You're your own best teacher: A Self-Supervised Learning Approach For Expressive Representations in Time Series Generation

Johan Vik Mathisen

June 14, 2024

Abstract

Time series generation (TSG) focuses on learning the distribution of time series data through generative modeling. State-of-the-art approaches in TSG, such as TimeVQVAE, utilize vector quantization-based tokenization to effectively model complex distributions. These models learn discrete latent representations of time series, and our research aims to enhance the *expressiveness* of these representations. We introduce a novel framework for integrating non-contrastive self-supervised learning into the tokenization model, termed NC-VQVAE. Our model is evaluated on a subset of the UCR archive, using SVM and KNN accuracies to assess the expressiveness of the latent representations, and IS, FID, and CAS metrics to evaluate the quality of the generated samples. Our results demonstrate that NC-VQVAE learns representations that improve classification and clustering of time series, while also enhancing the quality of synthetic samples.

Sammendrag

Tidsrekkegenerering (TSG) fokuserer på å lære fordelingen til tidsrekker ved generativ modellering. De beste metodene i TSG, som TimeVQVAE, utnytter vektorkvantisering for å effektivt modellere komplekse fordelinger. Disse modellene lærer diskrete latente representasjoner av tidsrekker, og vår forskning forsøker å gjøre disse mer *uttrykksfulle*. Vi introduserer et nytt rammeverk for integrering av ikke-kontrastiv selvstyrt læring i VQVAE. Vår modell, NC-VQVAE, evalueres på en undermengde av UCR-arkivet. Vi bruker SVM- og KNN-nøyaktighet til å bedømme uttrykksfullheten til de latente representasjonene, samt IS-, FID- og CAS-beregninger til å bedømme kvaliteten til de genererte tidsrekkene. Resultatene fører oss til konklusjonen at NC-VQVAE lærer representasjoner som forbedrer klassifisering og klynging av tidsrekker, samtidig som genereringskvaliteten øker.

Acknowledgements

This thesis is the culmination of 6 years of studies in Trondheim. There are many people that deserve big thank you for making these years so memorable. If you are reading this, chances are quite high that your name should be here.

I would firstly like to thank my supervisors, Erlend Aune and Daesoo Lee, for introducing me to this fascinating research field, and being open minded and supportive throughout the process. Erlend Lokna, a dear friend and great collaborator, for your indispensable contributions on this project. Developing advanced machine learning models is both challenging and time consuming, and synchronously banging our heads made it all a little easier. From the many late nights of writing, experimentation and discussion, to trash talking over the ping pong table, this project would not be the same without you. It was sweet to end on a 9-1 victory, which once and for all cemented me as the top dog, and nothing (except a rematch) could change that.

I would too like to thank my family, mom, dad, Otto, Andreas and Tiril for being a continuous source of inspiration and motivation. You move so graciously through life, sharing both of wisdom and skill, paving the way for the youngest in the flock. Erik, Ludvig and Sivert, my day ones. You have been there through all the ups and downs, with unwaivering support just a small phone call away. My roommates, Pernille and Eldrun, for being devilishly funny and handling me so well as I go full goblin mode. Preben, Elias and the rest of the Matteland mafia, despite placing bets on whether I would last one or two weeks into my bachelors, for their endless curiosity and fearlessness in face of difficult mathematics, pushing me to learn way more than I aught to have.

Finally, Tindegruppa, the university climbing group, my crew. You have such a special place in my heart. You reintroduced the joy of movement to my life, after abruptly retiring as professional handball player. You provided an arena to play, connect with nature and explore. Someone once told me, to have a great time as a student in Trondheim, you need to find your little community, be it on Samfundet or some part of NTNUI. I can wholeheartedly say I found mine, and to all my climber friends, such loving, crazy and interesting people, thank you. See you on top of some remote peak very soon.

Contents

Chapter 1

Introduction

Time series, as the name kinda spoils, is data with a time component. With the ever forward pointing arrow of time, it is really no surprise that these types of data are everywhere. Time series data is ubiquitous across various domains, ranging from finance and healthcare to environmental monitoring and industrial processes. Its importance lies in its ability to capture sequential observations over time, offering insights into underlying patterns, trends, and behaviors. Analyzing time series data can enable us to make informed decisions, predict future trends, and understand the dynamics of complex systems. However, traditional statistical models often struggle to capture the intricate structures present in time series data. These models typically assume linear relationships and stationary behavior, failing to adequately represent the complexities inherent in real-world time series. As a result, there is a pressing need for more sophisticated modeling approaches that can capture the non-linear, non-stationary, and high-dimensional nature of time series data.

Time series generation (TSG) is a branch of machine learning focused on synthesizing time series with similar characteristics as some observed data. TSG is applicable across numerous fields, whether it's generating synthetic data for training machine learning models, simulating realistic scenarios for predictive analytics, or augmenting limited datasets for robust analysis. In sectors with strict privacy regulations, such as healthcare, high quality synthetic data allow for privacy preserving analyses. The ability to generate plausible and diverse time series is highly valuable, but it is also notoriously difficult.

Unsupervised representation learning offers a promising path for enhancing time series generation, as it has for natural language and image data. By automatically discovering and extracting meaningful features from raw data, representation learning can assist the modeling of complex relationships and structures without the need for explicit supervision. This approach is particularly appealing for time series data, where the underlying patterns may be latent and not easily found through manual inspection. The ability to learn expressive representations

of time series data holds immense potential for advancing research and solving real-world problems.

There are many natural problems related to time series one wish to solve.

- **Forecasting** involves predicting future values of a time series based on its historical data.
- **Imputation** is the process of replacing missing or incomplete data points in a time series.
- **Anomaly detection** aims to identify unusual or unexpected patterns in time series data that do not conform to expected behavior.
- **Classification** involves categorizing time series data into predefined classes.
- **Clustering** groups similar time series or segments together based on their characteristics.
- **Synthetic sample generation** creates artificial time series data that mimics the properties of the training data.

A particularly intriguing property of generative models is the possibility to simultaneously solve several of the aforementioned problems. In this thesis we give confirmation that clustering and classification is also feasible, as our model can simultaneously classify, cluster and generate synthetic samples.

1.1 What is this thesis about?

The overarching theme of this thesis is machine learning, with a focus on two specialized areas: generative modeling and representation learning.

For the reader unfamiliar with the subject, we attempt to give a gentle introduction to the content of this thesis. Machine learning algorithms are fundamentally pattern finders, designed to pick up on patterns in data and utilize these patterns for various tasks. These tasks may include distinguishing images of dogs from cats, identifying fraudulent bank transactions from legitimate ones, or predicting tomorrow's weather. This is known as *predictive modeling*, which aims to use patterns in existing data to make predictions about unseen data. In *generative modeling*, the objective shifts to recognizing patterns that enable the creation of data that resembles the training data. A now familiar example of this is ChatGPT, a generative model capable of producing text similar to its training input.

The second major component of this thesis is representation learning, which involves viewing information from different perspectives. Just as diverse perspectives on an issue can highlight various aspects and be useful in different contexts, different data representations can reveal distinct patterns. In machine learning, representation learning is about finding perspectives that are computationally useful. For instance, consider the sequence of numbers $1, 3, 7, 15, 31, \ldots$. One approach to finding a generating formula might involve examining the differences

between consecutive numbers. However, a change in perspective, such as expressing the numbers in binary (i.e., $1, 11, 111, 1111, 11111$), makes the pattern more apparent—simply add another 1 to each subsequent number. While this is a simple example, it illustrates how changing perspectives can clarify patterns, a process that is crucial in complex scenarios commonly encountered in modern machine learning.

In this thesis we attempt, and in many ways succeed, to find a better perspective on time series data, such that we can create new ones that resemble, but not completely mimic, the training data. We do this by squishing the data into a smaller space, in a way that preserves the most important information. In this compressed space, we too push similar looking time series to the same regions, which in a sense organizes it. Finally, using a method similar to the one used by ChatGPT, we create new data in the compressed space, and decompress them to get new time series. The primary contribution of this thesis lies in this compressed but structured perspective on time series data.

Technically speaking, we investigate possible enhancements to the TimeVQVAE model presented in [1] by introducing a non-contrastive self-supervised loss to the tokenization model. We specifically examine if the representations learned are more expressive, in the sense that they improve the downstream classification accuracy, while simultaneously enable high quality reconstruction, and investigate how the learned representations affect the quality of the synthetic samples.

1.2 Research questions

We aim to leverage non-contrastive self-supervised learning to enhance the expressiveness of the discrete latent representations in TimeVQVAE. Non-contrastive SSL methods leverage augmented data as a supervisory signal during training, and the particular choice of augmentations typically have great influence on the learned representations. Hence, our research revolves around the following four questions:

RQ1: Will a self supervised learning approach enhance downstream classification, while simultaneously enable high quality reconstruction?
RQ2: How does different augmentations influence reconstruction quality and downstream classification accuracy?
RQ3: Will more expressive representations improve synthetic sample quality?
RQ4: How does different augmentations influence synthetic sample quality?

1.3 Sustainability impact

With respect for the incredible work done around the globe to address the UN sustainable development goals (SDG) [2], I will make no claim that sustainability

and considerations regarding environmental impact have been a part of our work. But, as new policy from NTNU demand I reflect around the works relation to the SDGs, I will do so.

The field of AI and machine learning is in rapid development, and the fear of being left in the dust in the gold rush push large actors to train ever larger models. Though the modern generative models such as ChatGPT and Stable Diffusion are very impressive, it is hard to ignore their colossal environmental impact. These large models are both power hungry and water thirsty. According to an article in Nature magazine [3], Kate Crawford suggests that large AI systems will soon likely need energy on the scale of entire nations. Furthermore in [4] they make the claim that the global demand for AI could account for freshwater withdrawals nearly half the volume of the United Kingdom by 2027. The current trajectory puts the AI field at odds with SDG 6, which aims to ensure the availability and sustainable management of water and sanitation for all.

While we believe that the generative AI has valuable applications and can be deployed without conflicting with the SDGs, it necessitates careful considerations. Our work on time series offers advantages because it is less resource-intensive than image and text generation. Additionally, work on time series has the benefit of being less flashy than image and text generation, and we consider TSG less likely to be used excessively, more easily aligning it with sustainable practices.

1.4 Collaboration and AI Declaration

During the work on our theses, Erlend Lokna and myself have collaborated extensively. This has been declared from the beginning and talked about continuously with our supervisors. Erlend, with his developer experience, has taken lead on code development for our model, and should get credit for his high level of programming skills. I have, in addition to code contribution and being a source of ideas, taken the lead on data processing and visualization. We have exchanged ideas, possible paths forward and the overall structure of the theses. Additionally, we have conducted the same experiments, hence our works will have many similarities. All writing though, is done independently. The collaboration has been very fruitful, and working closely with Erlend has made the process quite enjoyable, despite him being lousy at ping pong.

Content of this thesis have been to some degree been made with the assistance of LLMs. As a non-native english speaker, ChatGPT has been helpful in improving grammar and readability. It has provided assistance in producing and debugging LaTeX and Python code throughout.

1.5 Thesis Overview

We begin by building up the theoretical background for this thesis. There we introduce neural networks, and relevant architectures for our work, such as convolutional neural networks and transformers. We give an introduction to representation learning, highlighting standard evaluation protocols, and give an introduction to self-supervised learning (SSL), with emphasis on siamese architecture-based SSL. We present the Vector Quantized Variational Autoencoder (VQVAE), trying to highlight aspects brushed over in [5], building it up from Autoencoders. Lastly we go through the evaluation metrics used to assess our models.

In the related works we present models relevant to our work, including TimeVQVAE [1] and MaskGIT [6], as well as the non-contrastive SSL methods Barlow Twins [7] and VIbCReg [8]. This leads up to the methodology section, where we introduce the proposed model: NC-VQVAE. We give schematic overview of the architecture and details regarding the training objective.

In the experiments section we present the details regarding the setup for the main experiment, which include details regarding implementation, hyperparameters and datasets used. We then presents all results from our experiments, providing analyses regarding both the tokenization model and prior model, as well as going through visual inspections of both the learned representations and generated samples. Finally we conclude our results and provide possible paths forward for further research.

The code base for the project is found at `https://github.com/erlendlokna/Generative-SSL-VQVAE-modelling`.

Chapter 2

Theoretical Background

The field of machine learning draws on many parts of mathematics, statistics and computational sciences, and as much as the author would love deep diving into every aspect, certain sections are somewhat superficial. There are though, for the interested reader, a large number of sources going into greater detail referenced throughout.

2.1 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 [9]. The first practical implementation was by Frank Rosenblatt in 1957 [10]. 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 [11], 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.

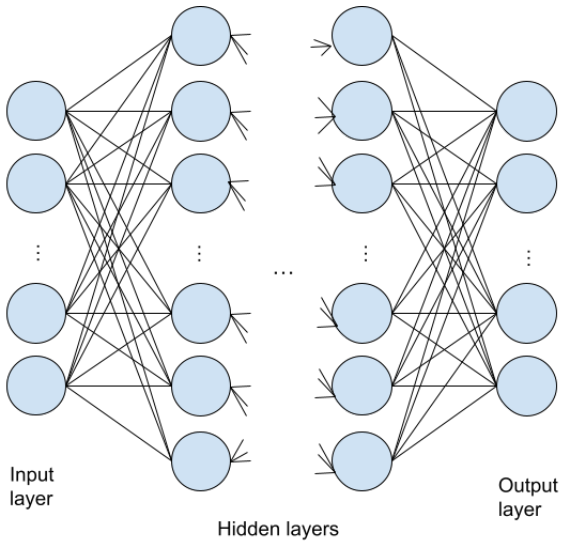


Figure 2.1: Illustration of a Neural Network model.

The variables $x = [x_1, ..., x_n]$ constitute the units of the *input layer*. 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 set of non-linear *activation functions* $\{\sigma_1, \ldots, \sigma_{K-1}\}$. 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.

2.1.1 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_θ 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.

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 [12].

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
```

2.2 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 is a neural network which replaces matrix multiplication with convolution in at least one of its layers [12].

Fully connected neural networks face 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 core idea of convolutional neural networks 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 [13], leading to the development of the neocognitron in 1980 [14], 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 [15]. Since then, CNNs have become a cornerstone of machine learning. For an in-depth exploration of the advancements and applications of convolutional neural networks, refer to [16].

2.2.1 The convolution operation

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

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

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau. \tag{2.1}$$

While the detailed mathematical criteria for the existence of this integral is outside 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* [12]. 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

$$(I * K)[n] = \sum_{m=-\infty}^{\infty} I[m]K[n-m]. \tag{2.2}$$

In practice I and K 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

$$(I * K)[i,j] = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} I[n,m]K[i-n, j-m]. \tag{2.3}$$

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 [12].

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.

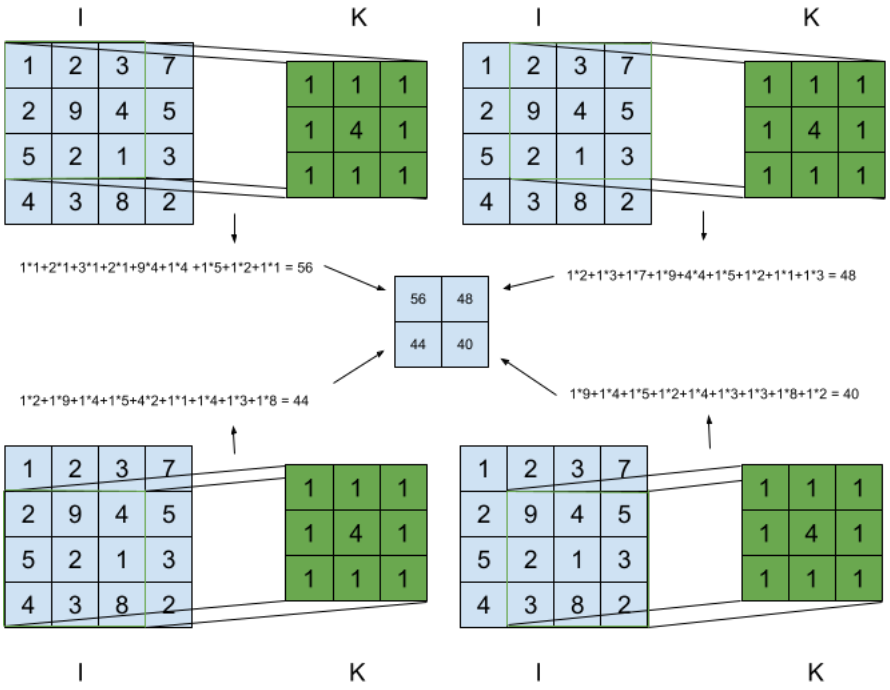


Figure 2.2: Illustration of discrete two dimensional convolution

2.2.2 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-

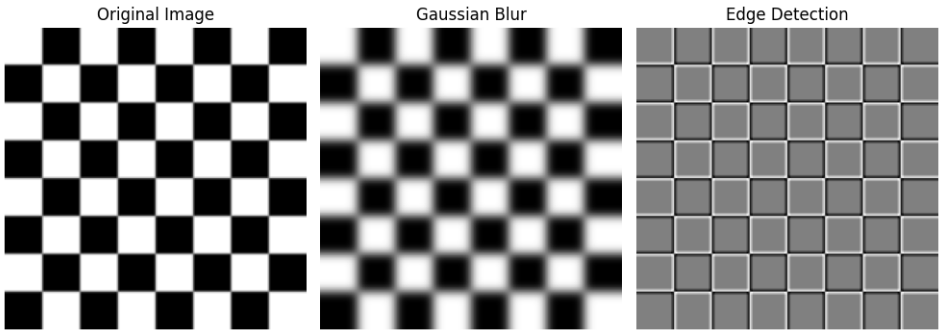


Figure 2.3: Illustration of discrete convolution applied to images. Image part of Scikit-image data package.

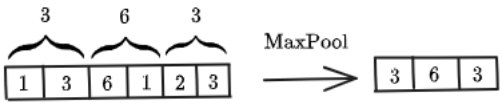


Figure 2.4: Max pooling of one dimensional array. Filter size: 2, stride: 2.

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.

2.2.3 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 [15] as an example of a general CNN.

A convolutional layer consists of several kernels. Each kernel is convolved with the entire input, producing the feature maps. 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 dimensions while retaining important information. The pooling layer helps to reduce the computational load and control overfitting by progressively reducing the size of the representations.

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

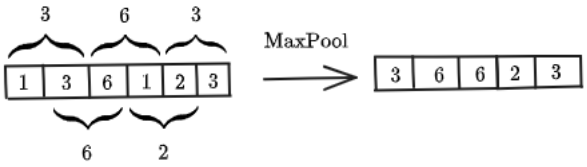


Figure 2.5: Max pooling of one dimensional array. Filter size: 2, stride: 1.

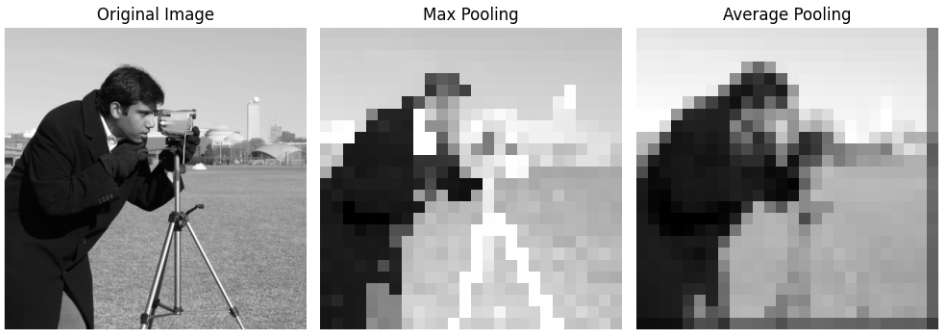


Figure 2.6: Illustration of mean and average pooling applied to images. Filter size and stride is 20×20. Original image size is 512×512, pooled images are 26×26. Image part of Scikit-image data package.

downsampled through convolutional and pooling layers. The fully connected layers aggregate the local features learned by convolutional layers, 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 [17], residual connections allow gradients to flow directly through the network, alleviating the vanishing gradient problem in deep networks [18]. 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 [17], which have achieved state-of-the-art results in various computer vision tasks.

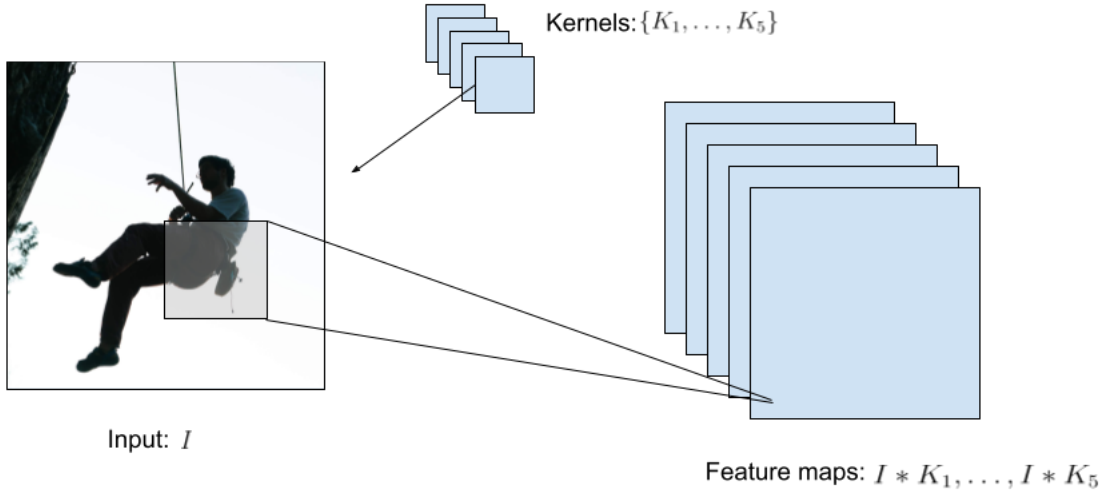


Figure 2.7: Convolutional layer.

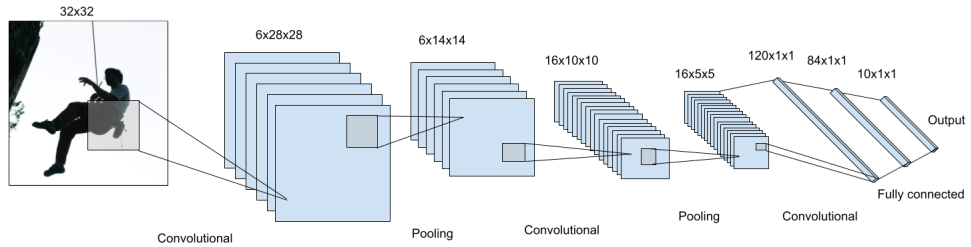


Figure 2.8: LeNet-5 network [15]

2.2.4 Transposed Convolutional Networks

Transposed convolution, also known as deconvolution or fractionally-strided convolution, is a technique used to reverse the downsampling effect of 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 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 2.6. These architectures are fundamental in many generative models. The encoder part of the network typically reduces the 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 dimensions, reconstructing the input or generating new data.

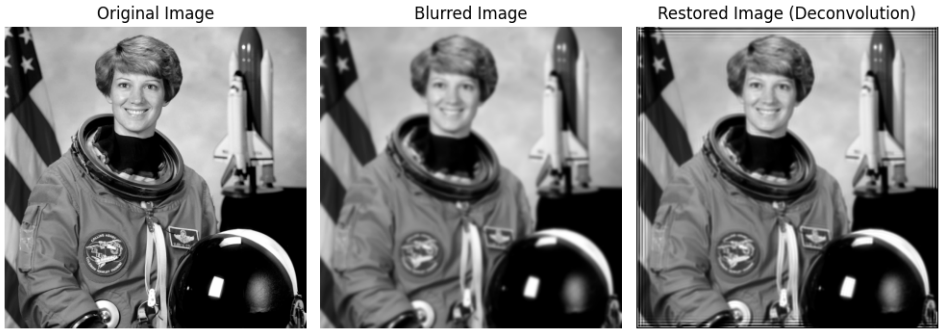


Figure 2.9: Illustration of deconvolution applied to images. Here using Gaussian blur and restoring using the Richardson Lucy algorithm with a Gaussian deconvolution kernel. Image part of Scikit-image data package.

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 that transposed convolutions can be used to approximately reverse the effects of convolutional operations.

2.3 Representation Learning

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 base, 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.

2.3.1 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 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 [19]. 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 [19].

2.3.2 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.

2.3.3 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 [20], 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 *head* 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.

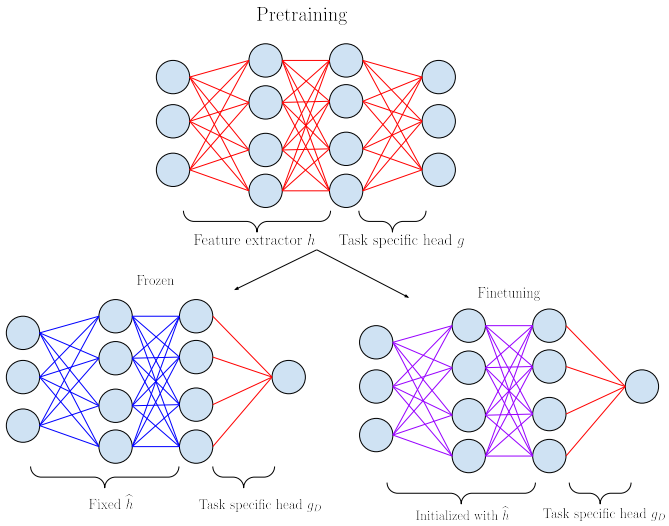


Figure 2.10: Illustration of representation learning protocols. Figure inspired by [21].

As illustrated in Figure 2.10, 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 layer 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 pretraining-finetuning. In this approach, the model typically 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 [21].

2.4 Transformers

Since their introduction in the seminal paper "Attention is all you need"[22], 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 [23], and similar trends are shown for other modalities such as audio [24] and time series [25].

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 [18]. 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 flattened and used as inputs. 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 [23], input images are split into patches and linearly projected before being processed by the transformer. Since then, *Vector-Quantized*-based tokenization introduced by [5] has emerged as a popular approach.

2.4.1 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.

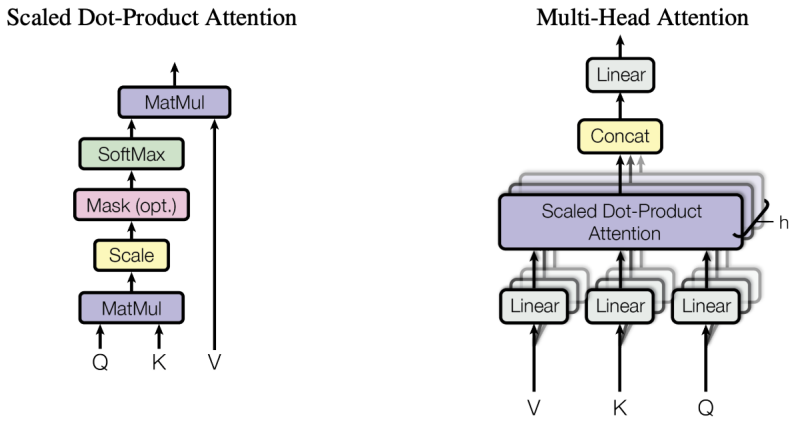


Figure 2.11: (left) Scaled dot-product attention. (right) Multi-headed attention. Taken with permission from [22]

Scaled dot-product attention is computed by

$$\text{Attention}(Q,K,V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.4}$$

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 sub-

space. The results from each subspace are then concatenated and linearly transformed. The process is defined as

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O, \tag{2.5}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \tag{2.6}$$

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 richer representations.

2.4.2 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.

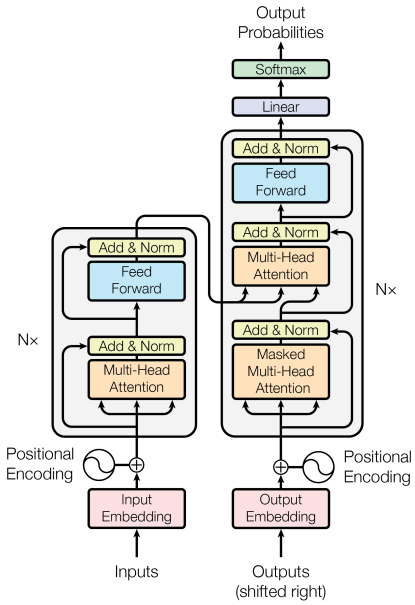


Figure 2.12: The Transformer - model architecture. Encoder (left), decoder (right). Taken with permission from [22]

Embeddings: The first step in the transformer architecture involves mapping each token in the input sequence to a vector through an embedding procedure. 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, leveraging the geometry of $\mathbb{R}^n$.

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 [22], 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 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 [26], 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 2.5.2.

2.5 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

$$\widehat{f_\theta} = \min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i). \tag{2.7}$$

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 [7, 27]. 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

$$\widehat{f}_\theta = \min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), p_i). \tag{2.8}$$

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, 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.

2.5.1 Siamese Architecture-based SSL

A siamese network architecture [28] consists of two branches with a shared encoder on each branch, as illustrated in Figure 2.13. 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*.

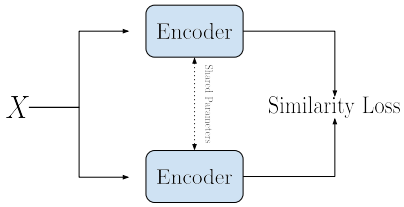


Figure 2.13: Schematics of a basic siamese architecture.

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 [29] and SimCLR [30]. 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 [8].

Non-Contrastive SSL methods, such as BYOL [31], BarlowTwins [7] and VIbCReg [8] 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.

Projector

Most siamese architecture-based self-supervised learning (SSL) methods utilize a projector to produce embeddings for the SSL loss. A projector is a neural network designed to map representations onto a subspace or expand them into a larger space, depending on the SSL method. In the literature, some differentiate between projection networks (projectors) and expanding networks (expanders). However, in our work, we use VICReg and Barlow Twins, both of which utilize expanding networks as projectors. For considerations on low-rank projectors, refer to [32].

Similar to the setup described in the representation learning section, we can view the projector as a head g in the model $f = g \circ h$, with the output of h considered as the representations. The primary reason for not using the projector's output as our representations is that these embeddings are too specialized for minimizing the SSL loss and generalize poorly. While the exact reasons for this behavior are unclear, empirical results consistently show the superiority of using the projector's input rather than its output for downstream tasks.

Experiments on the architecture of expanding projectors, simply referred to as projectors, in [7, 27] indicate a steady performance gain when increasing the projector's width (output dimension) from 256 to 8192. However, performance saturates when the depth (number of layers) reaches three.

Projectors used in SSL methods that employ variance/covariance regularization, such as Barlow Twins and VIbCReg, help enforce pairwise independence of the components of representations [33]. Independent representation features is in the literature considered a desired property, though not universally accepted [19, 34]. The article [33] attempt to explain why shallow and wide projectors typically result in improved representations. The main argument is that, as stated in Lemma 2, when the projector has orthogonal weights, minimizing the covariance of the projector output amounts to minimizing the pairwise correlation of the input components. For wider projectors, the orthogonality assumption is more accurate, as the dot product of randomly initialized vectors with zero mean tends to zero as the dimension increases, by the central limit theorem. They further argue that learning the projector is only crucial for maintaining variability, and that for wider networks, the weights need not move far from the random initialization to successfully maintain variance.

2.5.2 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 [26], 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 [35]. With the success of vision transformers [23], researchers began tokenizing images and pretraining transformers in a manner similar to BERT. For example, MaskGIT [6] 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 [36].

2.6 Vector Quantized Variational Autoencoder (VQVAE)

Our model is based on the Vector Quantized Variational Autoencoder (VQVAE) introduced in [5], and includes an Autoencoder (AE) branch. Therefore it is natural to dive into the models. We first start with introducing autoencoders, then present the variational variation VAE before presenting VQVAE.

2.6.1 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 2.14.

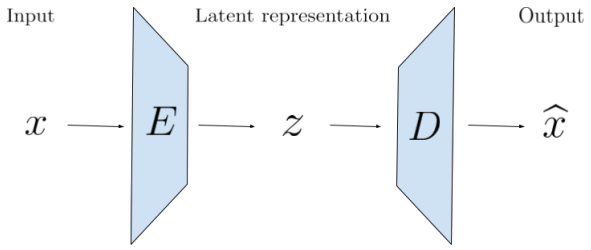


Figure 2.14: Schematics of the autoencoder architecture.

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 in order to reconstruct well. 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.

Even 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.

2.6.2 Variational Autoencoder (VAE)

Variational Autoencoders (VAE) were introduced in [37] and is a variational Bayes approach to approximate inference and learning with directed probabilistic models. While the architecture of VAEs, presented in Figure 2.15, is similar to that of traditional autoencoders, 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 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. Actually, 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 $p_\theta(z) \sim \mathcal{N}(0, I)$
- Likelihood $p_\theta(x|z) \sim \mathcal{N}(D(z), I)$
- Variational posterior $q_\phi(z|x) \sim \mathcal{N}(E(x)) = \mathcal{N}(\mu_x, \Sigma_x)$

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 $\epsilon \sim \mathcal{N}(0, I)$ 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.

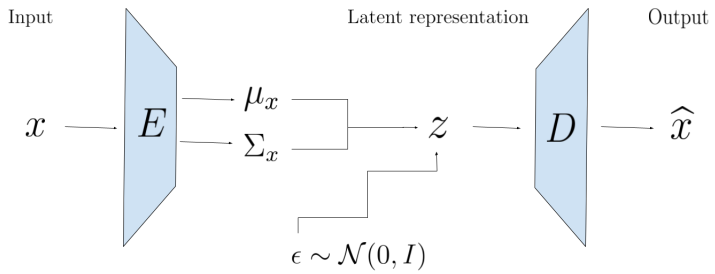


Figure 2.15: Schematics of the VAE architecture with Gaussian reparameterization.

Training objective

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

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{p_\theta(x,z)}{q_\phi(z|x)}\right) \tag{2.9}$$

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

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)} \log(p_\theta(x)) + \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{p_\theta(z|x)}{q_\phi(z|x)}\right) \\ &= \log(p_\theta(x)) - \mathrm{KL}(q_\phi(z|x)||p_\theta(z|x)).\end{aligned} \tag{2.10}$$

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 [38].

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

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)} \log(p_\theta(x|z)) - \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{q_\theta(z|x)}{p_\phi(z)}\right) \\ &= \underbrace{\mathbb{E}_{q_\phi(z|x)} \log(p_\theta(x|z))}_{\text{Expected reconstruction log likelihood}} \underbrace{- \mathrm{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{Regularizer}}.\end{aligned} \tag{2.11}$$

Assuming a Gaussian likelihood, we have

$$\log p_\theta(x|z) = -\frac{1}{2}\left[k\log(2\pi) + (x-\widehat{x})^T(x-\widehat{x})\right], \tag{2.12}$$

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.

2.6.3 VQVAE

The Vector Quantized Variational Autoencoder (VQVAE) was first introduced in [5] 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 $\mathcal{Z} = \{z_k\}_{k=1}^{K}$, 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

$$z_q = \arg\min_{z_k \in \mathcal{Z}} ||E(x) - z_k||. \tag{2.13}$$

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

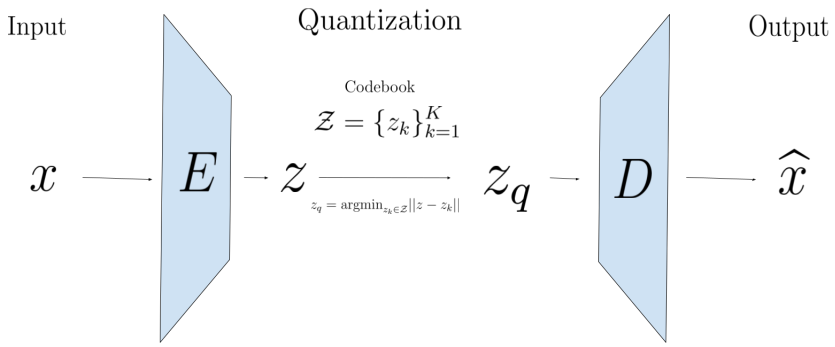


Figure 2.16: Schematics of the VQVAE architecture.

ELBO

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

$$q(z=k|x) = \begin{cases} 1 & \text{for } k = \arg\min_j ||E(x) - z_j||_2 \\ 0 & \text{otherwise} \end{cases} . \tag{2.14}$$

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

In the original article [5], 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 2.11 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

$$\begin{aligned} \text{KL}(q(z|x)||p(z)) &= \sum_z q(z|x) \log\left(\frac{q(z|x)}{p(z)}\right) \\ &= q(z=k|x) \log\left(\frac{q(z=k|x)}{p(z=k)}\right), \; q \text{ is deterministic} \\ &= \log\left(\frac{1}{1/K}\right), \text{ uniform prior} \\ &= \log(K). \end{aligned} \tag{2.15}$$

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 Euclidean distance between the input x and the reconstructed output $\widehat{x}$

$$\mathcal{L}_{\text{recons}} = ||x - \widehat{x}||_2^2. \tag{2.16}$$

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, is introduced to keep the distance between codewords from growing arbitrarily large. It is given by

$$\mathcal{L}_{\text{codebook}} = ||sg(z) - z_q||_2^2 + \beta||z - sg(z_q)||_2^2, \tag{2.17}$$

where β is a tuning parameter, typically set to be 0.25, and $sg()$ is the stop-gradient operation, defined as the identity with zero partial derivatives. The total loss function is given by the sum of the two components

$$\mathcal{L}_{\text{VQ}} = \mathcal{L}_{\text{codebook}} + \mathcal{L}_{\text{recons}}. \tag{2.18}$$

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 of [5].

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, which allows the encoder to focus on learning the optimal mapping from the data space to the discrete latent space without being influenced by a constraining 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, one learn 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 [39] was used for image data, while WaveNet [40] was used for audio. More recently in TimeVQVAE [1] utilized a bidirectional transformer [6] for prior learning in the time series domain.

2.7 Evaluation metrics

Downstream Classification

When evaluating the performance of representations, downstream classification tasks provide an insightful measure of how well the learned representations can be utilized. 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 captures local structures in the representations. 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

2.7.1 Generative Model Evaluation

Evaluating generative models is 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 benchmark datasets and classification models.

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 quantitat-

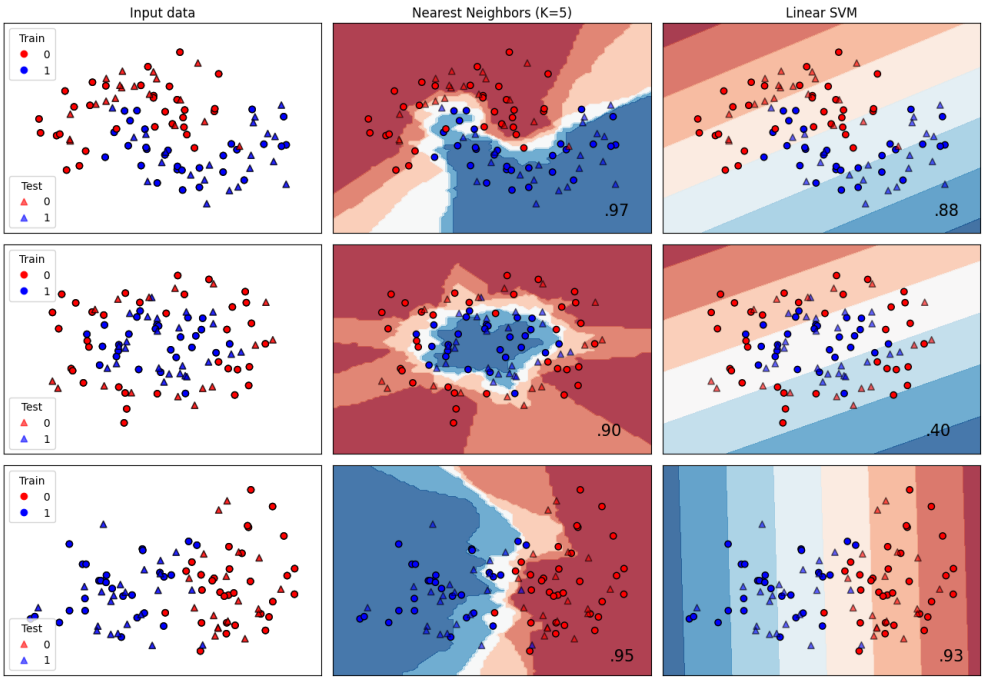


Figure 2.17: Modified example taken from `https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html`.

ive 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. In [41] a fully convolutional network (FCN) was proposed 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 [42] 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 $p(y|x)$, meaning that given a sample x the classifier should be certain of its label. Diverse samples should lead to high entropy in the marginal label distribution $p(y)$, meaning that if we average the conditional label

distributions for a set of generated samples, the different labels should be approximately equally probable. Thus, if the KL-divergence between these distributions are high, the samples should be of high quality. The Inception Score is defined as

$$\mathrm{IS}(\theta) = \exp\left(\mathbb{E}(D_{\mathrm{KL}}(p_\theta(y|\mathbf{x})||p_\theta(y)))\right). \tag{2.19}$$

The Inception Score ranges from 1 to the number of classes, and a higher value indicates higher quality.

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. Additionally IS has no mechanism for considering the diversity of generated samples within a class, which stresses the importance of reporting additional metrics that indicate whether the model overfits.

Fréchet Inception Distance (FID)

To address some limitations of IS, the Fréchet Inception Distance (FID) was introduced in [43]. Since then FID has been the standard for assessing generative models [44]. 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

$$\begin{aligned} d_F(f,g) &= \left(\inf_{\gamma\in\Gamma(f,g)} \int_{\mathbb{R}^n\times\mathbb{R}^n} ||x-y||_2^2 d\gamma(x,y)\right)^{\frac{1}{2}} \\ &= \left(\inf_{\gamma\in\Gamma(f,g)} \mathbb{E}_{(x,y)\sim\gamma}||x-y||_2^2\right)^{\frac{1}{2}}, \end{aligned} \tag{2.20}$$

where $\Gamma(f,g)$ is the set of all *couplings* of f and g. For two Gaussian distributions, as proven in [45], the Fréchet distance is explicitly computable as

$$d(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\mu',\Sigma'))^2 = ||\mu-\mu'||_2^2 + \mathrm{Tr}\left(\Sigma+\Sigma'-2(\Sigma\Sigma')^{\frac{1}{2}}\right). \tag{2.21}$$

In FID, the mean and covariance of the feature representations of real and generated samples are computed using a pretrained model. They argue in [43] 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 [46, 47].

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 leverage non-contrastive self-supervised learning (SSL) to improve representation learning in a Vector Quantized Variational Autoencoder (VQVAE) type model. 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 utilize 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 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, 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 [23], 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 [5] for details on the prior model training.

3.1.1 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 3.1. 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.

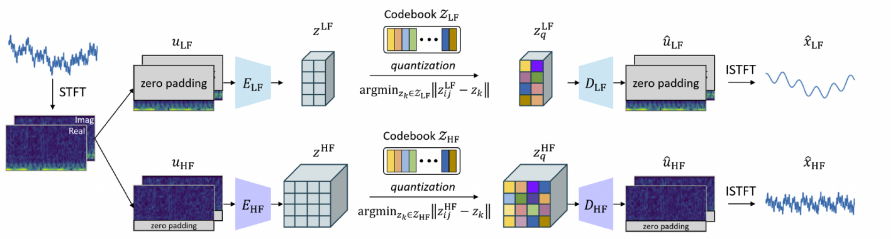


Figure 3.1: Stage 1: Tokenization. Figure taken with permission from [1]

Loss

The codebook loss of TimeVQVAE is similar to the codebook loss presented in Section 2.6 but reflects the HF-LF split

$$\begin{aligned}\mathcal{L}_{\text{codebook}} = &||\,\text{sg}[E_{LF}(\mathcal{P}_{\text{LF}}(\text{STFT}(x)))] - z_q^{\text{LF}}||_2^2 \\ &+ ||\,\text{sg}[E_{HF}(\mathcal{P}_{\text{HF}}(\text{STFT}(x)))] - z_q^{\text{HF}}||_2^2 \\ &+ \beta||E_{LF}(\mathcal{P}_{\text{LF}}(\text{STFT}(x))) - \text{sg}[z_q^{\text{LF}}]||_2^2 \\ &+ \beta||E_{HF}(\mathcal{P}_{\text{HF}}(\text{STFT}(x))) - \text{sg}[z_q^{\text{HF}}]||_2^2,\end{aligned} \tag{3.1}$$

where $\mathcal{P}_{[\cdot]}$ denotes the zero-padding operation. The reconstruction loss is computed both on time and time-frequency reconstructions and is given by

$$\begin{aligned}\mathcal{L}_{\text{recons}} = &||x_{\text{LF}} - \widehat{x}_{\text{LF}}||_2^2 + ||x_{\text{HF}} - \widehat{x}_{\text{HF}}||_2^2 \\ &+ ||u_{\text{LF}} - \widehat{u}_{\text{LF}}||_2^2 + ||u_{\text{HF}} - \widehat{u}_{\text{HF}}||_2^2.\end{aligned} \tag{3.2}$$

The total loss is given by

$$\mathcal{L}_{\text{VQ}} = \mathcal{L}_{\text{codebook}} + \mathcal{L}_{\text{recons}}. \tag{3.3}$$

To update the codebooks, TimeVQVAE uses an exponential moving average method as presented in Appendix A.1 of [5].

3.2 MaskGIT

The Masked Generative Image Transformer (MaskGIT), introduced in [6], 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 [48] was used for this purpose, focusing on improving the second stage. Therefore, our discussion will primarily address this aspect of the model.

3.2.1 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, denoted 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

$$s_{\text{Mask}} = s \odot (1_N - M) + M \cdot \texttt{M}, \tag{3.4}$$

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 $\{0, \ldots, N-1\}$ 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 3.2, where the bidirectional transformer predicts the probabilities $p(s_i|s_{\text{Mask}})$ of each masked token. The training objective is to minimize the negative log likelihood of the masked tokes, conditional on the unmasked

$$\mathcal{L}_{\text{Mask}} = -\mathbb{E}_{s \in \mathcal{D}} \left[\sum_{i \in I} p(s_i|s_{\text{Mask}}) \right], \tag{3.5}$$

and $\mathcal{L}_{\text{Mask}}$ is computed as the cross entropy between the ground truth one-hot token and the predicted token probabilities.

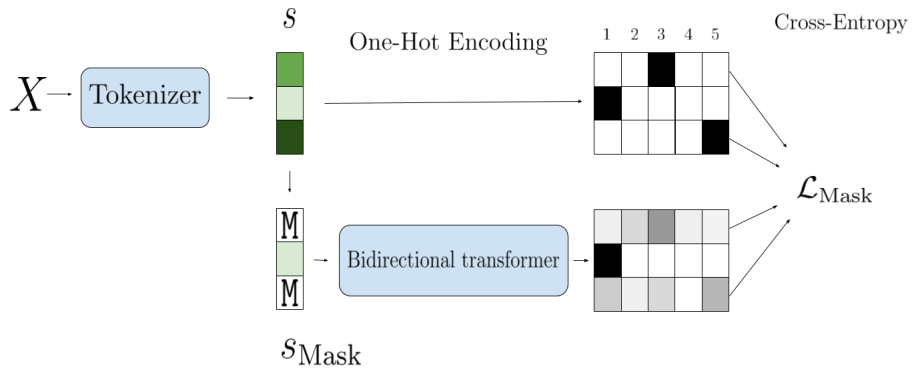


Figure 3.2: MaskGIT forward computation.

3.2.2 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, [6] 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 $s_i^{(t)}$ 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

$$m_i^{(t+1)} = \begin{cases} 1, \text{ if } c_i < \text{Sort}([c_1^{(t)}, \ldots, c_N^{(t)}])[n] \\ 0, \text{ otherwise} \end{cases} \tag{3.6}$$

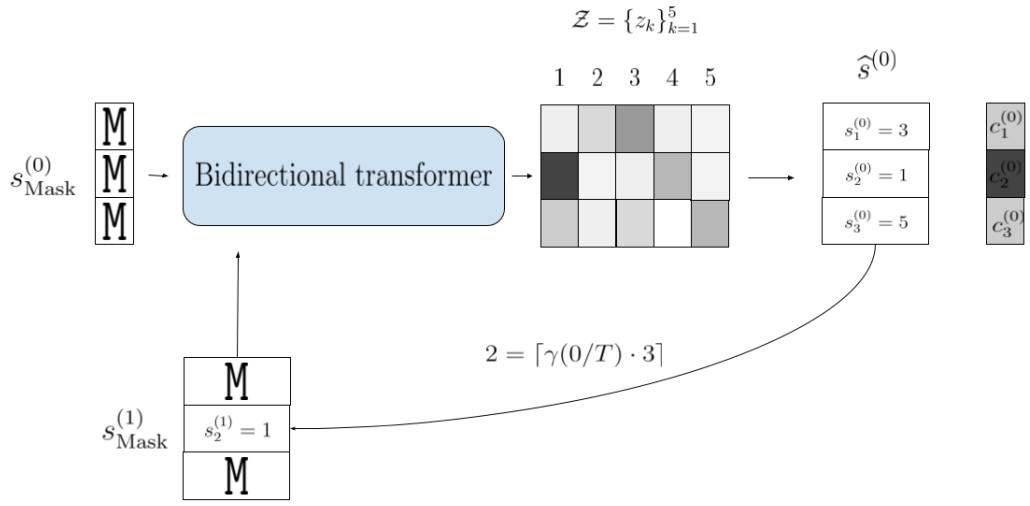


Figure 3.3: Illustration of first pass of the iterative decoding algorithm.

This process is illustrated in Figure 3.3, 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.

3.3 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* [49] 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 enforces both distortion invariance and decorrelated features in the representations.

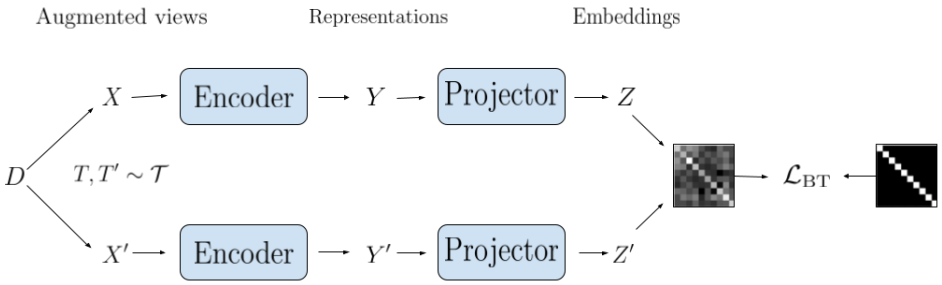


Figure 3.4: Overview of the Barlow Twins architecture. Figure inspired by [7]

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

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

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 [8] is a non-contrastive SSL model with a siamese architecture based on VICReg [27]. It improves upon VICReg by incorporating better covariance regularization and IterNorm [50]. 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 then *whitened* using IterNorm [50] before calculating the similarity loss.

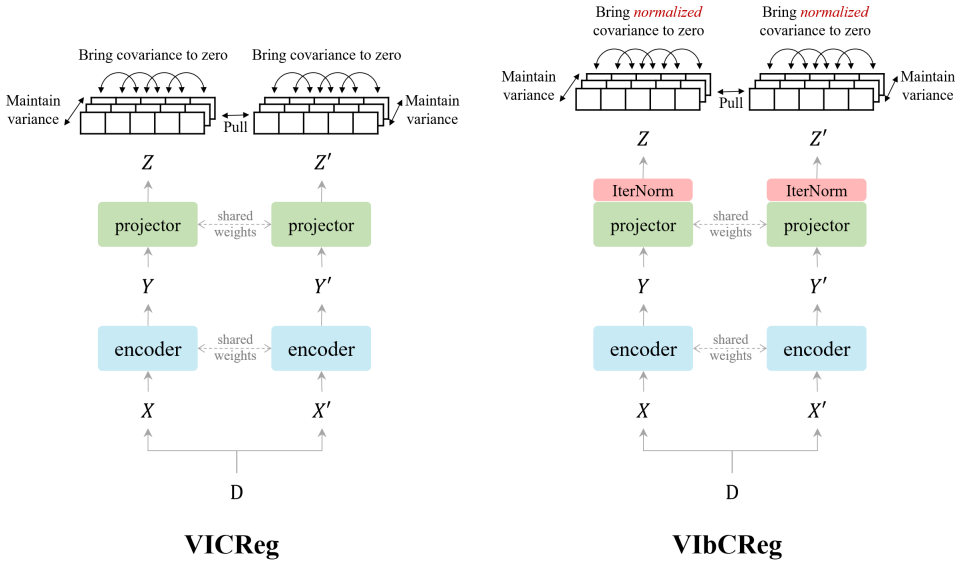


Figure 3.5: Overview of VIbCReg, and comparison with VICReg. Taken with permission from [8]

The loss consists of a similarity loss between the branches, a feature decoration (FD) term 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 keep the range of the FD loss small, independent of data, and eases hyperparameter tuning across datasets.

Chapter 4

Methodology

Our work in this thesis, as mentioned earlier, 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 encode semantic information into the discrete latent representation and 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, which should structure the discrete latent space in such a way that data with similar characteristics inhabit distinct regions. Additionally, 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, as well as preventing the encoder from ignoring the augmentations.

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 with non-contrastive SSL algorithms. NC-VQVAE uses the two-stage modeling approach and can be considered an extension of their naive TimeVQVAE. Our model primarily extends the tokenization stage, incorporating Barlow Twins and VIbCReg 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.

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.

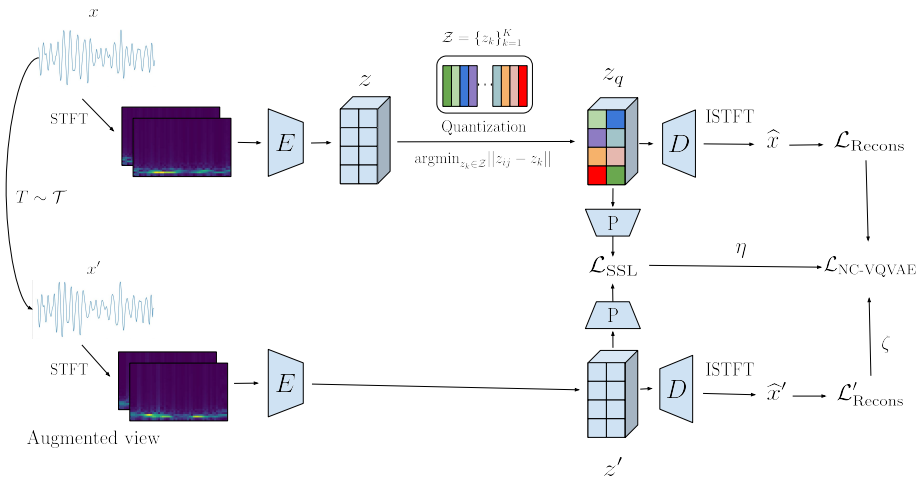


Figure 4.1: Overview of proposed model: NC-VQVAE.

Loss

Our training objective mirrors the training objective from TimeVQVAE in equation 3.3, 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 Euclidean distance between continuous and discrete latent representations, along with a commitment loss to prevent the codewords from diverging. In our setup, the codebook loss reduces to

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

and the reconstruction loss to

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

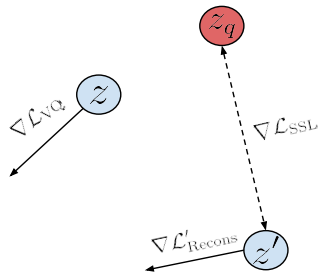


Figure 4.2: Illustration of the effect of different loss terms during training.

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 utilize 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 experiments 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. An illustration of the effect of the different loss terms on the latent space during training is presented in Figure 4.2.

4.1.2 Stage 2: Prior learning

In our model, the input embedding for the bidirectional transformer is initialized with the codebook, which has additional structure from the 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 Main Experiments

To evaluate our model, NC-VQVAE, and address the research questions, we compare it to the naive VQVAE using both Barlow Twins and VIbCReg as self-supervised learning methods. Additionally, as we are interested in the effect of augmentations on representation learning, for each SSL method we train three distinct models using different sets of augmentations. Furthermore, each configuration is trained using four different seeds, resulting in a total of 364 models trained at each stage. All experiments were run on the Idun HPC cluster [51], using several compute nodes. The total training time of all 728 models was approximately 700 hours.

5.1.1 Implementation details

In our implementation, we closely follow to the methodology outlined in [1], particularly in the design and deployment of the encoder, decoder, and codebook. The code base for the project is found at `https://github.com/erlendlokna/Generative-SSL-VQVAE-modelling`

Time Frequency Modelling

We utilize the Short Time Fourier Transform (STFT) and its inverse (ISTFT) for time-frequency modeling, using the functions `torch.stft` and `torch.istft`, respectively. Consistent with [1], we set the primary parameter $\texttt{n}_{\texttt{fft}}$ to 8 and use default parameters for others. This configuration yields a frequency axis spanning $[1, 2, 3, 4, 5]$ and a time axis half the length of the original.

Encoder and decoder

The encoder and decoder architectures closely resemble those described in [52], with further adaptations from [1].

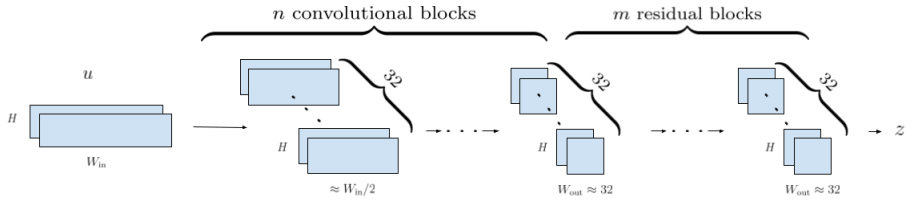


Figure 5.1: Overview of the encoder architecture. The decoder architecture is simply obtained by reversing the arrows and switching out the convolutional block for transposed convolutional blocks.

As illustrated in Figure 5.1, the encoder consists of n downsampling convolutional blocks (`Conv2d - BatchNorm2d - LeakyReLU`), followed by m residual blocks (`LeakyReLU - Conv2d - BatchNorm2d - LeakyReLU - Conv2d`). Downsampling convolutional layers are implemented with parameters:

```
kernel size=(3,4),stride=(1,2),padding=(1,1),
```

downsampling the temporal axis by a factor of 2 per block. Residual convolutional layers have parameters:

```
kernel size=(3,3),stride=(1,1),padding=(1,1).
```

The decoder mirrors this architecture, featuring m residual and n upsampling layers using transposed convolutional blocks with identical parameters.

The downsampling rate is determined by 2^n, ensuring the resulting latent representation (z) has a width of 32. While we set the number of residual blocks $m = 4$. For an in depth walkthrough of the TimeVQVAE implementation details, we direct readers to Appendix C.3 of [1].

VQ

The implementation for VQVAE is taken from [**VQ_repo**]. We adopt a codebook size of 32 and a dimension of 64, utilizing exponential moving average with a decay of 0.9 and a commitment loss weight $\beta = 1$. Codebook embeddings learned in stage 1 is used to initialize the codebook embeddings in stage 2.

SSL

For both Barlow Twins and VIbCReg, we implement the projector following the guidelines in [8]. We use identical projectors for both methods, consisting of 3 linear layers. The first two are normalized using `BatchNorm1d`. Both hidden and output layer has dimension size set to 4096.

For Barlow Twins the weight of the redundancy reduction term, λ, is set to 0.005. The weight of the Barlow Twins loss in the NC-VQVAE is set to $\eta = 1/D$ where D is the projector output dimension.

For VIbCReg, the weights of the similarity loss, λ, and variance loss, μ, are both set to 25, while for the covariance loss, ν is set to 100. The weight of the VIbCReg loss in the NC-VQVAE is set to $\eta = 0.01$.

Augmentations

For the reconstruction loss on the augmented brach we set the weight ζ to 0.1.

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 transforms in both x and y direction. The window warp augmentation randomly selects a section of the time series and speeds it up or down, and the factor is choose randomly from the interval $[0.9, 2.0]$. We interpolate in order to obtain a time series of equal length as the original. It has similar qualities to phase shift, but not uniformly and keeps endpoints fixed. The amplitude resize multiplies the time series by a factor of $1 + N(0, 0.2)$. This set was 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 as seen in Figure 5.2. In many cases we will refer to this set of augmentations simply as warp.

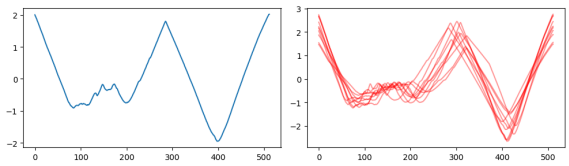


Figure 5.2: ShapesAll: Original (left), augmented (right). 15 instances of Amplitude Resizing + Window Warp applied to the original sample.

Slice and Shuffle crops the time series into four randomly selected 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. In many cases we will refer to this augmentation simply as slice.

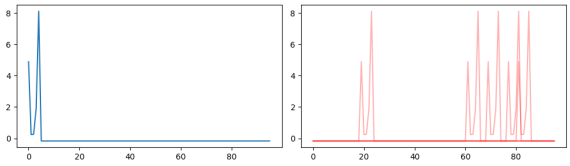


Figure 5.3: ElectricDevices: Original (left), augmented (right). 5 instances of Slice and Shuffle applied to the original sample.

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. In many cases we will refer to this augmentation simply as Gaussian or gauss.

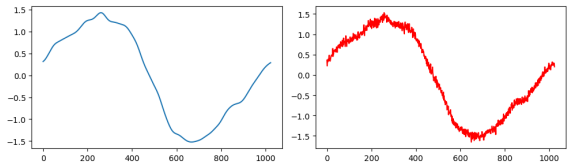


Figure 5.4: StarLightCurves: Original (left), augmented (right). One instance of Gaussian noise applied to the original sample.

Prior learning

Adopting the implementation from [6] we set the number of iterations T in the iterative decoding algorithm to 10 and use cosine as mask scheduling function (γ).

The bidirectional transformer is implemented with parameter presented in Table 5.1.

Training

We utilize the AdamW optimizer with batch sizes set to 128 for stage 1 and 256 for stage 2, an initial learning rate of 10^{-3}, cosine learning rate scheduler, and a

Hidden dimmension size	256
Number of layers	4
Number of heads	2
Feed-forward ratio	1

Table 5.1: Sizes of different components of the bidirectional transformer.

weight decay of 10^{-5}. Both stage 1 and stage 2 training procedures run for 1000 epochs.

5.1.2 Evaluation

For downstream classification, K-nearest neighbors (KNN) and Support Vector Machines (SVM) are implemented using scikit-learn, with $K = 5$ for KNN and a linear kernel for SVM.

SupervisedFCN is employed for calculating Inception Score (IS), Fréchet Inception Distance (FID), and Classification Accuracy Score (CAS). See Appendix B and C.2 of [5] for detailed information.

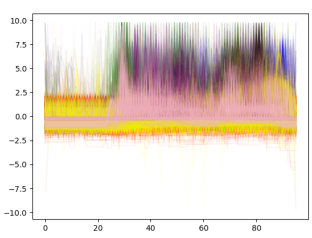
5.2 UCR Time Series Classification Archive

The evaluation of our model NC-VQVAE is done on a subset of the UCR Time Series Archive [53]. The UCR archive is a collection of 128 labeled datasets of univariate time series for 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.

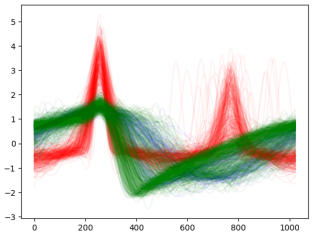
The subset selected for our experiments is presented in Table 5.2. We choose to test on a subset, rather than on the entire UCR Archive, due to computational limitations as well as to more thoroughly investigate the effect of our models and the role of augmentations. The subset is chosen such that they span a wide range of train set sizes, lengths, classes and type, while the class distributions have visually different characteristics which can be seen from table 5.2 and figure 5.5.

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

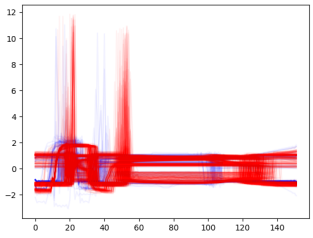
Table 5.2: The subset of the UCR Archive considered for our experiments.



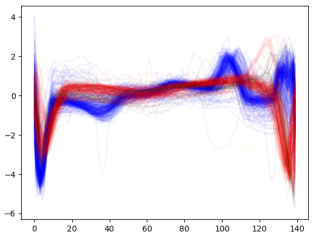
ElectricDevices



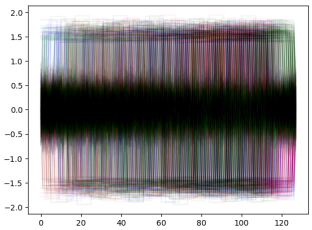
StarLightCurves



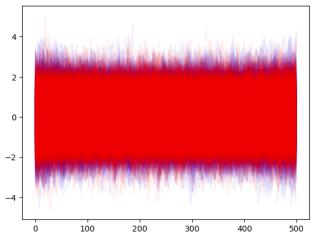
Wafer



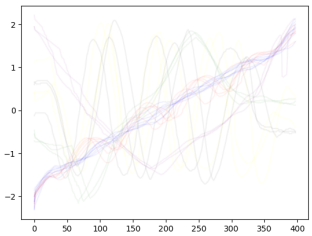
ECG5000



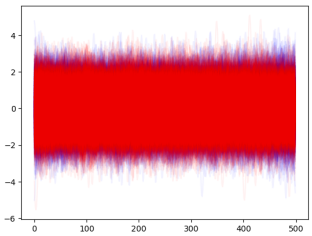
TwoPatterns



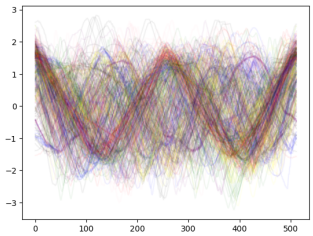
FordA



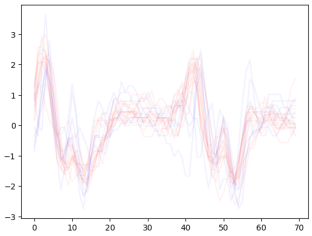
Symbols



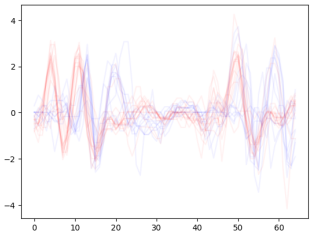
FordB



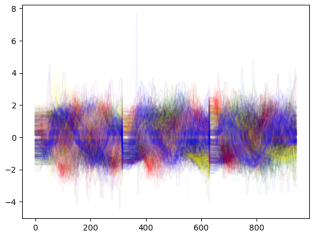
ShapesAll



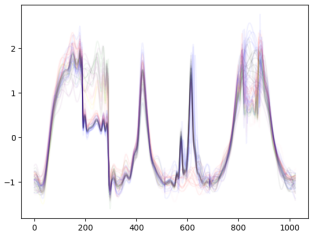
SonyAIBORobot-Surface1



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 thesis, we focus on two main objectives, which relates back to the research questions. Firstly, in Stage 1, we aim to determine whether NC-VQVAE can learn more expressive representations compared to VQVAE. Specifically, we investigate whether NC-VQVAE can achieve reconstruction performance on par with VQVAE while simultaneously enhancing downstream classification. In Stage 2, our interest lies in examining the impact of NC-VQVAE on synthetic sample quality.

Our evaluation process begins with assessing the tokenization models, focusing on their reconstruction capability and performance in downstream classification tasks. Subsequently, we then evaluate the performance of the generative models using metrics such as IS (Inception Score), FID (Fréchet Inception Distance), and CAS (Classification Accuracy Score). Additionally, visual inspections are conducted to provide further insights into the models' performance. A small ablation study on the effect of augmentation reconstruction weight on reconstruction and probe accuracy is presented towards the end.

6.1 Stage 1

In this section, we present the findings concerning the tokenization model. We find that some configuration of NC-VQVAE outperform naive VQVAE across the majority of datasets, both in terms of reconstruction quality and downstream classification, providing significant improvements in probe accuracy.

6.1.1 Reconstruction

We present the top 1 and mean reconstruction losses 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.

It's evident that NC-VQVAE achieves comparable reconstruction performance to the baseline model, and certain configurations even outperform the naive VQVAE in terms of mean reconstruction loss for 9 out of 13 datasets.

In Figure 6.1, we observe minimal differences in reconstruction loss across most datasets, regardless of SSL methods and augmentations. However, VIbCReg generally demonstrates slightly better performance compared to Barlow Twins, except for FordA. Additionally, the use of Gaussian augmentation introduces less regularization compared to the other augmentation methods, except for Slice and Shuffle on ECG5000. These findings suggest that incorporating a non-contrastive loss does not compromise the reconstruction capabilities compared to naive VQVAE.

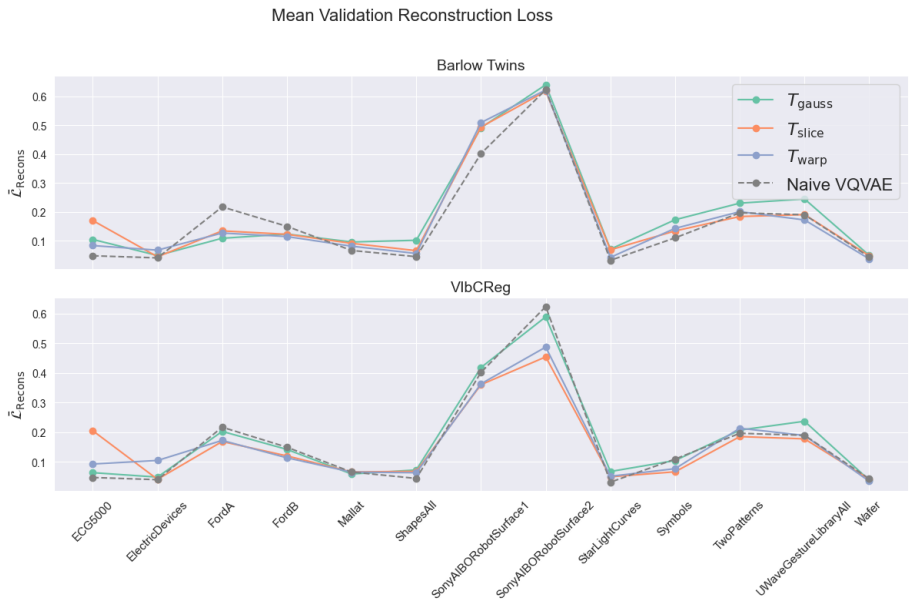


Figure 6.1: Mean validation reconstruction loss for the two models, compared to naive VQVAE

To explore the impact of the reconstruction loss of the augmented branch on validation reconstruction, a small ablation stydy was conducted, which is presented in Section 6.3. The results indicate that the validation reconstruction loss is robust to the specific value of the augmentation reconstruction weight, indicating a minor role played.

Through investigation into the development of validation reconstruction loss during training, we observed that the right configuration for NC-VQVAE can serve as a regularizer. This is illustrated in Figure 6.2 which depicts the development on FordA.

6.1.2 Classification

We present the mean and top 1 downstream classification accuracy in Table 6.3 and Table 6.4, respectively.

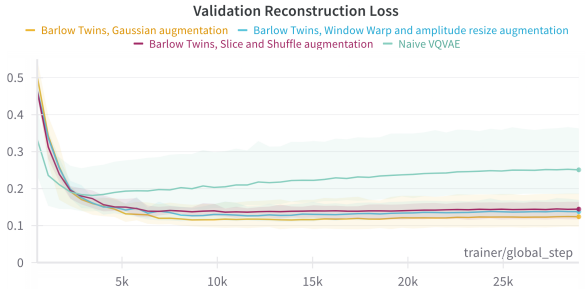


Figure 6.2: Development of the validation reconstruction loss for Barlow Twins and naive VQVAE on FordA during training. Averaged across all four runs.

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.

A clear improvement in probe accuracy is observed with NC-VQVAE compared to the naive VQVAE. Across 12 out of 13 datasets, a configuration of our model performs best, with the only exception showing a negligible one percent difference for both SVM and KNN. The most significant improvements are observed in FordA, FordB, Mallat, ShapesAll, TwoPatterns, and UWaveGestureLibraryAll.

In Figure 6.3, we observe that while the choice of augmentation has a substantial impact, all options lead to significantly improved probe accuracy across most datasets. Notably, both SSL methods yield comparable probe accuracies for a given augmentation, underscoring the importance of selecting appropriate augmentations.

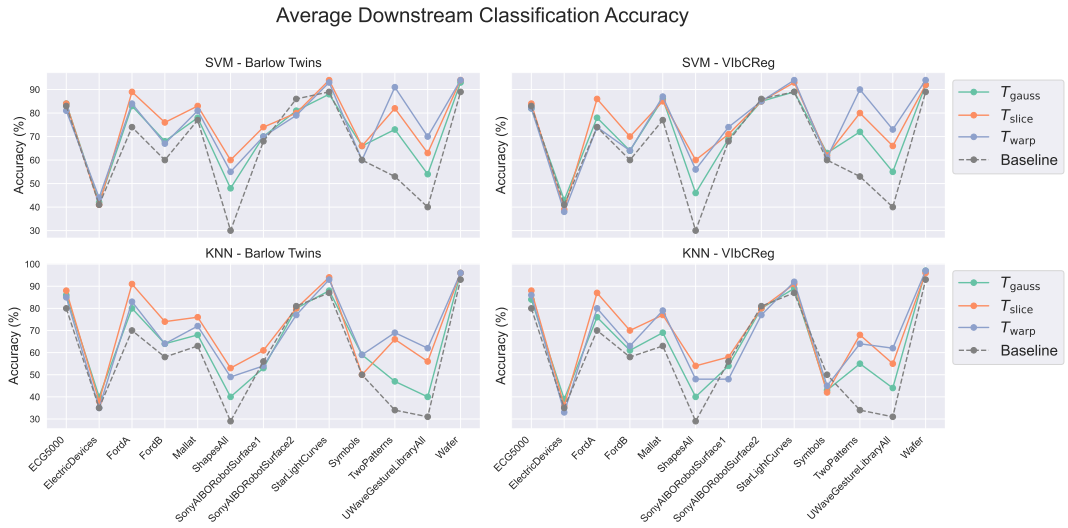


Figure 6.3: Mean probe accuracies.

Further analysis reveals that Slice and Shuffle, as well as Window Warp and Amplitude Resize, result in the most substantial accuracy gains, whereas Gaussian noise consistently yields less pronounced improvements. We hypothesize that since Slice and Warp often generate augmented views that differ considerably from the original, the SSL loss pushes the representations in different directions, potentially leading to better utilization of the latent space. Visualizations in Figure 6.4, 6.5 and 6.6 illustrate the effect of NC-VQVAE on the discrete latent representations of FordA, TwoPatterns and UWaveGestureLibraryAll. These visualizations demonstrate that representations learned with NC-VQVAE exhibit greater structure than those of the naive VQVAE, with similar samples, typically with the same label, clustered closer together in latent space. This suggests that the SSL loss introduces semantic information into the latent representations.

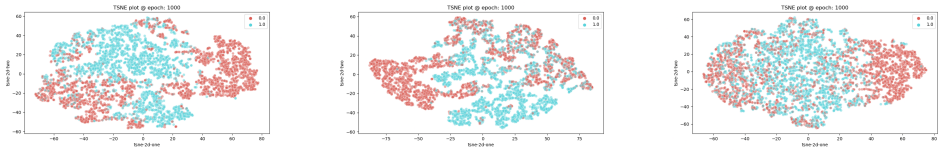


Figure 6.4: t-SNE plots of FordA. Barlow (left) and VIbCReg (center) with Slice and Shuffle, naive VQVAE (right).

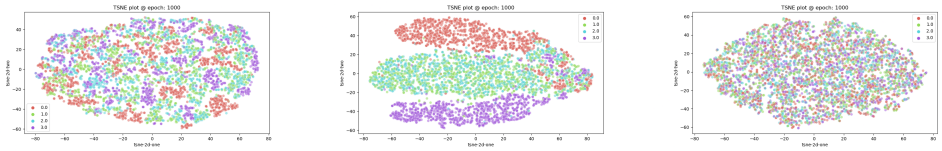


Figure 6.5: t-SNE plot of discrete latent representations from VIbCReg with Slice and Shuffle (left), Barlow Twins with Window Warp and Amplitude Resize (center) and naive VQVAE (right). Dataset is TwoPatterns.

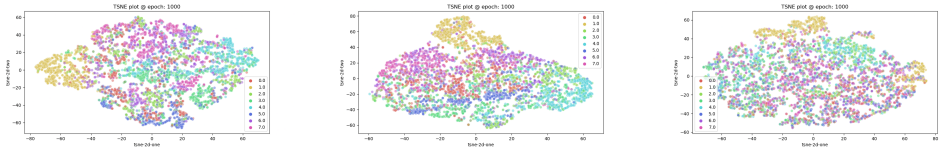


Figure 6.6: t-SNE plot of discrete latent representations from VIbCReg with Window Warp and Amplitude Resize (left), Barlow Twins with Window Warp and Amplitude Resize (center) and naive VQVAE (right). Dataset is UWaveGestureLibraryAll.

Summary of Stage 1

In summary, the results from stage 1 indicate that NC-VQVAE can reconstruct on par with the naive VQVAE, and in some cases improve the reconstruction loss, while substantially improving probe accuracy for most datasets. From the visual inspection of the discrete latent representations, we observed that NC-VQVAE effectively cluster samples from the same class. Addressing research question 1, we conclude that representations learned with NC-VQVAE are more expressive compared to the naive VQVAE, encoding more class-specific information without compromising reconstruction quality. Regarding research question 2, the choice of augmentation plays a pivotal role in the results, with warp and slice typically outperforming Gaussian augmentation, particularly in terms of probe accuracy. Additionally, significant variations across datasets support the hypothesis that the optimal choice of augmentations is highly dependent on the dataset.

6.1.3 Losses

We investigate some trends in the development of different loss terms during training. Notably, 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 6.7, we observe the typical pattern of the SSL loss during training.

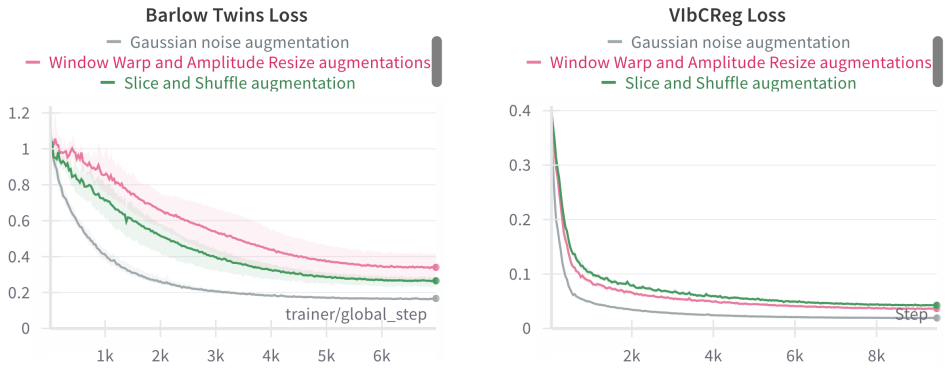


Figure 6.7: SSL loss during training on UWaveGestureLibraryAll. Averaged across four runs.

The relatively straightforward minimization of the losses might be attributed to the fact that the gaussian augmentation affects the samples in a predictable way. Additionally, we note that the VIbCReg loss declines more rapidly than Barlow Twins across all datasets.

Previously, in figure 6.3, we observed that Gaussian augmentation typically resulted in lower probe accuracy than warp and slice. In figure 6.8 we see that, for datasets where probe accuracy increased significantly, the augmentations that result in a more challenging SSL loss typically has higher downstream classification accuracy. Furthermore, we observe relatively stable patterns for specific augmentations, suggesting their significant impact on probe accuracy compared to variation in SSL loss.

Reconstruction losses during training are consistently minimized across models and augmentations, with augmented reconstruction loss slightly higher in models utilizing Slice and Shuffle. Differences in VQ loss primarily stem from the codebook loss, with VIbCReg consistently exhibiting more effective minimization than Barlow Twins. Gaussian augmentation present the easiest minimization challenge, followed by Window Warp and Slice and Shuffle.

Both VIbCReg and Barlow Twins, when coupled with Gaussian augmentation, consistently perform comparably to the naive VQVAE in terms of VQ loss during

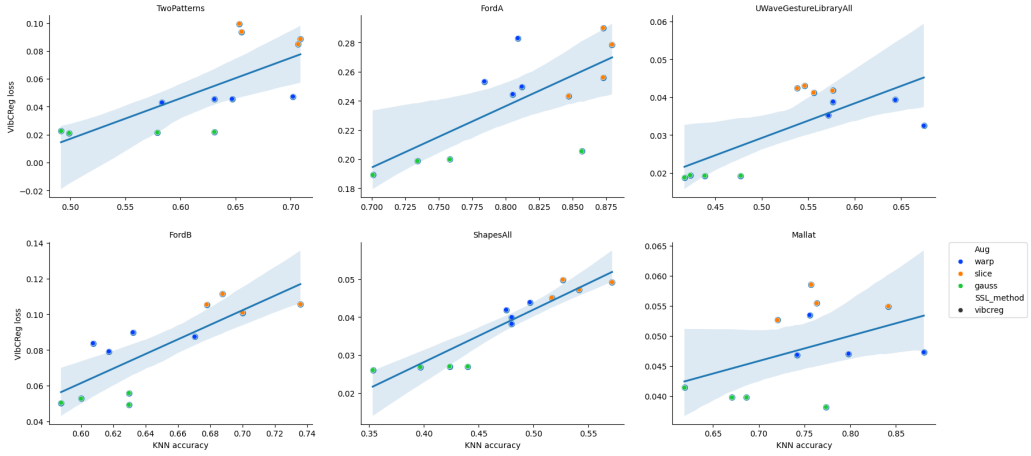


Figure 6.8: KNN accuracy plotted against VIbCReg loss. Each point correspond to a single run of the model. Similar tendency is shown for Barlow Twins.

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.

6.2 Stage 2

The generative quality of our models is assessed using FID, IS, and CAS, with all results presented in this section. However, it's important to note that for datasets with limited samples or few samples per class, the generative scores should be interpreted cautiously. Both the classifier and evaluation metrics rely on a sufficient number of samples to ensure reliability. Therefore, visual inspection is prioritized for such cases.

6.2.1 FID and IS

Tables 6.5 and 6.6 present the top 1 and mean scores across the four runs for both FID and IS. Analysis of the tables reveals that our model achieves higher IS scores for 12 out of 13 datasets and lower FID scores for 10 out of 13 datasets in both mean and top 1.

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 6.9, an overview of the results highlights that both Barlow Twins and VIbCReg generally yield better samples than the naive VQVAE in terms of FID and IS. There is a difference in VIbCReg and Barlow Twins, in that VIbCReg is more robust to the particular choice of augmentation, which is particularly evident when looking at the Slice and Shuffle augmentation. Furthermore, the use of Gaussian augmentation leads to the most significant improvements across most datasets. The high IS scores suggest that NC-VQVAE captures the conditional distributions more accurately than the naive VQVAE. The improved FID scores indicate that the synthetic samples more closely resemble the test data. This will be explored further in Section 6.2.5. 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. However, the discrete latent representations from NC-VQVAE offer additional class-specific information, as evidenced by the improved

downstream classification accuracy observed in stage 1. This supplementary class-specific information appears to assist the prior learning process in capturing class conditional distributions.

Figure 6.9: 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 important to note that the FID and IS scores are calculated using the SupervisedFCN, which is trained on the UCR Archive. Consequently, there may be a bias towards samples that mimic the training data.

6.2.2 CAS

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 observe that some configuration of NC-VQVAE outperform the naive VQVAE on all datasets. Additionally, VIbCReg with gaussian augmentation outperforms the baseline on 12 out of 13 datasets. Generally the difference in CAS is small for the different models, with the exception of Mallat, where VIbCReg with Gaussian augmentaion leads to an improvement of 0.437 compared to baseline. This spesific case will be investigated further in Section 6.2.5.

6.2.3 Prior Loss

In Figure 6.11, we illustrate the evolution of the validation prior loss during training. We observe that naive VQVAE outperforms all configurations of NC-VQVAE, and this pattern observed is representative for most datasets. The exception being SonyAIBORobotSurface1 and 2, where Barlow Twins with warp augmentation exhibits slightly better performance. These observations suggest that minimizing the validation prior loss does not correlate with improved generative scores.

6.2.4 The influence of stage 1 on stage 2

We aim to address research questions 3 and 4, which focus on how expressive representations influence the quality of synthetic samples and the role of augmentations. To explore this, we examine the relationship between probe accuracy and FID and IS metrics. Specifically, we focus on datasets where NC-VQVAE demonstrated a significant increase in probe accuracy, namely FordA, FordB, Mallat, ShapesAll, TwoPatterns, and UWaveGestureLibraryAll.

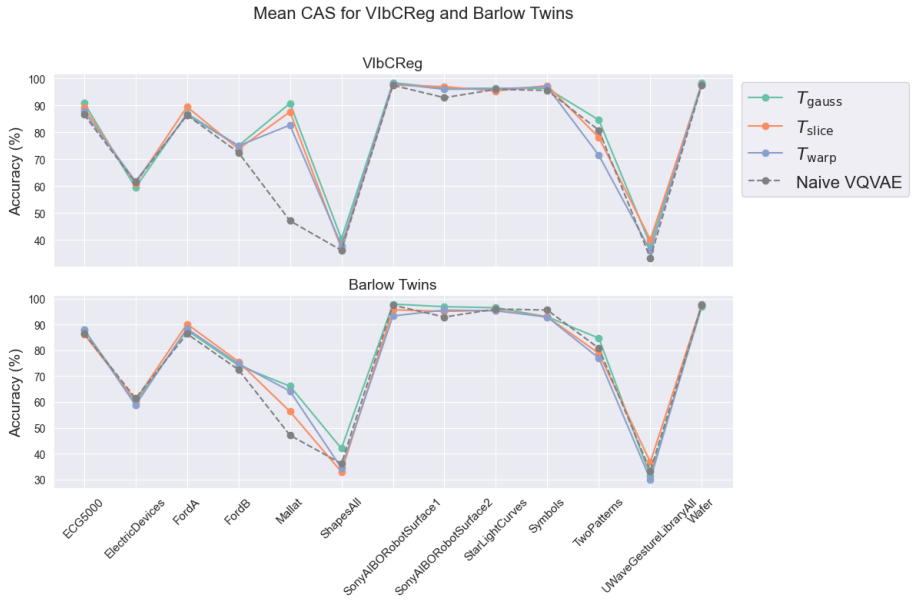


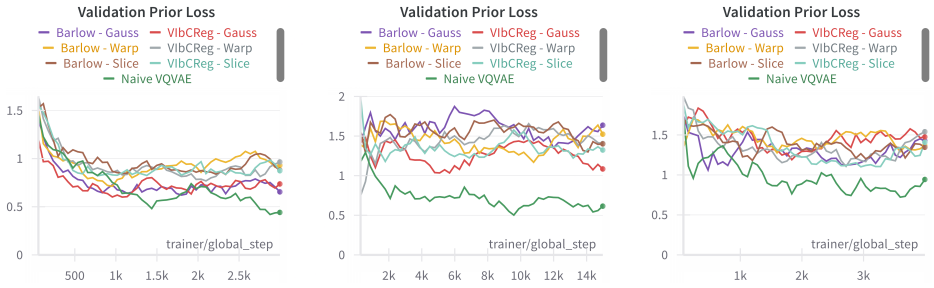
Figure 6.10: Mean CAS across all datasets.

In Figure 6.12 and 6.13, we present scatterplots of KNN against FID and IS, respectively, along with the least square regression line. Similar trends are observed in the corresponding plots with SVM accuracy. From Figure 6.12, we observe a trend, where higher probe accuracy correlates with higher IS. Upon closer inspection, we see a pattern of the prominent effect of augmentations. For each specific augmentation, the correlation between KNN and IS is close to 0. It appears that augmentations leading to higher KNN accuracy tend to result in higher IS scores, though the pattern is not consistent across all datasets. In Figure 6.13, we notice that the specific augmentation serves as a better indicator of FID score than KNN accuracy. Both figures also illustrate the sensitivity of model performance to initialization, particularly in terms of probe accuracy.

6.2.5 Visual inspection

In the following we present generated samples from naive VQVAE and NC-VQVAE for selected datasets. The ground truth, both test and train, are additionally included in order to better make sense of the IS, FID and CAS scores.

Some datasets, such as FordA and FordB, are poorly suited for this type of visual inspection, as illustrated previously in Figure 5.5. As a result, the selection of datasets is primarily based on how well they lend themselves to this type of presentation. For each figure in the following sections, 50 samples are generated from each model. For the ground truth, we plot a subset of 50 randomly selec-



ShapesAll FordA TwoPatterns

Figure 6.11: Validation prior loss during training for selected datasets. Averaged across all runs. Results averaged using exponential moving average with decay 0.7.

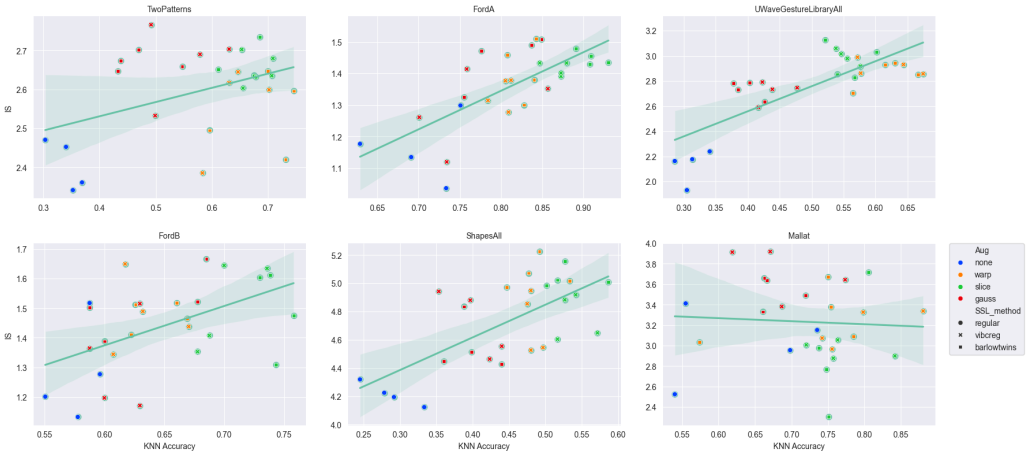


Figure 6.12: KNN plotted against Inception Score on the subset of datasets with significant improvement in probe accuracy.

ted samples, or the entire set if the dataset contain less than 50 samples. For the datasets with complex distribution, or many classes, it is very difficult to visually asses the unconditional distribution. Thus, we mainly provide class conditional samples. We surprisingly only observe minor differences in the generated samples from NC-VQVAE trained with different augmentations.

Typically naive VQVAE has more difficulty with capturing the global consistency of the samples when training samples are scarce and diverse, as seen on ShapesAll and Symbols. In contrast, our method will tend to overfit in these cases. The overfitting issue is most prominent in the class conditional distributions.

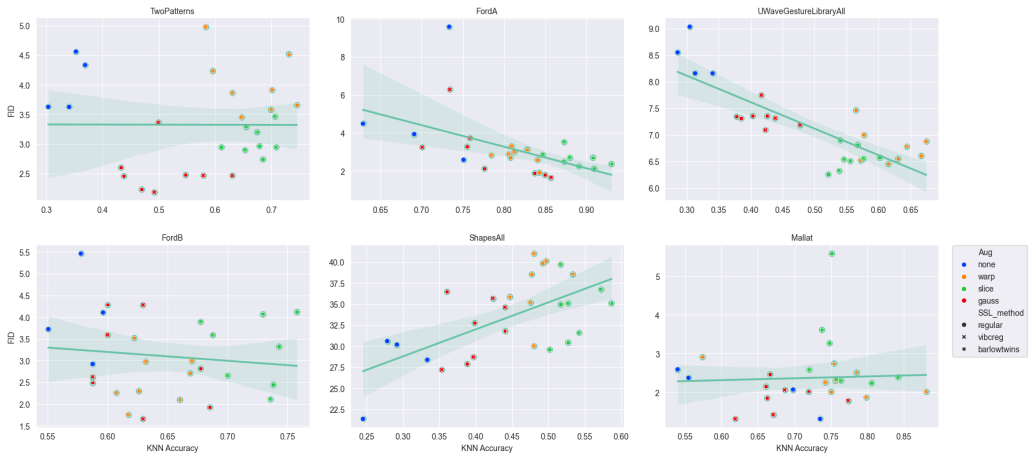


Figure 6.13: KNN plotted against Fréchet Inception Distance on the subset of datasets with significant improvement in probe accuracy.

ECG5000

In Figure 6.14 we present generated samples from naive VQVAE and NC-VQVAE trained with Window Warp and Amplitude Resize augmentations on ECG5000.

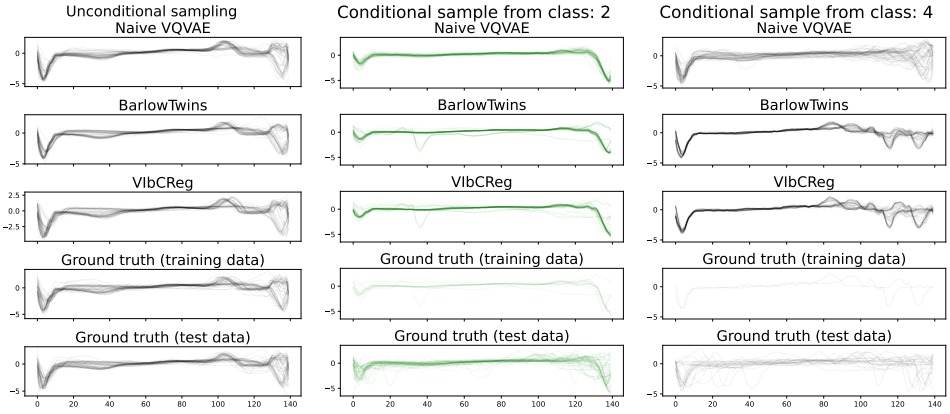


Figure 6.14: Dataset: ECG5000. Barlow and VIbCReg both trained with window warp and amplitude resize augmentations. 50 samples from each model.

We see some evidence that VIbCReg maintains more variability than Barlow Twins, while both have good mode coverage. In class 4, where the training data only consists of 2 samples, both Barlow Twins and VIbCReg catches the pattern, while producing some variation which resemble the training samples. The naive VQVAE samples does not capture this distribution, but when compared to the test data, it is more similar. This could explain the minor increase in CAS for NC-VQVAE. Looking at the unconditional sample, it is not evident why naive VQVAE

performs better in terms of FID score than NC-VQVAE.

Mallat

Mallat is a simulated dataset, where the classes have very little variability and training and test distribution are almost indistinguishable, except for sample size.

We observe that VIbCReg is superior in capturing the variability, compared to Barlow Twins and naive VQVAE. This is most evident in the first 300 timesteps of class 5 in Figure 6.15. Looking as class 7, we see Barlow Twins completely collapsing, essentially producing the same sample over and over. These figures explain the significant increase in CAS seen in Figure 6.10, particularly for VIbCReg. It also explains why VIbCReg with Gaussian augmentation both increases IS and reduces FID.

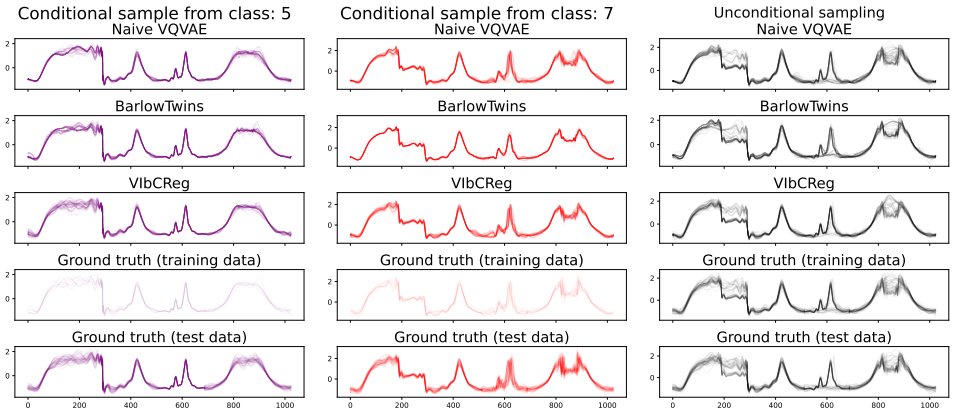


Figure 6.15: Class conditional distribution for some selected classes of Mallat, in addition to unconditional samples. Barlow and VIbCReg both trained with Gaussian augmentation.

By inspecting the PCA plots of both data space and the discrete latent representations of samples from Mallat, compared to synthetic samples form VIbCReg and Barlow Twins in Figure 6.17, we see a clear case of representation collapse for Barlow Twins. We hypothesize that the variance term in VIbCReg assists in maintaining variability in the representations. Making it more effective in avoiding this type of collapse.

Symbols

The Symbols dataset consists of several distinct, but simple, patterns. The dataset is very small, where each class contsists of less than 5 training samples.

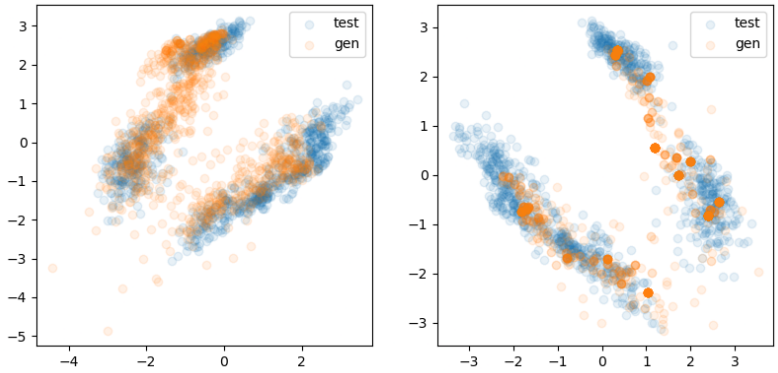


Figure 6.16: PCA of discrete latent representation from Mallat. Both VIbCReg (left) and Barlow Twins (right) are trained with gaussian augmentation.

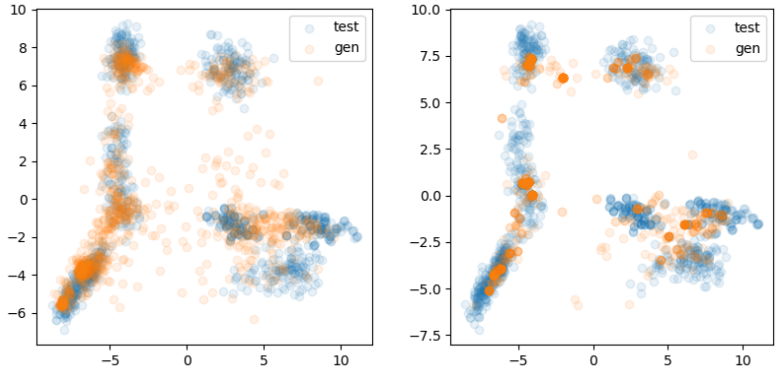


Figure 6.17: PCA of generated time series from Mallat. Both VIbCReg (left) and Barlow Twins (right) are trained with gaussian augmentation.

In Figure 6.18, particularly the unconditional sample, we see that naive VQVAE does not capture the entire underlying distribution, some classes are not represented/not recognizable, while global consistency for the sinusoids are poor, particularly towards the end. In class 3, we observe that both Barlow Twins and VIbCReg mimic the training data to a large degree. At first glance the naive VQVAE looks to produce the most desirable distribution, but upon closer inspection we see an excessive amount of noise and lack of consistency.

The IS on Symbols, for both VIbCReg and BarlowTwins, is substantially higher than naive VQVAE. While this is not surprising after inspecting the samples, it exposes an issue with the IS metric. It fails to take intraclass diversity into account, and is therefore oblivious to overfitting.

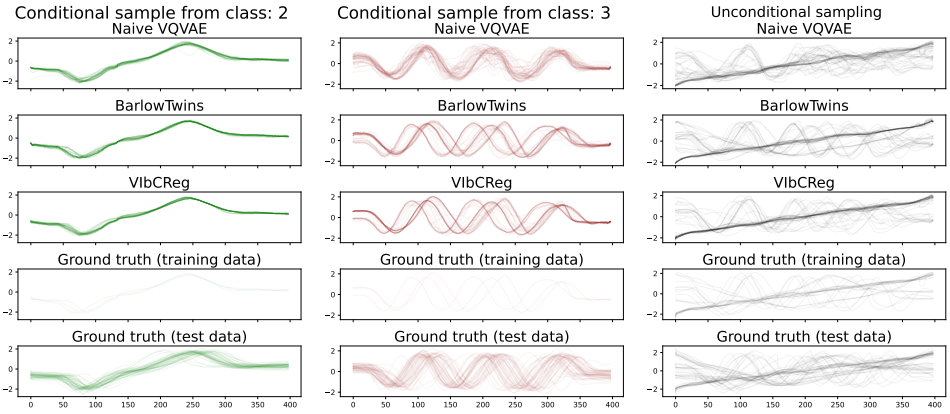


Figure 6.18: Class conditional distribution for some selected classes of Symbols. Barlow and VIbCReg both trained with gaussian augmentation.

ShapesALL

The dataset ShapesAll consists of 60 classes, with 10 samples within each class. Each class has distinct patterns, with varying complexity.

In Figure 6.19, we observe clearly that naive VQVAE struggles with capturing the global consistency of the samples. We too observe that Barlow Twins mimic the training slightly more closely than VIbCReg, which provides some insight as to why Barlow Twins improves CAS by about 10 percent compared to naive VQVAE. Both Barlow Twins and VIbCReg improve IS, but fail to improve FID. By inspecting the samples, it is not evident why NC-VQVAE fails to improve FID. As FID is calculated from unconditional samples, the issue is most likely due to a issue not observable from the conditional samples. As the dataset has 60 classes, it is probable that an insufficient number of synthetic samples were generated to represent the large variability in the unconditional distribution.

In Symbols and ECG5000 we saw that NC-VQVAE can overfit when there are very few samples in a class. Reassuringly, the mode coverage is quite good, and similar tendencies are observed for ShapesAll as well.

UWaveGestureLibraryAll

The dataset UWaveGestureLibraryAll contains time series with distinct discontinuities and sharp changes in modularity. As noted in [1], such datasets are challenging to model.

In Figure 6.20 a selected subset of classes are illustrated. We observe upon close inspection that VIbCReg maintains variability in the samples to a greater degree than Barlow Twins, as well at slightly better capturing the dead-spots follow-

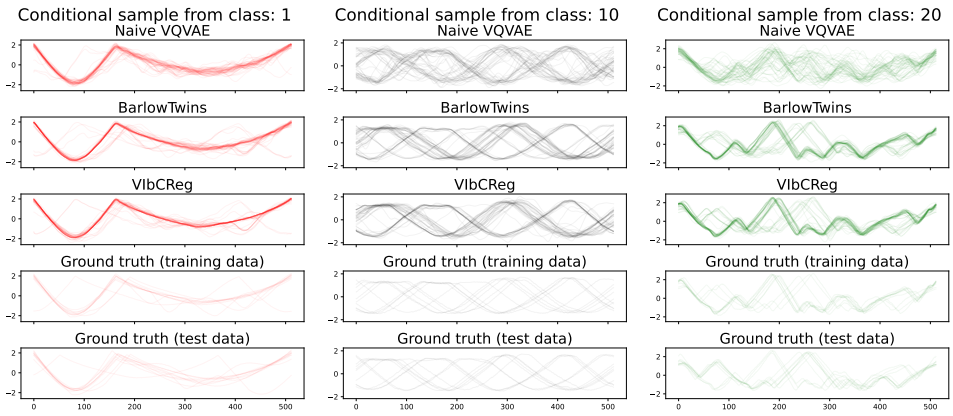


Figure 6.19: Class conditional distribution for some selected classes of ShapesAll. Barlow and VIbCReg both trained with gaussian augmentation.

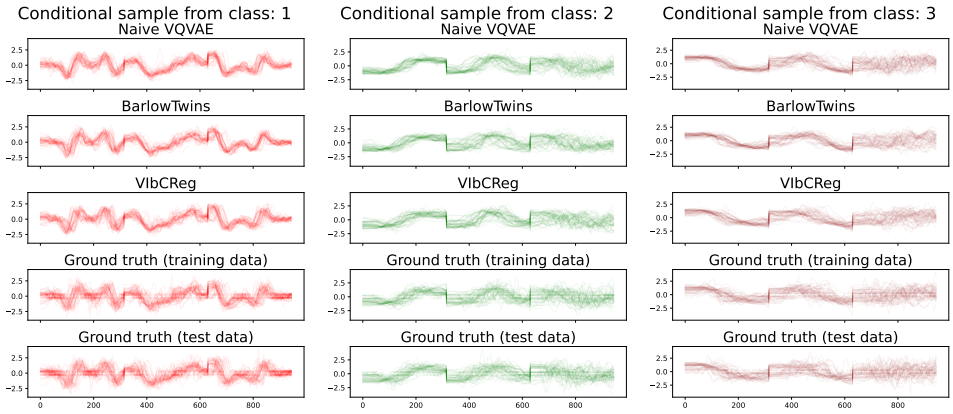
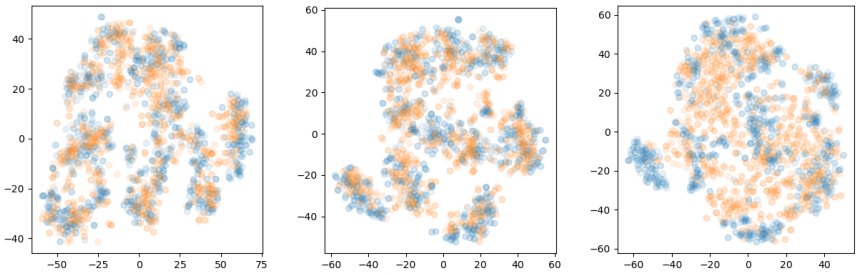


Figure 6.20: Class conditional distribution for some selected classes of UWaveGestureLibraryAll. Barlow and VIbCReg both trained with window warp and amplitude resize augmentations.

ing the discontinuities. By investigating the t-SNE plots in Figure 6.21 it becomes more evident that NC-VQVAE captures the distribution better than naive VQVAE.

Summary of Stage 2

In summary, the results from stage 2 suggest that NC-VQVAE offers advantages over the naive VQVAE in capturing conditional distributions, as indicated by higher IS and CAS scores. Additionally, NC-VQVAE demonstrates improved sample quality compared to ground truth data, with lower FID scores. Visual inspections further reveal that NC-VQVAE achieves better mode coverage and captures global sample consistency to a greater extent than its naive counterpart. Additionally, from the visual inspection we observe that the quality of the generated samples often are much higher, despite a moderate decrease in FID.



(a) VIbCReg **(b)** Barlow Twins **(c)** Naive VQVAE

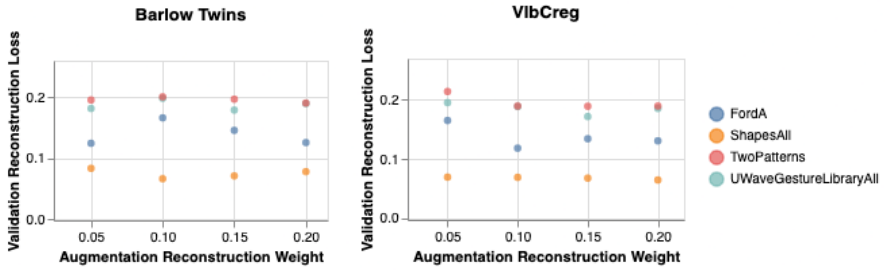
Figure 6.21: t-SNE of generated (orange) and test data (blue). Dataset: UWaveGestureLibraryAll

Addressing research question 3, we conclude that the expressive representations learned from NC-VQVAE contribute to learning class-specific details and enhancing the quality of synthetic samples. However, it's worth noting that NC-VQVAE is prone to overfitting when faced with small datasets or classes with few samples, whereas the naive VQVAE struggle to capture global consistencies effectively.

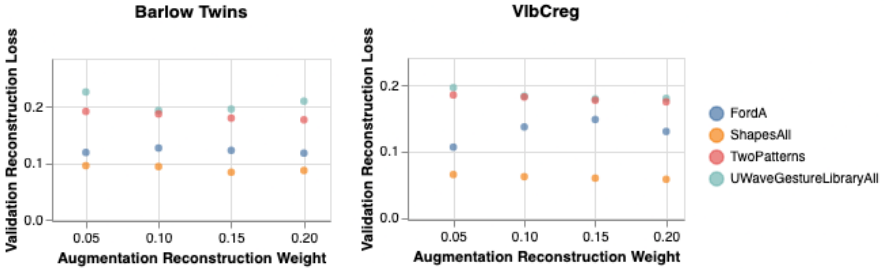
Regarding research question 4, we observe that the generative performance is less sensitive to the choice of augmentations compared to the downstream classification accuracy observed in stage 1. Nonetheless, Gaussian noise yields the least variability in performance compared to the baseline, outperforming the naive VQVAE consistently in terms of IS, FID, and CAS metrics.

6.3 Ablation Study

During the development of our model, we investigated the effect of the augmentation reconstruction weight, ζ, both on validation reconstruction and downstream classification. This experiment was carried out on a small scale, focusing on two sets of augmentations: Window Warp and Amplitude Resize, and Slice and Shuffle. We conducted our experiment on a subset of the UCR Archive consisting of FordA, ShapesAll, TwoPatterns and UWaveGestureLibraryAll. The tokenization model underwent training for 250 epochs, with all other parameters maintained as in the main experiment setup. We explored the effect of ζ across a range of values, specifically $\zeta \in \{0.05, 0.10, 0.15, 0.20\}$. The influence on reconstruction is visualized in Figure 6.22, while the impact on KNN and SVM accuracy is presented in Figure 6.23 and 6.24, respectively.

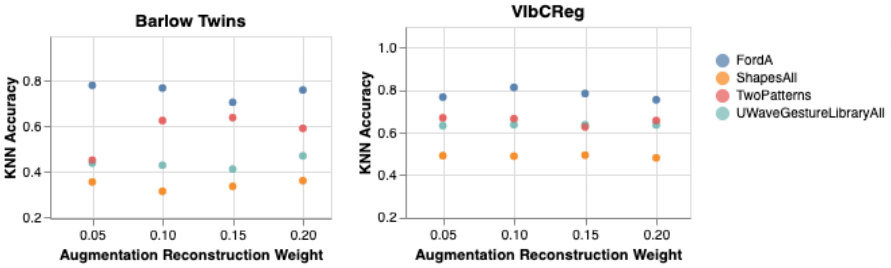


Augmentation: Window Warp and Amplitude Resize.

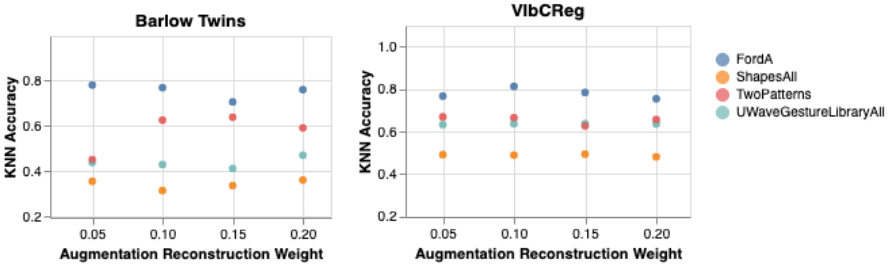


Augmentation: Slice and Shuffle.

Figure 6.22: The effect of the augmentation reconstruction weight on validation reconstruction. Results averaged across 2 runs.

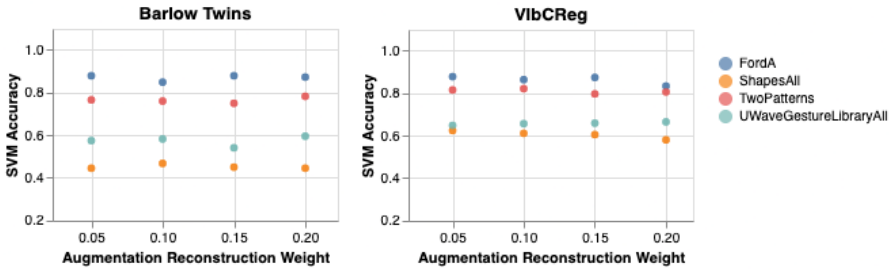


Augmentation: Window Warp and Amplitude Resize.

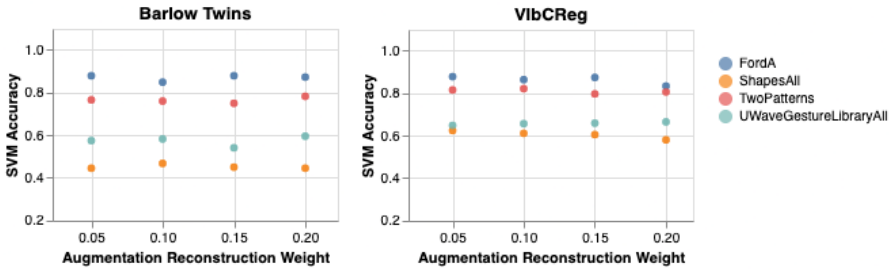


Augmentation: Slice and Shuffle.

Figure 6.23: The effect of the augmentation reconstruction weight on KNN accuracy. Results averaged across 2 runs.



Augmentation: Window Warp and Amplitude Resize.



Augmentation: Slice and Shuffle.

Figure 6.24: The effect of the augmentation reconstruction weight on SVM accuracy. Results averaged across 2 runs.

Observing Figure 6.22, we note that the reconstruction loss remains relatively stable across most datasets, exhibiting robustness to variations in the augmentation reconstruction weight. However, FordA displays some degree of variability, and the variability is augmentation dependent. This suggests that when optimization of model performance could benefit from jointly optimizing choice of augmentation and augmentation reconstruction weight.

Examining Figure 6.23 and 6.24, we observe that augmentation reconstruction weight affects KNN to a larger degree than SVM, indicating that the weight introduces differences in local structure of the discrete latent space, while the global structure is less affected. Furthermore, it's evident that VIbCReg consistently produces high probe accuracies and demonstrates greater robustness compared to Barlow Twins across different augmentation reconstruction weights.

6.4 Discussion

Difficulty in assessing generative samples

The difficulty in visually assessing generative performance for datasets with complicated distributions is still highly relevant. One would like to be able to confidently rely on the evaluation metrics when assessing the model performance, but as observed, the generative metics are flawed. We observe a discrepancy in the

improvement in generative scores and the quality observed in the visual inspection.

Many datasets in the UCR Archive and our selected subset is heavily imbalanced. As the SupervisedFCN used to evaluate FID and IS is trained on these imbalanced datasets, they may underrepresented the quality of synthetic samples from the minority classes. We have from the visual inspection seen that NC-VQVAE is superior to naive VQVAE in its ability to capture the conditional distributions, even then the sample size is very small. Additionally, the SupervisedFCN may be better at recognizing and evaluating features of the dominant classes and worse at recognizing features of minority classes. This can lead to biased scores that favor synthetic samples resembling the overrepresented classes.

The current metrics are dependent on high quality data with large sample sizes, balanced classes etc. There is currently a lack of large high quality datasets in the time series domain, and something akin to ImageNet for time series would be beneficial for the community.

The role of augmentations

We observe that both probe accuracies and generative scores are dependent on the particular choice of augmentations. The probe accuracies are generally better when augmentation are clearly different from the original view, as for warp and slice, while the generative scores has a tendency to follow the opposite pattern, though not as clearly. Additionally, we have observed that the optimal choice of augmentations is highly dataset dependent, and a systematic analysis of different augmentations on specific datasets is needed to understand the dynamics better.

Temporal vs frequency influence of augmentations

The Gaussian augmentation introduces a HF component, though its influence in the time domain is visually clear, its effect on the spectrograms is minor, as the LF components typically has much larger magnitude. Window Warp and Amplitude Resize mainly alter LF components, often changing the exact location of dominant frequencies on the time axis, but not their order. The effect of Slice and Shuffle is variable, but has a tendency to create sharp discontinuities, which in many cases introduce a significant HF component.

We assessed and chose our different sets of augmentations based on their effect on the temporal representation of the time series. As we model the time frequency domain, future work should more thoroughly investigate the effects of augmentation on spectrograms. Additionally, all models considered compress the input only along temporal axis in the encoder, which in a sens puts more emphasis in the frequency components rather than the exact temporal structure.

Differences in Barlow Twins and VIbCReg

From the visual inspections we observed that VIbCReg maintain the variability in the generated samples a bit better than Barlow Twins. We hypothesize that it is due to the variance term present in VIbCReg, and wonder whether increasing its weight might assist in producing more diverse samples.

In terms of the generative scores, VIbCReg is more robust to choice of augmentations, highlighted by the their different response to the Slice and Shuffle augmentation. This can likely be attributed the difference in variance/covariance regularization on the two models. Since VIbCReg regularizes each branch separately, it responds better to situations where the input of each branch differs significantly.

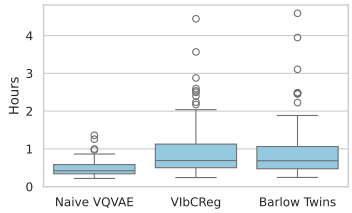
VIbCReg generally demonstrated slightly better performance in terms of validation reconstruction than Barlow Twins. In the ablation study we observed that VIbCReg was more robust to augmentation reconstruction weight, both in terms of the validation reconstruction and downstream classification.

Overfitting problem

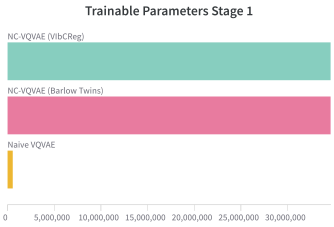
We have observed on multiple occasions that when the sample size is small and the patterns in the data are simple NC-VQVAE has a tendency to overfit, and memorize the training data to a large degree. In our experiments we have not employed any form of dropout or other regularization techniques when training on small datasets, therefore this issue is to be expected. As the naive VQVAE in the same cases fails to capture the global consistency, we see our model as a step in the right direction, but stress the need for regularization. Additionally, investigating prior learning models and sampling procedures tailored for the more expressive representations, leveraging the encoded semantic information, could be beneficial.

Training Time and Model Size

The addition of the self-supervised loss in NC-VQVAE comes at a cost in terms of model size, and consequently training time. The added complexity is a result of the high dimensionality of the projector used in both Barlow Twins and VIbCReg. As the projector is discarded after stage 1, the increased complexity only affect training of the tokenization model. Examining Figure 6.25, we observe a significant increase in training time and trainable parameters.



(a) Training time distributions for Stage 1.



(b) Number of parameters in Stage 1 for models trained on Wafer.

Figure 6.25: Overview of training time and model size for Stage 1.

Chapter 7

Conclusion

We are able to simultaneously reconstruct well and significantly improve downstream classification accuracy. Additionally, from the t-SNE plots 6.4, 6.5 and 6.6, we observe that NC-VQVAE can efficiently perform clustering as well. 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.

NC-VQVAE is more adept at mimicking training data and less likely to overlook underrepresented classes. However, for datasets with limited training samples, our model can be prone to overfitting. Some challenges associated with TimeVQVAE remain, particularly in modeling data with sharp differences in modularity.

While issues persist, we believe our model represents a step forward. The representations learned by NC-VQVAE are more expressive than those from the naive VQVAE, showing that multiple objectives can be optimized without sacrificing reconstruction capability. These expressive representations facilitate capturing the semantics of the training data and producing synthetic samples with better global consistency.

In summary, NC-VQVAE demonstrate the ability to classify accurately and perform effective clustering while maintaining reconstruction capabilities, as well as enhancing generative quality.

7.1 Further work

There are many exiting directions this work can be taken further. In [54] they underscore 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. An easy to use library of time series augmentations would be very beneficial to the community.

As we considered a simplified version of TimeVQVAE, it is natural to investigate effective ways of incorporating the HF-LF split. A possible approach could be to concatenate HF and LF representations and pass them through the projector. This would also open the possibility of applying augmentations tailored for HF and LF. In the same direction of sort of obvious extension, would be to further optimize the relationship between the reconstruction loss on the augmented branch and choice of augmentations, and understanding their interplay. Undergoing a more thorough analysis on the effect of the different losses, would be interesting.

It is of interest to better understand the geometry and topology of the discrete latent representations. A possible direction of future research could be to further investigate the effect of different augmentations and SSL methods by applying topological data analysis techniques.

The discrepancy between validation prior loss and generative quality observed with NC-VQVAE is not understood. NC-VQVAE consistently produces samples with higher fidelity and improved generative scores, but it fails to model the latent space in such a way that the validation prior loss is minimized. It would be very interesting to investigate further, and one possibility would be to consider the attention maps and try to deduce relationships between the tokens.

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] D. o. E. United Nations and S. A. .-. S. Development, *Transforming our world: The 2030 agenda for sustainable development*, General Assembly, 2015. [Online]. Available: https://sdgs.un.org/2030agenda.

[3] K. Crawford, *Generative ai is guzzling water and energy*, https://www.nature.com/articles/d41586-024-00478-x.pdf, 2024.

[4] P. Li, J. Yang, M. A. Islam and S. Ren, *Making ai less "thirsty": Uncovering and addressing the secret water footprint of ai models*, 2023. arXiv: 2304.03271 [cs.LG].

[5] 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.

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

[7] 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].

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

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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].

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

[18] 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.

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

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

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

[22] 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].

[23] 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].

[24] 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].

[25] 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].

[26] 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].

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

[28] 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.

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

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

[31] 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].

[32] K. Gupta, T. Ajanthan, A. van den Hengel and S. Gould, *Understanding and improving the role of projection head in self-supervised learning*, 2022. arXiv: 2212.11491 [cs.LG].

[33] G. Mialon, R. Balestriero and Y. LeCun, *Variance covariance regularization enforces pairwise independence in self-supervised representations*, 2024. arXiv: 2209.14905 [cs.LG].

[34] F. Träuble, E. Creager, N. Kilbertus, F. Locatello, A. Dittadi, A. Goyal, B. Schölkopf and S. Bauer, *On disentangled representations learned from correlated data*, 2021. arXiv: 2006.07886 [cs.LG].

[35] 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].

[36] 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].

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

[38] 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.

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

[40] 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].

[41] 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].

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

[43] 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].

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

[45] 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.

[46] 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].

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

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

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

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

[51] *Idun cluster*, https://www.hpc.ntnu.no/idun/, 2024.

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

[53] 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.

[54] 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].