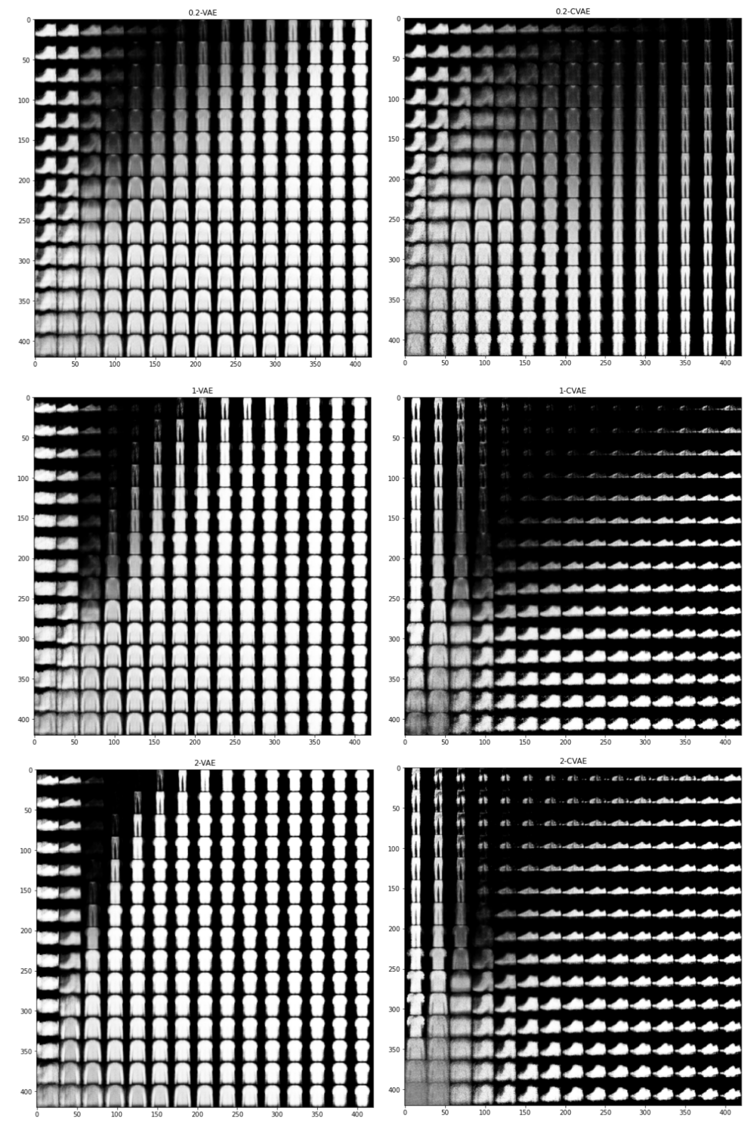

**Figure 6.22:** Manifolds generated by adding two as a constant to the input samples with the models trained using the Fashion-MNIST.

## 6.3 Conclusions

In this bachelor's thesis, we have performed an analysis over the different variations of the VAE (using MLPs and using CNNs) from some models from the state of the art. To that end, we have developed the **tfVAE** library that has allowed us to experiment with the VAE in a simple and flexible way.

To begin with, we can state that the FID computed with a custom model instead of a inception v3 model, works as expected. It was also visible that the magnitude of the FID depends not only on the generative model that is being used, also on the accuracy of the trained model. In our case, the custom model showed to have a $99\%$ accuracy for the MNIST, and a $90\%$ for the Fashion-MNIST. For this reason, during the whole experimentation process we were able to compare the performance of different models trained with the same dataset with the FID, but it was impossible, only by looking at the FID, to compare the models trained with different datasets.

At the same time, the FID appeared to work better for the MNIST than for the Fashion-MNIST because of some strange behavior of the metric during the training process (Figure 6.12). But we have to take into account that it could be happening because the models do not learn properly the structure of the Fashion-MNIST, which could be caused because the architecture used was found performing a grid search using the MNIST, which is a less complex-to-learn dataset. Also, the MNIST does not use the whole spectrum of the greyscale, and is limited to just black and white. The Fashion-MNIST uses a bigger stake of the greyscale spectrum, which could be an explanation of why the FID is lower for this dataset, because the noise is not as penalized as in the MNIST. The models appear to create clothing-shaped images but only using black and white, which could explain that the FID gets higher during the training in some cases.

Now, we will proceed to discuss about the generative capability of the $\beta$ models. Before jumping into any conclusion, we should take into account that the grid search was only focused on the structure of the NNs, and not on every hyperparameter of the models because a search like that would have taken much more time and computing power than we had. When observing the generated manifolds we should also take into account that they were created with less powerful models than the ones used for experimenting because the dimensionality of the latent space was reduced from five to two.

By analyzing the results, we can conclude that the $\beta < 1$ models tend to have a lower FID and to generate less blurry images. Also, when applying the sum of the constant to the manifold, the $0.2$-CVAE shows the most diverse generations for both datasets.

At the same time, the $\beta > 1$ and $\beta = 1$ models are not so different in generative terms. The $\beta = 2$ model shows a slightly better generative performance, and lower FID values. The blurriness of the generated images is not greater than the blurriness of the generated images of the common model. Although, that can also be conditioned to the fact that the value of $\beta$ is not big enough to generate images clearly blurry compared to the ones generated by the original model.

To conclude with the analysis of the $\beta$ models, we can say that, although as observed in [22],the $\beta$-VAE, when $\beta > 1$, presents a more disentangled latent representation of the features of the data, it does not grant a better generative capability. Meanwhile, even though we do not know if the latent representation of the features of the data is more or less disentangled, the $\beta < 1$ models show a better performance generating new data, and therefore, generate images with a very good quality. Besides, a poor performance can also be due to the fact that the sampling is based on the supposition that a model has learned in the same magnitude how to decode to the prior distribution and to reconstruct, which is not always true. After training any VAE or CVAE model, it would be highly recommendable to perform a post-processing exercise to learn how to approximate the distributions that are mapped to the latent space. This way, the sampling could be made over the approximated distributions instead of the prior distribution, allowing the model use its full generative potential.

Another interesting observation regarding the experiments is the fact that when multiplying the samples by a big number like one-hundred, the generations would improve its quality in some cases. This is an odd behavior because it was not intuitive that by augmenting the deviation of the samples, some generated images would look with more contrast. These *augmented* samples, showed to be very helpful with the blurriness, by minimizing it even though it would not always favor the generated image.

A possible explanation to this phenomenon would be that the high variance in the input of the decoder, would lead to a high variance in the values of the output image, discovering the underlying shape of the original sample. This event could be enhanced by the models we have trained in the experiments, models with the ReLU activation function across the whole model but in the last layer, where the sigmoid function was used, By exaggerating the features inside a latent vector, we could unconsciously make the model decide what shape to generate more categorically. The opposite of having samples with very low variance making the model struggle to find a proper reconstruction although it was initially trained to do so with input data with similar features.

Another interesting feature regarding this phenomenon is that the model in where this helps the generation the most is the $0.2$-VAE and $0.2$-CVAE. If the hypothesis that by increasing the variance of the data, we get the underlying shape of the original sample happens to be true, it would set the $\beta < 1$ models as the ones with the richest latent space, disentangled or not. The FID of this generation technique compared to the FID of the common one can be seen in Appendix C.

Regarding to the stick-breaking models, as said in [20], these models tend to collapse in the latent space. This would explain the null learning of the decoder of the model.

The model needs a very specific setup to work properly and not collapse, and, as shown, does not work under some conditions. The conclusion over this model is that it was so useful to expand the state of the art of the VAE, but it is not a reliable model.

To finish, the usefulness of **tfVAE** has been shown through the whole experimentation process. It granted all the necessary tools for creating and training the models easily and with flexibility. The library gave a whole abstraction layer to the VAE that not only helped testing with the model, but to implement new variations easily, quickly and in a structured way.

CHAPTER 7

# Future Work

As the scope of a bachelor's thesis is limited, many tasks could not be introduced in this work. The future work sets ideas and research lines to follow the work that has been already done. Among those lines we can find the following:

- Perform a wider grid search not only searching through few possibilities of layer distribution, but to search through every possible hyperparameter with proper pruning techniques to avoid exploring pointless configurations.
- Perform several executions of the same experiments with different random seeds and document their results to make the experimentation sounder.
- As the $\beta < 1$ models showed a better performance, see what would happen with a 0-VAE, or with $\beta \simeq 0$. It could lead to a very bad latent space, but in exchange, to generations with better quality.
- Measure the disentanglement of the latent space of the models in the experiments with the metric mentioned in the state of the art, which was originally used for the $\beta$-VAE.
- Related to the previous line, modeling the latent space after the training to adapt the generation of samples to the latent space. We could integrate different distribution estimation models, such as copulas to the latent space, which are also differentiable.
- Research more exhaustively about the phenomenon of multiplying the samples by a big number like one-hundred. Try to find a model that exploits that phenomenon.
- Search for a way to fix the collapse of the latent space of the SB-VAE keeping the essence of the model. If the model gets to work properly, try to add the $\beta$ hyperparameter to it.
- Expand the library to more implementations and kind of neural networks like the Recurrent Variational Auto-Encoder (RVAE) implemented with Recurrent Neural Networks (RNNs)

- Expand the library to support models for mode data types, and not only images (e.g. videos, audios...).
- Expand the library to grant support for a hybrid VAE that can contain any kind of layer in the same model. Also see if a hybrid model performs better than a pure one like VAE or CVAE.
- Lead the library to the next level and integrate it to the Python package-management system, so it can be easily installed by any user from any device.

CHAPTER 8

# Project Management

Before the start of an engineering project, some tasks need to be identified and added to a list of tasks required to support a larger strategic plan, which is commonly called backlog, in order to complete the project.

At the same time, the risks that threat the successful accomplishment of the engineering project need to be identified. As a consequence, some actions will be taken to avoid the identified risks or alternative plans should be drawn in order to cope with these situations.

## 8.1 Plan

This section will cover the planning followed during the development of the project related to this bachelor's thesis. This is based on an initial planning that may suffer some changes during the development of this project.

Every task under this project defines an independent action that will grant additional individual value to the project. The tasks are usually divided in two groups: optional and mandatory. The mandatory tasks are those ones that are strictly needed to reach the Minimum Viable Product (MVP), and, accordingly, the optional tasks are those ones that are not needed to reach the MVP but can boost the quality of the project and should be done if there is time. If a task is optional but does not boost the quality of the project, it should be removed from the backlog.

A very good practice to organize a project is organizing it by phases. The phases are a group of tasks that serve a specific context inside the project, they can be developed either sequentially or in parallel depending of their nature. Following the same logic as with tasks, the phases can also be optional or mandatory. A phase is mandatory if some tasks that compose it are mandatory, and to accomplish this phase we have to finish, at least, all the mandatory tasks inside it. Then, a phase is optional if and only if all the tasks that make it up are optional, and to accomplish this kind of phases it is needed to specify which optional tasks from it would be essential to be done in order to reach the goal specified by the optional phase.

The development of this bachelor's thesis has been taken as a project itself. So, before starting any kind of work, some phases and tasks have been planned and set in the own backlog of the project:

- **Phase 0:** Research.
    - **P0-T0:** Investigate about MultiLayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs)
    - **P0-T1:** Investigate about the Variational Auto-Encoder (VAE)
- **Phase 1:** Development of the library.
    - **P1-T0:** Set up the developing environment
    - **P1-T1:** Structure the programming project
    - **P1-T2:** Implement the functionalities and models of the library
    - ○ **P1-T3:** Create a documentation for the library
- **Phase 2:** Development of the memory.
    - **P2-T0:** Create an index and structure it
    - **P2-T1:** Write a detailed memory
- **Phase 3:** Experimentation.
    - **P3-T0:** Plan the experiments
    - **P3-T1:** Implement the experiments using the library
    - **P3-T2:** Analyze the results and draw conclusions
    - **P3-T3:** Propose the future research lines
    - ○ **P3-T4:** Upload the experiments in collab notebooks so that they can be reproduced by anyone

In the plan specification above, the bullet points represent the mandatory tasks, and the circles are the optional tasks. As it can be seen, the four phases are mandatory for the development of the project and there are just two phases with optional tasks, one optional task per phase. As it was explained above, we can see how these optional tasks do not affect the development of this project until its completion, but they are able to add great value to it.

In Figure 8.1 we can see a network diagram that explains how the phases are developed. Since the kick-off of the project, the research phase has to be finished in order to start with the phases of the development of the library and the memory, which are started at the same time in parallel. Once the development of the library is finished, the experiments phase are started and keep being developed in parallel within the memory. Once the experiments and the memory are finished, the project is considered as officially finished.

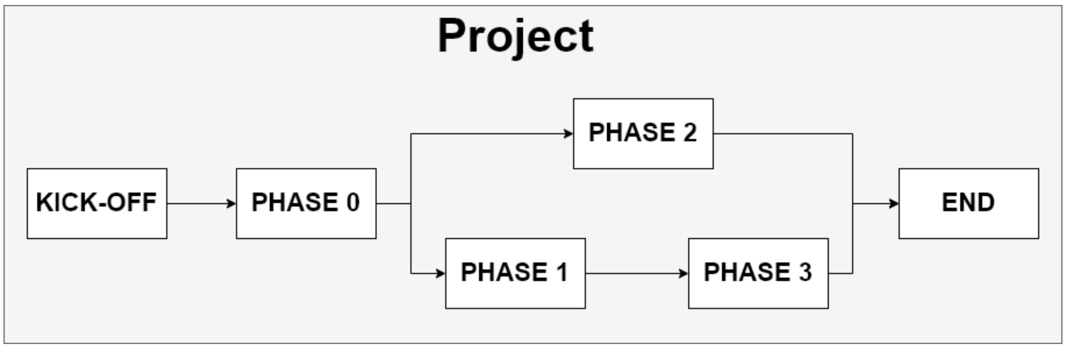

**Figure 8.1:** Network diagram of the project phases

To finish with the planning of the project, a Gantt chart has been created to estimate the times for the project's tasks and phases and set some a priory deadlines . This chart can be found in Figure 8.2, the time unit is weeks because the project follows weekly sprints in where each Tuesday the state of the project is evaluated and discussed with the thesis advisor.

The phase number three has been estimated to take less time than normal thanks to the development of the library. One of the strong points of creating code libraries is that it speeds up some processes, allowing the projects to be developed in less time. It is therefore expected that the library created in the second phase boosts the experiments of the third phase.

The two optional tasks of this project start when the status of the project allows so, and they are up to be done until the end of the project. They do not have a deadline because of their optional nature, so they inherit the deadline of the project itself.

## 8.2 Risks

During the development of any project there are risks that threat its success. The best way to mitigate these risks is to identify them beforehand and make a plan for them.

In this project, the risk management task was performed at the start of it. Identifying the main risks and defining some strategies to either cope with or prevent them. Table 8.1 registers all the identified risks, and how to avoid or cope with them if possible.

| **Risk** | **Explanation** | **How to avoid/cope with** |
|---|---|---|
| Lack of time | During the development of the project I will be working full time as a software developer. This situation can threat the project because I will have less time. Also, unexpected events could happen that would take time from me. | Schedule how I will distribute my free time to develop the project. If the scheduling is not enough. The project will be deferred to gain some extra time. |
| Electronic failure | The experiments, memory or code can be digitally lost if stored locally and the electronic devices used for the development of the project fail in some way. Also, the failure of the electronic devices used for the development of the project could block the progress of it | Use different cloud platforms like GitHub and overleaf for the data loss, and keep a good maintenance of the electronic devices to lower their possibilities to fail. |

**Table 8.1:** Risks that could threat the project

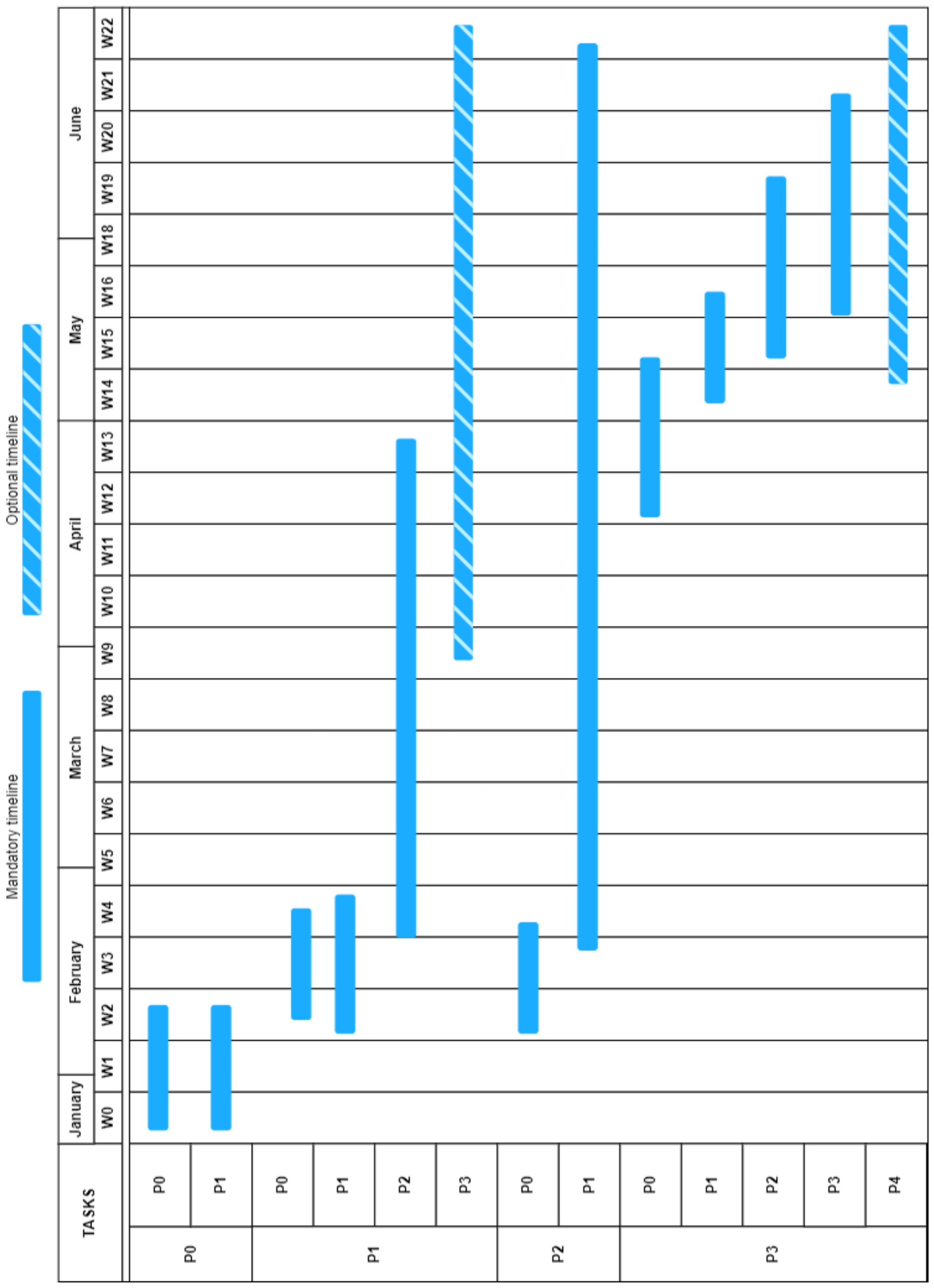

**Figure 8.2:** Gantt chart of the project

# Bibliography

[1] Walaa Medhat, Ahmed Hassan, et al. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal*, 5(4):1093–1113, 2014. See section 1.1.

[2] Adam Lopez. Statistical machine translation. *ACM Computing Surveys (CSUR)*, 40(3):1–49, 2008. See section 1.1.

[3] Wenyi Zhao, Rama Chellappa, et al. Face recognition: A literature survey. *ACM computing surveys (CSUR)*, 35(4):399–458, 2003. See section 1.1.

[4] Heng-Da Cheng, X_ H_ Jiang, et al. Color image segmentation: advances and prospects. *Pattern recognition*, 34(12):2259–2281, 2001. See section 1.1.

[5] Haizhou Li, Bin Ma, et al. Spoken language recognition: from fundamentals to practice. *Proceedings of the IEEE*, 101(5):1136–1159, 2013. See section 1.1.

[6] Frank Seide, Gang Li, et al. Conversational speech transcription using context-dependent deep neural networks. In *Twelfth annual conference of the international speech communication association*, 2011. See section 1.1.

[7] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. See section 1.1.

[8] Alex Krizhevsky, Ilya Sutskever, et al. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. See section 1.1.

[9] OpenAI. OpenAI - an ai research laboratory. `https://openai.com/`, 2021. See section 1.1.

[10] Aditya Ramesh, Mikhail Pavlov, et al. Dall·e 2: Exploring the limits of vision-and-language pre-training. `https://openai.com/dall-e-2/`, 2021. Accessed: April 26, 2023. See section 1.1.

[11] Prajit Ramachandran, Michael Janner, et al. Jukebox: A generative model for music. `https://openai.com/blog/jukebox/`, 2020. Accessed: April 26, 2023. See section 1.1.

[12] Vidyadhar Upadhya and PS Sastry. An overview of restricted boltzmann machines. *Journal of the Indian Institute of Science*, 99:225–236, 2019. See section 1.1.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. See sections 1.1, 3.3.

[14] Antonia Creswell, Tom White, et al. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018. See section 1.1.

[15] Martín Abadi, Ashish Agarwal, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. See sections 1.2, 5.2.1.

[16] François Chollet. Keras. `https://keras.io`, 2015. See sections 1.2, 5.2.1.

[17] Charles R. Harris, K. Jarrod Millman, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. See section 1.2.

[18] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. See section 1.2.

[19] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. See section 1.2.

[20] Weonyoung Joo, Wonsung Lee, et al. Dirichlet variational autoencoder. *Pattern Recognition*, 107:107514, 2020. See sections 1.2, 4, and 6.3.

[21] Eric Nalisnick and Padhraic Smyth. Stick-breaking variational autoencoders. *arXiv preprint arXiv:1605.06197*, 2016. See sections 1.2, 4.

[22] Irina Higgins, Loic Matthey, et al. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. See sections 1.2, 4, and 6.3.

[23] Hind Taud and JF Mas. Multilayer perceptron (MLP). *Geomatic approaches for modeling land change scenarios*, pages 451–455, 2018. See section 2.2.

[24] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. See section 2.2.

[25] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018. See section 2.2.

[26] David E Rumelhart, Geoffrey E Hinton, et al. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. See section 2.2.

[27] Yann LeCun, Léon Bottou, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. See sections 2.2, 2.3.

[28] Ian Goodfellow, Yoshua Bengio, et al. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. See sections 2.3, 3.1.1.

[29] Matthew D. Zeiler, Dilip Krishnan, et al. Deconvolutional networks. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2528–2535. IEEE, 2010. See section 2.3.2.

[30] Dominik Scherer, Andreas Müller, et al. Evaluation of pooling operations in convolutional architectures for object recognition. In Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis, editors, *Artificial Neural Networks – ICANN 2010*, pages 92–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. See section 2.3.3.

[31] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. See section 3.1.1.

[32] Kyunghyun Cho. Boltzmann machines and denoising autoencoders for image denoising. *arXiv preprint arXiv:1301.3468*, 2013. See section 3.1.1.

[33] Angshul Majumdar and Aditay Tripathi. Asymmetric stacked autoencoder. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 911–918. IEEE, 2017. See section 3.1.2.

[34] George Casella and Roger L Berger. *Statistical Inference*. Duxbury Press, Belmont, CA, 2nd edition, 2002. Chapter 2. See section 3.2.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. See section 3.3.

[36] Ziyu Wang, Yiyi Zhang, et al. Pianotree VAE: Structured representation learning for polyphonic music. *arXiv preprint arXiv:2008.07118*, 2020. See section 4.

[37] Ali Razavi, Aaron van den Oord, et al. Generating diverse high-fidelity images with vq-vae-2. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. See section 4.

[38] Pablo A Mitnik and Sunyoung Baek. The kumaraswamy distribution: median-dispersion re-parameterizations for regression modeling and simulation-based estimation. *Statistical Papers*, 54:177–192, 2013. See section 4.

[39] Kai Wang Ng, Guo-Liang Tian, et al. Dirichlet and related distributions: Theory, methods and applications. 2011. See section 4.

[40] Yadolah Dodge. *Gamma Distribution*, pages 215–216. Springer Science & Business Media, New York, NY, 2008. See section 4.

[41] Vincent Fortuin, Dmitry Baranchuk, et al. GP-VAE: Deep probabilistic time series imputation. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1651–1661. PMLR, 26–28 Aug 2020. See section 4.

[42] Jakub Tomczak and Max Welling. VAE with a vampprior. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1214–1223. PMLR, 09–11 Apr 2018. See section 4.

[43] Feihong Li, Wei Huang, et al. A new VAE-GAN model to synthesize arterial spin labeling images from structural MRI. *Displays*, 70:102079, 2021. See section 4.

[44] Steffen Schneider, Jin Hwa Lee, et al. Learnable latent embeddings for joint behavioural and neural analysis. *Nature*, 617(7960):360–368, May 2023. See section 4.

[45] Scott Chacon and Ben Straub. *Pro git*. Apress, 2014. See section 5.1.

[46] Lukasz Langa, Richard Si, et al. Black code formatter. `https://github.com/psf/black`. Accessed March 01, 2023. See section 5.1.1.

[47] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004. See section 5.2.1.

[48] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. See section 5.2.2.

[49] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012. See section 6.1.1.

[50] Han Xiao, Kashif Rasul, et al. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. See section 6.1.1.

[51] Luc Devroye, Abbas Mehrabian, et al. The total variation distance between high-dimensional gaussians. *arXiv preprint arXiv:1810.08693*, 6, 2018. See section 6.1.2.

[52] Artem Obukhov and Mikhail Krasnyanskiy. Quality assessment method for gan based on modified metrics inception score and fréchet inception distance. In *Software Engineering Perspectives in Intelligent Systems: Proceedings of 4th Computational Methods in Systems and Software 2020, Vol. 1 4*, pages 102–114. Springer, 2020. See section 6.1.2.

[53] Christian Szegedy, Vincent Vanhoucke, et al. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. See section 6.1.2.

[54] Kalyanmoy Deb and Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies: Introductory tutorials in optimization and decision support techniques*, pages 403–449. Springer, 2013. See section 3.

# Appendices

APPENDIX A

# More Generation Examples

During the experiments of the $\beta$-VAE and $\beta$-CVAE, after training each model for each dataset, some random generations were made, as depicted in Figure A.3, Figure A.2, Figure A.3 and Figure A.4. Each model generated two sets of random images: One with normal sampling and the other with augmented sampling.

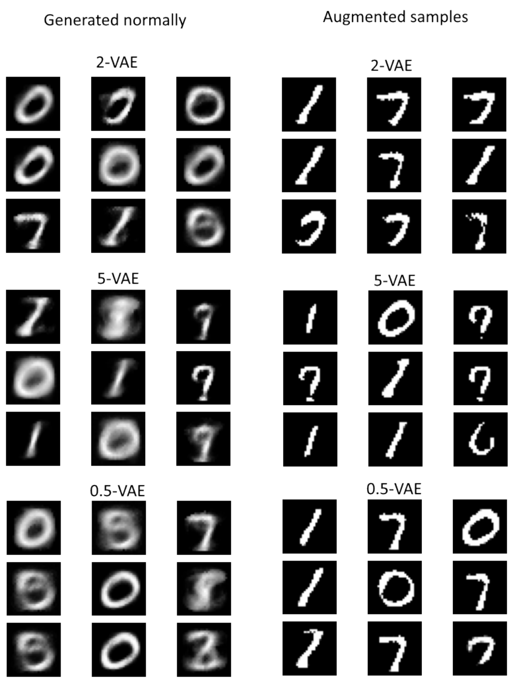

**Figure A.1:** Random generations of the $\beta$-VAE models with the MNIST dataset after the training.

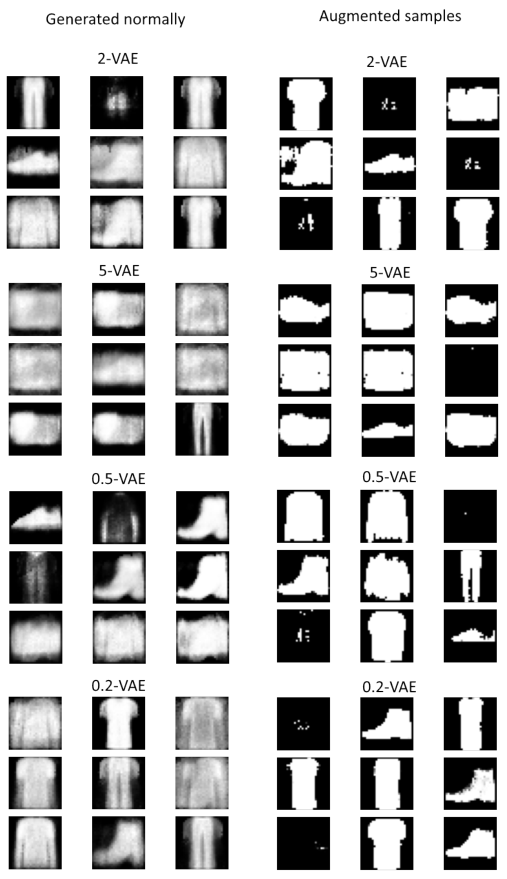

**Figure A.2:** Random generations of the $\beta$-VAE models with the Fashion-MNIST dataset after the training.

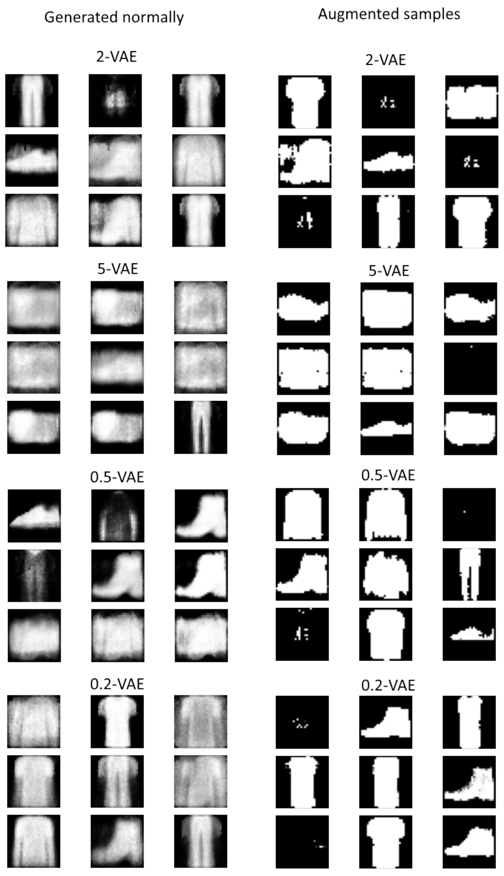

**Figure A.3:** Random generations of the $\beta$-CVAE models with the MNIST dataset after the training.

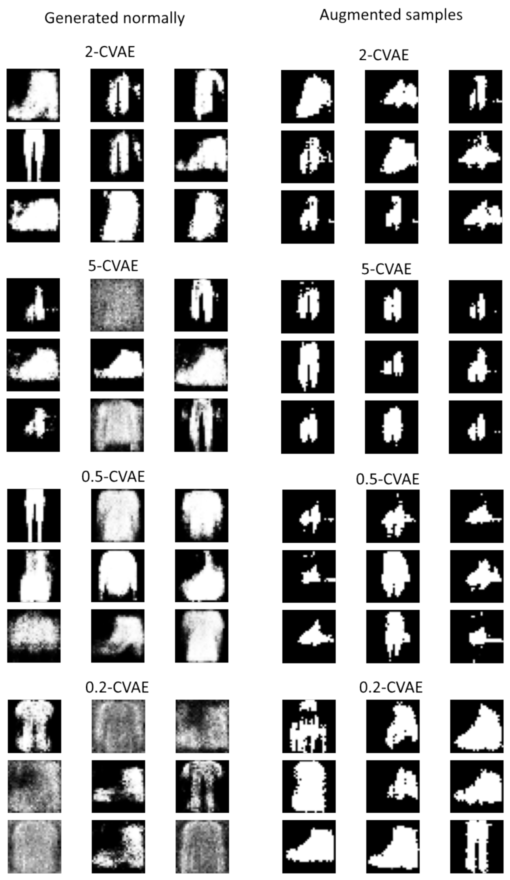

**Figure A.4:** Random generations of the $\beta$-CVAE models with the Fashion-MNIST dataset after the training.

APPENDIX B

# Kullback-Leibler Divercence Evolutions

In Figure B.1 and Figure B.2 we can observe how the KL divergence value evolved through the training of each model of the experiments of the $\beta$-VAE and $\beta$-CVAE for both used datasets.

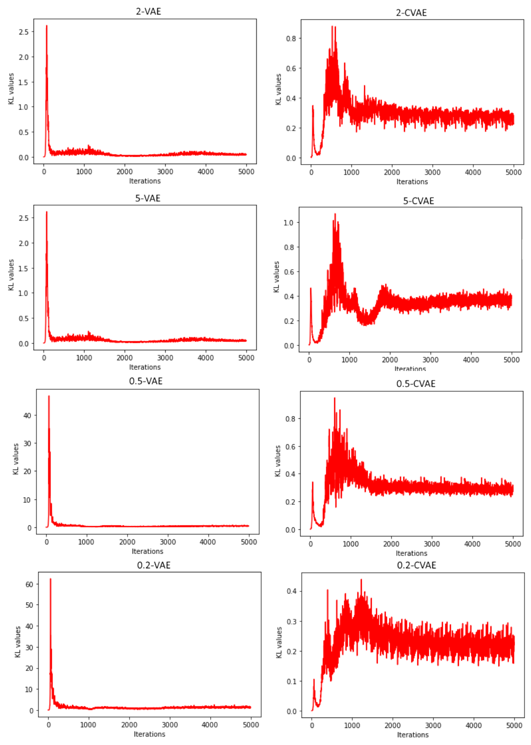

**Figure B.1:** KL divergence values through the training of the models with the MNIST dataset.

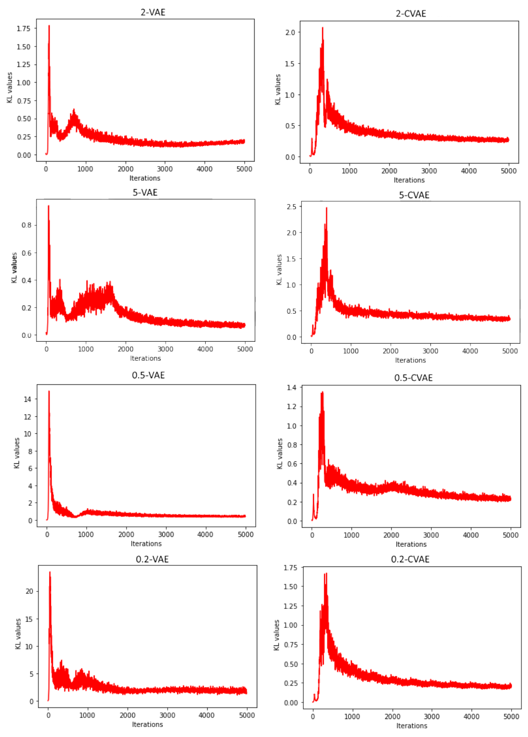

**Figure B.2:** KL divergence values through the training of the models with the Fashion-MNIST dataset.

APPENDIX C

# FID Values for the Augmented Sampling

The FID of the so called augmented sampling generation has been caculated for each model in the experiments of the $\beta$-VAE and $\beta$-CVAE, and has been compared with the values with the normal sampling, see Table C.1.

| **Model** | **Dataset** | **Augmented Sampling** | **Normal Sampling** |
|---|---|---|---|
| 2-VAE | MNIST | 123.9771 | 168.8981 |
| 5-VAE | MNIST | 137.6824 | 178.9140 |
| 0.5-VAE | MNIST | 91.6738 | 165.7083 |
| 0.2-VAE | MNIST | 91.7873 | 212.6919 |
| 2-VAE | Fashion-MNIST | 37.1376 | 91.6461 |
| 5-VAE | Fashion-MNIST | 62.2354 | 114.8022 |
| 0.5-VAE | Fashion-MNIST | 21.9546 | 43.5730 |
| 0.2-VAE | Fashion-MNIST | 26.0234 | 86.9979 |
| 2-CVAE | MNIST | 116.2057 | 85.1756 |
| 5-CVAE | MNIST | 153.2155 | 123.6261 |
| 0.5-CVAE | MNIST | 87.9924 | 101.0071 |
| 0.2-CVAE | MNIST | 67.2741 | 73.3358 |
| 2-CVAE | Fashion-MNIST | 63.7893 | 81.2080 |
| 5-CVAE | Fashion-MNIST | 98.1643 | 100.1480 |
| 0.5-CVAE | Fashion-MNIST | 78.0636 | 67.5634 |
| 0.2-CVAE | Fashion-MNIST | 37.3632 | 41.0419 |

**Table C.1:** Comparison of the FID value using two different sampling methods over the exact same model and dataset.

It can be noticed that for the MLP models, the FID is lower for the generations with augmented sample, but the trend changes for the CNN models.