You’re your own best teacher: A Self-Supervised Learning Approach For Expressive Representations

Johan Vik Mathisen

May 31, 2024

Abstract

The `ntnuthesis` document class is a customised version of the standard LaTeX `report` document class. It can be used for theses at all levels – bachelor, master and PhD – and is available in English (British and American) and Norwegian (Bokmål and Nynorsk). This document is ment to serve (i) as a description of the document class, (ii) as an example of how to use it, and (iii) as a thesis template.

Sammendrag

Dokumentklassen `ntnuthesis` er en tilpasset versjon av LaTeX' standard `report`-klasse. Den er tilrettelagt for avhandlinger på alle nivåer – bachelor, master og PhD – og er tilgjengelig på både norsk (bokmål og nynorsk) og engelsk (britisk og amerikansk). Dette dokumentet er ment å tjene (i) som en beskrivelse av dokumentklassen, (ii) som et eksempel på bruken av den, og (iii) som en mal for avhandlingen.

Contents

Chapter 1

Introduction

In this thesis we investigate possible improvements on the TimeVQVAE model presented by... We investigate how a "self supervised learning extension" of the tokenization affects the learned representations, and the effect on the prior learning. In particular we investigate if the learned representations are more informative, in the sense that they simultaneously enables high quality reconstruction, and improved the downstream classification accuracy. For the generative model we investigate if the learned representations enables faster convergence during training, and how the quality of the synthetic samples are affected.

1.1 Acknowledgements

Supervisors, Erlend, IDUN, UCR Archive creators

1.2 Motivation

- The role and importance of time series. - The need for models that capture complex structures for which traditional statistical models fail. Real world time series data is often incomplete (missing datapoints), irregular (datapoints not evenly spaced in time) and noisy. ML4ITS. - - Why do people care about time series generation (TSG)? - Applications - Why is (unsupervised) representation learning for time series interesting? Distributions of time series in their original temporal representation are complex and difficult to model. One would like to translate time series to a space where modelling is easier. This is one of the reasons to investigate representation learning for time series. Time series are recorded at record speed from sensors of various kinds (IoT, wearable devices). Unfortunately many of these do not have easily recognizable patterns for human observers, which makes labeling of such data quite difficult. In order to take advantage of this vast amount of unlabeled data we need techniques that can extract useful patterns without supervision. This is one of the reasons for investigating possible unsuper-

vised models. A subcategory of unsupervised learning called self-supervised learning has in recent times shown great potential for learning informative and useful representations without the need of labeled data in the fields of computer vision and natural language processing. Most notably the GPT models from OpenAI which utilizes masked language modelling for pre-training.

1.3 Overview/structure

- Main inspirations [1]
- Collaboration with Erlend
- Structure of the thesis

1.4 Research questions

Stage1

RQ1: Will self a supervised learning approach enhance downstream classification while simultaneously reconstruct well?

RQ2: How does augmentations influence reconstruction and downstream classification?

Stage 2

RQ3: Will more expressive representations improve prior model learning?

RQ4: How does augmentations influence prior learning and synthetic sample quality?

- - - How does SSL VQVAE compare when we train a powerful prior model on top of it (MaskGIT)?

1.5 Sustainability impact

Ethical and environmental impact consideration with basis in UN sustainability goals.

The field of AI and machine learning is in rapid development and the fear of being left in the dust in the gold rush makes many actors scrape all data they can find to train ever larger models.

One downside, not considering the massive copyright disputes, questionable privacy and the spread of misinformation by hallucinations and use of deepfakes, is their colossal environmental impact.

AI and machine learning are not universal tools though the modern LLMs make it seem so.

1.6 AI Declaration

Chapter 2

Theoretical Background

TODO: Introduce the section, what we think and the philosophy of presenting material in such a way.

2.1 Notation

- Encoder E
- Decoder D
- Estimated values are presented with a hat, $\widehat{x}$ for a reconstructed value, $\widehat{f}$ for a trained model etc.
- Parameters θ
- Dataset $X = \{x_i\}_{i=1}^{N}$

2.2 Information theoretic/basic stats used in evaluation

- maximum entropy distribution
- mutual information

Definition 1 (Differential entropy) *The differential entropy of a random variable X with pdf or pmf p defined on a sample space $\mathcal{X}$ is*

$$h(X) = \mathbb{E}[-\log(f(X))] - \sum_{x \in \mathcal{X}} p(x)\log(p(x)), \text{ if } X \text{ is discrete,} \tag{2.1}$$

$$h(X) = \mathbb{E}[-\log(f(X))] - \int_{\mathcal{X}} p(x)\log(p(x)), \text{ if } X \text{ is continuous} \tag{2.2}$$

- perplexity

Definition 2 (KL-Divergence) *For probability distribution P and Q defined on the same sample space $\mathcal{X}$, if for all $x \in \mathcal{X}$ $Q(x) = 0$ implies $P(x) = 0$, the Kullback-Leibler divergence is defined as*

$$\mathrm{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right), \text{ if } P \text{ and } Q \text{ are discrete,} \tag{2.3}$$

$$\mathrm{KL}(P||Q) = \int_x p(x) \log\left(\frac{p(x)}{q(x)}\right) dx, \text{ if } P \text{ and } Q \text{ are continuous.} \tag{2.4}$$

`https://stats.stackexchange.com/questions/188903/intuition-on-the-kullback-leibler-kl-189758#189758`

- Cross entropy
- Graphical probabilistic models - Ancestral sampling
- Generative models

2.3 Time Series Inference

- short-time-fourier transform etc.

2.4 Neural Network

An *artificial neural network* or simply *neural network* is a fundamental model in machine learning, and more specifically in *deep learning*. Neural networks are loosely inspired by the way neurons are assembled in the brain. The concept dates back to 1943 when Warren McCulloch and Walter Pitts developed the first artificial neuron, which is considered the first neural model ever invented[2]. The first practical implementation was by Frank Rosenblatt in 1957 [3]. However, it wasn't until the development of the backpropagation algorithm in its modern form in the 1980s that neural networks gained significant traction. Since then, neural networks have been highly influential in the machine learning field, with a broad range of impressive applications, including face recognition, defeating humans in chess, Go, and Starcraft, self-driving cars, and predicting protein structures. The remarkable effectiveness of neural networks across diverse tasks can partly be explained by the *universal approximation theorem*, proven by Kurt Hornlik in 1989 [4], which roughly states that a neural network can approximate any (Borel measurable) function to any desired degree of accuracy.

A neural network takes in a vector $x \in \mathbb{R}^n$ and builds a nonlinear function $f(x)$ to predict the response $y \in \mathbb{R}^m$.

More specifically, a neural network maps an input vector x to an output vector y through a series of nonlinear functions applied to linear combinations of the input. This particular structure, illustrated in Figure 2.1, distinguishes neural networks from other nonlinear prediction models. The variables $x = [x_1, ..., x_n]$ constitute the units of the *input layer*.

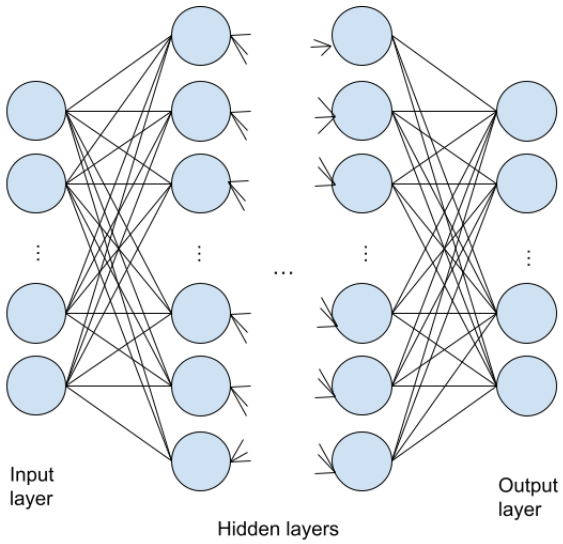


Figure 2.1: Illustration of a Neural Network model.

The intermediate layers are called the *hidden layers*, and the final mapping to y is called the *output layer*. A neural network is parameterized by a set of *weight* matrices W_i and *bias* vectors b_i, together with a specified non-linear *activation function* σ. We denote the collection of parameters $\{W_1, ..., W_{K-1}, b_1, ..., b_{K-1}\}$ by θ. Written out a K layered neural network is given by

$$f_\theta(x) = f_K \circ f_{K-1} \circ \ldots \circ f_2 \circ f_1(x),$$

where

$$f_i(x) = \sigma_i(W_i x + b_i), \quad i \in \{1, ..., K-1\},$$

and f_K is the output layer, with application dependent structure.
The introduction of nonlinearity by the activation function is what enables the model to approximate nonlinear signals, setting it apart from linear regression models. Two of the most commonly used activation functions are $\text{Sigmoid}(x) = \frac{1}{1+\exp(-x)}$ and $\text{ReLU}(x) = \max(0, x)$, but countless other options exists.

The architecture of neural networks, and most specializations thereof, is sequential in nature. They can effectively be described as compositions of some combination of a flavour of matrix multiplication, non-linear transformation and down or upsampling.

2.4.1 Training Neural Networks

Training a neural network involves finding the optimal values for the weight and bias parameters. The general idea behind neural network training is to optimize the parameters based on a distance metric between the predicted values and the

target values. This distance metric is referred to as the *loss function*, with mean squared error (MSE) being a common choice.

For a multivariable function F that is differentiable around a point a, the direction in which it decreases fastest from a, is given by the negative gradient at a, $-\nabla F(a)$. The gradient descent algorithm utilizes this property to find local minima of F by initializing the function with a value x_0 and iteratively updating

$$x_{n+1} = x_n - \gamma \nabla F(x_n).$$

If the *learning rate* γ is small enough, we are guaranteed that $F(x_{n+1}) \geq F(x_n)$, and the sequence $x_0, x_1, \ldots$ converges to a local minimum of F.

A neural network $f_\theta(x)$ is itself a multivariable function, and as long as the loss and activation functions are differentiable, the network is as well, both in terms of it arguments and parameters. By differentiating the network with respect to its parameters across its domain, the negative gradient indicates the fastest direction to update the parameters to minimize the loss. The optimization problem at hand, for a dataset $X = \{x_i\}_{i=1}^N$ with corresponding labels $Y = \{y_i\}_{i=1}^N$, is

$$\widehat{\theta} = \min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i),$$

and one would iteratively update the parameters by gradient descent

$$\theta_{n+1} = \theta_n - \frac{\gamma}{N} \sum_{i=1}^{N} \nabla_\theta \mathcal{L}(f_\theta(x_i), y_i).$$

If the dataset X is large, gradient calculations can be expensive. In such cases, which are common in modern machine learning, an effective estimator for the true gradient is used instead. Stochastic gradient descent (SGD) is a very popular approach. Instead of computing the gradient at each data point in every iteration, SGD updates the parameters by iterating through the dataset and using the gradient at each single data point. The dataset is then permuted and iterated through again until an approximate minimum is reached. Pseudocode for SGD is presented in Algorithm 1.

Algorithm 1 Stochastic Gradient Descent (SDG)

```
Initialize parameters θ and learning rate γ
while Not converged do
    Permute training set (X, Y)
    for i in 1, ..., N do
        θ ← θ − γ∇_θ L(f_θ(x_i), y_i)
    end for
end while
```

TODO: Mini batch SDG

For the actual gradient computations, the backpropagation algorithm is used. It provides an efficient way of computing gradients in neural networks by leveraging their compositional structure. Essentially, backpropagation is an efficient application of the Leibniz chain rule for differentiation.

For a thorough introduction to the subject of neural networks and their training, refer to Chapters 6 and 8 of [5].

2.5 Convolutional Neural Network

A convolutional neural network (CNN) is a specialized type of neural network designed to learn local features in data through the mathematical operation of convolution. Essentially, a CNN replaces matrix multiplication with convolution in at least one of its layers [5].

Fully connected neural networks face significant challenges when dealing with high-dimensional input data, such as images, due to their rapidly increasing computational complexity. CNNs address this issue by employing downsampling techniques and leveraging the convolution operation to reduce the dimensionality and complexity of the data. This makes CNNs particularly well-suited for tasks involving high-dimensional data.

The concept of CNNs is inspired by the structure and function of the visual cortex in mammals. In the 1960s, Hubel and Wiesel discovered the visual processing capabilities of the cortex [6], leading to the development of the neocognitron in 1980 [7], which is considered the precursor to modern CNNs. In 1989, Yann LeCun and his team introduced a contemporary framework for CNNs and showcased its effectiveness in handwritten digit recognition [8]. Since then, CNNs have become a cornerstone of machine learning research, particularly in the field of computer vision.

For an in-depth exploration of the advancements and applications of convolutional neural networks, refer to [9].

2.5.1 The convolution operation

The convolution operation is a fundamental mathematical concept with wide-ranging applications, particularly in the context of neural networks. It generalizes the notion of a moving weighted average and is deeply connected to the Fourier transform.

For real-valued functions f and g, their convolution is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.5}$$

While the detailed mathematical criteria for the existence of this integral are beyond the scope of this thesis, it suffices to note that if f and g are integrable (in the Riemann or Lebesgue sense) then the convolution is well-defined. Generally, the convolution of f and g is as smooth as the smoothest of f and g. Additionally, convolution is commutative, meaning $f * g = g * f$, which can be shown by a simple change of variables.

In the context of convolutional neural networks, the function g is referred to as the *kernel*, and it consists of learnable parameters. The function f is the input, and the result of the convolution is called the *feature map* [5]. Since machine learning typically involves discrete signals represented as multidimensional arrays, we use a discrete version of the convolution operation. For a discrete input I and kernel K, their convolution is defined as

$$(I * K)[n] = \sum_{m=-\infty}^{\infty} I[m]K[n-m]. \tag{2.6}$$

In practice I and K usually have finite support, meaning they are zero beyond a certain range, which avoids any convergence issues.

Convolutions can be naturally extended to higher-dimensional functions. For a two-dimensional image I and a kernel K, their convolution is calculated as

$$(I * K)[i,j] = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} I[n,m]K[i-n, j-m]. \tag{2.7}$$

This operation is illustrated in Figure 2.2.

In practice, the convolution operation used in machine learning often corresponds to cross-correlation rather than strict mathematical convolution, differing by a sign flip in the kernel arguments. While this makes the operation non-commutative, it does not affect the learning process, as the learned kernel parameters will be equivalent [5].

In digital signal processing, discrete convolutions are widely used with predefined kernels to manipulate signals predictably. Common examples include Gaussian kernels for blurring and edge detection kernels for highlighting areas of high intensity change. Figure 2.3 demonstrates the effects of these kernels. Edge detection provides insight into how CNNs learn features from data. In convolutional layers, kernels are learned to produce feature maps that aid in achieving the training objectives.

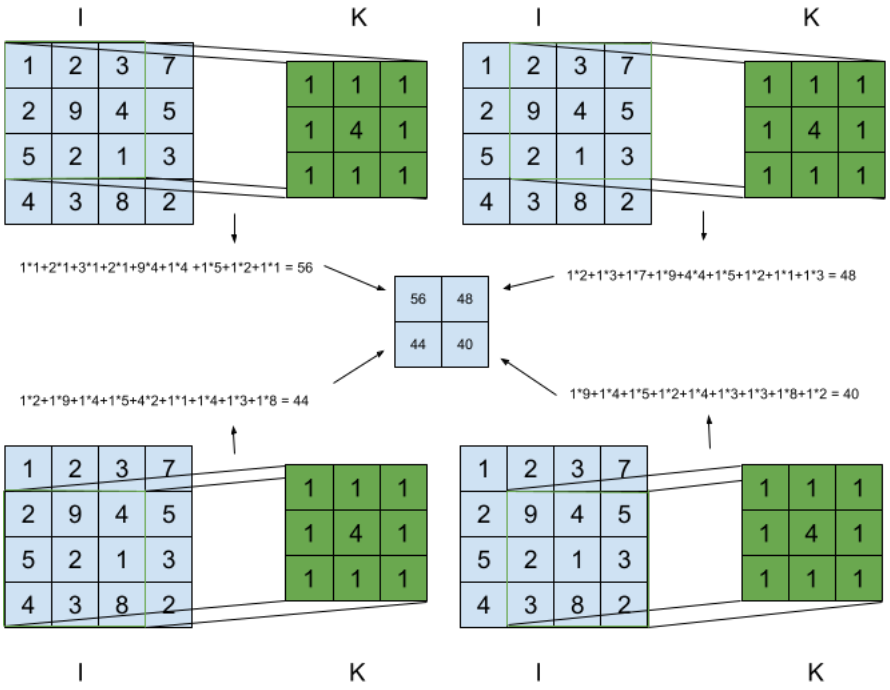


Figure 2.2: Illustration of discrete two dimensional convolution

2.5.2 Pooling

As mentioned earlier, CNNs are primarily used for high-dimensional data. To reduce the dimensionality to a manageable level, pooling operations are employed. Pooling is a downsampling technique applied to feature maps, where regions of the output are replaced with summary statistics. The aim with pooling is to retain the most important information while discarding less relevant details.

Two of the most common pooling methods are max pooling and average pooling. Max pooling replaces the region with its maximum value, while average pooling replaces it with the average value.

There are two key hyperparameters for any pooling operation: the filter size, which determines the region of values to calculate the summary statistic, and the stride length, which determines how the filter moves across the feature map. In addition to reducing dimensionality, pooling helps make representations approximately invariant to small distortions in the input.Illustrations of max pooling with different stride is presented in figure 2.4 and 2.4, while the effect of max and average pooling on image data is illustrated in 2.6.

Global pooling is a technique that involves calculating a summary statistic, such as the maximum or average, for an entire feature map. This effectively summarizes the presence of a feature across the entire input. This method is particu-

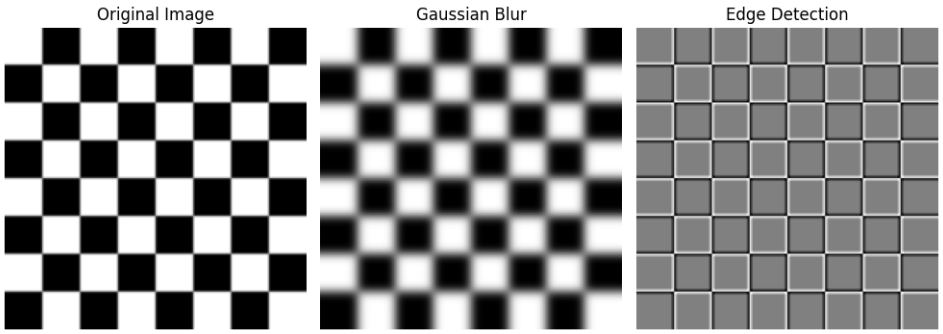


Figure 2.3: Illustration of discrete convolution applied to images. Image part of Scikit-image data package.

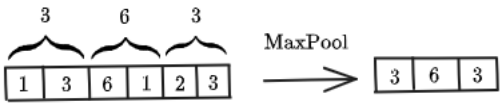


Figure 2.4: Max pooling of one dimensional array. Filter size: 2, stride: 2.

larly useful in the final layers of a convolutional neural network (CNN), where it reduces the spatial dimensions of the feature maps to a single value per map. This process condenses information and enhances computational efficiency. Additionally, global pooling helps CNNs handle variably sized inputs more effectively and serves as a method for downsampling representations for visualization purposes.

2.5.3 Architecture

There are many specific architectures within the category of convolutional neural networks (CNNs), but their basic components are largely the same. CNNs typically consist of convolutional layers, pooling layers, and fully connected layers. Figure 2.8 illustrates the original LeNet-5 [8] as an example of a general CNN.

A convolutional layer consists of several kernels used to compute different feature maps. Each kernel is convolved with the entire input, producing different feature maps by using different kernels. After convolution, a nonlinear activation function is applied pointwise to the feature maps, introducing nonlinearity to the model. The purpose of the convolutional layer is to extract local features such as edges, textures, and shapes from the input data.

A pooling layer is typically placed between convolutional layers and act as strong downsamplers that aim to enforce approximate translation invariance. A pooling operation, such as max pooling or average pooling, is applied across each feature map, reducing its spatial dimensions while retaining important information. The pooling layer helps to reduce the computational load and control overfitting by progressively reducing the spatial size of the representation.

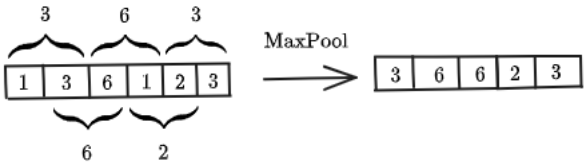


Figure 2.5: Max pooling of one dimensional array. Filter size: 2, stride: 1.

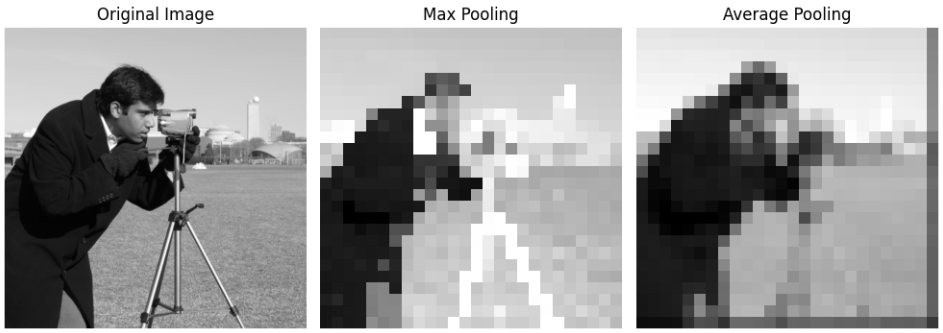


Figure 2.6: Illustration of mean and average pooling applied to images. Filter size and stride is 20×20. Original image size is 512×512, pooled images are 26×26. Image part of Scikit-image data package.

Fully connected layers are standard hidden layers in a neural network, connecting every input to every node in the output. Due to computational constraints, fully connected layers are introduced only after the input data has been sufficiently downsampled through convolutional and pooling layers. The fully connected layers aggregate the local features learned by convolutional layers across the entire image, learning high level reasoning and enabling the network to make final classifications or predictions.

In addition to these basic layers, modern CNN architectures often incorporate other techniques and layers to improve performance. Batch Normalization layers, for example, normalizes the inputs, which assist in stabilizing and speeding up the training process. They allow for higher learning rates and reduces the dependency on careful initialization.

Additionally, residual connections are frequently used in modern architectures. Introduced by He et al. in [10], residual connections allow gradients to flow directly through the network, alleviating the vanishing gradient problem in deep networks [11]. They enable the training of much deeper networks by providing shortcut connections that bypass one or more layers. This means that instead of learning the underlying mapping directly, the network learns the residual function, which is often easier to optimize. Residual connections have been a key innovation, enabling the development of very deep networks like ResNet, which

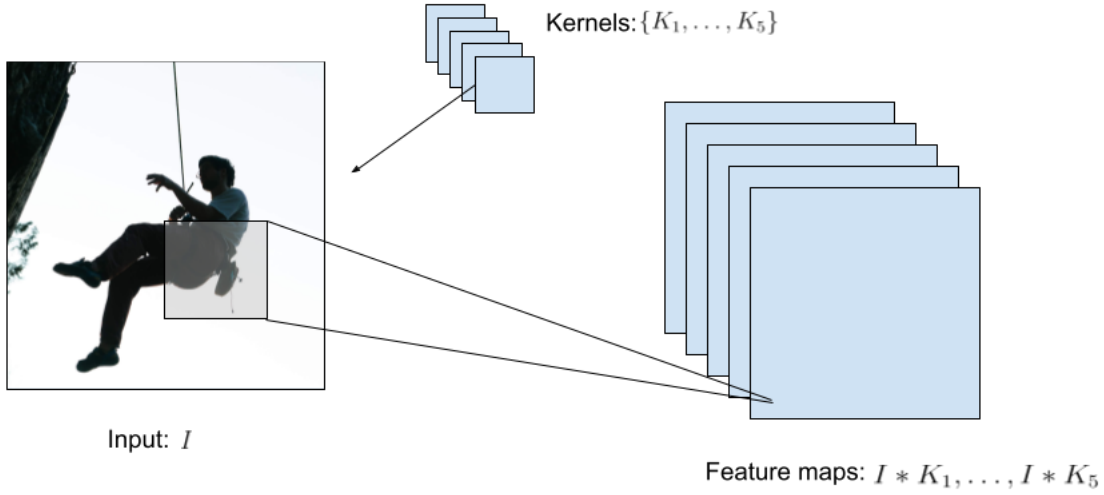


Figure 2.7: Convolutional layer.

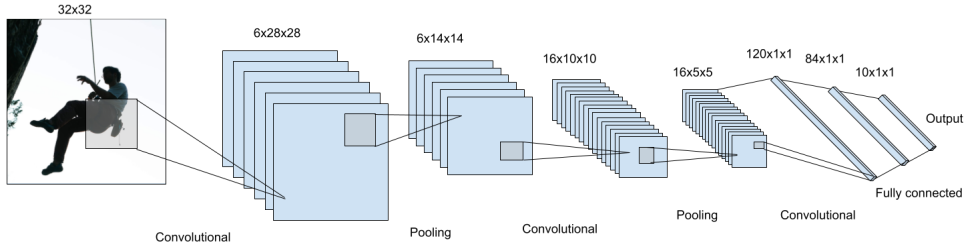


Figure 2.8: LeNet-5 network [8]

have achieved state-of-the-art results in various computer vision tasks.

2.5.4 Transposed Convolutional Networks

Transposed convolution, also known as deconvolution or fractionally-strided convolution, is a technique used to reverse the downsampling effect of standard convolutions. In essence, it is an inpainting or upsampling technique familiar from digital signal processing. The flexibility of learning data-dependent transposed convolutional kernels enables the effective reversal of the downsampling performed by convolutional layers.

Transposed convolutions are extensively used in combination with convolutional downsampling in encoder-decoder architectures presented in section 2.9. These architectures are fundamental in many generative models. The encoder part of the network typically reduces the spatial dimensions through a series of convolutional and pooling layers, capturing the essential features of the input data. The decoder part then uses transposed convolutions to upsample the encoded feature

maps back to the original spatial dimensions, reconstructing the input or generating new data.

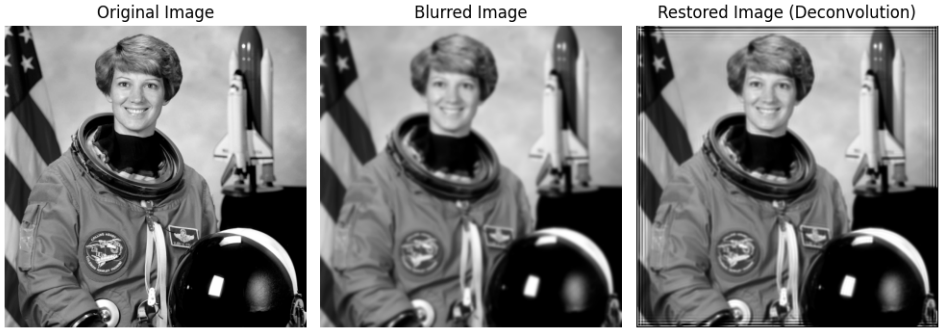


Figure 2.9: Illustration of deconvolution applied to images. Here using Gaussian blur and restoring using the Richardson Lucy algorithm with a Gaussian deconvolution kernel. Image part of Scikit-image data package.

Figure 2.9 illustrates the application of deconvolution to images. In this example, a Gaussian blur is applied, and the image is restored using the Richardson-Lucy algorithm with a Gaussian deconvolution kernel. This process demonstrates how transposed convolutions can be used to approximately reverse the effects of convolutional operations.

2.6 Representation Learning

The focus of this thesis is representation learning for improved time series generation.

2.6.1 What a representation?

Representation learning is a term that can be difficult to define due to its abstract nature. To understand it better, let's first consider what is meant by the representation of information.

Take the base ten integer $(4)_{10} = 4$ as an example. This number can be equivalently expressed in other bases, such as binary $(4)_2 = 100$. The base we choose depends on our intention. For digital electronics, a binary representation is useful because transistors have two states. For human arithmetic, base ten is natural because we have ten fingers. Thus, a particular representation can make a task easier or harder, although the information content remains unchanged. What changes is the ease or difficulty of certain information processing tasks. Representation learning is the process of learning a representation of information that facilitates these tasks.

Representations are highly dependent on who or what will process the information. Consider the example of time. Humans have developed a standardized system for writing timestamps that works well for us. However, if we want to model time-dependent phenomena using tabular data, the DateTime representation might be unhelpful to a tree-based model. The numerical representation of timestamps close in time is not necessarily close in numerical value, such as 23:59 and 00:00. A possible solution is to change the representation to reflect the periodic nature of time by mapping it to a circle. This new representation might be less intuitive for humans but more useful to a computer.

In machine learning, representation learning involves designing algorithms that learn representations useful for a specific objective. The goal is to find representations that make it easier for the machine learning models to process and understand the data, improving performance on the given tasks.

2.6.2 Why do we care about representation learning?

Those who has worked with data science or machine learning and has come across feature engineering are familiar the effect good feature engineering has on a models performance. The same people too knows the level of domain expertise, creativity and time is needed to feature engineer well. In reality much of the actual time spent in the process of deploying machine learning methods revolves around constructing good data pipelines and applying transformations that produce representations beneficial for the algorithm at hand [12]. Thus the ability to automate such tasks would be incredibly beneficial, and ease the use of ML algorithms significantly. It is here one of the intriguing and promising features of neural networks, with its many specializations and architectures, comes into play. They have shown the ability to learn useful abstract representations of the data and provide automatic feature engineering [12].

2.6.3 What is a good representation?

The quality of a representation is determined by how effectively it facilitates subsequent tasks. The idea of universally good representations is ill-defined because any representation derived from a non-invertible function can be specifically countered by designing a *downstream tasks* that relies on the lost information, leading to poor performance. In essence, there is no free lunch in representation learning either.

To evaluate the quality of a representation, one must specify a set of predefined downstream tasks and assess performance based on those tasks. A representation is considered better if it enables the model to perform well on the downstream tasks. Additionally the more general a representation is, meaning they facilitate high performance on a larger set of tasks, the higher its quality.

2.6.4 How does one evaluate representations?

There are several evaluation protocols in representation learning. These typically involve training a model on a *pretext task*, which, as defined in [13], is a task designed for the network to solve with the primary goal of learning useful representations. The learned representations are then evaluated on a downstream task, generally solved in a supervised manner using human-annotated data.

In an N-layered network $f = f_N \circ ... \circ f_1$, the intermediate value of the data x in some layer n represents the network's learned feature representations. When focusing on the learned representations, it is helpful to decompose the model f, notation wise, into a *feature extractor* h and an *output function* g such that it can be factored as $f = g \circ h$. Representation learning algorithms typically follow this pattern

- Train $f = g \circ h$ on a pretext task.
- Discard g
- Use the learned feature extractor $\widehat{h}$ as part of a new model.
- Evaluate the new model on the downstream task.

The standard evaluation protocol involves training a linear head g_D on top of the *frozen* representations in a supervised manner and evaluate this model's performance. This means training $f_D = g_D \circ \widehat{h}$ by only updating the parameters of the linear model g_D, and evaluate $\widehat{f_D}$ on a test set. This protocol is sometimes referred to as *linear probing*. A common downstream task is classification, where the idea is that good and informative representations should differentiate data in such a way that it is easy to separate them.

An alternative protocol, similar to the one above, allows all parameters of f_D to be learnable on the downstream task. This protocol is referred to as a pretraining-finetuning. In this approach, the model is pretrained on a large, potentially unlabeled dataset, and then fine-tuned on a smaller, labeled dataset specific to the downstream task.

Evaluating f_D involves assessing accuracy on the downstream task, training time, and the sensitivity of these metrics to the training data size. The baseline comparison is typically an identical but randomly initialized model. It is highly advantageous if one can pretrain a feature extractor using abundant, cheap data, ensuring faster convergence on a downstream task where data is expensive or scarce.

It is worth mentioning that in cases where the sole goal is to create the best-performing model, a more complex task-specific head g_D is often used. For a comprehensive survey on representation learning, see[14].

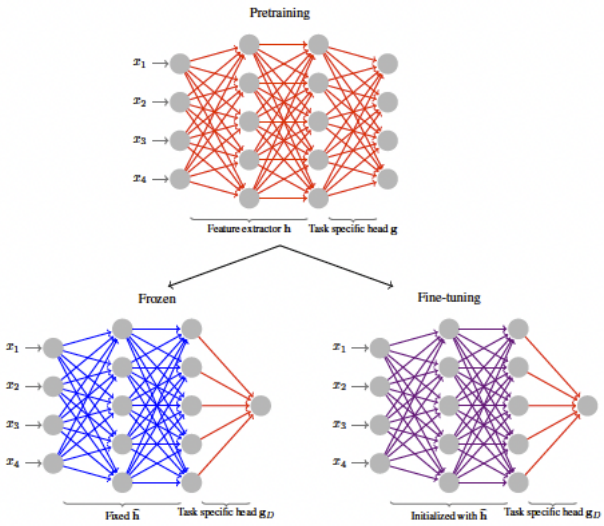


Figure 2.10: Taken from [14]

2.7 Transformers

Since their introduction in the seminal paper "Attention is all you need"[15], transformers have revolutionized the fields of machine learning and artificial intelligence. This architecture is the backbone of large language models (LLMs) such as Google's BERT, Meta's Llama and OpenAI's ChatGPT. Following their initial success in natural language processing, transformers have been adapted for other modalities, including computer vision with Vision Transformers [16], and similar trends are shown for other modalities such as audio [17] and time series [18].

Transformers were developed for sequential modeling and have the capability of learning long-range dependencies in data through the *multi-headed attention mechanism*. Previously, recurrent neural networks (RNNs) and long short-term memory (LSTM) networks were the state-of-the-art for sequential tasks. However, they struggled with modeling long-term dependencies due to the vanishing or exploding gradient problem [11]. One of the main novelties of the transformer architecture is not relying on recurrence, and instead solely using the attention mechanism to capture dependencies between input and output.

The transformer takes as input a sequence of symbols. For different modalities like text, speech, images, and time series, this requires a process called *tokenization*. For example, in natural language processing (NLP), a simple word-level tokenization involves creating a dictionary of all words in the dataset and assigning each word an integer. Text can then be mapped to a sequence of integers. Other modalities, such as speech, images, and time series, are usually represented

as matrices which can be flattend and used as inpus. However, directly modeling high-dimensional data as such sequences would require significant computational resources and make it challenging to model long-range dependencies. Therefore, various tokenization methods are used to represent the data as coarser sequences.

Transformers have also been successfully adopted for generative tasks. Autoregressive transformers, such as the GPT series, are trained to predict the next token in a sequence given the preceding tokens. By treating images as sequences of patches, Vision Transformers can generate new images or inpaint missing patches based on the input. With the introduction of Vision Transformers [16], input images are split into patches and linearly projected before being processed by the transformer. Since then, *Vector-Quantized*-based tokenization introduced by [19] has emerged as a popular approach.

2.7.1 The Attention Mechanism

The attention mechanism is a key innovation in transformer architectures, enabling them to handle long-range dependencies effectively. Attention can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and outputs are all vectors. This mechanism allows the model to focus on different parts of the input sequence simultaneously.

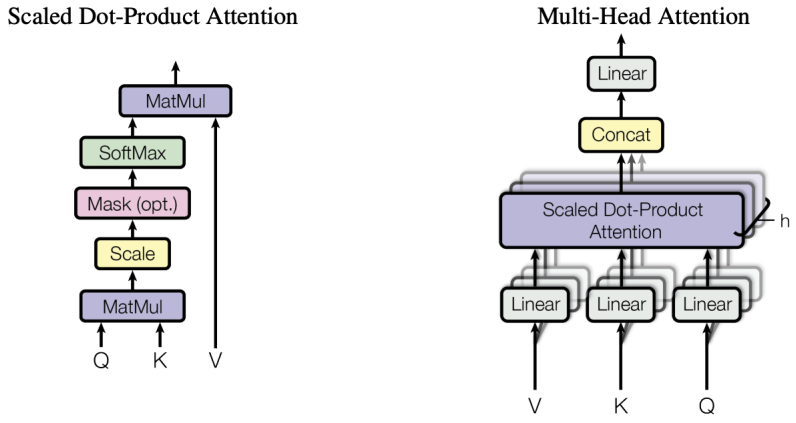


Figure 2.11: (left) Scaled dot-product attention. (right) Multi-headed attention. Taken with permission from [15]

Scaled dot-product attention is computed by

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.8}$$

This equation allows the model to weigh the importance of different input

tokens based on their relevance to the query, enabling it to focus on the most relevant parts of the input.

To enhance the model's ability to capture different aspects of the input, multi-headed attention is used. This involves projecting the queries Q, keys k, and values V into multiple subspaces and applying scaled dot-product attention in each subspace. The results from each subspace are then concatenated and linearly transformed. The process is defined as

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O, \tag{2.9}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \tag{2.10}$$

Multi-headed attention enables the transformer to attend to different parts of the input sequence and capture various features simultaneously. By having multiple attention heads, the model can learn different relationships and dependencies within the data, leading to a richer representation.

2.7.2 Architecture

The original transformer architecture, as presented in Figure 2.12, consists of several distinct components that work together to process sequential data effectively. This architecture includes an encoder and a decoder, each composed of multiple layers that leverage the attention mechanism to handle dependencies within the input data.

Embeddings: The first step in the transformer architecture involves mapping each token in the input sequence to a vector through an input embedding. This process is essentially a lookup table where each token is converted into a continuous vector representation. These embeddings allow the model to process and learn from the input data more effectively.

Positional encoding: Transformers do not inherently account for the order of the input sequence, as they lack the sequential nature of recurrent models. To address this, positional encoding is added to the input embeddings. Positional encoding provides information about the position of each token in the sequence, allowing the model to differentiate between tokens based on their order. In the original paper [15], sinusoidal functions were used to generate unique positional encodings for each position.

Encoder: The transformer encoder processes the input embeddings with positional encodings through multiple identical layers. Each encoder layer consists of two main components. The multi-head attention layer enables the model to focus on different parts of the input sequence simultaneously, capturing various dependencies. This is followed by a fully connected layer which processes the

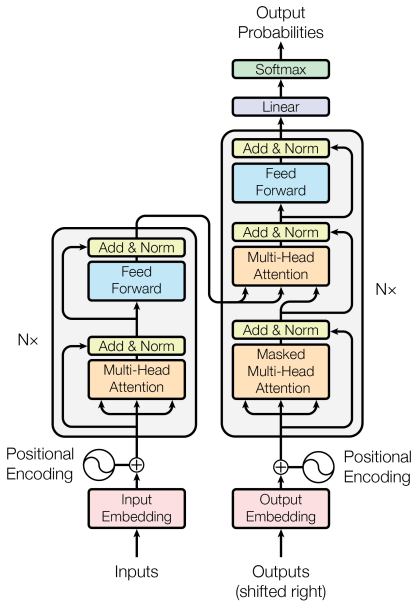


Figure 2.12: The Transformer - model architecture. Encoder (left), decoder (right). Taken with permission from [15]

output of the multi-head attention, further transforming the data. Each sublayer incorporates residual connections and normalization.

Decoder: The transformer decoder is similarly structured, consisting of multiple identical layers. They consist of a masked attention layer, which is similar to an attetion layer but prevents the model from attending to future tokens in the sequence during training, ensuring that predictions are based only on past and present tokens. A multi-head attention which is applied to the output of the encoder and a fully connected neural network. All sublayers employ residual connections and normalization. The final output of the decoder is converted to probabilities using a linear layer followed by a softmax function.

Bidirectional transformer models, such as BERT [20], utilize an encoder-only architecture. This design allows the attention mechanism to consider information from both directions within the sequence, enabling the model to capture a more comprehensive understanding of the context. The issue of "seeing into the future" is resolved through a training technique called *masked modeling*, where some tokens are masked and the model is trained to predict them based on the surrounding context. This will be looked at further in Section 2.8.2.

2.8 Self-Supervised learning

Self-supervised learning (SSL) has achieved great success in natural language processing (NLP) and computer vision in recent years and is rapidly being applied to other modalities. In our work, we leverage SSL for representation learning in the time series domain. Before diving into SSL, it is important to understand the broader context of machine learning, which can be coarsely divided into two categories: supervised and unsupervised learning.

Supervised Learning

Supervised learning involves training models using labeled data.For a given input x, the desired output y is known during training, allowing for direct supervision of the model's parameters by comparing its output to the true value. A bit more formally, for a dataset $X = \{x_i\}_{i=1}^N$ with corresponding human-annotated labels $Y = \{y_i\}_{i=1}^N$, the objective of a supervised learning algorithm is to fit a function f_θ that minimizes the loss across the data

$$\widehat{f_\theta} = \min_\theta \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), y_i). \tag{2.11}$$

Common approaches to supervised learning for neural networks is to calculate some distance metric between the predicted value $\widehat{y}$ and the true value y and update parameters using backpropagation.
Supervised learning models, including classical statistical models, support vector machines, and decision tree-based models, have seen tremendous success but are limited by the need for labeled data, which is often scarce and expensive to obtain.

Unsupervised Learning

Unsupervised learning, on the other hand, refers to models that learn exclusively from unlabeled data, discovering intrinsic patterns without explicit labels. Examples of unsupervised learning techniques include clustering methods (e.g., K-means, K Nearest Neighbor, Gaussian mixture models), dimensionality reduction techniques (e.g., PCA, SVD), and neural network architectures such as autoencoders. These techniques are widely used in exploratory data analysis, data visualization, and clustering. While unsupervised learning is valuable in a world abundant with unlabeled data, it has historically struggled to match the performance of supervised approaches.

Self-Supervised Learning

Self-supervised learning is a subcategory of unsupervised learning that has shown remarkable success in NLP and computer vision, approaching the performance of

state-of-the-art supervised representation learning [21, 22]. SSL involves using the data itself to generate a supervisory signal, rather than relying on external labels. Although SSL is technically unsupervised, its learning formulation closely resembles that of supervised learning.

For a dataset $X = \{x_i\}_{i=1}^N$ with *pseudo labels* $P = \{p_i\}_{i=1}^N$, the objective of a self-supervised learning algorithm is to fit a function f_θ hat minimizes the loss across the data

$$\widehat{f_\theta} = \min_\theta \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), p_i). \tag{2.12}$$

Pseudo labels are automatically generated labels from the data. In *siamese* architectures, these pseudo labels are typically augmentations of the original data. SSL has become an integral part of pretraining large models, such as those used in LLMs and vision transformers, enabling them to learn from vast amounts of unlabeled data. Pretraining with SSL allows models to capture the semantics of data without the need for extensive labeled datasets, significantly reducing the resources required for subsequent supervised training.

2.8.1 Siamese Architecture-based SSL

A siamese network architecture [23] consists of two branches with a shared encoder on each branch. These networks are trained to produce similar representations for different views of the same data. A major challenge in this architecture is avoiding trivial solutions, where the networks ignore the input and produce identical constant embeddings, a problem known as *collapse*.

In computer vision, several representation learning algorithms utilizing siamese architectures have been proposed. These can be broadly categorized into *contrastive* and *non-contrastive* SSL methods.

Contrastive SSL uses positive and negative samples to learn representations by pulling positive pairs closer together and pushing negative pairs further apart. Examples include MoCo [24] and SimCLR [25]. Contrastive methods often require large batch sizes as well as a substatial number of negative pairs compared to positive in order to learn representations effectively [26].

Non-Contrastive SSL methods, such as BYOL [27], BarlowTwins [22] and VIbCReg [26] avoid the need for positive and negative pairs. These methods use augmentations of the input data as pseudo labels and introduce different mechanisms to prevent collapse. For instance, BYOL introduces architectural asymmetry and the stop-gradient operation, Barlow Twins minimizes information redundancy between branches, and VICReg maintains variability while decorrelating features in each branch.

2.8.2 Masked modelling

Masked modeling is a straightforward self-supervised learning technique for generative models. The core idea is to mask or cover a portion of the data and train the model to predict the masked portions. By comparing the predictions against the unmasked data, the model learns useful representations without explicit supervision. Introduced in NLP by BERT [20], masked modeling has become the standard for self-supervised pretraining of language models.

In computer vision, masked modeling has also gained attention. Initial approaches involved masking image patches directly [28]. With the success of vision transformers [16], researchers began tokenizing images and pretraining transformers in a manner similar to BERT. For example, MaskGIT [29] proposed masked visual token modeling, using vector-quantized-based tokenization and a bidirectional transformer. For a comprehensive survey on masked modeling in the vision domain, refer to [30].

2.9 Vector Quantized Variational Autoencoder (VQVAE)

Our model is based on the Vector Quantized Variational Autoencoder (VQVAE) introduced in [19], and includes an Auto-encoder (AE) branch. Therefore it is natural to dive into the models. We first start with introducing auto-encoders, then present the variational variation VAE before presenting VQVAE.

2.9.1 Autoencoder (AE)

An autoencoder consists of two neural networks: an encoder E and a decoder D. These networks map data between two spaces: the data space X and the latent space Z. The encoder compresses the data from the data space to the latent space, while the decoder reconstructs the data from the latent space back to the data space. A schematic of the architecture is presented in Figure 2.13.

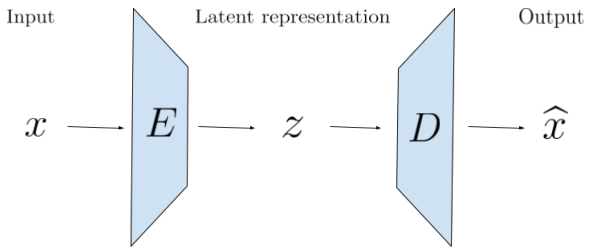


Figure 2.13: Schematics of the autoencoder architecture.

The primary goal of an autoencoder is to learn an efficient encoding of the data. The encoder compresses the data into a lower-dimensional representation, and the decoder reconstructs the original data from this compressed representation. The training objective is to make the composition of the encoder and decoder

approximate the identity function, meaning that the output should be as close as possible to the input. Autoencoders are typically trained to minimize the reconstruction error, which typically is the mean square error (MSE) between the input and the reconstructed output. This process forces the model to learn efficient latent representations of the data.

One of the critical aspects of autoencoders is the *information bottleneck*. The dimension of the latent space is much smaller than that of the data space. This bottleneck forces the model to capture the most important features of the data while discarding less relevant information. By compressing the data, the encoder learns a compact representation that retains the essential features needed for accurate reconstruction by the decoder However, learning effective latent representations can be challenging. Without proper regularization, the latent space may capture noise or irrelevant details, leading to overfitting.

Event though Autoencoders are outdated, they serve as the foundational building block for more complex models such as Variational Autoencoders (VAEs) and Vector Quantized Variational Autoencoders (VQVAEs), which further enhance the ability to learn meaningful and useful representations from data.

2.9.2 Variational Autoencoder (VAE)

Variational Autoencoders (VAE) were introduced in [31] and represent a variational Bayes approach to approximate inference and learning with directed probabilistic models. While the architecture of VAEs is similar to that of traditional autoencoders (AEs), their mathematical formulation is fundamentally different. Variational inference is a statistical technique used to approximate complex distributions by finding the closest approximation within a simpler, but flexible, parametric family.

In the VAE framework, we assume that the dataset $X = \{x_i\}_{i=1}^N$ consists of iid samples from a random variable $\mathbf{x}$. We further assume that the data is generated by some unobservable random process. Specifically, that there is a latent variable $\mathbf{z}$ such that $x_i \sim p_{\theta^*}(x|z_i)$, where $z_i \sim p_{\theta^*}(z)$. The distribution $p_{\theta^*}(z)$ is referred to at the true prior, and $p_{\theta^*}(x|z_i)$ as the true likelihood. Since $\mathbf{z}$ is unobservable and the true distributions are unknown, one has to assume their form. In general the prior and likelihood are assumed to be from parametric families $p_\theta(z)$ and $p_\theta(x|z)$. These distributions are typically chosen to be differentiable to facilitate gradient-based learning.

VAEs have two main components: the probabilistic encoder (or inference model) $q_\phi(z|x)$, which approximates the true posterior, and the probabilistic decoder (or generative model) $p_\theta(x|z)$, which approximates the likelihood. These models are typically parameterized by neural networks, with ϕ and θ representing

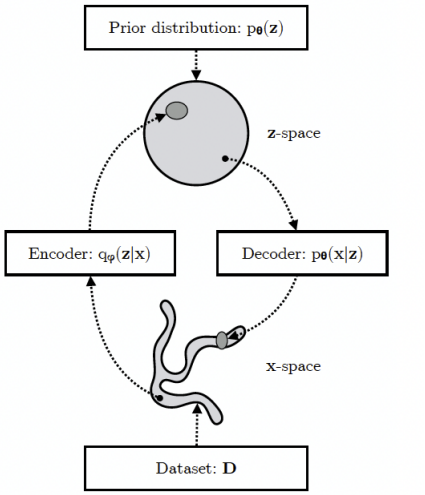


Figure 2.14: [32], need to ask for permission or make own

their weights and biases. Given a datapoint x_i, the probabilistic encoder provides a distribution over the possible values of the latent variable **z**. Similarly, given a latent representation z_i, the probabilistic decoder produces a distribution over the possible corresponding values of **x**.

The probabilistic nature of the encoder and decoder involves sampling. For a given x or z, the output is a distribution, and the same input will yield the same output distribution if the network parameters are unchanged. Actuallt, x and z are mapped to the parameters of a distribution, which uniquely determines the distribution in a particular family.

Common assumptions for the distributions are:

- Prior $p_\theta(z) \sim N(0, I)$
- Likelihood $p_\theta(x|z) \sim N(D(z), I)$
- Variational posterior $q_\phi(z|x) \sim N(E(x)) = N(\mu_x, \Sigma_x)$

The encoder maps datapoints to the parameters of the variational distribution q_ϕ,

$$x \mapsto E(x) = (\mu_x, \Sigma_x).$$

A latent representation z is then sampled from q_ϕ, which constitutes the random part of the algorithm. The decoder maps z to the expected value of the likelihood $p(x|z)$,

$$z \mapsto D(z) = \widehat{x}.$$

There are several reasons for choosing Gaussian distributions, one being that the Gaussian distribution a scale-location family. This enables us to employ the reparameterization trick, and circumvent the problematic random component in

the VAE when using gradient based learning. It involves introducing an auxillary variable $\epsilon \sim N(0, I)$ and rewriting $z = \mu_x + L_x\epsilon$, where L_x is the Cholesky decomposition of Σ_x. This allows the gradient to flow through the deterministic parts of the model.

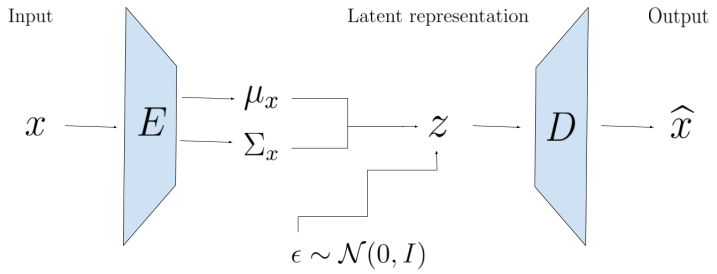


Figure 2.15: Schematics of the VAE architecture with Gaussian reparameterization.

Training objective

VAEs are optimized with respect to the *evidence lower bound* (ELBO). For jointly distributed variables **x** and **z** with distribution p_θ. Then for any distribution q_ϕ the ELBO is defined as

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{p_\theta(x,z)}{q_\phi(z|x)}\right) \tag{2.13}$$

This can be reformulated in terms of the marginal likelihood and KL divergence between the variational and true posterior

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)} \log\left(p_\theta(x)\right) + \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{p_\theta(z|x)}{q_\phi(z|x)}\right) \\ &= \log\left(p_\theta(x)\right) - \mathrm{KL}(q_\phi(z|x)||p_\theta(z|x)).\end{aligned} \tag{2.14}$$

Due to the non-negativity of the KL-divergence, we see that the ELBO bounds the marginal log likelihood of the data from below. By maximizing the ELBO with respect to the model parameters ϕ and θ, one simultaneously maximizes the marginal likelihood and minimizes the KL divergence, improving both the generative and inference models [32].

An alternative formulation shows a more evident connection to autoencoders, with the prior acting as a regularizer on the posterior

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)} \log\left(p_\theta(x|z)\right) - \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{q_\theta(z|x)}{p_\phi(z)}\right) \\ &= \underbrace{\mathbb{E}_{q_\phi(z|x)} \log\left(p_\theta(x|z)\right)}_{\text{Expected reconstruction log likelihood}} \underbrace{- \mathrm{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{Regularizer}}.\end{aligned} \tag{2.15}$$

Assuming a Gaussian likelihood, we have

$$\log p_\theta(x|z) = -\frac{1}{2}\left[k\log(2\pi) + (x-\widehat{x})^T(x-\widehat{x})\right], \tag{2.16}$$

which is equivalent, as an optimization objective of $\widehat{x}$, to $||x-\widehat{x}||_2^2$. Consequently the likelihood in the loss is implemented as the MSE of the input x and the output $\widehat{x}$.

Generative model

The prior in a VAE serves two main roles: as a regularizing constraint for the posterior during training, and as a means of generating new samples via ancestral sampling. Ancestral sampling refers to **z** being the parent node of **x** in the VAE, and that we can generate samples x by drawing a sample $z \sim p(z)$ and decode $D(z)$.

Limitations

Despite their strengths, VAEs can suffer from issues such as posterior collapse, where the latent variable **z** fails to capture meaningful information about the input data. This can lead to poor reconstruction quality and less informative latent representations. Additionally, VAEs may struggle with variance issues, where the variance of the learned latent distributions is not well-calibrated, affecting the quality of generated samples.

These limitations have motivated the development of more advanced models, such as the Vector Quantized Variational Autoencoder (VQVAE), which aims to address some of these challenges by introducing discrete latent variables and a learnable prior distribution.

2.9.3 Vector Quantization (VQ)

Dictionary learning model [33]

2.9.4 VQVAE

The Vector Quantized Variational AutoEncoder (VQVAE) was first introduced in [19] and presented a novel approach to training VAEs with discrete latent variables. It was the first model to achieve similar performance to continuous VAEs while utilizing discrete latent representations. VQVAE was developed to enhance representation learning by allowing the encoder network to output discrete codes and by learning the prior distribution rather than assuming it to be static. This approach provides more flexibility in capturing the underlying data distribution while simultaneously avoiding posterior collapse and variance issues observed in VAEs.

Architecture

The overall architecture of VQVAE comprises three main components: an encoder, a decoder, and a *codebook*. The encoder and decoder function similarly to those in traditional autoencoders, while the codebook, also referred to as the latent embedding space, acts as a lookup table for vector quantization.

The codebook, denoted by $\mathcal{Z} = \{z_k\}_{i=1}^{K}$, consists of K latent vectors with dimensionality D. During the encoding process, the output of the encoder $E(x)$ is quantized by finding the nearest neighbor in the codebook

$$z_q = \arg\min_{z_k \in \mathcal{Z}} ||E(x) - z_k||. \tag{2.17}$$

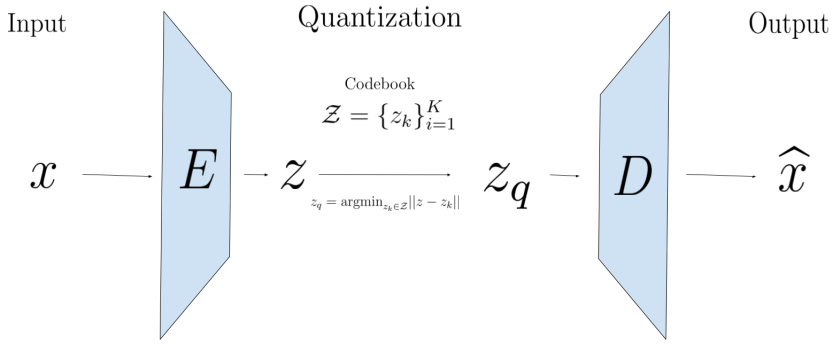


Figure 2.16: Schematics of the VQVAE architecture.

From 2.16 we can see that the VQVAE can be seen as an autoencoder with a nonlinearity introduced by the vector quantization.

ELBO

Unlike traditional VAEs, where the posterior distribution is assumed to be Gaussian, the posterior in VQVAE is categorical. The probabilities are defined as

$$p(z = k|x) = \begin{cases} 1 & \text{for } k = \arg\min_j ||E(x) - z_j||_2 \\ 0 & \text{otherwise} \end{cases}, \tag{2.18}$$

Sampling from the posterior amounts to quantizing the output of the encoder, as it is deterministic.

In the original article [19], the authors propose to learn the prior distribution separately after training the encoder, decoder, and codebook. Initially, the prior is assumed to be uniform, allowing the posterior to learn without constraints. The VQVAE can too be viewed as a VAE, allowing the use of the ELBO 2.15 to bound the marginal likelihood. Since the variational posterior is deterministic and the

prior (during training) is uniform over $\{1, ..., K\}$ we get that the regularizing term simplifies to a constant

$$\begin{aligned} \mathrm{KL}(q(z|x)||p(z)) &= \sum_z q(z|x) \log\left(\frac{q(z|x)}{p(z)}\right) \\ &= q(z=k|x) \log\left(\frac{q(z=k|x)}{p(z=k)}\right), \; q \text{ is deterministic} \\ &= \log\left(\frac{1}{1/K}\right), \text{ uniform prior} \\ &= \log(K). \end{aligned} \tag{2.19}$$

Thus, the ELBO reduces to the reconstruction term only.

Loss funciton

The VQVAE employs a loss function designed to facilitate gradient-based learning despite the non-continuous nature of the quantization process. As the input of the decoder has the same dimension, we circumvent the discontinuities by simply copying the gradients from the decoder to the encoder. The gradients from the decoder contains relevant information to how the encoder should change its output, and in the subsequent forward pass the encoder output can be quantized to different discrete latent codes.

The overall loss function in VQVAE consists of two main components, the reconstruction loss and the codebook loss. The reconstruction loss ensures that the output of the decoder closely matches the input, while the codebook loss manages the relationship between the encoder output and the quantized output.

The reconstruction loss is the typical mean square error of the input x and the reconstructed output $\widehat{x}$

$$\mathcal{L}_{\text{recons}} = ||x - \widehat{x}||_2^2. \tag{2.20}$$

The codebook loss has two parts. The first part measures the euclidean distance between the encoder output and the quantized output, while the second part, known as the commitment term, which is introduced to keep the distance between codewords from growing arbitrarily large. It is given by

$$\mathcal{L}_{\text{codebook}} = ||sg(z) - z_q||_2^2 + \beta||z - sg(z_q)||_2^2, \tag{2.21}$$

where β is a tuning parameter, typically set to be 0.25, and $sg()$ is the stop-gradient operation, behaves as the identity with zero partial derivatives.

The total loss function is given by the sum of the two components

$$\mathcal{L}_{\text{VQ}} = \mathcal{L}_{\text{codebook}} + \mathcal{L}_{\text{recons}}. \tag{2.22}$$

The codebook loss only affects the codebook, and can hence alternatively be updated as a function of moving averages of the encoder outputs z. Further details on this method can be found in Appendix A.1 in [19].

To understand the necessity of the commitment term, consider a simple example where the data is either 0 or 1, and the codebook is initialized with $\mathcal{Z} = \{-1, 1\}$. The encoder output $E(x)$ is quantized to -1 if its negative and 1 if its positive. Suppose that the encoder and decoder will try to differentiate between the two classes by pushing $E(0)$ and $E(1)$ away from each other. Since the reconstruction loss only affects the encoder and decoder, and the codebook loss only affect the codebook, if the encoder and decoder parameter trains faster than the codebook the distance between the encodings and the codewords will increase and the codebook loss diverge. This behavior is observed experimentally.

Prior Learning

In VQVAE, the prior learning process is designed to enhance the flexibility and quality of the generative model. During the initial training, the prior is uniform. This approach allows the encoder to focus on learning the optimal mapping from the data space to the discrete latent space without being influenced by a predefined prior distribution. After the initial training, the prior distribution is refined by fitting a model on the latent variables, referred to as prior learning. By training the prior with a generative objective, on learns a more accurate prior which can improve the quality of generated samples. The choice of prior model depends on the particular application. In the original VQVAE paper PixelCNN [34] was used for image data, while WaveNet [35] was used for audio. More recently in TimeVQVAE [1] utilized a bidirectional transformer [29] for prior learning in the time series domain.

2.10 Evaluation metrics

Downstream Classification

When evaluating the performance of representation learning, downstream classification tasks provide an insightful measure of how well the learned representations can be utilized for practical applications. Two commonly used classifiers are Support Vector Machines (SVM) with linear kernel and K-Nearest Neighbors (KNN), each with distinct inductive biases. The SVM method helps to determine to what degree the representations are linearly separable, while KNN better capturing local structures in the data. The difference in inductive biases between SVM and KNN can be leveraged to evaluate the robustness and generalizability of the learned representations. An illustration of the difference in decision boundary between the two classifiers is shown in Figure 2.17

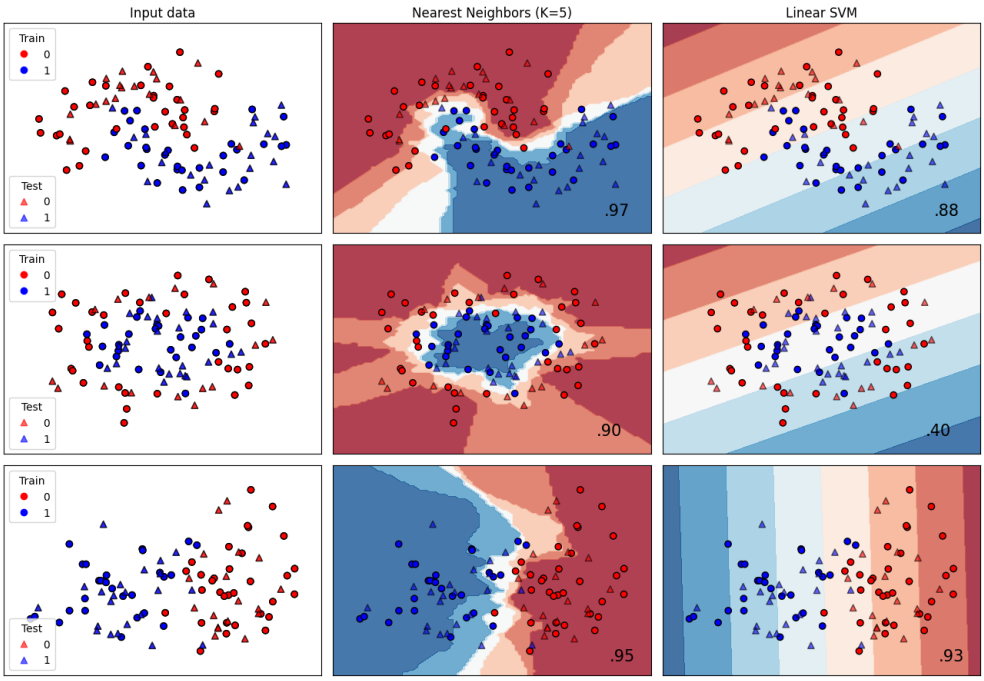


Figure 2.17: Modified example taken from `https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html`.

2.10.1 Generative model

Evaluating generative models remains a challenging task, as it requires assessing how well the generated samples match the real data distribution. While some data modalities have established evaluation standards, such as ImageNet and Inception v3 in the computer vision domain, others like time series lack widely accepted benchmarks.

The traditional sanity check for generative models is human visual inspection. However, this approach is subjective and varies significantly between data modalities. For example, while most people can easily judge the quality of generated images, interpreting generated time series data often requires domain-specific knowledge. This is one of the obstacles in TSG. Despite this, according to [1] the most common evaluation protocols in the TSG literature is PCA and t-SNE analyses to visually see similarities of distributions, complemented by quantitative metrics where possible.

Following the approach in [1], we employ three primary evaluation metrics in this thesis: Inception Score (IS), Fréchet Inception Distance (FID), and Classification Accuracy Score (CAS).

All the mentioned evaluation metrics depend on a pre-trained classification

model. Unlike computer vision, where models like Inception v3 are standard, there is no universally adopted classification model for time series data. However, [36] proposed a fully convolutional network (FCN) as a baseline model for time series classification. Since the original model is not readily available, we utilize an open-source FCN trained on the UCR Archive, as presented in [1], for our evaluations.

Inception Score (IS)

The Inception Score (IS) was first introduced in [37] as an automatic method for evaluating the quality of synthetic samples. IS measures the clarity and diversity of generated samples by using a pre-trained classifier to predict labels. The intuition behind IS is that high-quality samples should result in low entropy in the conditional label distribution $p(y|x)$, while diverse samples should lead to high entropy in the marginal label distribution $p(y)$. Thus if the KL-divergence between these distributions are high, the samples should be of high quality. The Inception Score is defined as

$$\mathrm{IS}(\theta) = \exp\left(\mathbb{E}(D_{\mathrm{KL}}(p_\theta(y|\mathbf{x})||p_\theta(y)))\right). \tag{2.23}$$

In the context of time series, we use a fully convolutional network (FCN) trained on the UCR Archive [1] to obtain an estimate for $p(y|x)$. For a synthetic dataset $X_{\mathrm{gen}} = \{x_{i,\mathrm{gen}}\}_{i=1}^{N}$, the marginal label distribution is obtained by averaging across all the synthetic data as follows

$$p(y) = \frac{1}{N}\sum_{i=1}^{N} p(y|x_{i,\mathrm{gen}}).$$

Issues with IS [38].The primary concern with IS is that it does not use any statistics of real world samples to compare with the statistics of the generated samples. Important to report different metrics that indicate that the model has not overfitted.

Also issues with IS [39].

Fréchet Inception Distance (FID)

To address some limitations of IS, the Fréchet Inception Distance (FID) was introduced in [40]. Since then FID has been the standard for assessing generative models [39]. FID compares the distributions of real and generated samples using the Fréchet distance, providing a measure of how similar the generated samples are to the real data. For any two probability distributions, f, g over $\mathbb{R}^n$, with finite

mean and variances, their Fréchet distance is defined as

$$d_F(f,g) = \left(\inf_{\gamma \in \Gamma(f,g)} \int_{\mathbb{R}^n \times \mathbb{R}^n} ||x-y||_2^2 d\gamma(x,y) \right)^{\frac{1}{2}} \\ = \left(\inf_{\gamma \in \Gamma(f,g)} \mathbb{E}_{(x,y)\sim\gamma} ||x-y||_2^2 \right)^{\frac{1}{2}}, \tag{2.24}$$

where $\Gamma(f,g)$ is the set of all *couplings* of f and g. For two Gaussian distributions, as proven in [41], the Fréchet distance is explicitly computable as

$$d(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\mu',\Sigma'))^2 = ||\mu-\mu'||_2^2 + \mathrm{Tr}\left(\Sigma + \Sigma' - 2(\Sigma\Sigma')^{\frac{1}{2}}\right). \tag{2.25}$$

In FID, the mean and covariance of the feature representations of real and generated samples are computed using a pre-trained model. They argue in [40] that since the Gaussian distribution is the maximum entropy distribution over $\mathbb{R}^n$, for a given mean and covariance, it is a reasonable distribution to assume for the representations. But, despite its usefulness, FID has limitations, such as the Gaussian assumption not always holding true and the need for a large number of samples to estimate the covariance matrix reliably [42], [43].

Classification Accuracy Score (CAS)

Classification Accuracy Score (CAS) evaluates the model's ability to learn class-conditional distributions. This is done by training a classifier on synthetic data and testing it on real data (Train on Synthetic, Test on Real - TSTR). The process involves generating synthetic samples conditioned on class labels, training a classifier on these samples, and evaluating its accuracy on a real test dataset. High CAS values indicate that the generative model produces samples that capture relevant class-specific features.

In this thesis, we use the Supervised FCN introduced in [1] to evaluate CAS across all models considered, comparing the results against a baseline model to assess relative performance.

Chapter 3

Related Work

Our work involves building a framework that uses non-contrastive self-supervised learning (SSL) to improve representation learning in a Vector Quantized Variational Autoencoder (VQVAE) type model. Additionally, we fit a prior model on the learned representations. This section presents the works related to ours. To the best of our knowledge, enhancing VQ-based tokenization models with SSL methods has not yet been investigated in the time series domain. Therefore, the related works consist of models used as different components in our model.

We base our model on a simplified version of TimeVQVAE, which utilizes a bidirectional transformer model, MaskGIT, for prior learning. Although our non-contrastive SSL extension could use any model with a siamese architecture, we experiment with the proven methods Barlow Twins and VIbCReg.

3.1 MaskGIT

The Masked Generative Image Transformer (MaskGIT), introduced in [29], is a generative transformer model for image synthesis developed by Google Research. The innovation of MaskGIT lies in its token generation method. Unlike traditional autoregressive generative transformers that treat images as a sequence of tokens, MaskGIT employs a bi-directional transformer for image synthesis. This allows MaskGIT to predict tokens in all directions during training, offering a more natural approach to image modeling. During inference, MaskGIT begins with a blank canvas, predicting the entire image iteratively by conditioning on the most confident pixels from previous predictions.

MaskGIT's architecture relies on a tokenization procedure in its first stage. In the original paper, VQGAN [44] was used for this purpose, focusing on improving the second stage. Therefore, our discussion will primarily address this aspect of the model.

3.1.1 Masked Visual Token Modeling (Prior learning)

For prior learning, the codebook learned in the tokenization procedure is provided with a masking vector, which is the embedding of the special masking token, which we denote by `M`. The input embedding in the bidirectional transformer is initialized with this expanded codebook. For an image X in the dataset $\mathcal{D}$, let $z = \{z_{k_i}\}_{i=1}^N$ denote the sequence of codewords obtained by passing X through the VQ-Encoder. This sequence can equivalently be described as a sequence of indices $s = \{k_i\}_{i=1}^N$. Prior learning involves masking this sequence and training the bidirectional transformer to predict the masked indices.

Let $s = \{k_i\}_{i=1}^N$ be the sequence of indices described above, and denote the corresponding binary mask by $M = \{m_i\}_{i=1}^N$. During training, a subset of s is replaced by the masking token `M` according to the binary mask M. This is done by

$$s_{\text{Mask}} = s \odot (1_N - M) + M \cdot \texttt{M}, \tag{3.1}$$

where $\odot$ is the Hadamard product (pointwise multiplication), and 1_N is a vector with the same shape as M and s.

The sampling procedure, or choice number of tokens to mask, is parameterized by a mask scheduling function γ. The sampling can be summarized as follows

- Sample $r \sim U(0,1]$.
- Sample $\lceil \gamma(r) \cdot N \rceil$ indices I uniformly from $\{0,\dots,N-1\}$ without replacement.
- Create M by setting $m_i = 1$ if $i \in I$, and $m_i = 0$ otherwise.

A forward computation is illustrated in Figure 3.1, where the bidirectional transformer predicts the probabilities $p(s_i|s_{\text{Mask}})$ of each masked token, and $\mathcal{L}_{\text{Mask}}$ is computed as the cross entropy between the ground truth one-hot token and the predicted token probabilities.

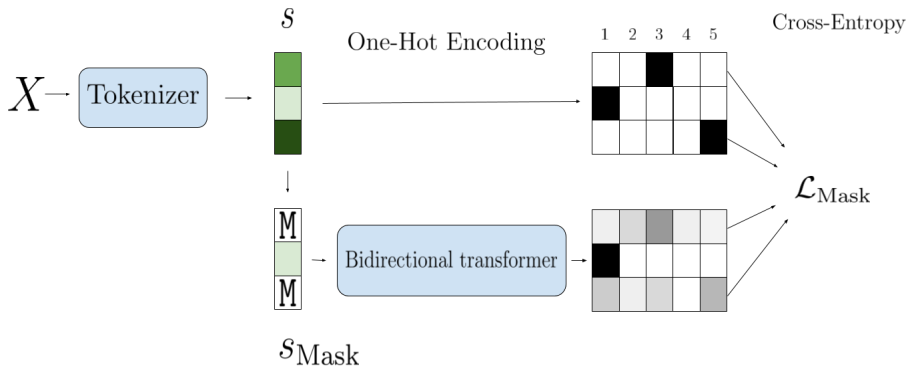


Figure 3.1: MaskGIT forward computation.

The training objective is to minimize the negative log likelihood of the masked tokes, conditional on the unmasked

$$\mathcal{L}_{\text{Mask}} = -\mathbb{E}_{s\in\mathcal{D}}\left[\sum_{i\in I} p(s_i|s_{\text{Mask}})\right] \tag{3.2}$$

3.1.2 Iterative Decoding (Sample generation)

The bi-directional transformer could in principle predict all masked tokens and generate a sample in a single pass by sampling according to the predicted probabilites $p(\widehat{s}_i|s_{\text{Mask}})$ from a forward pass of an all masked sequence. However, this approach presents certain challenges. To address these, [29] proposes a novel non-autoregressive decoding method to synthesize samples in a constant number of steps.

The decoding process goes from $t = 0$ to T. To generate a sample during inference, the process starts with an all masked sequence, denote as $s^{(0)}_{\text{Mask}}$. At iteration t, the model predicts the probabilities for all the mask tokens, $p(\widehat{s}_i|s^{(t)}_{\text{Mask}})$, in parallel. At each masked index i, a token $s^{(t)}_i$ is sampled according to the predicted distribution, and the corresponding probability $c^{(t)}_i$ is used as a measure of the confidence in the sample. For unmasked tokens, a confidence of 1 is assigned to the true position.

The number of tokens $s^{(t)}_i$ with highest confidence that are kept for the next iteration is determined by the mask scheduling function. Specifically, $n = \lceil \gamma(t/T) \cdot N \rceil$ of the lower confidence tokens are masked again by calculating $M^{(t+1)}$ as follows

$$m^{(t+1)}_i = \begin{cases} 1, \text{ if } c_i < \text{Sort}([c^{(t)}_1, \ldots, c^{(t)}_N])[n] \\ 0, \text{ otherwise} \end{cases} \tag{3.3}$$

This process is illustrated in Figure 3.2, which shows the first pass of the iterative decoding algorithm. The algorithm synthesizes a complete image in T steps. For image generation, the cosine scheduling function proved to be the most effective across all experiments in the original paper.

3.2 TimeVQVAE

TimeVQVAE is a time series generation model based on VQVAE and MaskGIT. It is the first to our and the authors knowledge that utilizes vector quantization (VQ) to address the time series generation problem. TimeVQVAE employs a two-stage approach similar to VQVAE, with a bidirectional transformer, akin to MaskGIT, for prior learning. The authors introduce VQ modeling in the time-frequency domain,

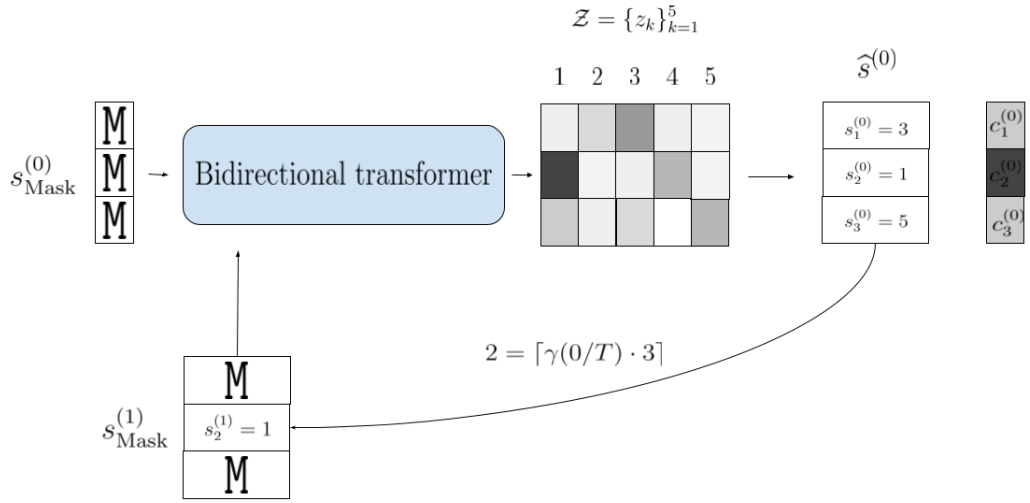


Figure 3.2: Illustration of first pass of the iterative decoding algorithm.

separating data into high and low-frequency components to better retain temporal consistencies and generate higher quality samples.

In addition to VQ modeling in the time-frequency domain, TimeVQVAE presents a process for sampling jointly from high and low-frequency latent spaces and enables guided class-conditional sampling. By appending a class token, similar to [16], the model can generate synthetic samples both conditionally and unconditionally.

Our work extends a variation of the TimeVQVAE model without the high-low frequency split, reducing the prior learning method to MaskGIT with the addition of guided class-conditional sampling. Here, we present the tokenization stage and refer the reader to [19] for details on the prior model training.

3.2.1 Tokenization

The tokenization stage of TimeVQVAE follows a structure similar to VQVAE, with the key difference being the frequency split. An overview of the model is presented in Figure 3.3. Initially, a time series is mapped to the time-frequency domain using the Short-time Fourier Transform (STFT). The time-frequency representation is then separated into two branches: one zero-padding the high-frequency (HF) region and the other zero-padding the low-frequency (LF) region. Each branch follows the VQVAE architecture, with separate encoders, decoders and codebooks denoted by E_{LF}, E_{HF}, D_{HF}, D_{HF} and $\mathcal{Z}_{\text{LF}}$, $\mathcal{Z}_{\text{HF}}$ respectively. The output of the decoders are again zero-padded giving $\widehat{u}_{\text{LF}}$ and $\widehat{u}_{\text{HF}}$, which are then mapped back to

the time domain using the Inverse Short-time Fourier Transform (ISTFT) to produce the reconstructed HF and LF components, $\widehat{x}_{\text{LF}}$ and $\widehat{x}_{\text{HF}}$, of the time series.

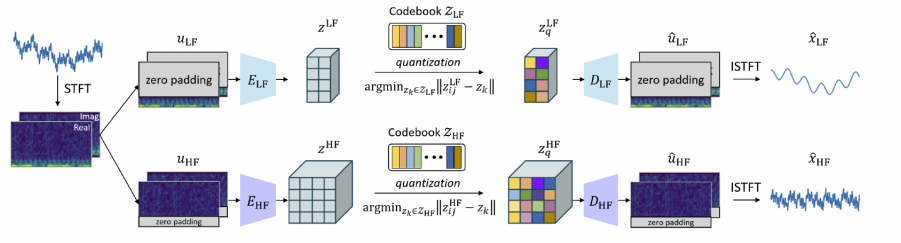


Figure 3.3: Stage 1: Tokenization. Figure taken with permission from [1]

Loss

The codebook loss of TimeVQVAE is similar to the codebook loss presented in Section 2.9but reflects the HF-LF split

$$\begin{aligned}\mathcal{L}_{\text{codebook}} = &|| \text{sg}[E_{LF}(\mathcal{P}_{\text{LF}}(\text{STFT}(x)))] - z_q^{\text{LF}}||_2^2 \\ &+ || \text{sg}[E_{HF}(\mathcal{P}_{\text{HF}}(\text{STFT}(x)))] - z_q^{\text{HF}}||_2^2 \\ &+ \beta||E_{LF}(\mathcal{P}_{\text{LF}}(\text{STFT}(x))) - \text{sg}[z_q^{\text{LF}}]||_2^2 \\ &+ \beta||E_{HF}(\mathcal{P}_{\text{HF}}(\text{STFT}(x))) - \text{sg}[z_q^{\text{HF}}]||_2^2,\end{aligned} \tag{3.4}$$

The reconstruction loss is computed both on time and time-frequency reconstructions and is given by

$$\begin{aligned}\mathcal{L}_{\text{recons}} = &||x_{\text{LF}} - \widehat{x}_{\text{LF}}||_2^2 + ||x_{\text{HF}} - \widehat{x}_{\text{HF}}||_2^2 \\ &+ ||u_{\text{LF}} - \widehat{u}_{\text{LF}}||_2^2 + ||u_{\text{HF}} - \widehat{u}_{\text{HF}}||_2^2.\end{aligned} \tag{3.5}$$

The total loss is given by

$$\mathcal{L}_{\text{VQ}} = \mathcal{L}_{\text{codebook}} + \mathcal{L}_{\text{recons}}. \tag{3.6}$$

To update the codebooks, TimeVQVAE uses an exponential moving average method as presented in Appendix A.1 of [19].

3.3 SSL

Our model leverages self-supervised learning algorithms to learn more expressive latent representations. In this section, we present the relevant SSL algorithms for our work: Barlow Twins and VIbCReg.

3.3.1 Barlow Twins

Barlow Twins is a non-contrastive SSL method that applies the *redundancy-reduction principle*[45] from neuroscientist H. Barlow to a pair of identical networks. The key idea is to encourage representations of similar samples to be alike while reducing redundancy between the components of the vectors.

The model produces two augmented views of each sample and projects their representations onto a feature space such that their cross-correlation is close to the identity matrix. This approach ensures that the representations are both invariant to distortions and decorrelated.

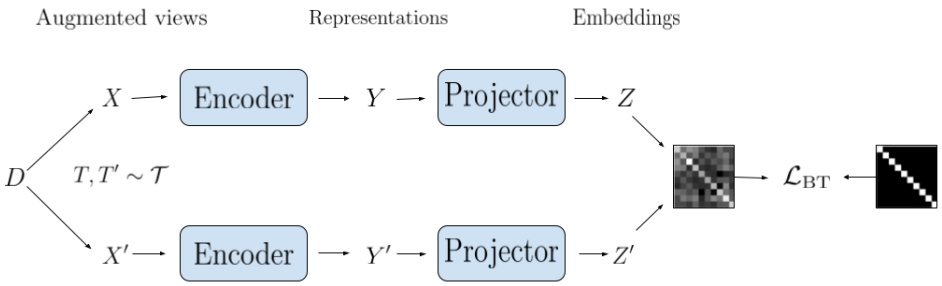


Figure 3.4: Overview of the Barlow Twins architecture. Figure inspired by [22]

The Barlow Twins algorithm, illustrated in Figure 3.4, starts out by creating two augmented views for each datapoint in a batch D. These augmentations are sampled from a collection of augmentations $\mathcal{T}$. The batches of augmented views $T(D) = X$ and $T'(D) = X'$, are then passed through an encoder to give representations Y and Y', which are further projected to produce embeddings Z and Z'. The embeddings are assumed to be mean centered across the batch dimension.

The loss function is calculated using the cross-correlation matrix $\mathcal{C}$ between Z and Z', measuring its deviation from the identity. The Barlow Twins loss is defined as

$$\mathcal{L}_{\text{BT}} = \overbrace{\sum_i (1 - \mathcal{C}_{ii})^2}^{\text{Invariance}} + \lambda \overbrace{\sum_i \sum_{j \neq i} \mathcal{C}_{ij}^2}^{\text{Redundancy reduction}}, \tag{3.7}$$

where

$$\mathcal{C}_{ij} = \frac{\sum_b z_{b,i} z'_{b,j}}{\sqrt{\sum_b (z_{b,i})^2} \sqrt{\sum_b (z'_{b,j})^2}}. \tag{3.8}$$

The *invariance term* helps make the embeddings invariant to the distortions introduced by the augmentations, pushing the representations closer together. The *redundancy reduction term* decorrelates the different vector components, reducing

the information redundancy.

3.3.2 VIbCReg

VIbCReg [**lee2024vibcreg**] is a non-contrastive SSL model with a siamese architecture based on VICReg [21]. It improves upon VICReg by incorporating better covariance regularization and IterNorm [46]. A key difference from Barlow Twins is that variance/covariance regularization is applied to each branch individually.

As with Barlow Twins, a batch D is augmented to create two views and passed through an encoder and projector. The embedding Z and Z' are *whitened* using IterNorm [46].

define or elaborate on this

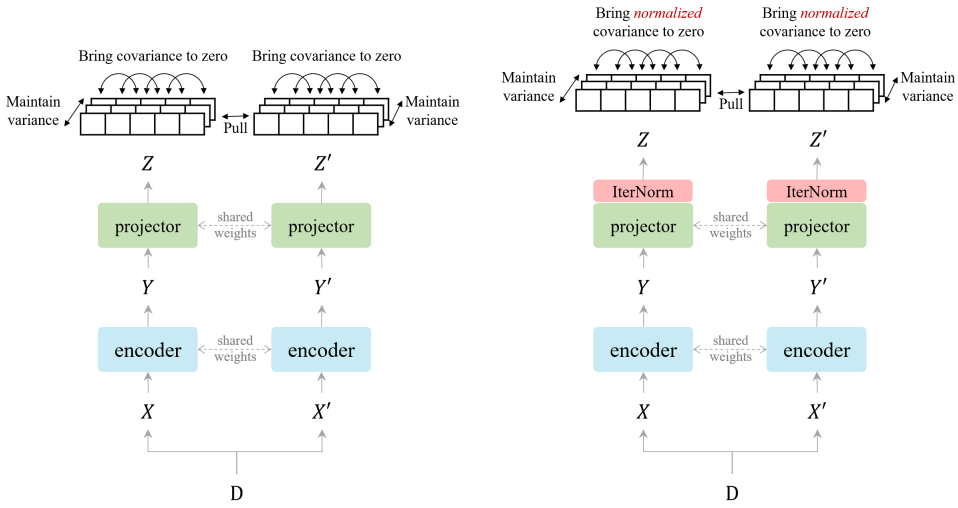


Figure 3.5: Overview of VIbCReg, and comparison with VICReg. Taken with permission from [26]

The loss consists of a similarity loss between the branches, and feature decoration (FD) loss, and a feature component expressiveness (FcE) term at each branch. Input data is processed in batches, with $Z \in \mathbb{R}^{B\times F}$ where B and F denotes the batch and feature sizes, respectively. We denote a row in Z by Z_b and column by Z_f, and similarly for Z'.

The similarity loss is defined as the mean square error between the two embeddings

$$s(Z,Z') = \frac{1}{B}\sum_{b=1}^{B} ||Z_b - Z'_b||_2^2, \tag{3.9}$$

which encourages the embeddings to be similar.

The FcE term, which encourages the variation across a batch to stay at a specified level γ, is defined as

$$v(Z) = \frac{1}{F}\sum_{f=1}^{F} \max(0, \gamma - \sqrt{\text{Var}(Z_f) + \epsilon}), \tag{3.10}$$

where Var() is a variance estimator, γ is a target value for the standard deviation, which both in VIbCReg and VICReg is set to 1, and ϵ is a small scalar to prevent numerical instabilities.

For the FD loss, the embeddings are mean-shifted and normalized along the batch dimension

$$\widehat{Z}_b = \frac{Z_b - \bar{Z}}{||Z_b - \bar{Z}||_2} \text{ where } \bar{Z} = \frac{1}{B}\sum_{b=1}^{B} Z_b, \tag{3.11}$$

giving

$$\widehat{Z} = [\widehat{Z}_1, ..., \widehat{Z}_B]^T. \tag{3.12}$$

The normalized covariance matrix is computed as

$$C(Z) = \frac{1}{B-1}\widehat{Z}^T\widehat{Z}, \tag{3.13}$$

before calculating the mean square across all off-diagonal elements to obtain the FD loss

$$c(Z) = \frac{1}{F^2}\sum_{i \neq j} C(Z)_{ij}^2. \tag{3.14}$$

The total loss is then given by

$$\mathcal{L}_{\text{VIbCReg}} = \lambda s(Z, Z') + \mu[v(Z) + v(Z')] + \nu[c(Z) + c(Z')] \tag{3.15}$$

where λ, μ and ν are hyperparameters that determine the importance of each term. The normalization of the covariance matrix keeps the range of the FD loss small, independent of data, and eases hyperparameter tuning across datasets.

Chapter 4

Methodology

Our work in this thesis builds upon the paper "Vector Quantized Time Series Generation with a Bidirectional Prior Model" [1]. We simplify the model architecture by omitting the high-low frequency split, reducing the model to what they refer to as naive TimeVQVAE in their paper. We expand upon naive TimeVQVAE by integrating a self-supervised learning (SSL) extension.

A schematic figure of our proposed tokenization model is given in Figure 4.1. To improve reconstruction, we add a regularizing term by reconstructing augmented views. We hypothesize that this approach enables the model to generalize better to unseen data by allowing the decoder to "see" the augmented views. Additionally, to improve class separation, we introduce a non-contrastive self-supervised loss. The intuition is that the SSL loss pushes the representations of original and augmented views closer together.

4.1 Proposed model: NC-VQVAE

Our model, termed NC-VQVAE, is a generative time series model that learns expressive discrete latent representations by combining VQVAE [19] with non-contrastive SSL algorithms. NC-VQVAE uses the two-stage modeling approach presented in [1] and can be considered an extension of their "naive TimeVQVAE." Our model primarily extends the tokenization stage, incorporating Barlow Twins [22] and VIbCReg [26] as our non-contrastive SSL methods, while the framework remains flexible. For the second stage, we model the prior using a bidirectional transformer similar to MaskGIT [29].

4.1.1 Stage 1: Tokenization

The architecture of the tokenization model, shown in Figure 4.1, consists of two branches: the original and augmented branch. The model takes a time series x as input and creates an augmented view x'. The original branch follows the naive

TimeVQVAE architecture from [1], while the augmented branch is an autoencoder, constructed by omitting the quantization layer. The views are passed through their respective branches, and we compute the SSL loss derived from the original discrete latent representation z_q and the augmented continuous latent z', before the decoder reconstructs each latent representation.

The SSL loss is calculated by concatenating the global average and max pool of both representations individually and passing the resulting vectors through the projector.

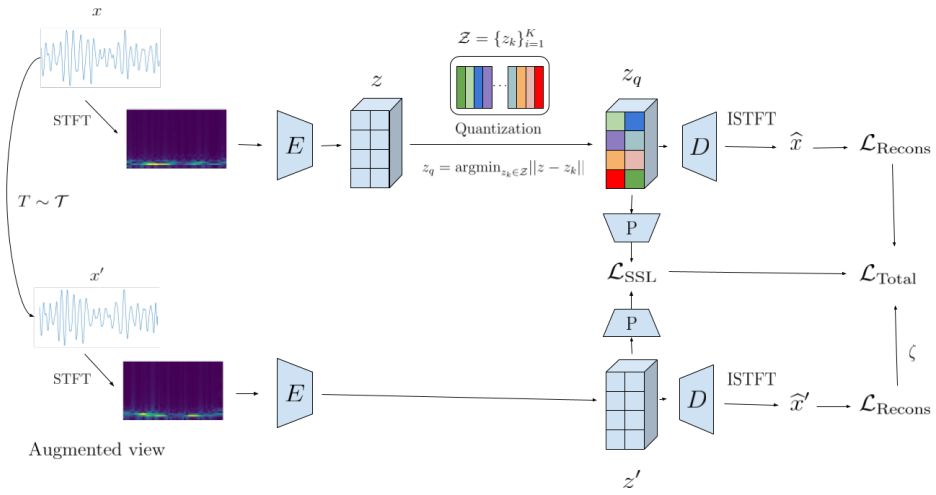


Figure 4.1: Overview of proposed model. NC-VQVAE. Fix index on Z, add weight to arrow from ssl loss to total. fix argmin index thing.

Loss

Our training objective mirrors the training objective from TimeVQVAE in equation 3.6, without the frequency split. Our contribution is the addition of an SLL loss together with a reconstruction loss on the augmented branch.

To refresh the reader's memory, the VQ loss consists of a reconstruction loss and a codebook loss, which is the MSE of continuous and discrete latent representations, along with a commitment loss to prevent the codewords from diverging. In our setup, the codebook loss reduces to

$$\begin{aligned} \mathcal{L}_{\text{codebook}} = &|| \text{sg}[z_q] - z_q||_2^2 \\ &+ \beta ||z_q - \text{sg}[z_q]||_2^2, \end{aligned} \tag{4.1}$$

and the reconstruction loss to

$$\mathcal{L}_{\text{recons}} = ||x - \widehat{x}||_2^2 + ||u - \widehat{u}||_2^2. \tag{4.2}$$

Our VQ loss is then given by

$$\mathcal{L}_{\text{VQ}} = \mathcal{L}_{\text{codebook}} + \mathcal{L}_{\text{recons}}. \tag{4.3}$$

The SSL loss varies depending on the SSL method used. It is calculated on derived values from z_q and z'. We consider Barlow Twins 3.7 and VIbCReg 3.15, both of which utilizes a projector. We apply a global average and max pool operation on both tensors and pass them through the projector before calculating the SSL loss.

The augmented reconstruction loss is simply given as

$$\mathcal{L}'_{\text{recons}} = ||x' - \widehat{x}'||_2^2 + ||u' - \widehat{u}'||_2^2. \tag{4.4}$$

This loss ensures that the encoder and decoder reconstruct the augmented view, which, in conjunction with the SSL loss, influences the codebook to encode information regarding the augmentations. Additionally, it helps prevent the encoder from ignoring reconstruction in favor of the SSL loss. Initial tests showed that omitting the augmentation reconstruction led to severe overfitting.

The total loss is given by

$$\mathcal{L}_{NC-VQVAE} = \mathcal{L}_{VQ} + \eta\mathcal{L}_{\text{SSL}} + \zeta\mathcal{L}'_{\text{recons}}, \tag{4.5}$$

where η and ζ are hyperparameters influencing the importance of each term in the total training objective.

4.1.2 Stage 2: Prior learning

In our model, the input embedding for the bidirectional transformer is initialized with the codebook, which has learned structure from both reconstruction and SSL loss. Instead of introducing an additional masking vector in the embedding matrix, we use the codebook directly and create a separate learnable masking vector to mask the embedded sequences. In order to separate this masking vector, we do the embedding stage before masking, effectively factoring the embedding out of the transformer. This approach ensures that the learning of the masked token embedding is independent of the other embeddings, further leveraging the learned codewords from stage 1 without unnecessary influence. Except for this adjustment, and the possibility of class conditional sampling from TimeVQVAE, our method is equivalent to MaskGITs.

The process for generating samples at inference time follows the same iterative steps as MaskGIT, ensuring robust and high-quality sample generation.

Chapter 5

Experiments

5.1 Implementation details

We follow [1] closely in the encoder/decoder/codebook implementation, and

Time Frequency Modelling

The short time fourier transform (STFT) and its inverse ISTFT are implemented with `torch.stft` and `torch.istft` respectively. We follow [1] and set the main parameter `nfft` to 8, and use default parameters for the rest. This results in a fequency axis with range $[1, 2, 3, 4, 5]$ and half as long time axis.

Encoder and decoder

The same encoder and decoder architecture as in [19] is used and further use the implementation from [47] with adaptations from [1].

The encoder presented in figure 5.1 consists of n downsampling convolutional blocks (`Conv2d - BatchNorm2d - LeakyReLU`), followed by m residual blocks (`LeakyReLU - Conv2d - BatchNorm2d - LeakyReLU - Conv2d`). The downsampling convolutional layers are implemented with parameters: `kernel size=(3,4), stride=(1,2), padding=(1,1)`. This results in downsampling of only the temporal axis, which is downsampled by a factor of 2 for each downsampling block. The residual convolutional layers have parameters: `kernel size=(3,3),stride=(1,1), padding=(1,1)`.

The decoder is implemented similarity with m residual followed n upsampling layers using transposed convolutional blocks with same parameters as in the encoder.

The downsampling rate is determined by 2^n where n is set such that z has width of 32. For more in depth consideration of the detail regarding TimeVQVAE

implementation we refer to Appendix C.3 of [1].

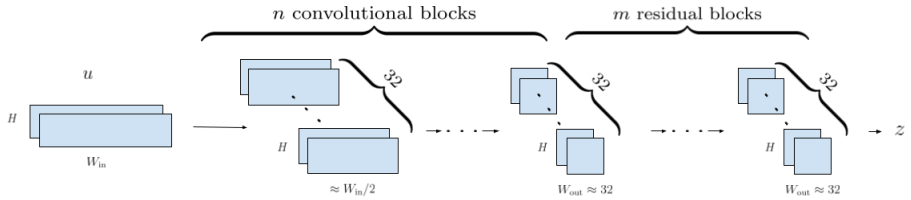


Figure 5.1: Overview of the encoder architecture. The decoder architecture is simply obtained by reversing the arrows and switching out the convolutional block for transposed convolutional blocks.

VQ

Codebook size $= 32$ and dimension $= 64$.
Use exponential moving average with decay 0.9, and commitment loss with weight $\beta = 1$.

Augmentations

window warp: window ratio: 0.4,min window warp: 0.9, max windowwarp: 2.0
amplitude resize: Amp Rrate: 0.2
gaussian noise: gaus mean: 0, gaus std: 0.05
slice and shuffle:

Projector

We follow the implementation from [26] for both Barlow Twins and VIbCReg.
Barlow Twins: Projector's dimension size is set to 4096. λ is set to $5 \cdot 10^3$.
VIbCReg: The dimension of the the projector is set to 4096. λ and μ are both set to 25, while ν is set to 100.

Prior learning

The number of iterations in the iterative decoding algorithm T, is set to 10, following [29]. We too use the cosine as mask scheduling function γ. The implementation is adopted from [1].

Optimizer

The AdamW optimizer with batch sizes for stage1: 128 and stage2: 256, initial learning rate 10^{-3}, cosine learning rate scheduler and weight decay of 10^{-5}. We run 1000 epochs for both stage 1 and 2.

Evaluation

KNN and SVM are implemented using scikit-learn. $K = 5$ in KNN and linear kernel in SVM.
FCN for IS, FID and CAS

5.2 Initial Experimentation and Model Development

The overarching objective in creating our model is to learn more expressive latent representations for better time series generation. We want to keep the reconstruction capabilities of the tokenization model at a high level, where the rationality is that if the tokenization model reconstructs well the latent representations contains all relevant information of the input. We simultaneously want enforce better class separability in the latent representations, as we hypothesize that such additional structure eases/improved learning of the generative model.

During development we encountered several problems:
When we attempted a siamese architecture, with quantization in the augmented branch, and to derive the SSL loss from the discrete representations there were a correlation problem. The codewords were very highly correlated, which resulted from the passing both views through the VQ. $SSL(z_q, z_q')$
In an attempt to solve this we attempted to derive the SSL loss from the continuous latent representations, but the resulting discrete latent representations performed poorly on the downstream classification task. Separability problem: $SSL(z, z')$
The solution was to remove the VQ in the augmented branch and rather derive the SSL loss from z_q and z'. Solution: $SSL(z_q, z')$

Overfitting problem: Using $SG()$ on augmented branch / Not using augRecons

5.3 Main Experiments

5.4 Stage 1

5.4.1 Augmentations

In our experiments we consider three sets of augmentations with different characteristics. They are

- Amplitude Resizing + Window Warp
- Slice and Shuffle
- Gaussian noise

Amplitude Resizing + Window Warp scales in both x and y direction. The window warp has similar qualities to phase shift, but not uniformly and keeps

endpoints fixed. They were considered as the observed conditional distribution in some datasets, such as ShapesALL 5.5, had similar overall shape, but peaked with different amplitude and at different locations. Thus the augmented view had similar characteristics as the conditional distribution of the original view 5.2.

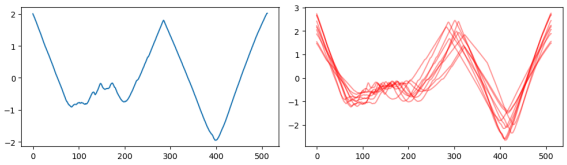


Figure 5.2: ShapesAll: Original (left), augmented (right). 15 instances of Amplitude Resizing + Window Warp applied to the original sample.

Slice and Shuffle crops the time series into four sections and permutes them. For datasets with sharp modularity and few peaks, such as ElectricDevices 5.5, the augmentation provides a view with peaks occurring at timestamps not seen in the training data, which is illustrated in figure 5.3. This could improve the reconstruction on unseen data, as well as encouraging the model to focus more on the existence of a peak rather than its specific location. For some datasets such as FordA 5.5, the semantics of the dataset is preserved under this augmentation, despite their continuous nature.

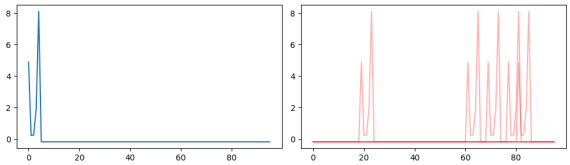


Figure 5.3: ElectricDevices: Original (left), augmented (right). 5 instances of Slice and Shuffle applied to the original sample.

Gaussian noise adds a nose $\epsilon \sim N(0, 0.05)$ to each datapoint in the time series. This introduces, in many cases, a substantial high frequency component as seen in figure 5.4. As the naive VQVAE described in [1] had trouble with reconstruction of HF components, this augmentation could provide more emphasis on these. The reconstruction of the augmented views can too provide more information regarding HF components for the decoder. Of the three augmentations, gaussian noise provides the most predictable augmented views from a numerical standpoint, which might result in a SSL loss which is easier to minimize.

5.4.2 Evaluation

The tokenization model, as we are interested in representation learning, is evaluated on two metrics. Firstly, and most importantly its ability to reconstruct the

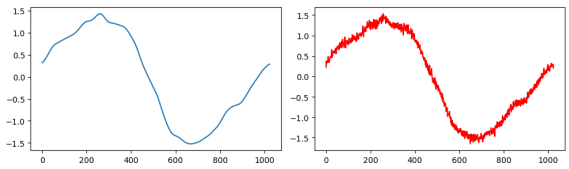


Figure 5.4: StarLightCurves: Original (left), augmented (right). One instance of Gaussian noise applied to the original sample.

input data once compressed into latent space. In essence the latent representation encodes "everything" (important information is preserved) about the original data if the model is able to reconstruct well. Secondly we evaluate linear classifiers on the latent representations, which provides good results if the model learns discriminative features of the different classes and produces an approximately linear separable space. Finally, as the tokenization model is a part of the generative model, the ultimate evaluation metric is the corresponding evaluation of the generative model.

5.5 Stage 2

5.5.1 Evaluation

- **IS**:
- **FID**:
- **CAS**:We evaluate the CAS for TSTR by using the Supervised FCN on all our models considered and compare against the baseline model to investigate the relative performance.
- **Visual inspection**:
- **Token usage**:

5.6 UCR Time Series Classification Archive

The evaluation of our model NC-VQVAE is done on a subset of the UCR Time Series Archive [48]. The UCR archive is a collection of 128 datasets of univariate time series for time series classification. The different datasets in the archive span a wide range characteristics and include among others sensor, device, image-derived and simulated data. Each dataset has a predefined training and test split.

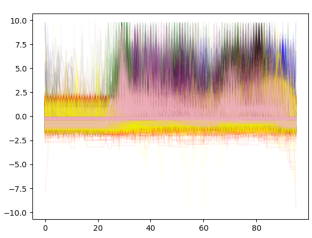
Our subset of the UCR archive is

We choose to test on a subset, rather than on the entire UCR Archive, due to computational limitations as well as to more thoroughly investigate the effect of our models and the role of augmentations. The subset is chosen such that they span a wide range of train set sizes, lengths, classes and type, while the class

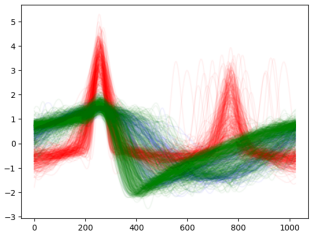
Type	Name	Train	Test	Class	Length
Device	ElectricDevices	8926	7711	7	96
Sensor	FordB	3636	810	2	500
Sensor	FordA	3601	1320	2	500
Sensor	Wafer	1000	6164	2	152
Simulated	TwoPatterns	1000	4000	4	128
Sensor	StarLightCurves	1000	8236	3	1024
Motion	UWaveGestureLibraryAll	896	3582	8	945
ECG	ECG5000	500	4500	5	140
Image	ShapesAll	600	600	60	512
Simulated	Mallat	55	2345	8	1024
Image	Symbols	25	995	6	398
Sensor	SonyAIBORobotSurface2	27	953	2	65
Sensor	SonyAIBORobotSurface1	20	601	2	70

Table 5.1: The subset of the UCR Archive considered for our experiments.

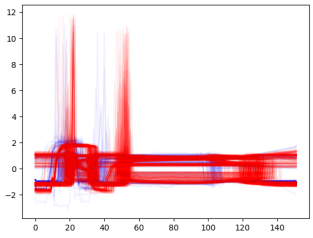
distributions have visually different characteristics which can be seen from table 5.1 and figure 5.5.



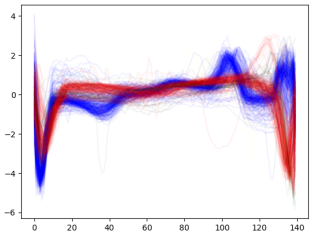
ElectricDevices



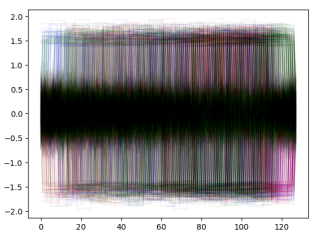
StarLightCurves



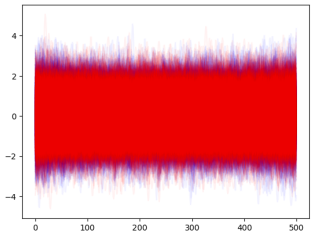
Wafer



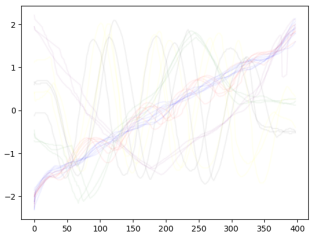
ECG5000



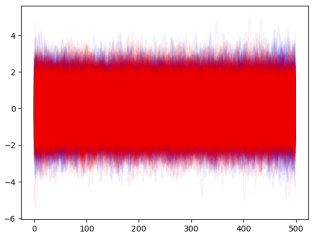
TwoPatterns



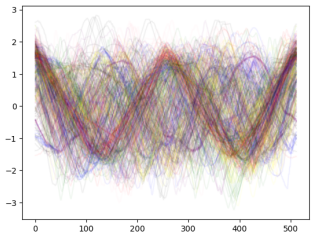
FordA



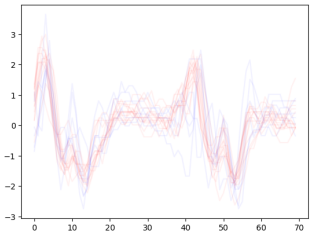
Symbols



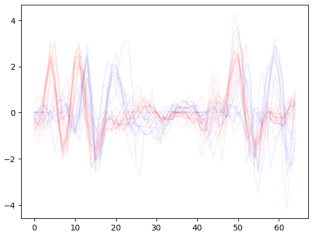
FordB



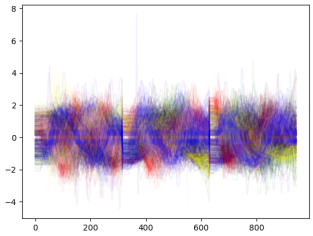
ShapesAll



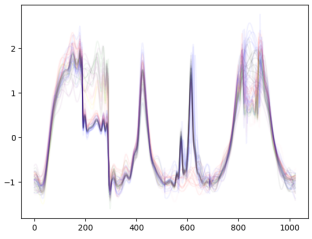
SonyAIBORobot-Surface1



SonyAIBORobot-Surface2



UWaveGestureLib-raryAll



Mallat

Figure 5.5: Our selected subset of the UCR Archive. All time series in the training set are plotted and color coded according to label.

Chapter 6

Results and Discussion

In this section we present the results from both the experiments on stage 1 and 2, while addressing the research questions.

To summarize NC-VQVAE is able to capture the conditional distribution of the data better than naive VQVAE for a wide variety of datasets. For datasets with few training samples, our model can be prone to overfitting. We see our model as a step in the right direction, but further development is needed to ensure better intraclass diversity, possibly through a more refined sampling procedure.

Key takeaways: We are able to simultaneously reconstruct well and significantly improve downstream classification accuracy, which is very interesting from a representation learning perspective. We improve both IS, FID and CAS for most datasets, indicating that the conditional distribution is better captured, as well as the synthetic data being closer to the ground truth. Additionally we see some differences in Barlow Twins and VIbCReg when it comes to sample diversity.

NC-VQVAE is better able to mimic the training data. When data is abundant, then our model better captures the entire distribution, while covering

Some of the issues of TimeVQVAE are still highly relevant, such as the difficulty in modelling data with sharp differences in modularity, such as TwoPatterns and ElectricDevices.

6.1 Stage 1

In this section we present the results of the tokenization model, in particular the reconstruction loss and the downstream classification accuracy. We address research question 1 and 2, if the proposed NC-VQVAE is able to reconstruct on par with the naive VQVAE and if the learned latent representations are more expressive, in the sense that they simultaneously improve the downstream classification

accuracy. We see that some configuration of NC-VQVAE is the top performer on the majority of datasets for both metrics, and provides significant increase in probe accuracy.

6.1.1 Reconstruction

We present top 1 and mean reconstruction loss across the four runs in table 6.2 and table 6.1 respectively.

Mean validation reconstruction error

Dataset	**Baseline**	**SSL Method**					
	Regular	Barlow Twins			VIbCReg		
	None	Warp	Slice	Gauss	Warp	Slice	Gauss
FordA	0.217	0.127	0.134	**0.108**	0.173	0.169	0.203
ElectricDevices	**0.041**	0.067	0.044	0.049	0.105	0.042	0.049
StarLightCurves	**0.032**	0.042	0.069	0.071	0.052	0.050	0.068
Wafer	0.044	0.037	0.048	0.049	**0.035**	0.042	0.039
ECG5000	**0.048**	0.083	0.170	0.104	0.093	0.205	0.064
TwoPatterns	0.197	0.201	**0.184**	0.230	0.214	0.186	0.207
UWaveGestureLibraryAll	0.190	**0.172**	0.190	0.245	0.189	0.178	0.237
FordB	0.150	0.115	0.122	0.123	**0.114**	0.121	0.142
ShapesAll	**0.045**	0.056	0.066	0.102	0.064	0.069	0.073
SonyAIBORobotSurface1	0.402	0.509	0.494	0.491	**0.360**	0.363	0.418
SonyAIBORobotSurface2	0.623	0.622	0.618	0.640	0.487	**0.454**	0.589
Symbols	0.110	0.143	0.134	0.173	0.078	**0.067**	0.105
Mallat	0.066	0.081	0.091	0.096	0.066	0.067	**0.060**

Table 6.1: Mean validation reconstruction error across all 13 datasets. Results are averaged over four runs.

Top 1 validation reconstruction error

Dataset	**Baseline**	**SSL Method**					
	Regular	Barlow Twins			VIbCReg		
	None	Warp	Slice	Gauss	Warp	Slice	Gauss
FordA	0.158	0.108	0.111	**0.087**	0.130	0.134	0.113
ElectricDevices	0.036	0.060	0.034	0.043	0.092	**0.031**	0.045
StarLightCurves	**0.026**	0.037	0.057	0.055	0.043	0.048	0.065
Wafer	0.038	0.031	0.045	0.043	**0.027**	0.031	0.038
ECG5000	**0.044**	0.069	0.156	0.084	0.080	0.181	0.056
TwoPatterns	0.181	0.184	**0.169**	0.208	0.200	0.172	0.185
UWaveGestureLibraryAll	0.159	**0.145**	0.167	0.201	0.155	0.169	0.233
FordB	0.117	0.094	0.090	0.103	**0.082**	0.094	0.102
ShapesAll	**0.035**	0.043	0.046	0.092	0.061	0.063	0.067
SonyAIBORobotSurface1	0.381	0.473	0.472	0.465	0.329	**0.328**	0.408
SonyAIBORobotSurface2	0.513	0.577	0.536	0.588	0.444	**0.414**	0.470
Symbols	0.088	0.111	0.122	0.150	0.062	**0.059**	0.090
Mallat	0.061	0.075	0.076	0.088	0.059	0.059	**0.057**

Table 6.2: Top 1 validation reconstruction error across all 13 datasets. Lowest value of the four runs for each model is selected.

From the tables we see that NC-VQVAE reconstructs on par with the baseline model, and that some configuration outperforms the naive VQVAE on mean reconstruction loss for 9 out of 13 datasets. In figure 6.1 observe that the difference in reconstruction loss is small for most datasets, both across SSL methods and augmentations. The use of gaussian augmentation introduces less of a regularizing effect compared to the two others, with the exception of Slice and shuffle on ECG5000. These results show that the introduction of a non contrastive loss does not hurt the reconstruction capabilities of our model compared to naive VQVAE.

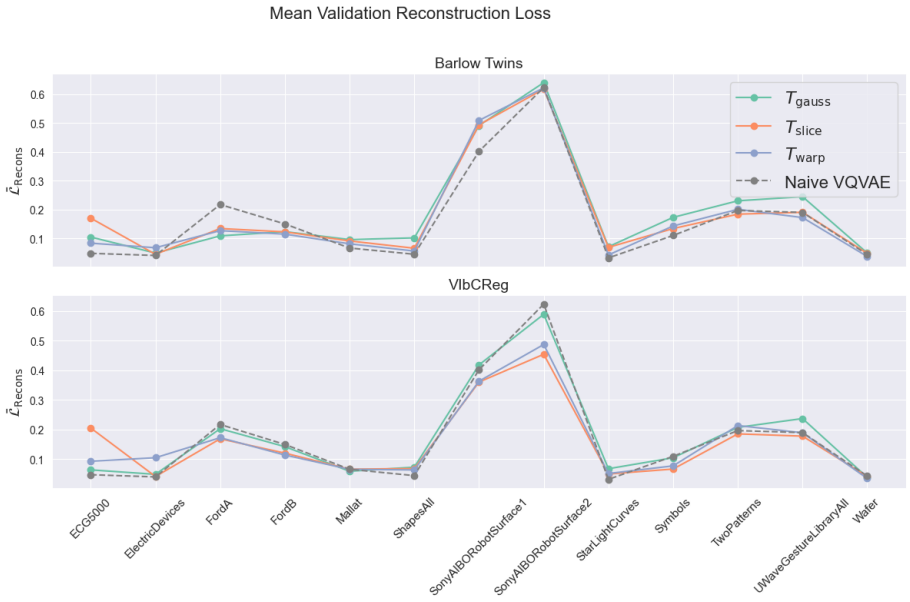


Figure 6.1: Mean validation reconstruction loss for the two models, compared to naive VQVAE

The right configuration for NC-VQVAE acts as a regularizer, in figure 6.2 we see how the validation reconstruction loss developes on FordA during training.

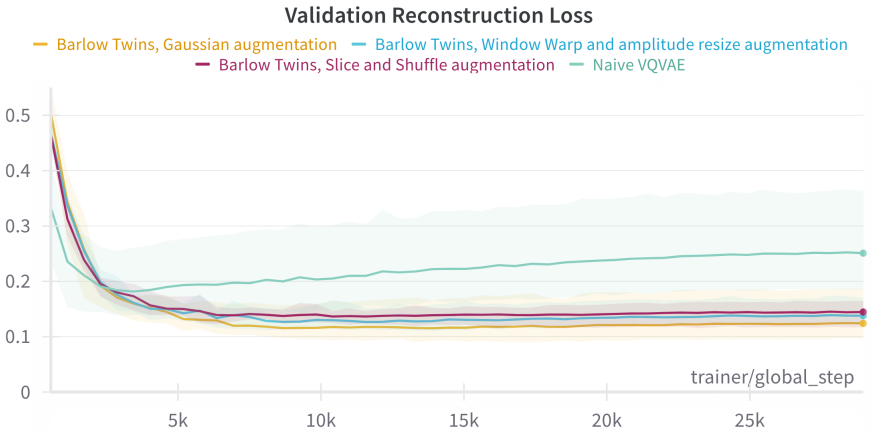


Figure 6.2: Development of the validation reconstruction loss for FordA during training. Averaged across all four runs.

6.1.2 Classification

We present the mean and max downstream classification accuracy in table 6.3 and 6.4 respectively.

Mean linear probe accuracy

Dataset	Baseline		SSL Method											
	Regular		Barlow Twins						VIbCReg					
	None		Warp		Slice		Gauss		Warp		Slice		Gauss	
	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM
FordA	0.70	0.74	0.83	0.84	**0.91**	**0.89**	0.80	0.83	0.80	0.74	0.87	0.86	0.76	0.78
ElectricDevices	0.35	0.41	0.35	**0.44**	0.38	0.41	**0.40**	0.42	0.33	0.38	0.36	0.39	0.39	0.43
StarLightCurves	0.87	0.89	0.93	0.93	**0.94**	**0.94**	0.88	0.88	0.92	**0.94**	0.91	0.93	0.89	0.89
Wafer	0.93	0.89	0.96	**0.94**	0.96	**0.94**	0.96	0.93	**0.97**	0.94	0.96	0.92	**0.97**	0.92
ECG5000	0.80	0.83	0.85	0.81	**0.88**	0.84	0.86	**0.84**	0.86	0.82	**0.88**	**0.84**	0.84	0.82
TwoPatterns	0.34	0.53	**0.69**	**0.91**	0.66	0.82	0.47	0.71	0.64	0.90	0.68	0.80	0.55	0.72
UWaveGestureLibraryAll	0.31	0.40	**0.62**	0.70	0.56	0.63	0.40	0.54	**0.62**	**0.73**	0.55	0.66	0.44	0.55
FordB	0.58	0.60	0.64	0.67	0.74	0.76	0.64	0.68	0.63	0.64	0.70	0.70	0.61	0.64
ShapesAll	0.29	0.30	0.49	0.55	0.53	**0.60**	0.40	0.48	0.48	0.56	**0.54**	**0.60**	0.40	0.46
SonyAIBORobotSurface1	0.56	0.68	0.54	0.70	**0.61**	**0.74**	0.53	0.70	0.48	**0.74**	0.58	0.71	0.54	0.69
SonyAIBORobotSurface2	**0.81**	**0.86**	0.77	0.79	0.80	0.80	0.80	0.81	0.77	0.85	0.80	0.85	0.80	0.85
Symbols	0.50	0.60	**0.59**	0.60	0.50	**0.66**	**0.59**	**0.66**	0.45	0.61	0.42	0.62	0.43	0.63
Mallat	0.63	0.77	0.72	0.81	0.76	0.83	0.68	0.78	**0.79**	**0.87**	0.77	0.85	0.69	**0.86**

Table 6.3: Summary of mean linear probe accuracy by SSL Method and Augmentation. Average across 4 seeds. Best result for KNN and SVM are highlighted in bold.

Top 1 linear probe accuracy

Dataset	Baseline		SSL Method											
	Regular		Barlow Twins						VIbCReg					
	None		Warp		Slice		Gauss		Warp		Slice		Gauss	
	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM
FordA	0.75	0.78	0.84	0.88	**0.93**	**0.92**	0.85	0.87	0.81	0.77	0.88	0.90	0.86	0.85
ElectricDevices	0.35	0.43	0.36	0.45	0.39	0.43	**0.45**	**0.46**	0.34	0.42	0.39	0.42	0.42	0.45
StarLightCurves	0.89	0.91	0.94	0.95	**0.96**	**0.96**	0.90	0.91	0.95	0.95	0.93	0.95	0.90	0.90
Wafer	0.94	0.89	**0.97**	**0.95**	**0.97**	**0.95**	**0.97**	0.93	**0.97**	**0.95**	**0.97**	**0.95**	**0.97**	0.94
ECG5000	0.83	0.84	0.88	0.86	**0.90**	**0.88**	**0.90**	**0.88**	0.88	0.85	0.89	0.86	0.86	0.85
TwoPatterns	0.37	0.62	**0.75**	**0.96**	0.68	0.85	0.55	0.75	0.70	0.92	0.71	0.81	0.63	0.76
UWaveGestureLibraryAll	0.34	0.43	**0.67**	0.74	0.60	0.67	0.43	0.54	**0.67**	**0.76**	0.58	0.67	0.48	0.58
FordB	0.60	0.63	0.67	0.71	**0.76**	**0.80**	0.69	0.74	0.67	0.65	0.74	0.77	0.63	0.68
ShapesAll	0.33	0.34	0.53	0.59	**0.59**	**0.65**	0.44	0.50	0.50	0.56	0.57	0.63	0.44	0.48
SonyAIBORobotSurface1	0.67	**0.80**	0.61	0.77	0.76	**0.80**	0.60	0.74	0.51	0.79	0.63	0.75	0.63	0.75
SonyAIBORobotSurface2	**0.84**	**0.89**	0.80	0.86	0.82	0.84	0.83	0.82	0.81	0.88	0.81	0.88	0.83	0.87
Symbols	0.56	0.66	0.65	0.69	0.55	0.73	**0.64**	**0.71**	0.51	0.65	0.45	0.67	0.46	0.69
Mallat	0.54	0.88	0.57	0.87	0.74	0.89	0.66	0.80	0.74	**0.92**	0.72	0.88	0.62	**0.90**

Table 6.4: Summary of max linear probe accuracy by SSL Method and Augmentation. Maximum value across 4 seeds. Best result for KNN and SVM are highlighted in bold.

From the tables we see a significant improvement in probe accuracy with NC-VQVAE compared to naive VQVAE. Some configuration is best on 12 out of 13 datasets, while the one where our model falls short, the difference is one percent for both svm and knn. The largest differences are seen on FordA, FordB, Mallat, ShapesAll, TwoPatterns and UWaveGestureLibraryAll.

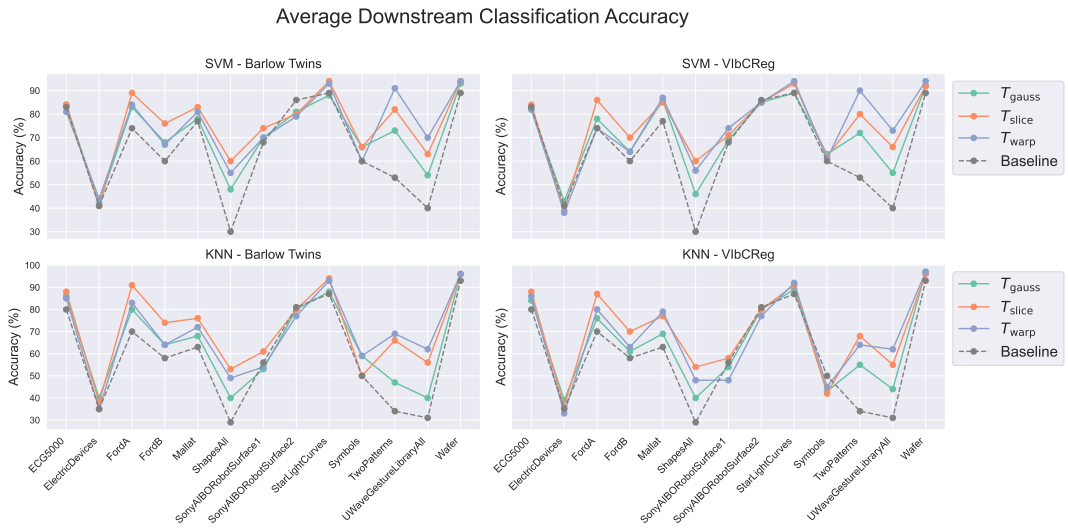


Figure 6.3: Mean probe accuracies.

The NC-VQVAE model is able to reconstruct on par with naive VQVAE, and in some cases improve the reconstruction loss, while significantly improving the probe accuracy for the discrete latent representations for most datasets. The NC-VQVAE produces representations that separates classes more effectively, and could in turn encode more class specific information.

6.1.3 Losses

TODO: How does the minimization of diffrerent losses influence FID/IS?

We investigate some trends in the development of different loss terms in this section. To summarize, using VIbCReg results in more easily minimizable losses compared to Barlow Twins, and the Gaussian augmentation results in significantly easier minimization of the SSL loss as well as reducing the VQ loss.

In figure 6.4 we observe the typical pattern of the SSL loss during training.

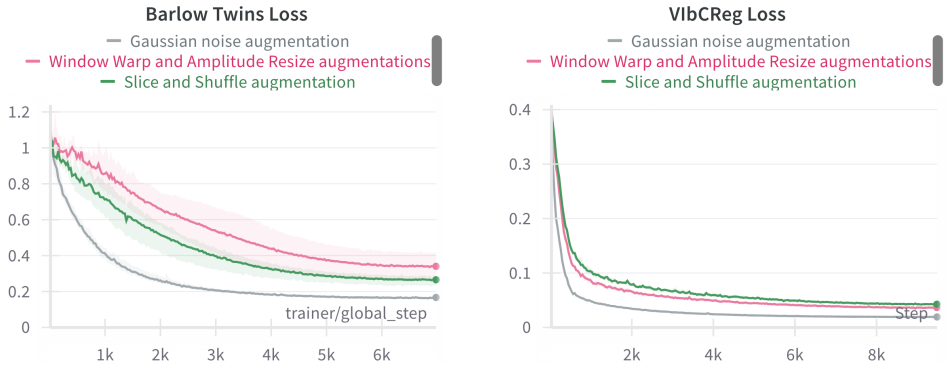


Figure 6.4: SSL loss during training on UWaveGestureLibraryAll. Averaged across four runs. The pattern shown here is typical across datasets. The Gaussian noise augmentation leads to an easier minimization, while the two others are relatively similar with the occasional dataset where one or the other is minimized the most. In general the VIbCReg loss is minimized faster than Barlow Twins.

We see that the Gaussian augmentation results in a SSL loss which is easier to minimize, which might be attributed to the fact that it affects the samples in a more predictable way. We too see that the VIbCReg loss decreases more rapidly than the Barlow Twins loss. Both observations are too seen across datasets. Previously in figure 6.3 we have seen that the gaussian augmentation often resulted in lower probe accuracy than the other two. In figure 6.5 we see that on the datasets with a significant increase in probe accuracy, the augmentations that result in a more challenging SSL loss typically has higher downstream classification accuracy. We also see that for a specific augmentation, the pattern is rather flat, indicating that the particular augmentation plays the most prominent role in probe accuracy. The SSL loss for a specific augmentation varies very little compared to probe accuracy. This could be due to the fact that augmentations such as Slice and Shuffle often alter the semantics of the sample and could make the SSL loss push latent representation to more unpopulated areas of the latent space, making it easier to classify, but harder to minimize.

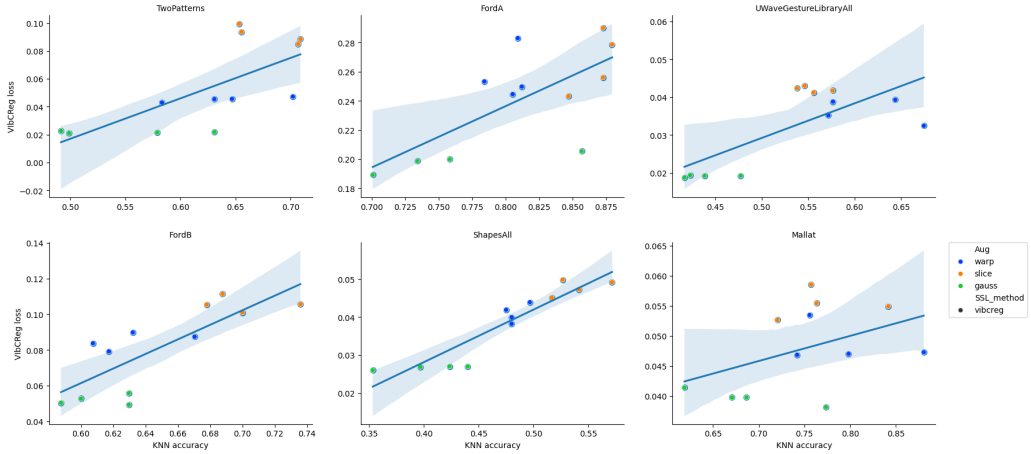


Figure 6.5: KNN accuracy plotted against VIbCReg loss. Each point correspond to a single run of the model. Similar tendency is shown for Barlow Twins.

The training reconstruction losses are heavily minimized, both across models and augmentations. The only consistent noticeable difference is the augmented reconstruction loss, where models using Slice and Shuffle have a slightly higher loss. The differences in VQ loss for the different models is mainly due to the codebook, where we too observe that VIbCReg minimizes more effectively than Barlow Twins, and again that gaussian augmentations results in the hardest minimization followed by Window Warp and then Slice and Shuffle. Both VIbCReg and Barlow Twins with Gaussian augmentation routinely perform on par with naive VQVAE in terms of VQ loss during training. The minimization of the codebook loss indicates that the encoder is properly aligned with the discrete latent codes. We hypothesize that when the SSL loss is not properly minimized, the encoder must adjust its weights more throughout training which keeps the encoder outputs and the discrete codes from aligning completely.

TODO: What effect might this have on stage 2??

6.1.4 Visual inspection

After training the stage 1 model, we map the training and test data to discrete latent representations using the learned encoder and codebook. The representations are then global average pooled before being mapped onto the plane by TSNE showed in figure 6.6.

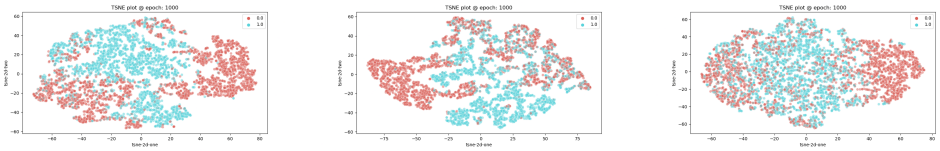


Figure 6.6: TSNE plots of FordA. Barlow (left) and VIbCReg (center) with Slice and Shuffle, naive VQVAE (right). Best performing model in terms of KNN accuracy is chosen.

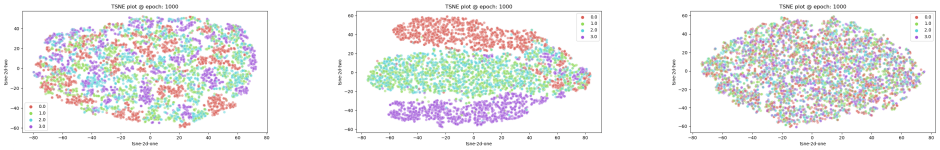


Figure 6.7: TSNE plot of discrete latent representations from VIbCReg with Slice and Shuffle (left), Barlow Twins with Window Warp and Amplitude Resize (center) and naive VQVAE (right). Dataset is TwoPatterns. The latent space is significantly more structured with NC-VQVAE.

TODO: PCA-TSNE-UMAP plots

TODO: Latent space plots

Illustrate the clustering in latent space

6.2 Stage 2

6.2.1 Generative quality

The generative quality of or models are evaluated according to FID, IS and CAS. We present the top 1 results in table 6.5, and the mean score across the four runs in table 6.6. From the tables we see that our model produces better IS score for 12 out of 13 datates, and better FID for 10 out of 13.

Top 1 FID and IS

Dataset	Baseline		SSL Method											
	Regular		Barlow Twins						VIbCReg					
	None		Warp		Slice		Gauss		Warp		Slice		Gauss	
	FID↓	IS↑	FID↓	IS↑	FID↓	IS↑	FID↓	IS↑	FID↓	IS↑	FID↓	IS ↑	FID↓	IS↑
FordA	2.59	1.30	1.93	**1.51**	2.13	1.48	1.80	**1.51**	2.83	1.38	2.50	1.43	**1.66**	1.41
ElectricDevices	12.05	3.97	11.82	4.20	**8.91**	4.07	9.89	3.86	12.38	**4.23**	11.08	3.94	13.96	3.71
StarLightCurves	**0.74**	1.99	0.89	**2.43**	1.50	2.36	0.75	2.39	0.92	2.39	0.85	**2.40**	0.79	2.26
Wafer	5.27	**1.39**	3.31	1.29	3.82	**1.26**	2.77	1.35	3.33	1.29	3.60	1.30	**2.52**	1.34
ECG5000	1.56	2.01	2.43	2.02	2.27	2.00	2.15	2.02	2.15	**2.03**	2.21	2.00	**1.52**	2.02
TwoPatterns	3.63	2.47	3.59	2.65	2.74	2.73	**2.24**	2.70	3.45	2.64	2.90	2.70	**2.19**	**2.77**
UWaveGestureLibraryAll	8.16	2.24	6.45	2.94	**6.26**	**3.13**	7.31	2.79	6.52	2.99	6.33	3.06	7.09	2.79
FordB	2.92	1.52	2.10	1.52	2.44	1.61	1.93	**1.67**	1.76	1.65	2.12	1.64	**1.66**	1.52
ShapesAll	**21.35**	4.32	35.89	**5.22**	29.61	5.16	27.91	4.83	30.03	4.95	31.59	4.92	27.20	4.94
SonyAIBORobotSurface1	18.21	1.27	26.20	1.32	28.90	1.28	21.63	1.32	21.98	1.36	25.20	1.38	**15.73**	**1.55**
SonyAIBORobotSurface2	3.85	1.69	2.50	1.82	3.34	1.79	**0.82**	1.82	2.61	1.81	2.75	1.83	1.24	**1.84**
Symbols	8.50	2.43	5.86	3.20	7.39	2.82	**4.25**	**3.50**	6.78	3.39	7.21	3.23	8.21	3.30
Mallat	**1.31**	3.41	2.01	3.67	2.24	3.72	1.85	3.66	1.87	3.34	2.30	3.05	**1.31**	**3.92**

Table 6.5: Summary of FID and IS scores by SSL Method and Augmentation. Best achieved results are highlighted in bold

Mean FID and IS

Dataset	Baseline		SSL Method											
	Regular		Barlow Twins						VIbCReg					
	None		Warp		Slice		Gauss		Warp		Slice		Gauss	
	FID↓	IS↑	FID↓	IS↑	FID↓	IS↑	FID↓	IS↑	FID↓	IS↑	FID↓	IS ↑	FID↓	IS↑
FordA	5.15	1.16	2.59	1.41	2.36	**1.45**	**2.28**	**1.45**	3.01	1.34	2.90	1.41	3.73	1.29
ElectricDevices	13.48	3.75	16.51	3.95	**10.20**	3.93	11.54	3.75	13.99	**4.17**	11.82	3.85	15.20	3.55
StarLightCurves	**1.01**	1.93	1.29	2.35	1.91	2.32	1.08	2.25	1.07	2.35	1.19	**2.36**	1.05	2.22
Wafer	5.72	**1.33**	3.70	1.25	4.20	1.24	2.85	1.31	3.67	1.26	3.86	1.26	**2.84**	1.31
ECG5000	**1.62**	1.94	2.61	**2.00**	2.56	1.98	2.47	**2.00**	2.60	1.99	2.39	**2.00**	1.76	1.99
TwoPatterns	4.04	2.41	4.00	2.54	2.96	2.66	**2.44**	**2.67**	4.05	2.56	3.15	2.66	2.62	**2.67**
UWaveGestureLibraryAll	8.48	2.13	6.77	2.86	6.64	2.96	7.35	2.73	6.80	2.91	**6.49**	**2.99**	7.34	2.72
FordB	4.05	1.28	2.66	1.48	3.49	1.50	2.88	**1.52**	**2.49**	1.48	3.07	1.51	3.04	1.31
ShapesAll	**27.64**	4.22	38.22	5.07	32.54	**5.04**	32.25	4.56	36.59	4.72	35.79	4.76	31.56	4.71
SonyAIBORobotSurface1	23.71	1.20	30.65	1.22	31.97	1.21	25.29	1.28	26.11	1.32	28.20	1.32	**18.61**	**1.44**
SonyAIBORobotSurface2	5.42	1.62	3.35	1.77	4.41	1.74	**1.78**	**1.81**	4.43	1.74	3.32	1.79	2.36	1.79
Symbols	13.62	1.99	9.78	2.92	9.78	2.67	8.61	3.14	8.84	3.20	9.74	3.03	**8.58**	**3.24**
Mallat	2.09	3.01	2.54	3.29	3.68	2.94	2.12	3.53	2.11	3.18	2.40	2.96	**1.65**	**3.72**

Table 6.6: Summary of FID and IS scores by SSL Method and Augmentation. Best mean achieved FID and IS are highlighted in bold

In figure 6.8 we get a better overview of the results, and observe that both Barlow Twins and VIbCReg produces better samples than the naive VQVAE in terms of FID and IS. Additionally we see that the use of gaussian augmentation results in the largest improvements for most datasets. The high IS scores indicate that NC-VQVAE captures the conditional distributions better than naive VQVAE in many datasets. This will be explored further in section 6.2.5. The improved FID scores indicates that the synthetic samples more closely resemble the test data. The moderate decrease in FID, compared to the increase in IS, could indicate that the generated samples does not generalize too well to the test data. The discrete latent representations from NC-VQVAE provides more information regarding the classes, as we saw from the improved downstream classification accuracy in stage 1. This additional class specific information seems to assist the prior learning in capturing class conditional distributions.

Figure 6.8: Mean FID and IS scores for Barlow Twins and VIbCReg VQVAE. FID is plotted on a log scale because of the large difference in values across datasets.

It is worth mentioning that the FID and IS score is calculated using the SupervisedFCN, which is also trained on the UCR Archive. Thus, the FID and IS scores could have a bias toward samples that mimic the training data.

6.2.2 Class conditional sampling

TODO: How do we calculate the CAS? How many samples etc.

We present the mean CAS for all models across datasets in table 6.7.

Mean CAS

Dataset	**Baseline**	**SSL Method**					
	Regular	Barlow Twins			VIbCReg		
	None	Warp	Slice	Gauss	Warp	Slice	Gauss
FordA	0.864	0.884	**0.902**	0.878	0.864	0.895	0.870
ElectricDevices	0.614	0.588	0.607	0.599	**0.618**	0.610	0.594
StarLightCurves	0.960	0.953	0.955	**0.965**	0.962	0.954	0.964
Wafer	0.976	0.977	0.978	0.968	0.979	0.976	**0.984**
ECG5000	0.866	0.881	0.863	0.880	0.877	0.892	**0.910**
TwoPatterns	0.808	0.770	0.788	**0.847**	0.715	0.781	0.846
UWaveGestureLibraryAll	0.333	0.300	0.367	0.313	0.360	**0.401**	0.383
FordB	0.725	0.748	**0.756**	0.741	0.750	0.738	0.750
ShapesAll	0.361	0.344	0.329	**0.420**	0.379	0.367	0.404
SonyAIBORobotSurface1	0.975	0.933	0.957	0.979	0.982	0.976	**0.985**
SonyAIBORobotSurface2	0.929	0.956	0.951	0.969	0.960	**0.970**	0.964
Symbols	0.956	0.929	0.930	0.930	0.969	**0.974**	0.963
Mallat	0.471	0.642	0.563	0.661	0.827	0.876	**0.908**

Table 6.7: Mean CAS score across datasets. Results averaged across four runs.

We see that some configuration of NC-VQVAE outperforms the naive VQVAE on all datasets, as well as VIbCReg with gaussian augmentation outperforming the baseline on 12 out of 13, where the one dataset where it falls short its within one percent. In general we observe that NC-VQVAE performs well across all datasets, and in particular with gaussian augmentation. The dataset where we see the most dramatic increase is Mallat, with an improvement of 0.437. This particular case will be investigated in section 6.2.5.

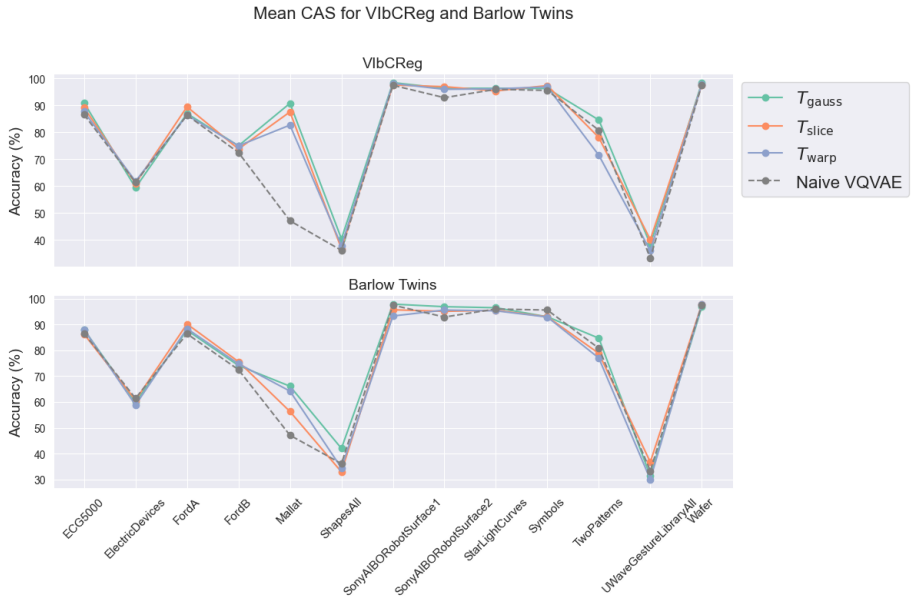


Figure 6.9: Mean CAS across all datasets.

6.2.3 Prior loss

Mention that during experiments with our stage 2 modification, embed / fine-tune, we observed that the val prior loss with our modification was higher, but with similar shape as without. If we had time and computational resources to re-run the experiments, then we would omit the stage 2 modification. The FID/IS in our main experiments are in many cases better than baseline VQVAE, despite higher val prior loss.

6.2.4 Token usage

We say memorization/overfitting when the selected probabilities are mainly >0.9.

wafer: Barlow and VIbCReg are more certain of tokens than naive. Often at time T=3 the main proportion of selected samples have probability >0.9. Barlow to a greater degree than VIbCReg.

ShapesAll: Barlow a bit more uncertain than VIbCReg. For a seed they both collapse and basically sample tokens with probability 1 from T=1.

Sony2: Barlow is much more certain earlier for several models. Both are significantly more certain than naive.

Mallat: All models overfit quite hard. VIbCReg and Barlow has some more variability, vibcreg best of SSL.

FordB: more healthy distributions. VIbCReg a bit more certain, in a good way i think.

ECG5000: For several models barlow and vib overfits hard. Naive has healty distributions, mostly.

TwoPatterns: Naive consistently uncertain. VIbCReg and Barlow develops similarly as T increases, and looks very good. This is a prime example of what i consider good.

UWave: VIbCReg and Barlow has several cases of severe overfitting, . Naive looks healthy.

Symbols: VIbCReg has significantly more diversity than barlow. Still overfits quite a bit. Naive overfits in some cases, but generally healthier.

ElectricDevices: Similar behavior. After T=4 almost certain.

StarLightCurves: Overfitting in some cases. Otherwise a more healty distribution than naive.

Sony1: Barlow overfits more than VIbCReg . Both have some quite extreme cases. Warp produces the best distributions.

FordA: One model each with some overfitting (both gaussian).

Note: Does overfitting etc, happen mostly for Gauss? It is the case for StarLightCurves.

Include something on the differences in sampling/token usage between naive VQVAE and NC-VQVAE. NC-VQVAE has a tendency to be more certain of tokens selected. For small datasets such as Mallat, the certainty is close to 1 for most sampled tokens.

TODO: Investigate this further. Compare/relate the selected probability histograms with token usage histograms / perplexity

Would be interesting to investigate different values for T in maskgit iterative sampling.

Higher masking ratios during training etc.

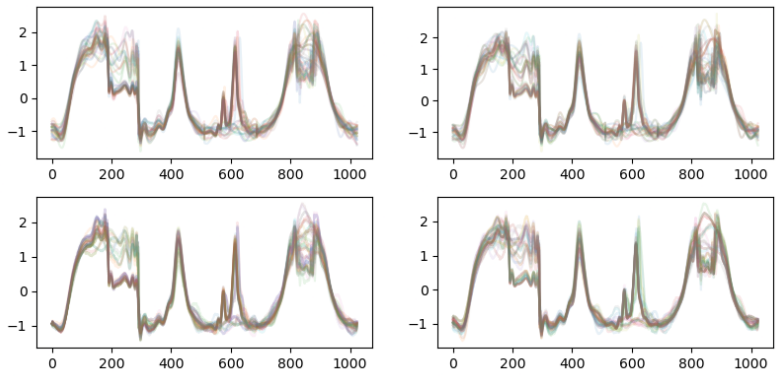
6.2.5 Visual inspection

Simple patterns, such as sinusoids, are very easily captured (Symbols/ShapesAll). Sharp changes in modularity and frequency are much harder(TwoPatterns).

Datasets or classes with very few samples might be mimicked/overfitted.

Different samples sizes, how easy patterns are etc should be considered when setting nr of epochs, and T in maskgit (lower for simple patterns).

Mallat



Naive VQVAE.
Real (Top) vs synthetic (Bottom)

VIbCReg with Gaussian augmentation. Real (Top) vs synthetic (Bottom)

Figure 6.10: Difference in synthetic samples between the top performing naive VQVAE and VIbCReg VQVAE with Gaussian augmentation. The VIbCReg VQVAE samples are more varied in the first 300 timesteps, which from figure 5.5 contains much class specific information.

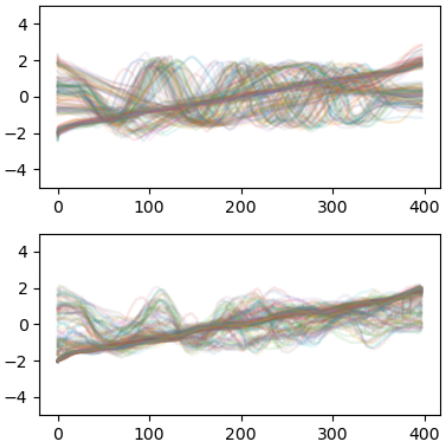
Symbols

Naive VQVAE: Does not capture the entire underlying distribution, some classes are not represented/not recognizable. Global consistency for the sinusoids are poor, particularly towards the end.
VIbCReg: Good mode coverage, but underrepresents the sinusoids or lacks diversity in each class.

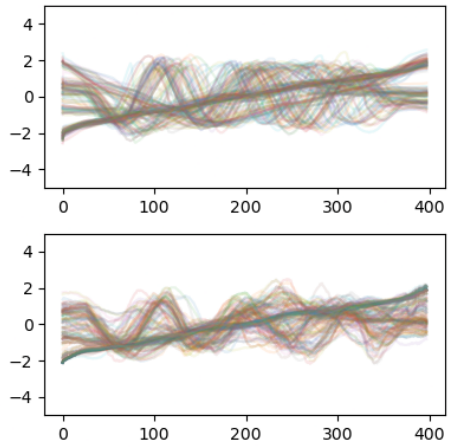
Barlow Twins: windowwarp: little variability in sinusoids, could it be that the ssl loss makes these too close in latent space?

Does the high IS scores correlate with good mode covarage? For symbols our model cover the modes much better than naive. But produces many very similar samples. Does this have something to do with the selected token histograms. Seems like our models select tokens with higher probability, sometimes much higher!

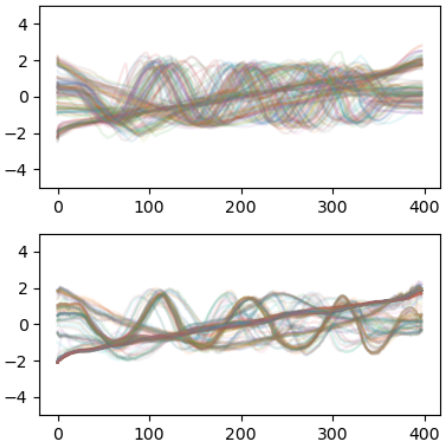
IS has a flaw in that it does not take intraclass diversity into account. Thus a model which generates the mode at each class will get a high IS score. Thus it can give high scores to models that overfit.



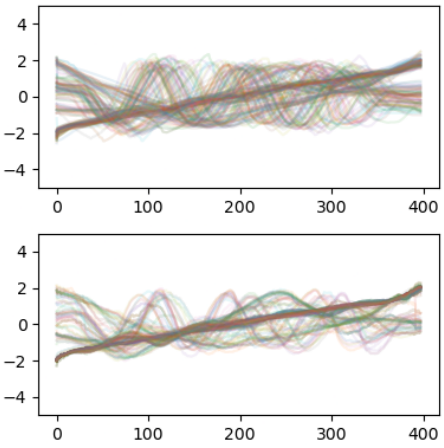
Collapse of global consistency for Naive VQVAE



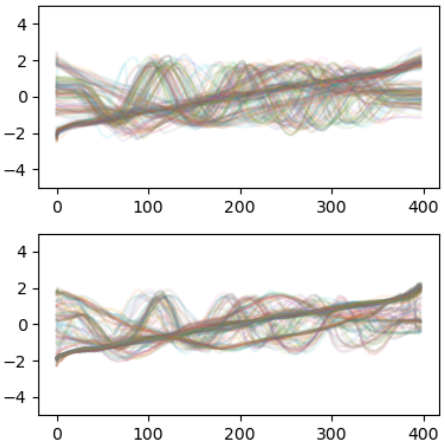
Best Naive VQVAE in terms of FID and IS



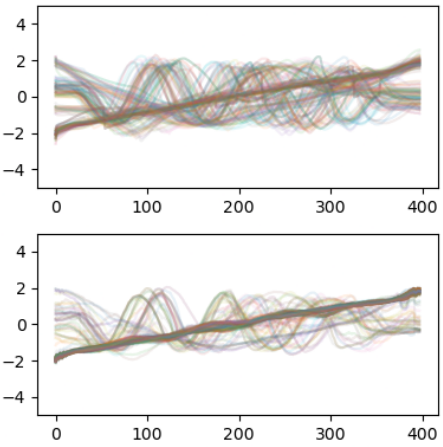
Barlow Twins with Gaussian augmentation, the best performing BT model in terms of FID and IS



Barlow Twins with Gaussian augmentation, the wors performing BT model in terms of FID.



VIbCReg with Slice and Shuffle augmentation, the best performing BT model in terms of FID. Among top performers in terms of IS



VIbCReg with Slice and Shuffle augmentation, the worst performing VIbCReg model in terms of FID and IS.

Figure 6.11: Visual inspection of the Symbols dataset

Sony2 is interesting

ShapesALL

A LOT BETTER class conditional sampling!
Generated vs real.

UWaveGestureLibraryAll

Our model follows the training data quite closely. It might be susceptible to generate outlier data.

The bias introduced by augmentation

6.3 The influence of stage 1 on stage 2

The best performing datasets in terms of probe accuracies: "FordA", "FordB", "Mallat", "ShapesAll", "TwoPatterns", "UWaveGestureLibraryAll"

Relationship between reconstruction in stage 1 and FID/IS/CAS: Does better reconstruction capabilities in stage 1 improve the generative model?

Relationship between probes in stage 1 and FID/IS/CAS: Does better probe accuracies (class separation) in stage 1 improve the generative model?

How does the best performing models from stage 1 transfer to stage 2?

Look at FordA, FordB, Mallat, ShapesALL, TwoPatterns and UWaveGestureLibraryAll. The datasets where probe accuracies are good compared to baseline. Slice is aug with best performance overall on these datasets.

6.4 Differences in Barlow Twins and VIbCReg

VIbCReg seems to keep the variablility in the conditional distribution a bit better than Barlow Twins. Can it be attributed to the variance term in VIbCReg?
TSNE and PCA of Mallat.

6.4.1 Overfitting problem

6.4.2 Thoughts

Better inception score and CAS of our models indicate that the class separability learned in latent space makes the conditional distributions more distinct easier to classify. The FID is variable, but in many cases better, which indicated that the

generative distributions are closer to the ground truth.

Gaussian noise aug seems to result in a lot easier the BT/VIbCReg loss to minimize.
Slice and shuffle is harder to minimize, but could seem to push representations for different classes further apart resulting in better linear probes.

Talk about the difficulty/ease in minimizing the SSL loss for the different augmentations. Does this affect linear probes / reconstruction / FID / IS / Prior loss

For datasets of smaller size with classes of different characteristics (a clear distributional difference in visual inspection [Sony2 and Symbols]) NC-VQVAE seems to perform better both in terms of FID and IS.

The biases introduced by augmentations in stage 1 seems to be included in the generated samples to some degree. In particular datasets with high frequency components, when applying Gaussian noise (easier to spot), has substantially better FID score.

Is there correlation between CAS and linear probe accuracy??

Temporal vs frequency influence of augmentations. We compress only along temporal axis in the encoder. Could this be a reason for Gaussian artifacts in generation and not slice?

6.5 Discussion

The added flexibility of NC-VQVAE, with possibility of choosing dataset specific augmentations, can in some applications be beneficial.

6.6 Further work

[49] suggest that focus on augmentations is of great importance. The hunt for good augmentations in the time series domain is ongoing and should probably get more attention.
HF-LF split - augmentations tailored for HF and LF, as they often have quite different characteristics.
Wavelet transform to improve HF-LF split.

Further optimize the relationship between aug recon loss and choice of augmentations.

Improving on the stage 2 learning to better handle the expressive representations, and be able to create more diverse samples. Higher masking ratio during training, lower value for T etc.

The differences in Barlow and VIbCReg indicate that further optimization of the SSL method/pretext task for generative performance is possible and could be an interesting extension of this project.

Chapter 7

Conclusion

7.1 Reconstruction

7.2 Classification

Bibliography

[1] D. Lee, S. Malacarne and E. Aune, *Vector quantized time series generation with a bidirectional prior model*, 2023. arXiv: 2303.04743 [cs.LG].

[2] W. S. McCulloch and W. Pitts, 'A logical calculus of the ideas immanent in nervous activity,' *Bulletin of Mathematical Biology*, vol. 52, no. 1, pp. 99–115, 1990, ISSN: 0092-8240. DOI: https://doi.org/10.1016/S0092-8240(05)80006-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0092824005800060.

[3] F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton Project Para*, ser. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957. [Online]. Available: https://books.google.no/books?id=P_XGPgAACAAJ.

[4] K. Hornik, M. Stinchcombe and H. White, 'Multilayer feedforward networks are universal approximators,' *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[5] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[6] D. H. Hubel and T. N. Wiesel, 'Receptive fields and functional architecture of monkey striate cortex,' *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968. DOI: https://doi.org/10.1113/jphysiol.1968.sp008455. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1968.sp008455. [Online]. Available: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1968.sp008455.

[7] K. Fukushima, S. Miyake and T. Ito, 'Neocognitron: A neural network model for a mechanism of visual pattern recognition,' *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 826–834, 1983. DOI: 10.1109/TSMC.1983.6313076.

[8] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, 'Backpropagation applied to handwritten zip code recognition,' *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[9] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, L. Wang, G. Wang, J. Cai and T. Chen, *Recent advances in convolutional neural networks*, 2017. arXiv: 1512.07108 [cs.CV].

[10] K. He, X. Zhang, S. Ren and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[11] Y. Bengio, P. Simard and P. Frasconi, 'Learning long-term dependencies with gradient descent is difficult,' *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. DOI: 10.1109/72.279181.

[12] Y. Bengio, A. Courville and P. Vincent, *Representation learning: A review and new perspectives*, 2014. arXiv: 1206.5538 [cs.LG].

[13] L. Jing and Y. Tian, *Self-supervised visual feature learning with deep neural networks: A survey*, 2019. arXiv: 1902.06162 [cs.CV].

[14] K. Nozawa and I. Sato, *Empirical evaluation and theoretical analysis for representation learning: A survey*, 2022. arXiv: 2204.08226 [cs.LG].

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[16] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [cs.CV].

[17] S. Latif, A. Zaidi, H. Cuayahuitl, F. Shamshad, M. Shoukat and J. Qadir, *Transformers in speech processing: A survey*, 2023. arXiv: 2303.11607 [cs.CL].

[18] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan and L. Sun, *Transformers in time series: A survey*, 2023. arXiv: 2202.07125 [cs.LG].

[19] A. van den Oord, O. Vinyals and K. Kavukcuoglu, 'Neural discrete representation learning,' *CoRR*, vol. abs/1711.00937, 2017. arXiv: 1711.00937. [Online]. Available: http://arxiv.org/abs/1711.00937.

[20] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[21] A. Bardes, J. Ponce and Y. LeCun, *Vicreg: Variance-invariance-covariance regularization for self-supervised learning*, 2022. arXiv: 2105.04906 [cs.CV].

[22] J. Zbontar, L. Jing, I. Misra, Y. LeCun and S. Deny, *Barlow twins: Self-supervised learning via redundancy reduction*, 2021. arXiv: 2103.03230 [cs.CV].

[23] J. Bromley, I. Guyon, Y. Lecun, E. Säckinger and R. Shah, 'Signature verification using a siamese time delay neural network.,' vol. 7, Jan. 1993, pp. 737–744.

[24] K. He, H. Fan, Y. Wu, S. Xie and R. Girshick, *Momentum contrast for unsupervised visual representation learning*, 2020. arXiv: 1911.05722 [cs.CV].

[25] T. Chen, S. Kornblith, M. Norouzi and G. Hinton, *A simple framework for contrastive learning of visual representations*, 2020. arXiv: 2002.05709 [cs.LG].

[26] D. Lee and E. Aune, *Computer vision self-supervised learning methods on time series*, 2024. arXiv: 2109.00783 [cs.LG].

[27] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos and M. Valko, *Bootstrap your own latent: A new approach to self-supervised learning*, 2020. arXiv: 2006.07733 [cs.LG].

[28] K. He, X. Chen, S. Xie, Y. Li, P. Dollár and R. Girshick, *Masked autoencoders are scalable vision learners*, 2021. arXiv: 2111.06377 [cs.CV].

[29] H. Chang, H. Zhang, L. Jiang, C. Liu and W. T. Freeman, *Maskgit: Masked generative image transformer*, 2022. arXiv: 2202.04200 [cs.CV].

[30] S. Li, L. Zhang, Z. Wang, D. Wu, L. Wu, Z. Liu, J. Xia, C. Tan, Y. Liu, B. Sun and S. Z. Li, *Masked modeling for self-supervised representation learning on vision and beyond*, 2024. arXiv: 2401.00897 [cs.CV].

[31] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2022. arXiv: 1312.6114 [stat.ML].

[32] D. P. Kingma and M. Welling, 'An introduction to variational autoencoders,' *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019, ISSN: 1935-8245. DOI: 10.1561/2200000056. [Online]. Available: http://dx.doi.org/10.1561/2200000056.

[33] R. Gray, 'Vector quantization,' *IEEE ASSP Magazine*, vol. 1, no. 2, pp. 4–29, 1984. DOI: 10.1109/MASSP.1984.1162229.

[34] A. van den Oord, N. Kalchbrenner and K. Kavukcuoglu, *Pixel recurrent neural networks*, 2016. arXiv: 1601.06759 [cs.CV].

[35] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior and K. Kavukcuoglu, *Wavenet: A generative model for raw audio*, 2016. arXiv: 1609.03499 [cs.SD].

[36] Z. Wang, W. Yan and T. Oates, *Time series classification from scratch with deep neural networks: A strong baseline*, 2016. arXiv: 1611.06455 [cs.LG].

[37] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, *Improved techniques for training gans*, 2016. arXiv: 1606.03498 [cs.LG].

[38] S. Barratt and R. Sharma, *A note on the inception score*, 2018. arXiv: 1801.01973 [stat.ML].

[39] A. Borji, *Pros and cons of gan evaluation measures: New developments*, 2021. arXiv: 2103.09396 [cs.LG].

[40] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, *Gans trained by a two time-scale update rule converge to a local nash equilibrium*, 2018. arXiv: 1706.08500 [cs.LG].

[41] D. Dowson and B. Landau, 'The fréchet distance between multivariate normal distributions,' *Journal of Multivariate Analysis*, vol. 12, no. 3, pp. 450–455, 1982, ISSN: 0047-259X. DOI: https://doi.org/10.1016/0047-259X(82)90077-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0047259X8290077X.

[42] S. Jayasumana, S. Ramalingam, A. Veit, D. Glasner, A. Chakrabarti and S. Kumar, *Rethinking fid: Towards a better evaluation metric for image generation*, 2024. arXiv: 2401.09603 [cs.CV].

[43] M. J. Chong and D. Forsyth, *Effectively unbiased fid and inception score and where to find them*, 2020. arXiv: 1911.07023 [cs.CV].

[44] P. Esser, R. Rombach and B. Ommer, *Taming transformers for high-resolution image synthesis*, 2021. arXiv: 2012.09841 [cs.CV].

[45] H. Barlow, 'Possible principles underlying the transformations of sensory messages,' *Sensory Communication*, vol. 1, Jan. 1961. DOI: 10.7551/mitpress/9780262518420.003.0013.

[46] L. Huang, Y. Zhou, F. Zhu, L. Liu and L. Shao, *Iterative normalization: Beyond standardization towards efficient whitening*, 2019. arXiv: 1904.03441 [cs.CV].

[47] Y.-H.Huang, K. Huang and P. Fernández, *Vq-vae*, https://github.com/nadavbh12/VQ-VAE, 2021.

[48] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista and Hexagon-ML, *The ucr time series classification archive*, https://www.cs.ucr.edu/~eamonn/time_series_data_2018/, Oct. 2018.

[49] W. Morningstar, A. Bijamov, C. Duvarney, L. Friedman, N. Kalibhat, L. Liu, P. Mansfield, R. Rojas-Gomez, K. Singhal, B. Green and S. Prakash, *Augmentations vs algorithms: What works in self-supervised learning*, 2024. arXiv: 2403.05726 [cs.LG].

Appendix A

Additional Material

Additional material that does not fit in the main thesis but may still be relevant to share, e.g., raw data from experiments and surveys, code listings, additional plots, pre-project reports, project agreements, contracts, logs etc., can be put in appendices. Simply issue the command `\appendix` in the main `.tex` file, and make one chapter per appendix.

If the appendix is in the form of a ready-made PDF file, it should be supported by a small descriptive text, and included using the `pdfpages` package. To illustrate how it works, a standard project agreement (for the IE faculty at NTNU in Gjøvik) is attached here. You would probably want the included PDF file to begin on an odd (right hand) page, which is achieved by using the `\cleardoublepage` command immediately before the `\includepdf[]{}` command. Use the option `[pages=-]` to include all pages of the PDF document, or, e.g., `[pages=2-4]` to include only the given page range.

NTNU

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

_____

_____ (oppdragsgiver), og

_____

_____

_____ (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra _____ til_____ .

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): _____

Oppdragsgivers kontaktperson (navn): _____

Student(er) (signatur): _____ dato _____

_____ dato _____

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur):_____ dato _____

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____