## Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



## Sammendrag

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

# Preface

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, Navn Navnesen, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, Navn Navnesen, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Max Lunde Hauge
Trondheim, June 2023

*'A famous person once said...'*

**— Max in writing this document**

# Contents

**Abstract** . . . . . **v**

**1 Introduction** . . . . . **1**

**2 Theoretical background** . . . . . **2**

2.1 Neural Networks . . . . . 2

2.1.1 Structure of Neural Networks . . . . . 2

2.1.2 Learning in Neural Networks . . . . . 3

2.1.3 Activation Functions . . . . . 3

2.1.4 Supervised and unsupervised learning . . . . . 3

2.2 Convolutional Neural Networks, CNN . . . . . 4

2.2.1 Convolutional layers . . . . . 4

2.2.2 Pooling . . . . . 5

2.2.3 Architecture of CNNs . . . . . 5

2.3 Encoder . . . . . 6

2.3.1 Downsampling blocks . . . . . 6

2.3.2 Residual block . . . . . 6

2.3.3 Encoder Architecture . . . . . 7

2.4 Decoder . . . 8
2.4.1 Upsampling Blocks . . . 8
2.4.2 Decoder Architecture . . . 8
2.5 Self Supervised Learning, SSL . . . 9
2.5.1 Contrastive Learning in SSL . . . 9
2.5.2 Non-Contrastive Learning in SSL . . . 9
2.5.3 The Role of Siamese Networks in SSL . . . 9
2.5.4 Projector . . . 10
2.6 Additional Machine Learning Algorithms . . . 11
2.6.1 Support Vector Machines, SVM . . . 11
2.6.2 K-Neares Neighbors, KNN . . . 11
2.6.3 KMeans and Silhouette Score . . . 12
2.7 Representation Learning . . . 13
**3 Related work / Literature review . . . 14**
**4 Methology . . . 16**
4.1 Overview . . . 16
4.2 The Vector Quantized Variational Auto-Encoder, VQVAE . . . 16
4.2.1 Evolution from Variational Auto-Encoder, VAE . . . 16
4.2.2 Transition to discrete latent variables . . . 18
4.3 VQVAE implementation . . . 20
4.3.1 Informational flow . . . 20
4.3.2 Learning . . . 21
4.4 Modifying the VQVAE for Enhanced Self Supervised Learning . . . 23
4.4.1 Barlow Twins . . . 23

4.4.2 Modifying the VQVAE encoder . . . 24
4.4.3 Augmentation Techniques . . . 26
**5 Experimental Setup . . . 28**
5.1 UCR Archive . . . 28
5.2 Reconstruction evaluation . . . 29
5.3 Downstream evaluation . . . 29
5.3.1 Extracting Discrete latent variables . . . 29
5.3.2 Downstream tests . . . 30
**6 Results and discussion . . . 31**
**7 Conclusion . . . 32**

Chapter 1

# Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 2

# Theoretical background

## 2.1 Neural Networks

Neural Networks (NNs), the cornerstone of many modern artificial intelligence applications, are computational models inspired by the human brain. They consist of interconnected groups of artificial neurons, where each connection represents a synaptic strength or weight. These weights are adjusted through a process known as learning.

### 2.1.1 Structure of Neural Networks

Typically, NNs are structured in layers, including:

- **Input Layer:** This layer receives the raw input signal for the neural network.
- **Hidden Layers:** One or more layers where computation occurs through a system of weighted connections. These layers perform non-linear transformations of the inputs entered into the network.
- **Output Layer:** This layer produces the final output of the neural network.

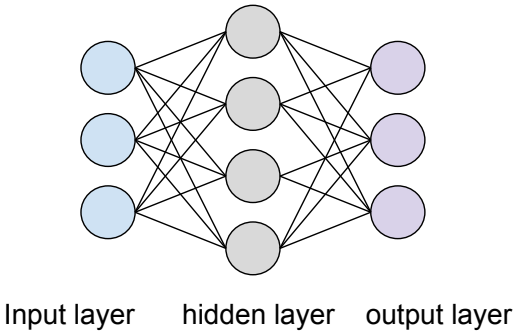


**Figure 2.1:** Simple neural network consisting of a single hidden layer.

### 2.1.2 Learning in Neural Networks

The learning process involves adjusting the weights of the connections in the network through a process known as backpropagation, which calculates the gradient of the loss function with respect to the neural network's weights.

### 2.1.3 Activation Functions

Non-linear activation functions play a key role in NNs by introducing non-linear properties to the network. This non-linearity is essential because most real-world data is non-linear, meaning it can't be represented accurately with just a straight line. The most common activation functions are sigmoid, tanh, and Rectified Linear Unit (ReLU).

ReLU has become a industry standard due to its simplicity and efficiency in backpropagation. It is expressed as: $ReLu(x) = \max(0, x)$, with a simple gradient 1 for positive inputs and 0 for non-positive. In addition to its simplicity it is more efficient at preventing vanishing gradients compared to sigmoid and tanh [VL21].

### 2.1.4 Supervised and unsupervised learning

Using a neural network there are two key learning paradigms. Supervised and unsupervised learning. Supervised learning refers to a learning process where the network has access to prelabelled inputs which acts as targets. For each training example there will be a set of input values alongside a designated output value, which is often human labeled. A typical example is classification, where we measure how well the network performes by how well it classifies labels.

For unsupervised learning the training set does not include any labels, therefor we are limited to looking at how well the model minimizes or maximizes an assosiated cost function.

## 2.2 Convolutional Neural Networks, CNN

Convolutional Neural Networks (CNNs)[ON15] are a specialized kind of neural network using convolutional layers and pooling for processing data that has a grid like topology. Examples include timeseries data, which can be thought of as a 1 dimensional grid taking samples at regular time intervals, and image data, which is a 2 dimensional grid of pixels.

### 2.2.1 Convolutional layers

What characterizes a CNN is the use of a mathematical operation called convolution, which is a special kind of linear operation. The convolution leverages the spatial or temporal structure of the data by enforcing a local connectivity pattern between neurons of adjacent layers. The parameters of the convolutional layers are composed of a set of learnable filters or kernels, which have a small receptive field, but is applied to all of the input data. This is achieved by computing the dot product between the entries of the filter and the input, producing a activation map of that filter. As a result the network learns filters that activate when they see some specific type of feature at some spacial or temporal position in the input.

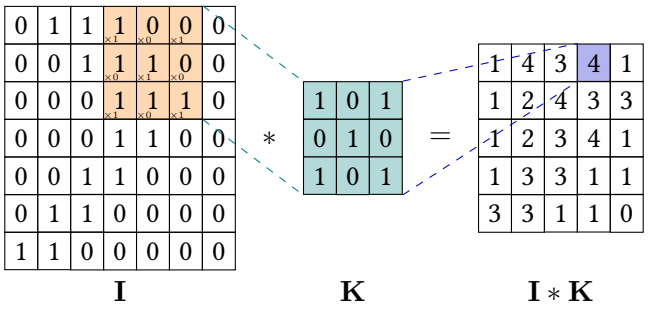


**Figure 2.2:** Example of convolutional operation. $\mathbf{I}$ is the input, $\mathbf{K}$ is the kernel and $\mathbf{I} * \mathbf{K}$ is the activation map. Illustration taken from the Random TikZ collection[Rie22]

In a convolutional layer, the configuration involves not only the selection of the filter (or kernel) size but also the specification of the stride length and the padding. The stride defines the number of units the filter shifts over the input data for each convolution operation. A stride of one

moves the filter one unit at a time, capturing fine-grained information, while larger strides can speed up the computation and provide a more global view of the input data, albeit with reduced spatial resolution. This reduction in spacial resolution is necessary for tasks like compression which we will discuss in the encoder section.

### 2.2.2 Pooling

A pooling layer is in essence a filter using a specified operation to reduce dimensionality. The pooling layer works similarely as a convolutional layer, they both slide through the temporal or spatial axis, computing a mathematical operation.

The most common pooling operations are:

- **Max Pooling:** Selects the maximum element from the region of the feature map covered by the filter. This method is effective at capturing the presence of features.
- **Average Pooling:** Computes the average of the elements in the region of the feature map covered by the filter, which helps in smooth feature representation.

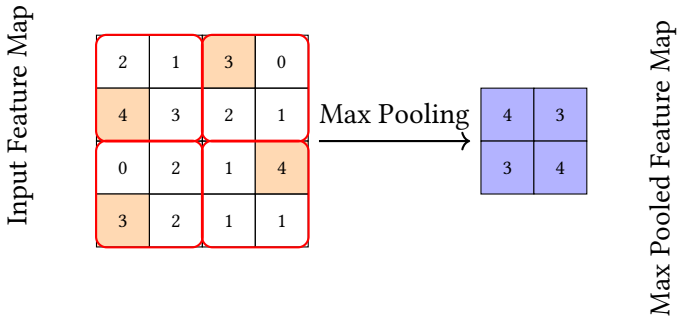


**Figure 2.3:** Illustration of a max pooling operation. The input feature map is reduced in size by applying a max pooling filter with size 2x2 (red boxes), which selects the maximum value in each filter region to produce the max pooled feature map.

Using a pooling layer in a neural network architecture helps the network to learn better feature representations. This is beneficial for enhancing the efficiency of convolutional filters.

### 2.2.3 Architecture of CNNs

A typical CNN architecture consists of the following layers:

- **Convolutional Layer** — Applies the learnable filters on the input data, producing a activation map. Each filter detects different features by convolving with the input.
- **Activation Function** — Typically, a ReLU is applied to introduce non-linearity into the model, allowing it to learn more complex functions.
- **Pooling** — This layer reduces the spatial size of the representation, reducing the number of parameters and computation in the network, and hence, also controlling overfitting.
- **Fully Connected Layer** — Neurons in a fully connected layer have full connections to all activations in the previous layer. Primarely used for processing the information obtained through the convolutional layers to perform a task like classification.

## 2.3 Encoder

The Encoder has the task of compressing the high-dimensional input data into a compact and manageable representation. It is a component of the Variational Auto-Encoder architecture, enabling the compression of input to latent variables. The encoded data should capture important characteristics and features from the input data necessary for decompression while also removing unessesary and reduntant information. It operates by using a block like structure, where each block is designed to process the input data sequentially.

### 2.3.1 Downsampling blocks

Downsampling blocks are integral components within encoder architectures, facilitating the reduction of dimensionality of the feature maps. Based on the principles of CNNs the downsampling is achieved through a convolutional layer, that not only detects patterns within the input data but also strategically reduces the dimensions of the input. The convolutional layer applies filters to the input, creating activation maps that highlight significant features. By adjusting the stride of the convolution, the layer can downsample the input, distilling the information into a more compact form.

Following the convolutional layer, batch normalization is employed. This technique normalizes the output of the convolution by adjusting and scaling the activations. It allows the network to use higher learning rates, as it reduces the internal covariate shift which in turn makes the learning process more efficient and stable, as described by Ioffe and Szegedy[IS15].

### 2.3.2 Residual block

To enhance the encoder's ability to learn complex patterns and facilitate the training of deeper networks, residual blocks are integrated. These blocks, which were first introduced by He et al. in their paper on deep residual learning [He+15], incorporate shortcut connections that bypass

one or more layers. The primary advantage of these shortcuts is to allow gradients to flow directly through the network, mitigating the vanishing gradient problem that often plagues deep neural networks.

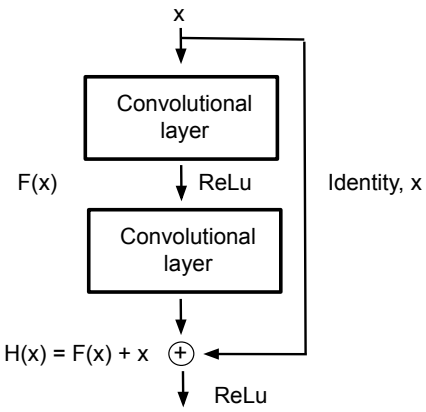


**Figure 2.4:** Illustration of the Resnet block from He et al's original paper[He+15]

### 2.3.3 Encoder Architecture

The complete encoder architecture comprises a sequence of downsampling blocks, each further distilling the data. This is then followed by a series of Residual blocks to form deeper representations efficiently. The overall structure is sequential, with each block building upon the previous one to gradually reduce the input data's dimensionality while capturing patterns and important features. The number of downsampling blocks is determined by a downsampling rate which is dependent of the size of the input data, and the amount of compression needed for the encoding task.

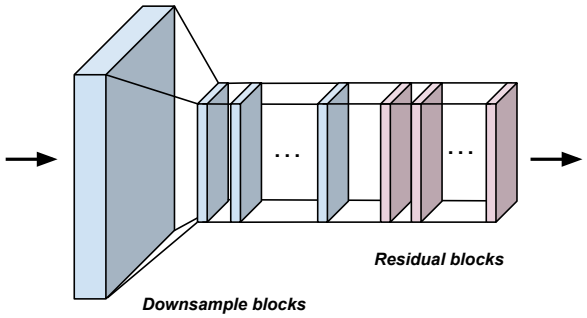


**Figure 2.5:** Illustration of the Encoder architecture used in the VQVAE implementation. The first downsample block reduces the dimensionality, followed by further downsample blocks to capture important features.

## 2.4 Decoder

While the encoder is responsible for data compression, the decoder is responsible for decompression. Instead of reducing the dimensions of the data, it will upsample it to a higher dimensional space.

### 2.4.1 Upsampling Blocks

Contrary to the encoder's downsampling blocks, the decoder employs upsampling blocks designed to incrementally reconstruct the higher-dimensional data from the lower-dimensional representation. The upsampling process is achieved through the use of transposed convolutional layers, often referred to as deconvolutional layers. These layers effectively reverse the operation of convolution by learning to expand a compressed feature map onto a larger spatial canvas. The transposed convolutions apply filters in a manner that distributes the encoded feature values over a higher-resolution grid, interpolating additional data points as necessary to achieve the desired output size. This process not only increases the spatial dimensions but also refines the feature map to recover details that were compressed during encoding. Following the transpose convolutional layer the upsample block uses batch normalization and ReLu activation function.

### 2.4.2 Decoder Architecture

The decoder's architecture is characterized by a series residual blocks followed by upsampling blocks that progressively restore the data's dimensionality. The number of upsampling blocks

is the same as number of downsampling blocks in the encoder.

## 2.5 Self Supervised Learning, SSL

Self-supervised learning (SSL) is a form of unsupervised learning where the data itself provides the supervision. SSL algorithms learn to predict unobserved or hidden parts of the input from observed parts. This is achieved trough carefully designed loss functions, which enforce the learning of usefull features by solving pretext tasks. Modern mainstream SSL frameworks can be divided into two categories, contrastive and non-contrastive learning.

### 2.5.1 Contrastive Learning in SSL

Contrastive learning is a popular method in SSL, particularly for learning visual representations. It relies on contrasting positive pairs (similar or related data points) against negative pairs (dissimilar or unrelated data points). The intuition is to learn embeddings such that similar samples are closer to each other in the embedding space, while dissimilar ones are farther apart. Several prominent contrastive learning methods, such as MoCo[He+19] and SimCLR[Che+20], effectively utilize positive and negative pairs to learn contrasted representations.

### 2.5.2 Non-Contrastive Learning in SSL

Non-Contrastive learning methods in SSL, by contrast, does not depend on negative pairs. Instead, these approaches aim to learn representations by encouraging similarity between different augmented views of the same data point. This approach is based on the principle that different transformations of the same data should yield similar representations, thereby ensuring consistency and robustness in the learned features.

Notable non-contrastive methods include BYOL (Bootstrap Your Own Latent)[Gri+20] and Barlow Twins [Zbo+21].

### 2.5.3 The Role of Siamese Networks in SSL

Siamese networks[LST15] is a neural network architecture designed for specialized tasks that require the comparison of input pairs. The defining characteristics of these networks is the dual-branch structure, where two identical sub-networks with shared parameters process two seperate inputs. The outputs of the sub networks, often refered to as twin embeddings, are then brought togheter to evaluate the degree of similarity or difference.

Many modern SSL approaches base their architecture on the siamese network as it is efficient for teaching a sub network to discriminate between different classes or types of data as explored by Lee and Aune in their paper on SSL methods on timeseries [LA21]

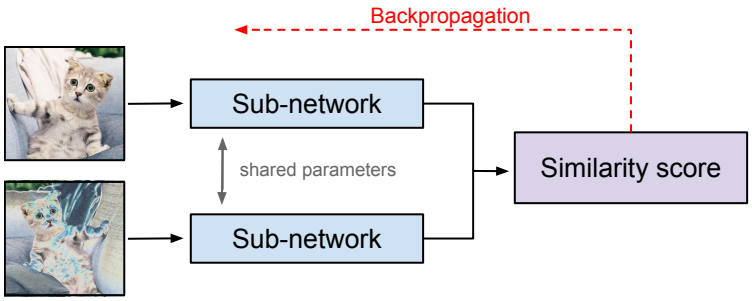


**Figure 2.6:** Example of a non contrastive siamese network using augmented views. Here the images are processed to calculate a similarity score, the backpropagation arrow indicates that the similarity score is then used to update the weights of the sub network. Pictures taken from www.pexels.com

### 2.5.4 Projector

The projector serves as a component in the sub-network component of siamese networks. Its main function is to transform a lower-dimensional encoded representation into a higher dimensional space suited for a SSL loss function. Common SSL techniques using projectors include BYOL, SimCLR and Barlow Twins.

#### Projector architecture

The projector architecture is designed with the objective of expanding the number of features through fully connected layers. The arcitecture used in this methology consist of the following components.

Bit vague

- **Global max pooling.** Condenses the input encodings into a condensed and robust set of features. This helps distill the most salient aspects of the encoded data.
- **Fully connected layers**. Broadens the dimensions and learns weights and biases that emphasize differences
- **batch normalization**
- **ReLu activation function**

## 2.6 Additional Machine Learning Algorithms

### 2.6.1 Support Vector Machines, SVM

Support Vector Machines (SVM) is a supervised machine learning algorithm widely used for classification and regression tasks. The core idea of SVM is to establish an optimal hyperplane that maximizes the margin between different class labels. This approach is effective in high-dimensional spaces, addressingthe curse of dimensionality through its dependence on support vectors. These support vectors are a critical subset of the training data that influences the construction of the decision boundary. SVM's adaptability is further enhanced by its capability to incorparate various kernel functions, such as linear and radial basis functions, to suit different types of data distributions. The foundation of SVM is largely based on the work done by Vapnik et al[CV95].

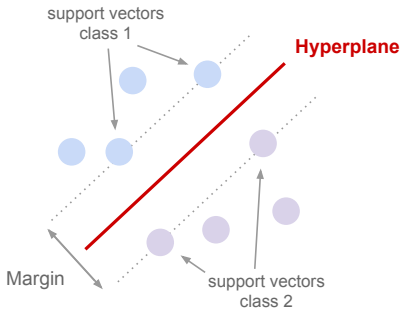


**Figure 2.7:** Vizualisation of the hyperplane, margin and support vectors in a SVM procedure.

### 2.6.2 K-Neares Neighbors, KNN

K-Nearest Neighbors (KNN) is a versatile supervised learning algorithm that classifies data points based on the majority class of their K nearest neighbors. It operates under the assumption that similar data points are likely to be close in proximity. One of the key strengths of KNN is its simplicity and effectiveness in classification tasks. However the KNNs performance is significally impacted by the choice of K and the distance metric used, visualized in fig 2.9. The choice of K is often determined by techniques like cross validation.

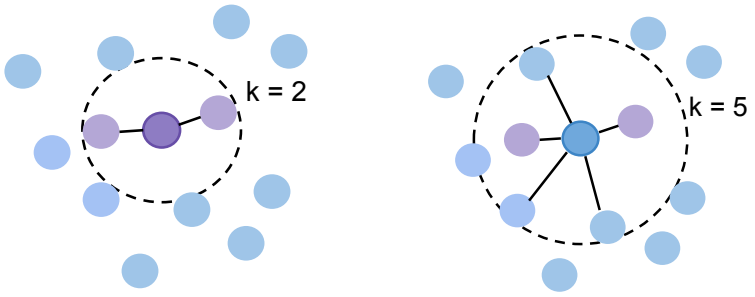


**Figure 2.8:** Classification of middle point in a KNN procedure. For $k = 2$ the majority vote is purple while for $k = 5$ it is blue.

### 2.6.3 KMeans and Silhouette Score

K-Means is a popular unsupervised learning algorithm used primarely for clustering. Its goal is to partition the data into K distinct clusters, each represented by its centroid, which is the mean of the points in the cluster. The algorithm iteratively assigns each data to the neares cluster, based on the Euclidian distance to the centroid, and then recalculates the centroids. This process repeats until the centroids stabilize or a certain iteration criteria is met.

The Silhouette Score is a metric for evaluating the effectiveness of the clustering process. It measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The score ranges from -1 to 1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.

- **Cohersion (a)**. It measures the average distance of the data point from all other points in the same cluster.
- **Seperation (b)**. It is the average distance of the data point to the points in the nearest cluster that the data point is not apart of.

The formula for the Silhouette score of a single data point ($i$) is given by:

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{2.1}$$

The overall silhouette score, $S$, for a dataset is the mean of the silhouette score of each data point.

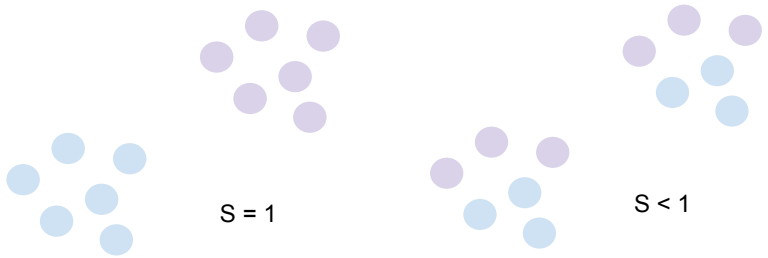


**Figure 2.9:** Illustration of the silhuette score metric applied. On the left we see well seperated clusters with a silhouette score equal to 1. On the right a not so well seperation, giving a silhouette score less than 1.

## 2.7 Representation Learning

Chapter 3

# Related work / Literature review

This work builds on the original paper by Van den Oord et al[OVK17], "Neural Discrete Representation Learning", which introduce the Vector Quantified-Variational AutoEncoder (VQ-VAE). The VQVAE represents a significant advancement in generative models, particularly in learning discrete latent representations. The model's distinctive approach of employing vector quantization mitigates issues commonly assosiated with posterior collapse in traditional Variational AutoEncoders (VAEs).

Following the introduction of VQVAE, there have been several notable advancements in this area. Researchers have explored various enhancements, such as improving the models ability to handle higher-dimensional data and increasing its efficiency in representation learning. A significant contribution in this domain was made by Razavi et al[ROV19], who extended the VQVAE model to VQVAE-2, offering improvements in generating high fidelity images.

Lee et al in their paper on VQVAE[LAM23] also proposed a approach to enhancing the VQ-VAEs ability capture high frequency information in timeseries by using frequency augmentation techniques. fill

Other work has focused on improving traditional VAEs using contrastive SSL loss funcitons. Abid et al[AZ19] showed that using a contrastive approach in VAEs proved to improve the salient information in the latent variables.

Parallel to the developments in VQVAE, the Barlow Twins approach, as proposed by Zbontar et al[Zbo+21], has gained attention in the field of SSL. This method emphasizes the learning of representations by making the features of augmented versions of the same image as similar as possible, while minimizing redundancy among features of different images. The Barlow Twins approach has shown promising results in enhancing the learning capabilities of neural networks, particularly in scenarios with limited or no labeled data. Research done by Lee

et al in their paper comparing SSL techniques on timeseries[LA21], show that the decorrelation technique that Barlow Twins provide is efficient on a variatity of datasets in the UCR Archive[Dau+18].

Chapter 4

# Methology

## 4.1 Overview

In this section, we present a comprehensive overview of our chosen methodologies, detailing their derivation and techniques. Our focus begins with the Vector Quantized Variational Autoencoder (VQVAE), an advanced model that represents an evolutionary step from the conventional Variational Autoencoder (VAE). We explore the foundational principles behind the VQVAE, including its development from the traditional VAE architecture, the core optimization challenges it addresses, and our approach to its implementation.

Further, we introduce a modified version of the VQVAE inspired by SSL and the Barlow Twins technique.

## 4.2 The Vector Quantized Variational Auto-Encoder, VQVAE

### 4.2.1 Evolution from Variational Auto-Encoder, VAE

The Vector Quantized Variational Autoencoder (VQ-VAE) evolves from the foundational principles of the Variational Autoencoder (VAE), sharing its dual-model architecture. At its core, a VAE consists of two models: the encoder, which compresses input data into latent representations, and the decoder, which reconstructs data from latent representations. Giving rise to the bottleneck structure characterizing autoencoders.

As a fundemental principle of latent variable models, the Variational Autoencoder explicates the relationship between the observed data and latent variables through the marginal distri-

bution. This relationship is central to understanding how a VAE encodes and decodes data, capturing the essence of the data's underlying structure.

Lets consider a dataset, denoted as $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$, which comprises observed data points. Corresponding to these data points, we have latent variables $\mathbf{z} = \{z_1, z_2, \cdots, z_n\}$, which represent hidden factors or features extracted by the VAE. The model parameters denoted as $\theta$, govern the transformation from data to latent space and vice versa.

The relationship is explained by the marginal distribution, expressed as follows:

$$p_\theta(x) = \int_z p_\theta(x, z)dz = \int_z p_\theta(z)p_\theta(x|z)dz \tag{4.1}$$

### Variational inference and optimization

The optimization problem is to optimize $\theta$ to efficiently transform input data into a latent representation and reconstruct the input based on the latent representation. Using the posteriors $p_\theta(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{x}|\mathbf{z})$ both parameterized by $\theta$.

### Challenges in Maximum Likelihood Estimation

Optimizing the model parameters $\theta$ through Maximum Likelihood Estimation poses significant computational challenges. The optimization objective for $\theta$ is to maximize the log likelihood of the observed data:

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log p(x_i|\theta) \tag{4.2}$$

However, this requires evaluating an integral over the latent variables that is intractable:

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log \int_{z_i} p_\theta(x|z)p_\theta(z) \tag{4.3}$$

### Variational Approximation

To circumvent the intractability of the integral, we approximate the posterior $p_\theta(z|x)$ with an encoder model $q_\phi(z|x)$, parameterized by a neural network with variational parameters $\phi$, that

maps the input data to the latent space. The decoder neural network aims to reconstruct the input data $\mathbf{x}$, thereby approximating the conditional likelihood $p_\theta(\mathbf{x}|\mathbf{z})$.

$$\mathbf{x} \rightarrow \underset{\sim q_\phi(\mathbf{z}|\mathbf{x})}{\text{EncoderNeuralNet}_\phi(\mathbf{x})} \rightarrow \mathbf{z} \rightarrow \underset{\sim p_\theta(\mathbf{x}|\mathbf{z})}{\text{DecoderNeuralNet}_\theta(\mathbf{z})} \rightarrow \hat{\mathbf{x}} \tag{4.4}$$

#### Evidence Lower Bound

Due to the infeasibility of optimizing the likelihood directly it becomes necessary to adopt an alternative objective function. This function incorporates the variational aproximated posterior, $q_\phi(\mathbf{z}|\mathbf{x})$, and is commonly known as the Evidence Lower Bound (ELBO), denoted as $\mathcal{L}_{\theta,\phi}(x)$.

$$\mathcal{L}_{\theta,\phi}(x) = \log p_\theta(x) - \mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x)) \tag{4.5}$$

The first term in the ELBO, $\log p_\theta(x)$, represents the log likelihood of the data under the model parameters $\theta$. This term encourages the reconstructed data $\hat{\mathbf{x}}$, generated by the decoder network, to be as close as possible to the original input data $\mathbf{x}$. The second term,

$$\mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x)) \tag{4.6}$$

is the Kullback-Leibler (KL) divergence between the approximate posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and the true underlying distribution, $p_\theta(\mathbf{z}|\mathbf{x})$.

By maximizing the ELBO with respect to $\theta$ and $\phi$ we achieve the following:

1. Enhancement of the marginal likelihood $p_\theta(x)$, leading to improved reconstruction of the input data by the decoder.
2. Reduction of the Kullback-Leibler (KL) divergence between the approximate posterior $q_\phi(z|x)$ and the true posterior $p_\theta(z|x)$, resulting in a more accurate encoder model for the latent variables.

By maximizing the ELBO, we get a two for one, we refine the model's parameters to enhance data reconstruction accuracy and improve the encoder's inference of the latent representations, aligning them more closely with the true assumed underlying distribution.[KW19]

### 4.2.2 Transition to discrete latent variables

VQ-VAE retains the encoder-decoder structure but introduces a discrete twist. It employs a vector quantization mechanism that translates the continuous latent variables of a traditional VAE into discrete ones, denoted as $\mathbf{z_q}$

The fundamentak shift from continous to discrete latent variables influences the dynamics of the model. In a VAE, the priors and posteriors are typically assumed gaussian. This assumption enables the application of the reparameterization trick[KW13], which is beneficial for effective training, primarily due to its contribution to reducing gradient variance. However its important to note that this approach imposes a constraint that both the prior and posterior must conform to the underlying assumption of its shapes. This requirement, while beneficial for certain aspects of training, introduces a limitation in the models flexibility to represent a wider variety of data distributions.

### Vector quantization

Vector quantization (VQ) is a key component of the VQ-VAE model, first proposed by van den Oord et al[OVK17]. Within the architecture of the VQ-VAE, the encoder transforms input data into a series of continuous latent vectors. These vectors are then discretized through vector quantization, which maps each continuous vector to the closest embedding vector in a predefined set known as the codebook.

The codebook, denoted by $Z$, is composed of $K$ discrete embedding vectors (or tokens), expressed as $Z = \{z_k\}_{k=1}^{K}$. Here $K$ represents the cardinality of the discrete latent space, implying that the latent space comprises K distinct categories. Each embedding vector $z_k$ resides in a $d$-dimensional space. $z_k \in \mathbb{R}^d$

Quantization is implemented through a nearest neighbor search within $Z$. For any given continuous latent vector $z_{ij}$, the quantized vector $(z_q)_{ij}$ is determined by locating the token $z_k \in Z$ that minimizes the euclidian distance to $z_{ij}$.

$$(z_q)_{ij} = \arg\min_{z_k \in Z} ||z_{ij} - z_k||$$

### Effect on KL divergence

As Van den Oord et al describes in their paper [OVK17] each latent vector index maps deterministically to a single embedding vector in the codebook. As a result, the probability distribution of the latent indices given any input is a one-hot distribution. This means that for any given input, the posterior assigns a probability of one to a single latent index and zero to all others.

$$q(z = z_q|x) = \begin{cases} 1 & \text{if } q = \arg\min_{z_q \in Z} ||z_{ij} - z_q|| \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

(Stemmer formuleringen her??)

We can investigate the effects on the KL divergence by viewing the model as a VAE, as Van den Oord in his paper. Meaning that we bound $\log p_\theta(x)$ by the ELBO $\mathcal{L}_{\theta,\phi}$. By now defining a simple uniform distribution over $z$ and using the one-hot deterministic $q(z = q|x)$ we can derive the KL-divergence as follows

$$\begin{aligned}\mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z)) &= \sum_i \underbrace{q(z=q|x)}_{=1} \log \frac{\overbrace{q(z=q|x)}^{=1}}{\underbrace{p(z=q)}_{\frac{1}{K}}} \\ &= \log K\end{aligned} \tag{4.8}$$

(Skriv om .. Noen bruker p(z) andre p(z—x) for Kullback. Brukte p(z—x) i ELBO)

The constant KL divergence indicates a stable relationship between the aproximate posterior and the prior. The relationship is stable regardless of any assumption of shapes and forms for the posterior. This efficiently addresses the ”posterior collapse” or ”KL vanishing”[WBC23] issue commonly encountered in traditional VAEs. Posterior collapse occurs when the latent variables become less informative as the decoder learns to ignore the information provided by the encoder. In VQVAE, by allowing the posterior to take any shape without being bound by the constraints of a Gaussian distribution, the model maintains significance of the latent variables throughout training.

## 4.3 VQVAE implementation

The implementation is based on the paper TimeVQVAE[LAM23]. Here they argue for a 2 stage approach. The first stage consists of optimizing the Encoder, Codebook and Decoder to efficiently compress the input into tokens and then reconstruct the input. This is achieved by minimizing the informational loss obtained by comparing the input $x$ with the reconstructed $\hat{x}$. The next stage consists of training a transformer to approximate the prior.

Stage 1 aligns with our objectives. They base their implementation on a improved approach to VQVAE, designed to extract better high frequency information. We base our implementation on the naive VQVAE as we are interested in the general case for VQVAEs on timeseries.

### 4.3.1 Informational flow

Prior to the encoder denoted by $E$, the input $\mathbf{x}$ (a timeseries) is passed through a Short Time Fourier Transform (STFT), producing $\mathbf{u}$.

The **STFT** converts the time-domain signal into a time-frequency representation. This trans-

formation allows the encoder to analyze the signal in both time and frequency domains simultaneously, providing a comprehensive view of the signal's spectral content as it varies over time. The encoder thus learns convolutional filters that capture and extract frequency patterns within the input data. Reversly the Inverse Short Time Fourier Transform (**ISTFT**) converts the time-frequency representation back to a time-domain representation. Both STFT and ISTFT are computational methods grounded in a series of fourier transforms. A key parameter is $n_{\text{fft}}$, dictating the resolution of the frequencies in the transformed domain.

Next the encoder compresses $\mathbf{u}$ into a continuous latent variable $\mathbf{z}$ followed by a quantization to $\mathbf{z_q}$. Transforming the continous latent space to a discrete latent space. The input is at this state represented as tokens in our codebook. The tokens or discrete latent variables is then passed through the decoder denoted as $D$, projecting the discrete latent space back into a time-frequency domain, $\hat{\mathbf{u}}$. Where it is then mapped back to a time-domain using ISTFT. Giving the reconstructed input $\hat{\mathbf{x}}$. This is summerized in the figure 4.1

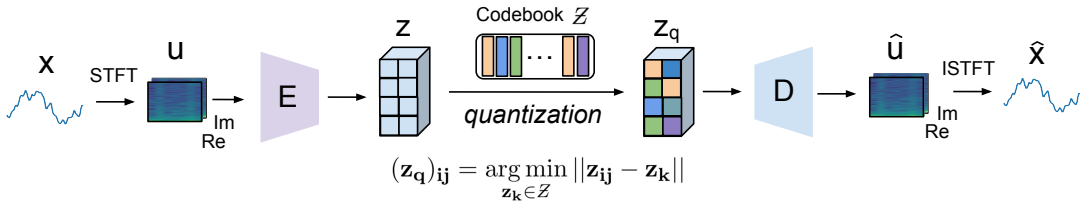


**Figure 4.1:** Illustration of the VQVAE. Illustration and implementation inspired by the TimeVQVAE paper[LAM23]

### 4.3.2 Learning

Learning is achieved through updating the networks parameters through backpropagation. This process requires a formulation of a suitable loss function that reflects the learning objective of the model.

Based on the TimeVQVAE[LAM23] paper and the ELBO formulation the total loss for the VQ-VAE is the sum of two parts. Reconstruction loss, $\mathcal{L}_{\text{recon}}$, and codebook loss, $\mathcal{L}_{\text{codebook}}$. Contrary to the VAE, we dont have to include the minimization of the KL divergence.

$$\mathcal{L}_{VQVAE} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{codebook}} \tag{4.9}$$

### Reconstruction

By minimizing the reconstruction loss, $\mathcal{L}_{\text{recon}}$, we align with maximization of $\log p_\theta(x)$ in the ELBO formulation 4.5. We define the reconstruction loss to be:

$$\mathcal{L}_{\text{recon}} = ||x - \hat{x}||_2^2 + ||u - \hat{u}||_2^2 \tag{4.10}$$

This gives us a metric or score on how well the reconstructed timeseries is compared to the input. As the metric acts on both time and time-frequency domain, we ensure that the model learns to reconstruct well for both of these domains.

To furhter clarify, the loss is calculated based on the Mean Squared Error (MSE), a standard metric in regression tasks. The MSE between the time domain input and reconstructed time domain input is defined as:

$$MSE(x, \hat{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{4.11}$$

Consequently, the reconstruction loss is expressed as the sum:

$$\mathcal{L}_{\text{recon}} = MSE(x, \hat{x}) + MSE(u, \hat{u}) \tag{4.12}$$

### Codebook-learning loss

The codebook learning loss will ensure that the quantized latent representations, $\mathbf{z_q}$, effectively approximate the continues latent space $\mathbf{z}$, thereby maintaining a structured and meaning full latent representation.

The codebook-learning loss is defined as

$$\mathcal{L}_{\text{codebook}} = ||\text{sg}\left[E(\text{STFT}(x))\right] - z_q||_2^2 + \beta||E(\text{STFT}(x)) - \text{sg}\left[z_q\right]||_2^2 \tag{4.13}$$

The first term in the codebook-learning loss,

$$||\text{sg}\left[E(\text{STFT}(x))\right] - z_q||_2^2 \tag{4.14}$$

measures the difference between the stop-gradient, $\text{sg}\left[\cdot\right]$, aplied encoded representation and the quantized latent vector $z_q$. This term ensures that the chosen quantized vectors from the codebook are as close as possible to the output of the encoder. By applying the stop gradient operator, we prevent the gradients from backpropagating through the encoder during this part of the loss calculation. Isolating the calculating to just act on the codebook network.

The second term,

$$\beta||E(\text{STFT}(x)) - \text{sg}\left[z_q\right]||_2^2 \tag{4.15}$$

encourages the encoder's output to move closer to the quantized vectors. The parameter $\beta$ acts as a balanding factor, controlling the strength. We typically set this parameter to be 1, ensuring that the encoder is updated at the same rate as the codebook. (Kilde?). By applying the stop gradient on the quantized vector $z_q$, this term focuses on updating the encoder's parameters to produce outputs that are more aligned with the existing codebook vectors.

Togheter, these two terms ensure that the codebook adapts to the encoded, $z$. Simultaneously, the encoder adapts to the quantized $z_q$. As a result the Encoder is tuned to produce representations that are effectively quantized by the codebook. As done in the reconstruction loss the norms, $|| \cdot ||_2^2$, is replaced with MSE in our loss calculation.

## 4.4 Modifying the VQVAE for Enhanced Self Supervised Learning

In this section we present our modification of the VQVAE, specifically tailored to leverage the strengths of Self Supervised Learning (SSL). The primary goal of this modification is to build upon the existing VQVAE architecture, while adapting its encoder to produce more robust and informative latent representations. To achieve this we propose the integration of the Barlow Twins method with the VQVAE encoder.

### 4.4.1 Barlow Twins

The Barlow Twins method, introduced by Zbontar et al in their 2021 article "Self-Supervised Learning via Redundancy Reduction"[Zbo+21], presents a alternative non-contrastive objective function for SSL. The core idea being an objective function that encourages the learning of feature representations by reducing redundancy while retaining critical information from the input data.

The rationale behind the objective function, $\mathcal{L}_{\text{BT}}$, is grounded in the properties of the identidy matrix. Viewing the identity matrix as a cross correlation matrix the zero valued off diagonal elements indicates no redundancy between different features. Since the diagonal elements are ones it suggest that each feature is similar, containing similar informational value. Thereby, if a cross correleation matrix equals the identity matrix we have achieved no redundancy and full informative value between the features.

#### Objective function of Barlow Twins

The objective function, or similarity score, of the Barlow Twins method can be formulized as follows:

$$\mathcal{L}_{BT}(\mathbf{z_1}, \mathbf{z_2}) = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2 \tag{4.16}$$

where $C$ is the cross correlation matrix computed between the outputs ($\mathbf{z_1}$ and $\mathbf{z_2}$) of two identical networks fed with distorted version of the same data (siamese style). $C_{ii}$ is the diagonal elements and $C_{ij}$ is the offdiagonal elements. The parameter $\lambda$ controls the trade-off between invariance and redundancy reduction.

### Siamese architecture

The barlow Twins method builds apon the siamese architecture, where each subnetwork consists of a encoder and a projector.

The procedure is as follows. First a batch normalization is applied to the distorted views ($Y^A$, $Y^B$) followed by a compression to latent variables by the Encoder, $E$. The projector, $P$, acts as a dimensionality expander with learnable parameters. It learns to expand the encodings into a space where the barlow twins loss can be efficiently applied. The intuition behind expanding the dimensions is that more features in the cross-correlation matrix enable a more efficient and robust comparison while also allowing the model to capture a broader range of characteristics within the encodings. Next the empirical cross correlation is compared with the Identity matrix as visualized in fig 4.2

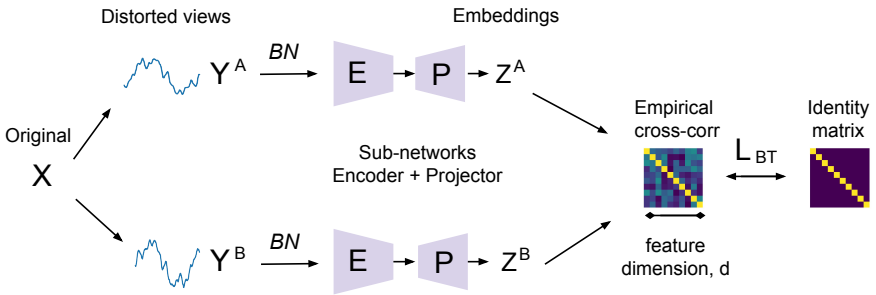


**Figure 4.2:** Illustration of the Barlow Twins procedure, inspired by the original paper[Zbo+21].

## 4.4.2 Modifying the VQVAE encoder

The proposed modification involves extending the VQVAEs encoder section into a two branch structure. Hence adopting a siamese architecture by augmenting the input $\mathbf{x}$ into two views, $(\mathbf{x_A}, \mathbf{x_B})$.

The same encoder in both branches then encodes both time-frequency views, ($\mathbf{u_A}$, $\mathbf{u_B}$). Producing two latent variables ($\mathbf{z_A}$, $\mathbf{z_B}$). We continue by calculating the barlow twins loss. First applying a batch normalization, $BN$, then using a projector denoted as $P$ to expand the dimensionality of the latent variables. As done by Zbontar et al in their paper[Zbo+21]. After the projection to a higher feature space, the cross correlation matrix, $C$, is calculated followed by the barlow twins loss function, $\mathcal{L}_{BT}$. This is summerized in fig 4.3. As illustrated, the upper branch is the regular VQVAE and the lower branch is the extension allowing the encoder to process two views.

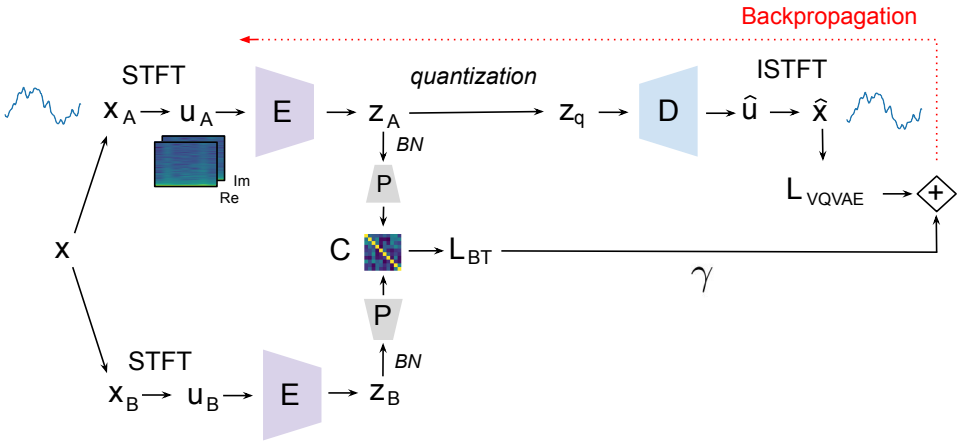


**Figure 4.3:** Illustration of the Barlow Twins modification to VQVAE including the loss function calculation.

### Learning

The modified VQVAE now has a additional SSL objective. As done by Bardes et al in their paper VICReg[BPL21] we scale the barlow twins loss by the projected feature dimension, $d$, visualized in fig 4.2. Our Barlow Twins loss is extended as follows:

$$\mathcal{L}_{BT}(\mathbf{z_1}, \mathbf{z_2}) = \frac{1}{d}\left[\sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2\right] \tag{4.17}$$

In the original barlow twins paper[Zbo+21] they found that a low valued $\lambda$ equal to $0.005$ gave the optimal balance between invariance and redundancy reduction. They also found that the dimensionality of the projector had a substantial effect on the efficiency of the barlow twins loss calculation. A higher dimensionality gave better results. Based on this find, we expand the encoded dimension to $d = 4096$ as done by Lee et al[LA21].

The total loss function, $\mathcal{L}_{\text{BT-VQVAE}}$, of our modified VQVAE becomes:

$$\mathcal{L}_{\text{BT-VQVAE}} = \mathcal{L}_{\text{VQVAE}} + \gamma \cdot \mathcal{L}_{\text{BT}} \tag{4.18}$$

The loss calculation is visualized in fig 4.3. The parameter $\gamma$ determines the weight of the barlow twins loss in the calculation. The $\mathcal{L}_{VQVAE}$ is calculated as done in eq 4.9 using the upper branch reconstruction and codebook.

### 4.4.3 Augmentation Techniques

The augmentation techniques employed in this implementation, inspired by the paper by Wen et al[Wen+20] and the paper by Lee and Aune[LA21], are categorized into time domain augmentations and time-frequency domain augmentations. A good augmentation should preserve overall schemantics of the timeseries during the transformation.

#### Time Domain Augmentations

These techniques manipulate the time series data directly in the time domain. A summerization of the implemented techniques are:

- **Flipping**: Reversing the sequence of the time series, simulating time-reversed scenarios.
- **Jittering**: Adding Gaussian noise to the time series, introducing random variations akin to sensor noise or measurement errors.
- **Amplitude Resizing**: Adjusting the amplitude of the time series, simulating variations in signal strength.
- **Adding Slope**: Incorporating a linear trend, emulating gradual shifts or drifts in the data.

#### Time-Frequency Domain Augmentations

These techniques operate on the time-frequency representation of the time series, offering a different perspective for augmentation:

**STFT Augmentation**: The most comprehensive technique in this implementation is the STFT (Short Time Fourier Transform) augmentation. It involves transforming the time series into the frequency domain using STFT, modifying the phase components using noise, and then reconstructing the time series. This process is particularly effective in introducing complex and subtle variations that are not easily achievable through direct time domain manipulations.

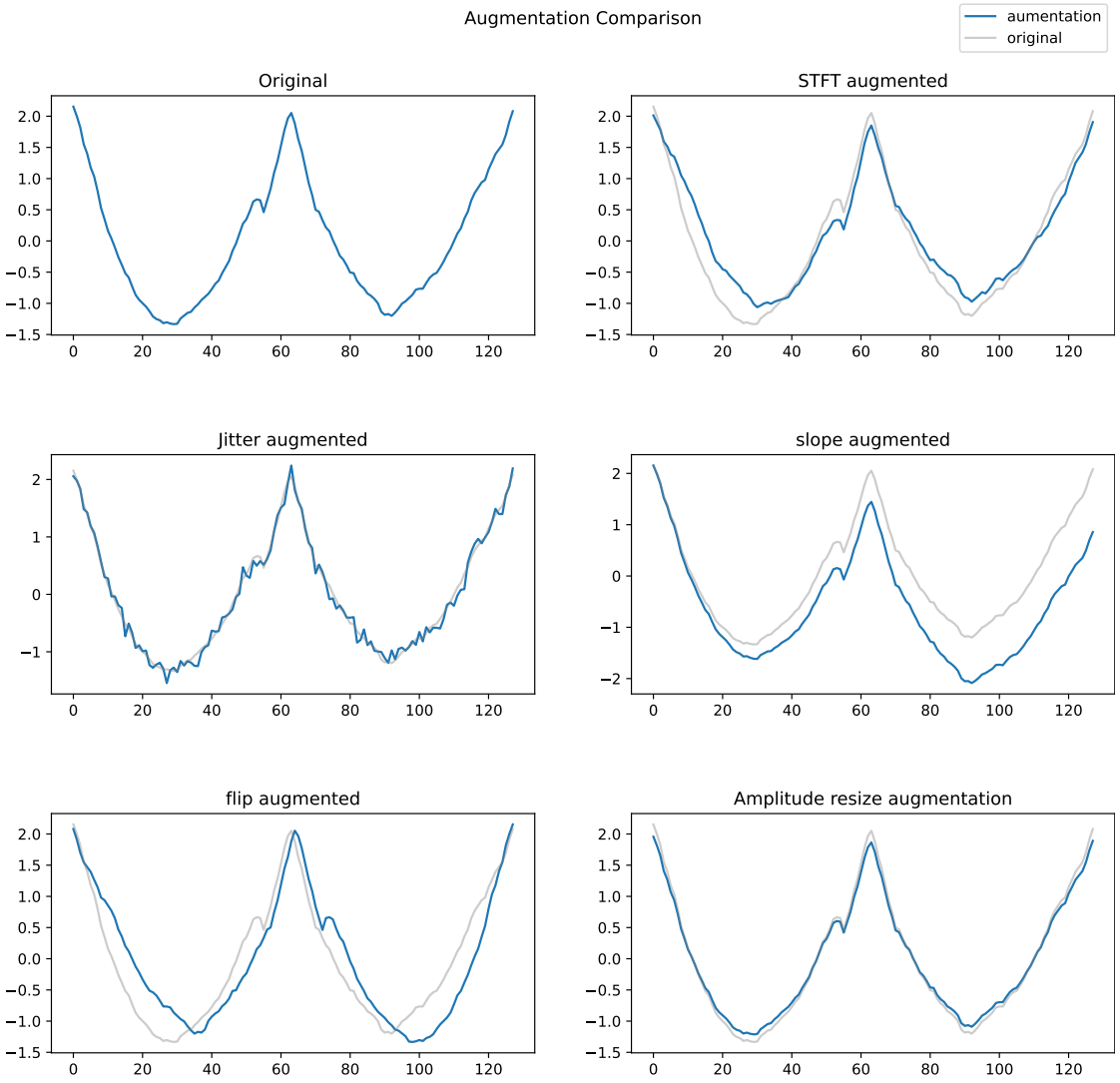


**Figure 4.4:** Visualization of the flip, slope and STFT augmentation aplied on a timeseries.

Chapter 5

# Experimental Setup

Our experimental investigation is structured into two primary focuses:

1. **Reconstruction Analysis**: This part of the experiment focuses on assessing the ability of our models to accurately reconstruct time series data. The effectiveness of the reconstruction process is a crucial indicator of the model's understanding and representation of the input data.
2. **Representation Learning Evaluation**: The second aspect of our experiment delves into the quality of the learned representations in the discrete latent space. Here, we aim to examine the intricacies of how the models, especially our modified version, encode and represent the time series data in the latent space.

As a benchmark for comparison, we establish the naive VQVAE as the baseline model. Against this baseline, we examine the performance of the Barlow Twins modified VQVAE (BT-VQVAE). We examine how the BT-VQVAE performs as a function of $\gamma$ in eq 4.18. The comparison is geared towards understanding the enhancements and differences brought about by the Barlow Twins modification, particularly in terms of the quality of the latent representations and the reconstruction accuracy.

## 5.1 UCR Archive

Our experiments are conducted using the UCR Archive [Dau+18], a widely-used open-source repository of time series datasets. The UCR Archive contains a diverse range of time series data, including various domains and characteristics, making it a good benchmark for evaluating time series models. The datasets in UCR Archive are divided into a training dataset aswell as a test dataset. In the experiment we train the models on the provided training dataset and validate

on the test set. The chosen subset for the experiment emphasize datasets that vary not only in size but also in the complexity and nature of their class distributions. The chosen subset also contains datasets with the highest number of samples, as done by Lee et al in their investigation of various SSL algorithms on the UCR Archive[LA21].

| Dataset Name | #Samples | #Classes | Length |
|---|---|---|---|
| Electric Devices | 16637 | 7 | 46 |
| StarLightCurves | 9236 | 3 | 1024 |
| Wafer | 7164 | 2 | 152 |
| ECG5000 | 5000 | 5 | 140 |
| TwoPatterns | 5000 | 4 | 128 |
| FordA | 4921 | 2 | 500 |
| UWaveGestureLibraryAll | 4478 | 8 | 945 |
| FordB | 4446 | 2 | 500 |
| ChlorineConcentration | 4307 | 3 | 166 |

**Table 5.1:** Chosen UCR Archive subset. Data collected by Lee et al[LA21]

## 5.2 Reconstruction evaluation

We evaluate the reconstruction accuracy by calculating $\mathcal{L}_{\text{recon}}$ in eq 4.12 using the validation set. Giving $\mathcal{L}_{\text{val-recon}}$. This gives us a score on how well the models generalize on the validation set.

## 5.3 Downstream evaluation

In the training phase of our models, we incorporate a set of downstream tasks. These tasks are distinct from the primary training process in that they do not influence the gradient updates during backpropagation. Instead, their primary role is to offer insights into various aspects of the model's performance and capabilities.

In our case, we are interested in how the fundamental structure of the discrete latent space change during training for the two models. Aswell as how the different types of timeseries are ordered spacially as discrete latent representations.

### 5.3.1 Extracting Discrete latent variables

The UCR archive include labels $Y$ alongside the set of timeseries $X$. Splitted in training and validation datasets respectively. The process of extracting the latent representations is by prop-

agating $X_{\text{train}}$ and $X_{\text{test}}$ through the encoder and quantization model seperatively. This gives us the models discrete latent representations, $Z_{\text{train}}$ and $Z_{\text{test}}$, as illustrated in fig 5.1. The concatenation of $Z_{\text{train}}$ and $Z_{\text{test}}$ is denoted as $Z$.

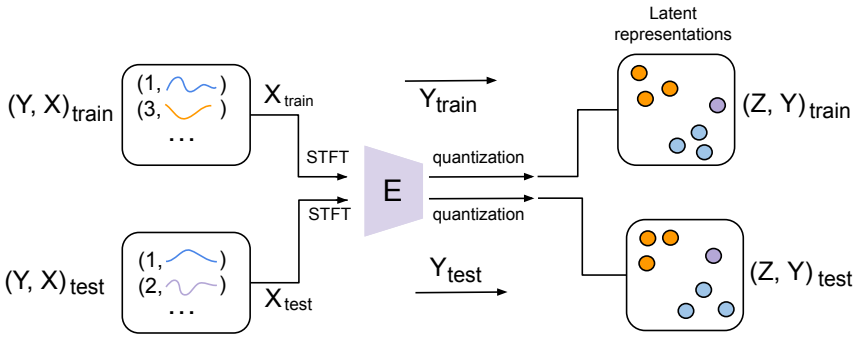


**Figure 5.1:** Illustration showing the processing of training and validation datasets to latent representations.

This gives us the encoder-codebook latent representations, which is usefull to examine how the

### 5.3.2 Downstream tests

The **intrinstic dimension** measure identifies the most straight forward yet comprehensive set of features required to grasp our data's structure. We approximate the intrinstic dimension by applying a Principal Component Analysis (PCA) on $Z$ and pinpoint the number of principal components that account for 95% of the variance. This provides a measure of the complexity of the discrete latent variables.

By fitting supervised models on $(Z, Y)_{\text{train}}$ and predicting on $(Z, Y)_{\text{test}}$ we can investigate the richness of information in the discrete latent variables related to the labels. We use two supervised approaches: **SVM** and **KNN**. The SVM is implemented with a linear kernel, hence measuring the linear seperatability of the latent space. A high classification accuracy would suggest that the labeled timeseries are linearly separable as discrete latent representations. For the KNN algorithm we experiment with three different values for K, namely $1, 5, 10$. The intuition being that a high accuracy with a $K = 1$ would suggest more complexity, compared to a high accuracy with $K = 10$.

Additionally to the richness of information related to the labels, we are also investigating if the discrete latent variables ($Z_{\text{train}}$, $Z_{\text{test}}$) are seperatable into clusters. We test this by designing a experiment using the unsupervised **KMeans** algorithm and **Silhouette score**. By fitting 15 Kmeans clusters on the latent representations and average the silhouette scores, we get a downstream metric of how well the discrete latent space is clusterable for each model.

Chapter 6

# Results and discussion

Chapter 7

# Conclusion

# Bibliography


[AZ19] Abubakar Abid and James Zou. 'Contrastive Variational Autoencoder Enhances Salient Features'. In: (Feb. 2019). arXiv: 1902.04601 [cs.LG].

[BPL21] Adrien Bardes, Jean Ponce and Yann LeCun. *VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning*. 2021. eprint: arXiv:2105.04906.

[Che+20] Ting Chen, Simon Kornblith, Mohammad Norouzi and Geoffrey Hinton. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. eprint: arXiv:2002.05709.

[CV95] C. Cortes and V. Vapnik. 'Support Vector Networks'. In: *Machine Learning* 20 (1995), pp. 273–297.

[Dau+18] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista and Hexagon-ML. *The UCR Time Series Classification Archive*. https://www.cs.ucr.edu/eamonn/timeseriesdata2018/. Oct. 2018.

[Gri+20] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos and Michal Valko. *Bootstrap your own latent: A new approach to self-supervised Learning*. 2020. eprint: arXiv:2006.07733.

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. eprint: arXiv:1512.03385.

[He+19] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie and Ross Girshick. *Momentum Contrast for Unsupervised Visual Representation Learning*. 2019. eprint: arXiv:1911.05722.

[IS15] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. eprint: arXiv:1502.03167.

[KW13] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. eprint: arXiv:1312.6114.

[KW19] Diederik P Kingma and Max Welling. 'An Introduction to Variational Autoencoders'. In: (June 2019). arXiv: 1906.02691 [cs.LG].

[LA21] Daesoo Lee and Erlend Aune. *Computer Vision Self-supervised Learning Methods on Time Series*. 2021. eprint: arXiv:2109.00783.

[LAM23] Daesoo Lee, Erlend Aune and Sara Malacarne. 'Masked Generative Modeling with Enhanced Sampling Scheme'. In: *arXiv preprint arXiv:2309.07945* (2023).

[LST15] Brenden M Lake, Ruslan Salakhutdinov and Joshua B Tenenbaum. 'Siamese Neural Networks for One-shot Image Recognition'. In: (2015).

[ON15] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. eprint: arXiv:1511.08458.

[OVK17] Aaron van den Oord, Oriol Vinyals and Koray Kavukcuoglu. 'Neural discrete representation learning'. In: (Nov. 2017). arXiv: 1711.00937 [cs.LG].

[Rie22] Janosh Riebesell. *Random TikZ Collection*. Software. Version 0.1.0. Available online at https://github.com/janosh/tikz. GitHub, Dec. 2022. DOI: 10.5281/zenodo.7486911. URL: https://github.com/janosh/tikz.

[ROV19] Ali Razavi, Aaron van den Oord and Oriol Vinyals. 'Generating diverse high-fidelity images with VQ-VAE-2'. In: (June 2019). arXiv: 1906.00446 [cs.LG].

[VL21] Leni Ven and Johannes Lederer. *Regularization and Reparameterization Avoid Vanishing Gradients in Sigmoid-Type Networks*. 2021. eprint: arXiv:2106.02260.

[WBC23] Yixin Wang, David M Blei and John P Cunningham. 'Posterior Collapse and Latent Variable Non-identifiability'. In: (Jan. 2023). arXiv: 2301.00537 [stat.ML].

[Wen+20] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang and Huan Xu. 'Time series data augmentation for deep learning: A survey'. In: (Feb. 2020). arXiv: 2002.12478 [cs.LG].

[Zbo+21] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun and Stéphane Deny. *Barlow Twins: Self-Supervised Learning via Redundancy Reduction*. 2021. eprint: arXiv:2103.03230.

# Appendix

Derivation of the Evidence Lower Bound (ELBO)

$$
\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x)\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log\left[\frac{p_\theta(x,z)}{p_\theta(z|x)}\right]\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log\left[\frac{p_\theta(x,z)}{q_\phi(z|x)}\frac{q_\phi(z|x)}{p_\theta(z|x)}\right]\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log\left[\frac{p_\theta(x,z)}{q_\phi(z|x)}\right]\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log\left[\frac{q_\phi(z|x)}{p_\theta(z|x)}\right]\right] \\
&= \mathcal{L}_{\theta,\phi}(x) + \mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x))
\end{aligned}
\tag{7.1}
$$

Here $\mathcal{L}_{\theta,\phi}(x)$ is the ELBO and $\mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x))$ is the Kullback-Leibler (KL) divergence which is non-negative.

$$
\implies \mathcal{L}_{\theta,\phi}(x) = \log p_\theta(x) - \mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x)) \leq \log p_\theta(x) \tag{7.2}
$$