

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

Johan Vik Mathisen

June 4, 2024

Abstract

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

Sammendrag

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

Contents

Abstract	iii
Sammendrag	v
Contents	vii
1 Introduction	1
1.1 Acknowledgements	1
1.2 What is this thesis about?	2
1.3 Motivation	3
1.4 Research questions	4
1.5 Sustainability impact	4
1.6 Collaboration and AI Declaration	4
2 Theoretical Background	7
2.1 Neural Network	7
2.1.1 Training Neural Networks	9
2.2 Convolutional Neural Network	10
2.2.1 The convolution operation	11
2.2.2 Pooling	12
2.2.3 Architecture	13
2.2.4 Transposed Convolutional Networks	15
2.3 Representation Learning	16
2.3.1 Why do we care about representation learning?	17
2.3.2 What is a good representation?	17
2.3.3 How does one evaluate representations?	18
2.4 Transformers	19
2.4.1 The Attention Mechanism	20
2.4.2 Architecture	21
2.5 Self-Supervised learning	23
2.5.1 Siamese Architecture-based SSL	24
2.5.2 Masked modelling	26
2.6 Vector Quantized Variational Autoencoder (VQVAE)	26
2.6.1 Autoencoder (AE)	26
2.6.2 Variational Autoencoder (VAE)	27
2.6.3 VQVAE	30
2.7 Evaluation metrics	33
2.7.1 Generative model	33

3 Related Work	37
3.1 MaskGIT	37
3.1.1 Masked Visual Token Modeling (Prior learning)	38
3.1.2 Iterative Decoding (Sample generation)	38
3.2 TimeVQVAE	39
3.2.1 Tokenization	40
3.3 SSL	41
3.3.1 Barlow Twins	42
3.3.2 VIbCReg	43
4 Methodology	45
4.1 Proposed model: NC-VQVAE	45
4.1.1 Stage 1: Tokenization	45
4.1.2 Stage 2: Prior learning	47
5 Experiments	49
5.1 Initial Experimentation and Model Development	49
5.2 Main Experiments	50
5.2.1 Implementation details	50
5.2.2 Evaluation	53
5.3 UCR Time Series Classification Archive	53
6 Results and Discussion	57
6.1 Stage 1	57
6.1.1 Reconstruction	58
6.1.2 Classification	60
6.1.3 Losses	62
6.2 Stage 2	64
6.2.1 FID and IS	64
6.2.2 CAS	66
6.2.3 Prior loss	67
6.2.4 The influence of stage 1 on stage 2	67
6.2.5 Visual inspection	68
6.2.6 Augmentation Reconstruction Weight	74
6.3 Discussion	74
6.3.1 Differences in Barlow Twins and VIbCReg	74
6.3.2 Overfitting problem	74
6.3.3 Temporal vs frequency influence of augmentations	75
7 Conclusion	77
7.1 Conclusion	77
7.2 Further work	78
7.3 Ethical Considerations	78
Bibliography	79
A Additional Material	83

Chapter 1

Introduction

TODO: How the order of things should be is not clear to me yet. Would like some pointers.

1.1 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, but the margins are sadly too narrow to include you all.

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. My roommates, Pernille and Eldrun, for being devilishly funny and handling me so well as i go full goblin mode. Preben and Elias, 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 cult, be it on Samfundet, in a band or some part of NTNUI. I can wholeheartedly say i found mine, and to all the members, such loving, crazy and interesting people, thank you. See you on top of some remote peak very soon.

1.2 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, 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 our research, we are interested in generative modelling of time series. Time series, as the name kinda spoils, is data with a time component. Time series data encompasses diverse measurements such as weather conditions, patient heart rates, economic indicators (e.g., GDP, inflation, unemployment rates), energy usage, social media activity, and sales figures, all recorded at successive points in

time. With the ever forward pointing arrow of time, it is really no surprise that these types of data are everywhere.

In this thesis we attempt, and in many ways succeed, to find a better perspective on time series, 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 clever 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 informative, 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.

1.3 Motivation

As we briefly mentioned above, 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 task with applications 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, the ability to generate plausible and diverse time series is highly valuable.

Unsupervised representation learning offers a promising avenue 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 rich representations of time series data holds immense potential for advancing research and solving real-world problems.

1.4 Research questions

Our research in this thesis revolves around the following four questions

- RQ1:** Will 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?

1.5 Sustainability impact

TODO: This is under construction

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

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

Monopoly of the largest tech companies.

The massive copyright disputes in LLMs.

Questionable privacy and the spread of misinformation by hallucinations and use of deepfakes.

their colossal environmental impact. Large models are power hungry. Water shortages

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

1.6 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 exchanges ideas, possible paths forward and the overall structure of the theses. Additionally 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.

Chapter 2

Theoretical Background

This thesis covers a lot of ground and the field of machine learning draws on many parts of mathematics, statistics and computational sciences. 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.

In this section we begin by introducing 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, trying to highlight aspects brushed over in [2], building it up from Autoencoders. Before finally we go through the evaluation metrics used to assess our models.

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[3]. The first practical implementation was by Frank Rosenblatt in 1957 [4]. 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 Hornik in 1989 [5], which roughly states that a neural network can approximate any (Borel

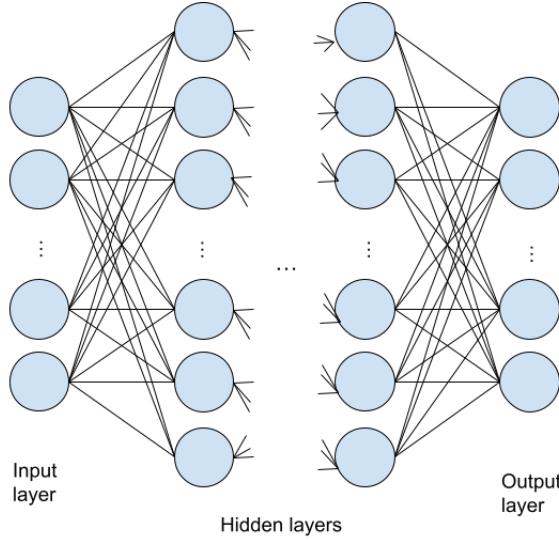


Figure 2.1: Illustration of a Neural Network model.

measurable) function to any desired degree of accuracy.

A neural network takes in a vector $x \in \mathbb{R}^n$ and builds a nonlinear function $f(x)$ to predict the response $y \in \mathbb{R}^m$. More specifically, a neural network maps an input vector x to an output vector y through a series of nonlinear functions applied to linear combinations of the input. This particular structure, illustrated in Figure 2.1, distinguishes neural networks from other nonlinear prediction models.

The variables $x = [x_1, \dots, x_n]$ 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* $\{\sigma_1, \dots, \sigma_{K-1}\}$. We denote the collection of parameters $\{W_1, \dots, W_{K-1}, b_1, \dots, b_{K-1}\}$ by θ . Written out a K layered neural network is given by

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

where

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

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, \dots converges to a local minimum of F .

A neural network $f_\theta(x)$ is itself a multivariable function, and as long as the loss and activation functions are differentiable, the network is as well, both in terms of its 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

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

and one would iteratively update the parameters by gradient descent

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

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

Algorithm 1 Stochastic Gradient Descent (SDG)

```

Initialize parameters  $\theta$  and learning rate  $\gamma$ 
while Not converged do
    Permute training set  $(X, Y)$ 
    for  $i$  in  $1, \dots, N$  do
         $\theta \leftarrow \theta - \gamma \nabla_{\theta} \mathcal{L}(f_{\theta}(x_i), y_i)$ 
    end for
end while

```

TODO: Mini batch SDG

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

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

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 replaces matrix multiplication with convolution in at least one of its layers [6].

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

The concept of CNNs is inspired by the structure and function of the visual cortex in mammals. In the 1960s, Hubel and Wiesel discovered the visual processing capabilities of the cortex [7], leading to the development of the neocognitron in 1980 [8], 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 [9]. 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 [10].

2.2.1 The convolution operation

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

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

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

While the detailed mathematical criteria for the existence of this integral are 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* [6]. 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

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

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

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

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

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 [6].

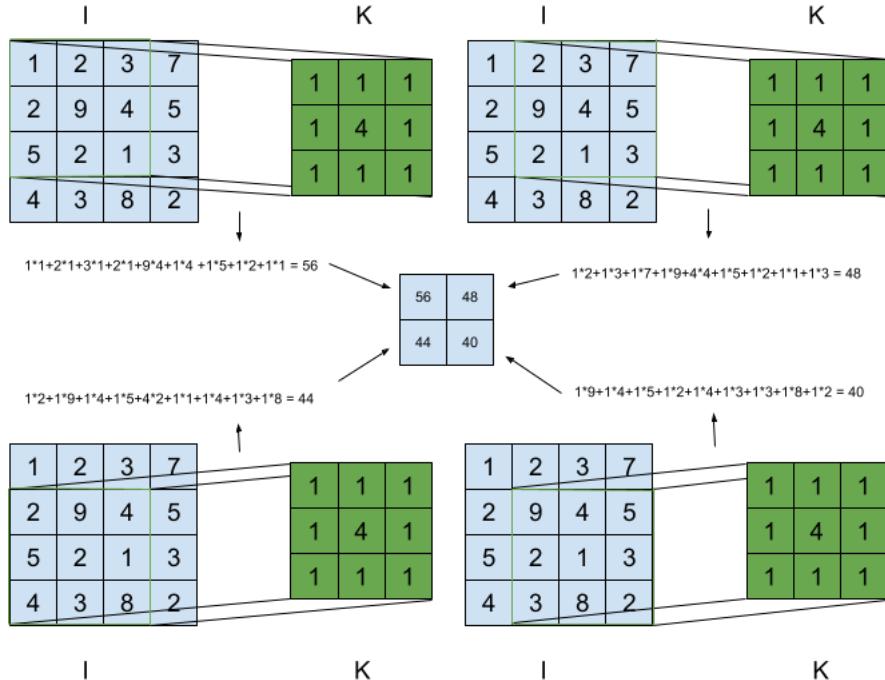


Figure 2.2: Illustration of discrete two dimensional convolution

In digital signal processing, discrete convolutions are widely used with pre-defined 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.

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

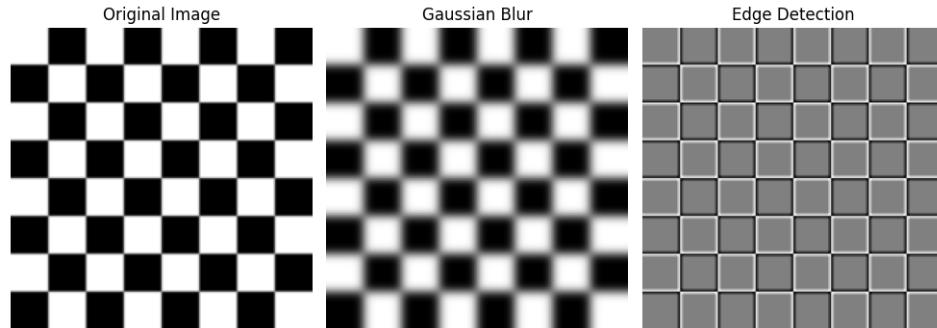


Figure 2.3: Illustration of discrete convolution applied to images. Image part of Scikit-image data package.

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.

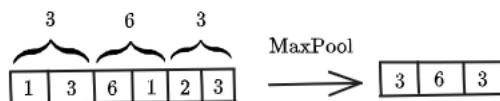


Figure 2.4: Max pooling of one dimensional array. Filter size: 2, stride: 2.

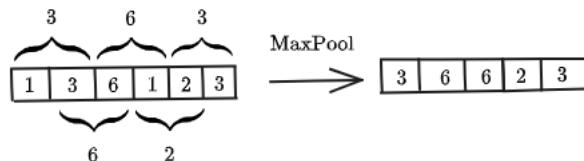


Figure 2.5: Max pooling of one dimensional array. Filter size: 2, stride: 1.

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 particularly 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

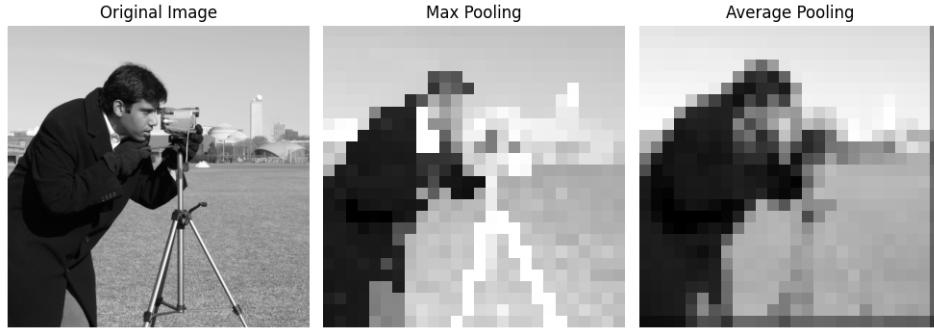


Figure 2.6: Illustration of mean and average pooling applied to images. Filter size and stride is 20×20 . Original image size is 512×512 , pooled images are 26×26 . Image part of Scikit-image data package.

2.8 illustrates the original LeNet-5 [9] 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.

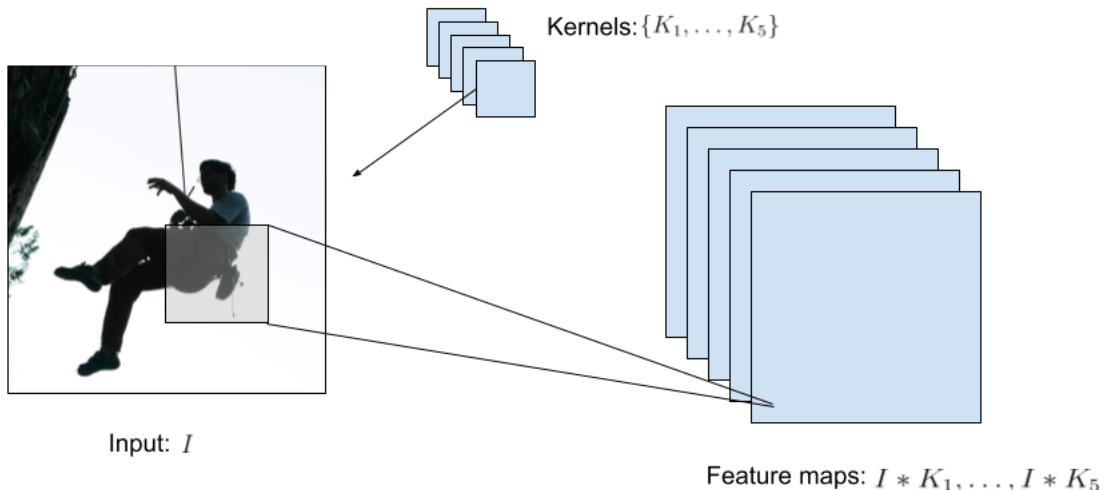


Figure 2.7: Convolutional layer.

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 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.

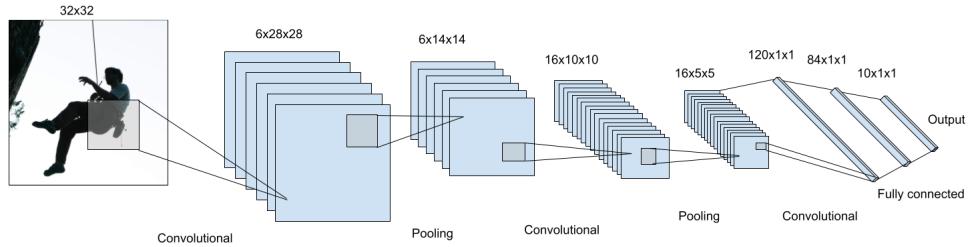


Figure 2.8: LeNet-5 network [9]

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 [11], residual connections allow gradients to flow directly through the network, alleviating the vanishing gradient problem in deep networks [12]. 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 [11], which have achieved state-of-the-art results in various computer vision tasks.

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 convolu-

tional 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.

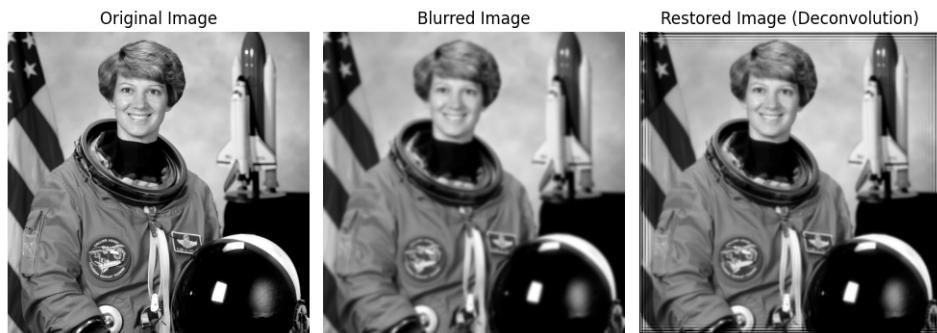


Figure 2.9: Illustration of deconvolution applied to images. Here using Gaussian blur and restoring using the Richardson Lucy algorithm with a Gaussian deconvolution kernel. Image part of Scikit-image data package.

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 is needed to feature engineer well. In reality much of the actual time spent in the process of deploying machine learning methods revolves around constructing good data pipelines and applying transformations that produce representations beneficial for the algorithm at hand [13]. 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 [13].

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

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

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

An alternative protocol, similar to the one above, allows all parameters of f_D to be learnable on the downstream task. This protocol is referred to as a pretraining-finetuning. In this approach, the model 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[15].

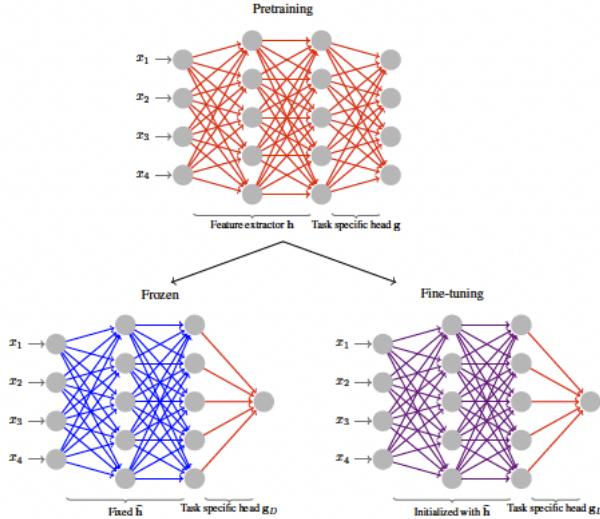


Figure 2.10: Taken from [15]

2.4 Transformers

Since their introduction in the seminal paper "Attention is all you need"[16], 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 [17], and similar trends are shown for other modalities such as audio [18] and time series [19].

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 [12]. 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*.

ation. 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 [17], input images are split into patches and linearly projected before being processed by the transformer. Since then, *Vector-Quantized-based* tokenization introduced by [2] 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.

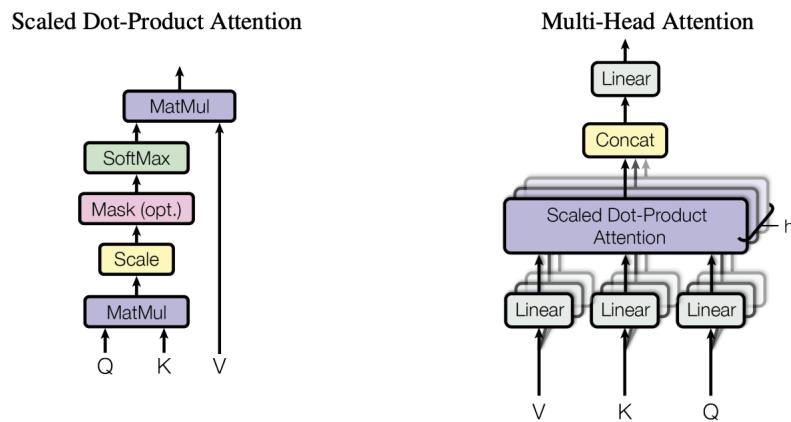


Figure 2.11: (left) Scaled dot-product attention. (right) Multi-headed attention.
Taken with permission from [16]

Scaled dot-product attention is computed by

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

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

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

where

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

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.

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

Positional encoding: Transformers do not inherently account for the order of the input sequence, as they lack the sequential nature of recurrent models. To address this, positional encoding is added to the input embeddings. Positional encoding provides information about the position of each token in the sequence, allowing the model to differentiate between tokens based on their order. In the original paper [16], sinusoidal functions were used to generate unique positional encodings for each position.

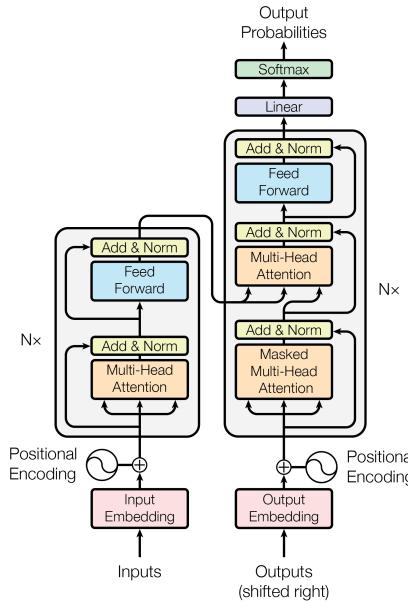


Figure 2.12: The Transformer - model architecture. Encoder (left), decoder (right). Taken with permission from [16]

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 attention layer but prevents the model from attending to future tokens in the sequence during training, ensuring that predictions are based only on past and present tokens. A multi-head attention which is applied to the output of the encoder and a fully connected neural network. All sublayers employ residual connections and normalization. The final output of the decoder is converted to probabilities using a linear layer followed by a softmax function.

Bidirectional transformer models, such as BERT [20], utilize an encoder-only architecture. This design allows the attention mechanism to consider information from both directions within the sequence, enabling the model to capture a more comprehensive understanding of the context. The issue of "seeing into the future" is resolved through a training technique called *masked modeling*, where some tokens are masked and the model is trained to predict them based on the surrounding context. This will be looked at further in Section 2.5.2.

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

$$\hat{f}_\theta = \min_{\theta} \frac{1}{N} \sum_{i=1}^N L(f_\theta(x_i), y_i). \quad (2.7)$$

Common approaches to supervised learning for neural networks is to calculate some distance metric between the predicted value \hat{y} and the true value y and update parameters using backpropagation.

Supervised learning models, including classical statistical models, support vector machines, and decision tree-based models, have seen tremendous success but are limited by the need for labeled data, which is often scarce and expensive to obtain.

Unsupervised Learning

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

Self-Supervised Learning

Self-supervised learning is a subcategory of unsupervised learning that has shown remarkable success in NLP and computer vision, approaching the performance of state-of-the-art supervised representation learning [21, 22]. SSL involves using the data itself to generate a supervisory signal, rather than relying on external labels. Although SSL is technically unsupervised, its learning formulation closely resembles that of supervised learning.

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

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

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

TODO: Figure of siamese

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

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

Non-Contrastive SSL methods, such as BYOL [27], BarlowTwins [22] and ViLBReg [26] avoid the need for positive and negative pairs. These methods use augmentations of the input data as pseudo labels and introduce different mechanisms to prevent collapse. For instance, BYOL introduces architectural asymmetry

and the stop-gradient operation, Barlow Twins minimizes information redundancy between branches, and VICReg maintains variability while decorrelating features in each branch.

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 [28].

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 [21, 22] 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 [29]. Independent representation features is in the literature considered a desired property, though not universally accepted [13, 30]. The article [29] 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 [20], masked modeling has become the standard for self-supervised pretraining of language models.

In computer vision, masked modeling has also gained attention. Initial approaches involved masking image patches directly [31]. With the success of vision transformers [17], researchers began tokenizing images and pretraining transformers in a manner similar to BERT. For example, MaskGIT [32] 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 [33].

2.6 Vector Quantized Variational Autoencoder (VQVAE)

Our model is based on the Vector Quantized Variational Autoencoder (VQVAE) introduced in [2], 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 2.13.

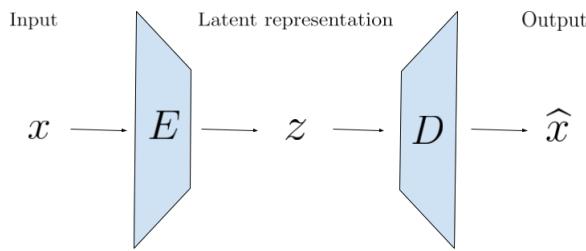


Figure 2.13: Schematics of the autoencoder architecture.

The primary goal of an autoencoder is to learn an efficient encoding of the data. The encoder compresses the data into a lower-dimensional representation,

and the decoder reconstructs the original data from this compressed representation. The training objective is to make the composition of the encoder and decoder approximate the identity function, meaning that the output should be as close as possible to the input. Autoencoders are typically trained to minimize the reconstruction error, which typically is the mean square error (MSE) between the input and the reconstructed output. This process forces the model to learn efficient latent representations of the data.

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

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

2.6.2 Variational Autoencoder (VAE)

Variational Autoencoders (VAE) were introduced in [34] and is a variational Bayes approach to approximate inference and learning with directed probabilistic models. While the architecture of VAEs, presented in Figure 2.14, 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 $p_\theta(z) \sim N(0, I)$
- Likelihood $p_\theta(x|z) \sim N(D(z), I)$
- Variational posterior $q_\phi(z|x) \sim N(E(x)) = N(\mu_x, \Sigma_x)$

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

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

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

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

There are several reasons for choosing Gaussian distributions, one being that the Gaussian distribution is 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 auxiliary variable $\epsilon \sim N(0, I)$ and rewriting $z = \mu_x + L_x \epsilon$, where L_x is the Cholesky decomposition of Σ_x . This allows the gradient to flow through the deterministic parts of the model.

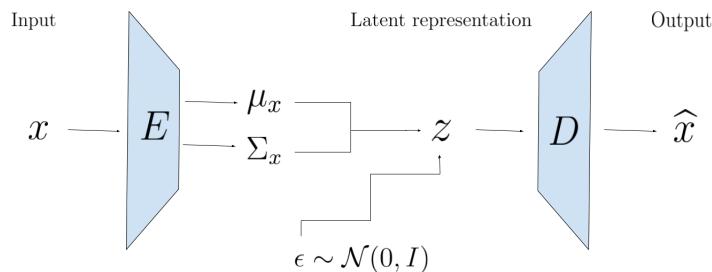


Figure 2.14: Schematics of the VAE architecture with Gaussian reparameterization.

Training objective

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

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

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

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

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 [35].

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

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

Assuming a Gaussian likelihood, we have

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

which is equivalent, as an optimization objective of \hat{x} , to $\|x - \hat{x}\|_2^2$. Consequently the likelihood in the loss is implemented as the MSE of the input x and the output \hat{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 \mathbf{z} being the parent node of \mathbf{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 \mathbf{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 [2] and presented a novel approach to training VAEs with discrete latent variables. It was the first model to achieve similar performance to continuous VAEs while utilizing discrete latent representations. VQVAE was developed to enhance representation learning by allowing the encoder network to output discrete codes and by learning the prior distribution rather than assuming it to be static. This approach provides more flexibility in capturing the underlying data distribution while simultaneously avoiding posterior collapse and variance issues observed in VAEs.

Architecture

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

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

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

From 2.15 we can see that the VQVAE can be seen as an autoencoder with a nonlinearity introduced by the vector quantization.

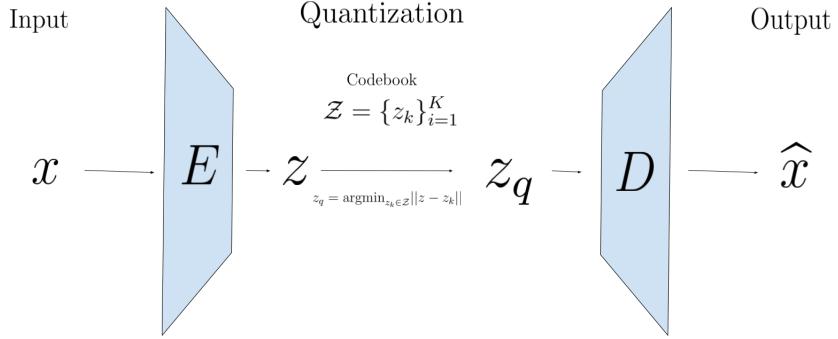


Figure 2.15: Schematics of the VQVAE architecture.

ELBO

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

$$p(z = k|x) = \begin{cases} 1 & \text{for } k = \operatorname{arg min}_j \|E(x) - z_j\|_2 \\ 0 & \text{otherwise} \end{cases}, \quad (2.14)$$

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

In the original article [2], 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, \dots, K\}\$ we get that the regularizing term simplifies to a constant

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

Thus, the ELBO reduces to the reconstruction term only.

Loss function

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

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

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

$$\mathcal{L}_{\text{recons}} = \|x - \hat{x}\|_2^2. \quad (2.16)$$

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

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

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

$$\mathcal{L}_{\text{VQ}} = \mathcal{L}_{\text{codebook}} + \mathcal{L}_{\text{recons}}. \quad (2.18)$$

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 [2].

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 learns a more accurate prior which can improve the quality of generated samples. The choice of prior model depends on the particular application. In the original VQVAE paper PixelCNN [36] was used for image data, while WaveNet [37] was used for audio. More recently in TimeVQVAE [1] utilized a bidirectional transformer [32] 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 2.16

2.7.1 Generative model

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

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

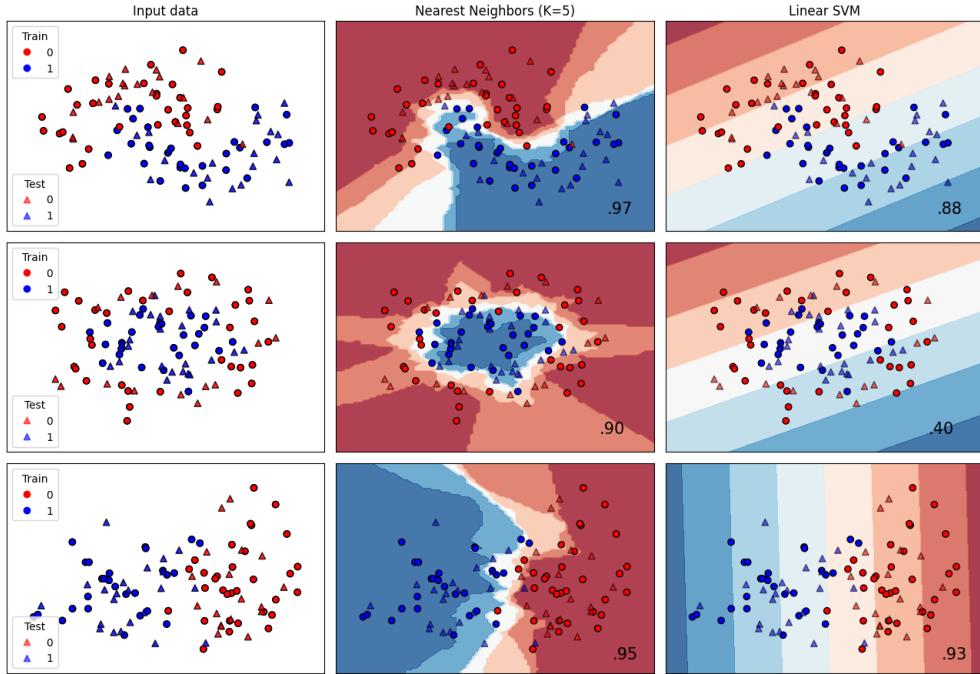


Figure 2.16: Modified example taken from https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html.

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. Unlike computer vision, where models like Inception v3 are standard, there is no universally adopted classification model for time series data. However, [38] proposed a fully convolutional network (FCN) as a baseline model for time series classification. Since the original model is not readily available, we utilize an open-source FCN trained on the UCR Archive, as presented in [1], for our evaluations.

Inception Score (IS)

The Inception Score (IS) was first introduced in [39] as an automatic method for evaluating the quality of synthetic samples. IS measures the clarity and diversity of generated samples by using a pre-trained classifier to predict labels. The intuition behind IS is that high-quality samples should result in low entropy in the conditional label distribution $p(y|x)$, while diverse samples should lead to high entropy in the marginal label distribution $p(y)$. Thus if the KL-divergence between

these distributions are high, the samples should be of high quality. The Inception Score is defined as

$$\text{IS}(\theta) = \exp(\mathbb{E}(D_{\text{KL}}(p_\theta(y|x)||p_\theta(y)))). \quad (2.19)$$

In practice one uses a pretrained classifier as estimate for $p(y|x)$. For a synthetic dataset $X_{\text{gen}} = \{x_{i,\text{gen}}\}_{i=1}^N$, the marginal label distribution is obtained by averaging across all the synthetic data as follows

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

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 [40]. Since then FID has been the standard for assessing generative models [41]. 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

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

where $\Gamma(f, g)$ is the set of all *couplings* of f and g . For two Gaussian distributions, as proven in [42], the Fréchet distance is explicitly computable as

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

In FID, the mean and covariance of the feature representations of real and generated samples are computed using a pretrained model. They argue in [40] that since the Gaussian distribution is the maximum entropy distribution over \mathbb{R}^n , for a given mean and covariance, it is a reasonable distribution to assume for the representations. But, despite its usefulness, FID has limitations, such as the Gaussian assumption not always holding true and the need for a large number of samples to estimate the covariance matrix reliably [43, 44].

Classification Accuracy Score (CAS)

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

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

Chapter 3

Related Work

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

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

3.1 MaskGIT

The Masked Generative Image Transformer (MaskGIT), introduced in [32], 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 [45] was used for this purpose, focusing on improving the second stage. Therefore, our discussion will primarily address this aspect of the model.

3.1.1 Masked Visual Token Modeling (Prior learning)

For prior learning, the codebook learned in the tokenization procedure is provided with a masking vector, which is the embedding of the special masking token, denoted by \mathbf{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 \mathbf{M} according to the binary mask M . This is done by

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

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

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

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

A forward computation is illustrated in Figure 3.1, where the bidirectional transformer predicts the probabilities $p(s_i|s_{\text{Mask}})$ of each masked token. The training objective is to minimize the negative log likelihood of the masked tokens, conditional on the unmasked

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

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

3.1.2 Iterative Decoding (Sample generation)

The bi-directional transformer could in principle predict all masked tokens and generate a sample in a single pass by sampling according to the predicted probabilities $p(\hat{s}_i|s_{\text{Mask}})$ from a forward pass of an all masked sequence. However, this approach presents certain challenges. To address these, [32] proposes a novel non-autoregressive decoding method to synthesize samples in a constant number

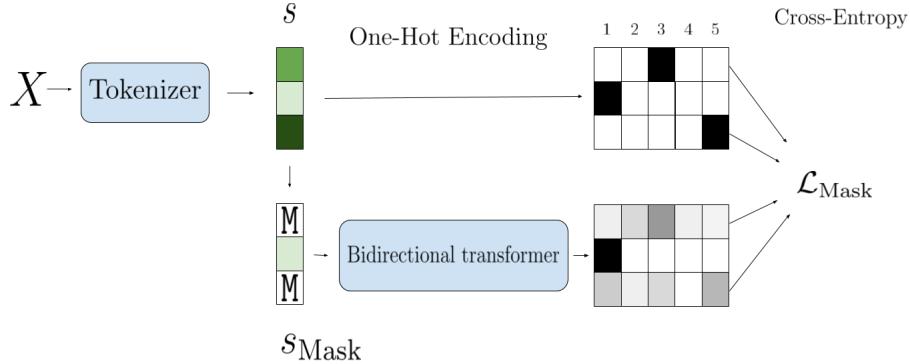


Figure 3.1: MaskGIT forward computation.

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 $s_{\text{Mask}}^{(0)}$. At iteration t , the model predicts the probabilities for all the mask tokens, $p(\hat{s}_i|s_{\text{Mask}}^{(t)})$, in parallel. At each masked index i , a token $s_i^{(t)}$ is sampled according to the predicted distribution, and the corresponding probability $c_i^{(t)}$ 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

$$m_i^{(t+1)} = \begin{cases} 1, & \text{if } c_i < \text{Sort}([c_1^{(t)}, \dots, c_N^{(t)}])[n] \\ 0, & \text{otherwise} \end{cases} \quad (3.3)$$

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

3.2 TimeVQVAE

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

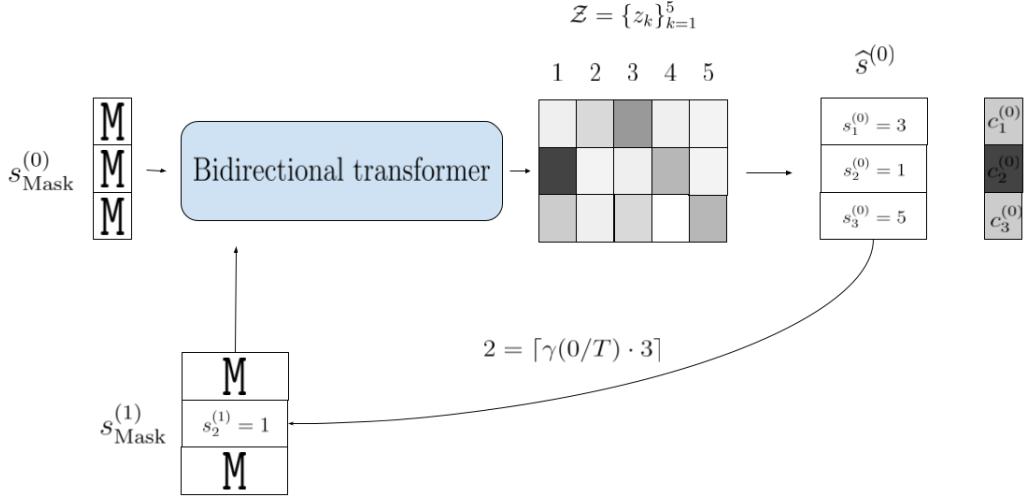


Figure 3.2: Illustration of first pass of the iterative decoding algorithm.

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 [17], 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 [2] for details on the prior model training.

3.2.1 Tokenization

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

coders are again zero-padded giving \hat{u}_{LF} and \hat{u}_{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, \hat{x}_{LF} and \hat{x}_{HF} , of the time series.

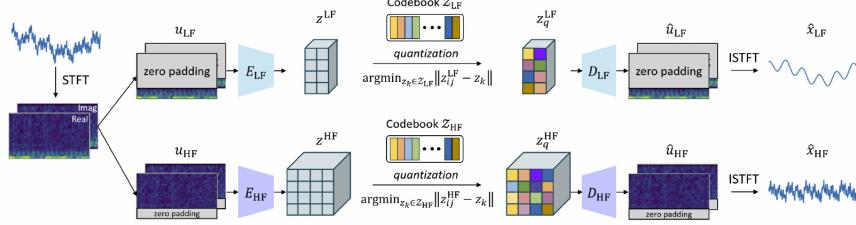


Figure 3.3: Stage 1: Tokenization. Figure taken with permission from [1]

Loss

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

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

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

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

The total loss is given by

$$\mathcal{L}_{\text{VQ}} = \mathcal{L}_{\text{codebook}} + \mathcal{L}_{\text{recons}}. \quad (3.6)$$

To update the codebooks, TimeVQVAE uses an exponential moving average method as presented in Appendix A.1 of [2].

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*[46] 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.

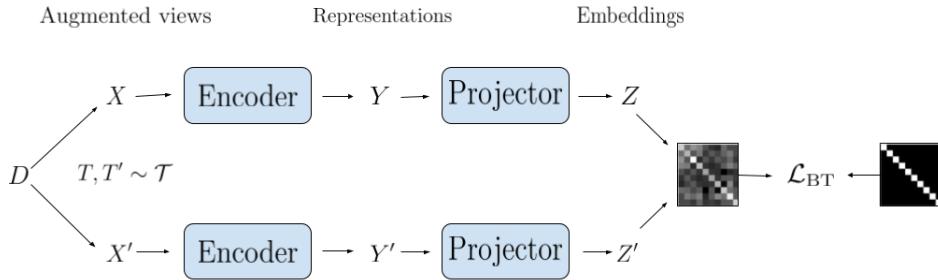


Figure 3.4: Overview of the Barlow Twins architecture. Figure inspired by [22]

The Barlow Twins algorithm, illustrated in Figure 3.4, starts out by creating two augmented views for each datapoint in a batch D . These augmentations are sampled from a collection of augmentations \mathcal{T} . The batches of augmented views $T(D) = X$ and $T'(D) = X'$, are then passed through an encoder to give representations Y and Y' , which are further projected to produce embeddings Z and Z' . The embeddings are assumed to be mean centered across the batch dimension. The loss function is calculated using the cross-correlation matrix \mathcal{C} between Z and Z' , measuring its deviation from the identity. The Barlow Twins loss is defined as

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

where

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

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

3.3.2 VIBCReg

VIBCReg [[lee2024vibcreg](#)] is a non-contrastive SSL model with a siamese architecture based on VICReg [21]. It improves upon VICReg by incorporating better covariance regularization and IterNorm [47]. A key difference from Barlow Twins is that variance/covariance regularization is applied to each branch individually.

As with Barlow Twins, a batch D is augmented to create two views and passed through an encoder and projector. The embedding Z and Z' are *whitened* using IterNorm [47].

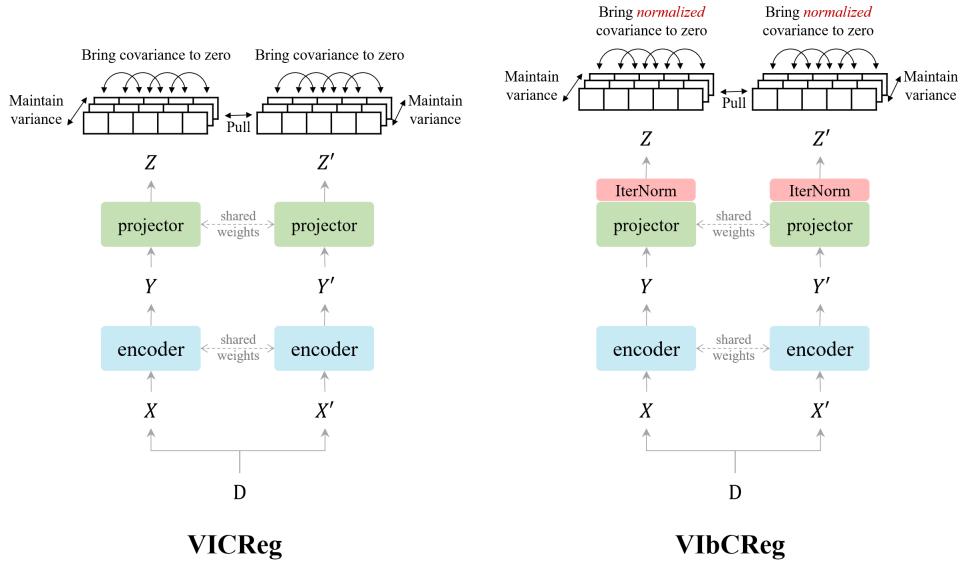


Figure 3.5: Overview of VIBCReg, and comparison with VICReg. Taken with permission from [26]

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

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

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

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

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

where $\text{Var}()$ 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

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

giving

$$\widehat{Z} = [\widehat{Z}_1, \dots, \widehat{Z}_B]^T. \quad (3.12)$$

The normalized covariance matrix is computed as

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

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

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

The total loss is then given by

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

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

Chapter 4

Methodology

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

A schematic figure of our proposed tokenization model is given in Figure 4.1. To improve 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 encode more semantic information into the latent representations. 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.

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 [2] with non-contrastive SSL algorithms. NC-VQVAE uses the two-stage modeling approach presented in [1] and can be considered an extension of their naive TimeVQVAE. Our model primarily extends the tokenization stage, incorporating Barlow Twins [22] and VlbCReg [26] as our non-contrastive SSL methods, while the framework remains flexible. For the second stage, we model the prior using a bidirectional transformer similar to MaskGIT [32].

4.1.1 Stage 1: Tokenization

TODO: Pseudocode

The architecture of the tokenization model, shown in Figure 4.1, consists of two branches: the original and augmented branch. The model takes a time series x as input and creates an augmented view x' . The original branch follows the naive TimeVQVAE architecture from [1], while the augmented branch is an autoencoder, constructed by omitting the quantization layer. The views are passed through their respective branches, and we compute the SSL loss derived from the original discrete latent representation z_q and the augmented continuous latent z' , before the decoder reconstructs each latent representation.

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

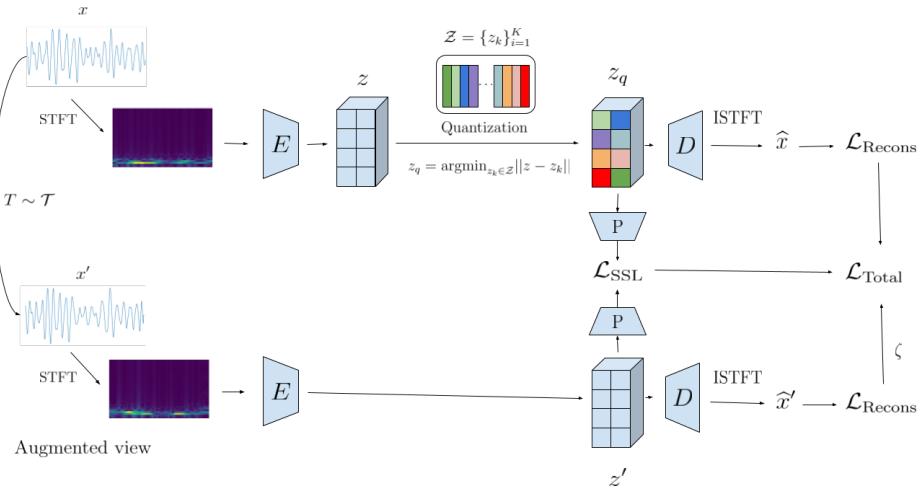


Figure 4.1: Overview of proposed model. NC-VQVAE. Fix index on Z, add weight to arrow from ssl loss to total. fix argmin index thing.

Loss

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

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

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

and the reconstruction loss to

$$\mathcal{L}_{\text{recons}} = \|x - \hat{x}\|_2^2 + \|u - \hat{u}\|_2^2. \quad (4.2)$$

Our VQ loss is then given by

$$\mathcal{L}_{\text{VQ}} = \mathcal{L}_{\text{codebook}} + \mathcal{L}_{\text{recons}}. \quad (4.3)$$

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

The augmented reconstruction loss is simply given as

$$\mathcal{L}'_{\text{recons}} = \|x' - \hat{x}'\|_2^2 + \|u' - \hat{u}'\|_2^2. \quad (4.4)$$

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

The total loss is given by

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

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

4.1.2 Stage 2: Prior learning

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

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

Chapter 5

Experiments

5.1 Initial Experimentation and Model Development

TODO: This section is under construction. Contemplating if i should even include it at all in some form.

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

During development we encountered several problems:

When we attempted a siamese architecture, with quantization in the augmented branch, and to derive the SSL loss from the discrete representations there were a correlation problem. The codewords were very highly correlated, which resulted from the passing both views through the VQ. $SSL(z_q, z'_q)$

In an attempt to solve this we attempted to derive the SSL loss from the continuous latent representations, but the resulting discrete latent representations performed poorly on the downstream classification task. Separability problem: $SSL(z, z')$

The solution was to remove the VQ in the augmented branch and rather derive the SSL loss from z_q and z' . Solution: $SSL(z_q, z')$

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

5.2 Main Experiments

TODO: This section i need som assistance with. Find it difficult to write anything smart.

To evaluate our model, NC-VQVAE, we compare it against the naive VQVAE using both Barlow Twins and VIBCReg as self-supervised learning methods. 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.

5.2.1 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.

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 `n_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 [48], with further adaptations from [1].

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].

The augmentation reconstruction weight ζ in NC-VQVAE is set to 0.1.

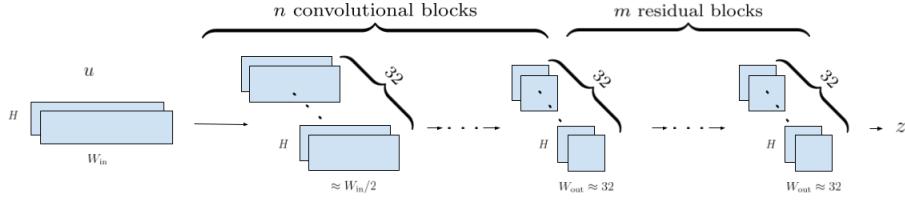


Figure 5.1: Overview of the encoder architecture. The decoder architecture is simply obtained by reversing the arrows and switching out the convolutional block for transposed convolutional blocks.

VQ

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 $\beta = 1$.

SSL

For both Barlow Twins and VlbCReg, we implement the projector following the guidelines in [26]. 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 VlbCReg, 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 VlbCReg loss in the NC-VQVAE is set to $\eta = 0.01$.

Augmentations

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

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

Amplitude Resizing + Window Warp 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.

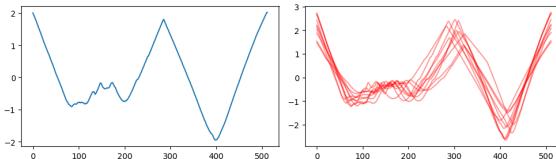


Figure 5.2: ShapesAll: Original (left), augmented (right). 15 instances of Amplitude Resizing + Window Warp applied to the original sample.

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.

slice and shuffle:

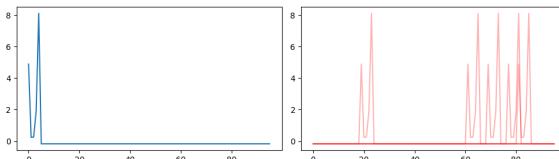


Figure 5.3: ElectricDevices: Original (left), augmented (right). 5 instances of Slice and Shuffle applied to the original sample.

Gaussian noise adds a noise $\epsilon \sim N(0, 0.05)$ 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.

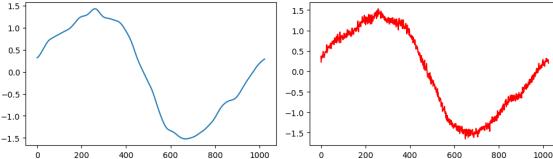


Figure 5.4: StarLightCurves: Original (left), augmented (right). One instance of Gaussian noise applied to the original sample.

Prior learning

We adopt the implementation from [32]. In the iterative decoding algorithm, the number of iterations T is set to 10, and use cosine as mask scheduling function (γ).

Optimizer

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

5.2.2 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 [2] for detailed information.

5.3 UCR Time Series Classification Archive

The evaluation of our model NC-VQVAE is done on a subset of the UCR Time Series Archive [49]. The UCR archive is a collection of 128 datasets of univariate time series for time series classification. The different datasets in the archive span a wide range characteristics and include among others sensor, device, image-derived and simulated data. Each dataset has a predefined training and test split.

The subset selected for our experiments is presented in Table 5.1.

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

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

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

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

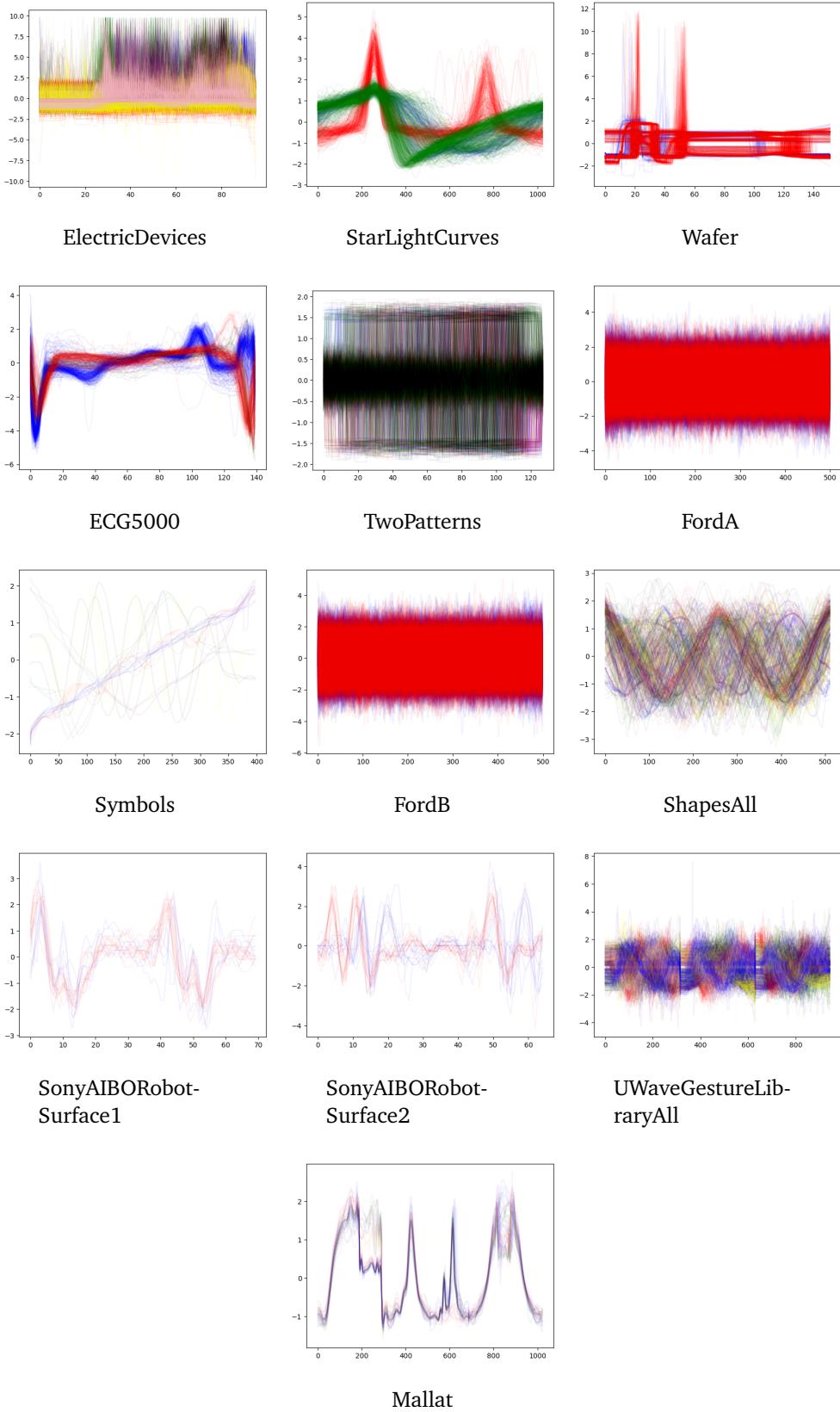


Figure 5.5: Our selected subset of the UCR Archive. All time series in the training set are plotted and color coded according to label.

Chapter 6

Results and Discussion

TODO: The spacing and figure placement is under development.

TODO: Have not yet polished formulations and language.

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 prior learning and time series generation.

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.

6.1 Stage 1

In this section we present the results of the tokenization model, in particular the reconstruction loss and the downstream classification accuracy. We address research question 1 and 2, if the proposed NC-VQVAE is able to reconstruct on par with the naive VQVAE and if the learned latent representations are more expressive, in the sense that they simultaneously improve the downstream classification accuracy. We see that some configuration of NC-VQVAE is the top performer on the majority of datasets for both metrics, and provides significant increase in probe accuracy.

6.1.1 Reconstruction

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

Mean validation reconstruction error							
Dataset	Baseline		SSL Method				
	Regular	Barlow Twins			ViLBReg		
		None	Warp	Slice	Gauss	Warp	Slice
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			ViLBReg		
		None	Warp	Slice	Gauss	Warp	Slice
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.

We observe that NC-VQVAE reconstructs on par with the baseline model, and that some configuration outperforms the naive VQVAE on mean reconstruction loss for 9 out of 13 datasets.

In figure 6.1 observe that the difference in reconstruction loss is small for most datasets, both across SSL methods and augmentations. Nevertheless we observe that ViBcReg generally performs slightly better than Barlow Twins, except for FordA. Additionally the use of gaussian augmentation introduces less of a regularizing effect compared to the two other sets of augmentations, with the exception of Slice and Shuffle on ECG5000. These results show that the introduction of a non contrastive loss does not hurt the reconstruction capabilities, compared to naive VQVAE.

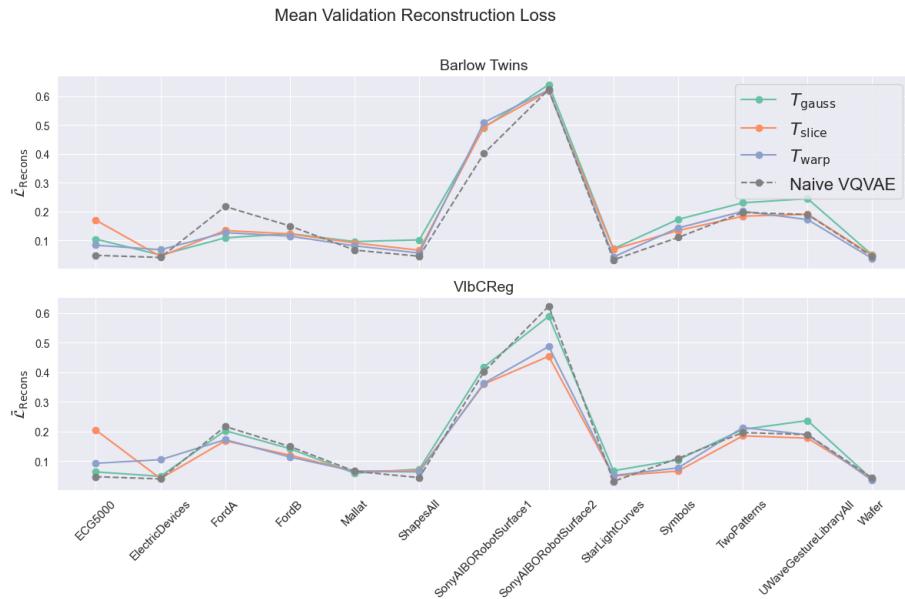


Figure 6.1: Mean validation reconstruction loss for the two models, compared to naive VQVAE

Regarding the effect of by the reconstruction loss of augmented branch on validation reconstruction, a small initial experiment was conducted. The results show that the validation reconstruction loss was quite robust to the particular value of augmentation reconstruction weight, indicating a minor role played, as seen in Figure 6.19 and 6.20.

By investigating the development of the validation reconstruction loss during training, we have observed that right configuration for NC-VQVAE can act as a regularizer. In Figure 6.2 we see the development on FordA.

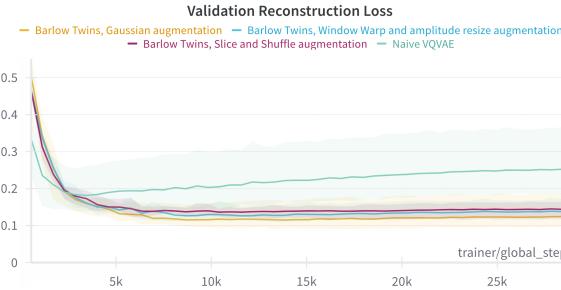


Figure 6.2: Development of the validation reconstruction loss for Barlow Twins and naive VQVAE on FordA during training. Averaged across all four runs.

6.1.2 Classification

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

Mean linear probe accuracy

Dataset	Baseline		SSL Method											
	Regular		Barlow Twins						VIbCReg					
	None		Warp		Slice		Gauss		Warp		Slice		Gauss	
	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM	KNN	SVM
FordA	0.70	0.74	0.83	0.84	0.91	0.89	0.80	0.83	0.80	0.74	0.87	0.86	0.76	0.78
ElectricDevices	0.35	0.41	0.35	0.44	0.38	0.41	0.40	0.42	0.33	0.38	0.36	0.39	0.39	0.43
StarLightCurves	0.87	0.89	0.93	0.94	0.94	0.88	0.88	0.92	0.94	0.91	0.93	0.89	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.

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

In Figure 6.3 we observe that even though the specific choice of augmentation has a large impact, on the majority of datasets, all choices result in significantly improved probe accuracy. We note that both SSL methods produce similar probe accuracies for a given augmentation, highlighting the importance of selecting suitable augmentations.

We observe that Slice and Shuffle, and Window Warp and Amplitude Resize result in the most dramatic increases in accuracy, while Gaussian noise consistently results in less drastic improvement. We hypothesize that, since Slice and Warp often result in augmented views that deviate quite a lot from the original view, the SSL loss pushes the representations in different directions, which again might result in a better utilization of the latent space. In Figure 6.4 and 6.5 we see the effect of NC-VQVAE on the discrete latent representations of FordA and TwoPatterns. From these visualizations it is evident that representations learned using NC-VQVAE are more structured than those of the naive VQVAE. Similar samples, typically with the same label, are clustered closer together in latent space. This further indicate that the SSL loss introduces information regarding shape and semantics into the latent representations.

To summarize the results from stage 1, NC-VQVAE is able to reconstruct on par with naive VQVAE, and in some cases improve the reconstruction loss, while signi-

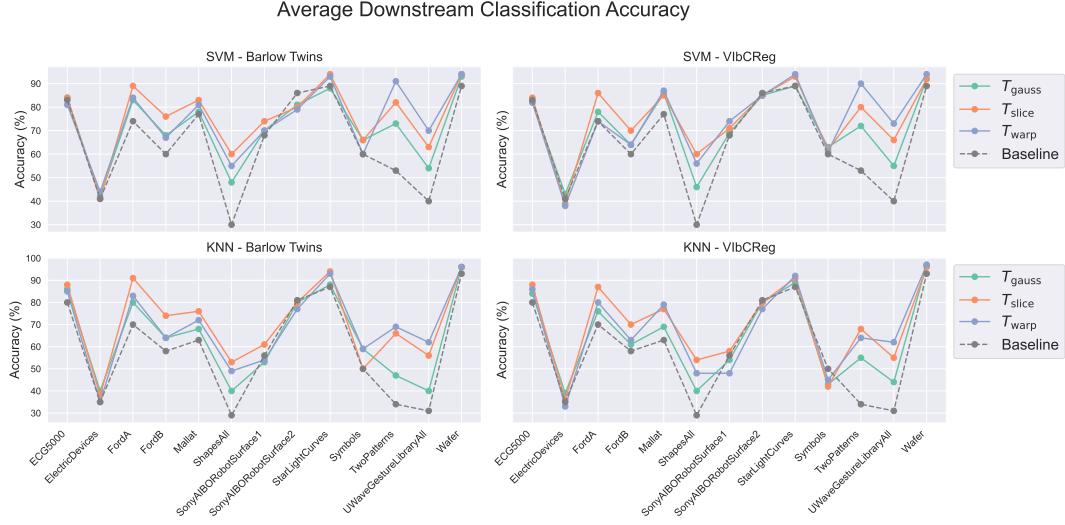


Figure 6.3: Mean probe accuracies.

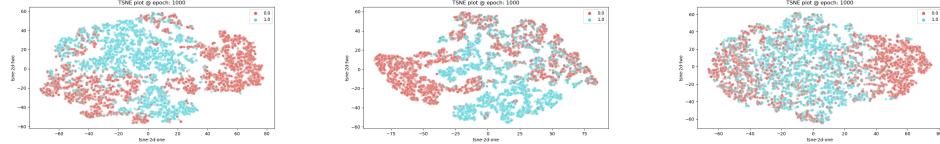


Figure 6.4: TSNE plots of FordA. Barlow (left) and ViLBReg (center) with Slice and Shuffle, naive VQVAE (right). Best performing model in terms of KNN accuracy is chosen.

fificantly improving the probe accuracy for most datasets. To address research question 1, we conclude that the representations learned using NC-VQVAE are more expressive compared to the naive VQVAE. As the representations separates classes more effectively, they could encode more class specific information. Since we are not sacrificing reconstruction quality, this could be beneficial for prior learning. To address research question 2, the specific augmentation used plays a prominent role in the results, and we observe that warp and slice typically leads to better performance than gaussian, especially in terms of probe accuracy. We additionally note that there are large differences across datasets, which support the hypothesis that the optimal choice of augmentations is highly dataset dependent.

6.1.3 Losses

We investigate some trends in the development of different loss terms in this section. To summarize, using ViLBReg 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.

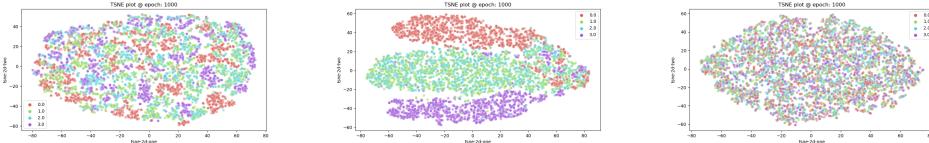


Figure 6.5: TSNE plot of discrete latent representations from VIBCReg with Slice and Shuffle (left), Barlow Twins with Window Warp and Amplitude Resize (center) and naive VQVAE (right). Dataset is TwoPatterns. The latent space is significantly more structured with NC-VQVAE.

In figure 6.6 we observe the typical pattern of the SSL loss during training.

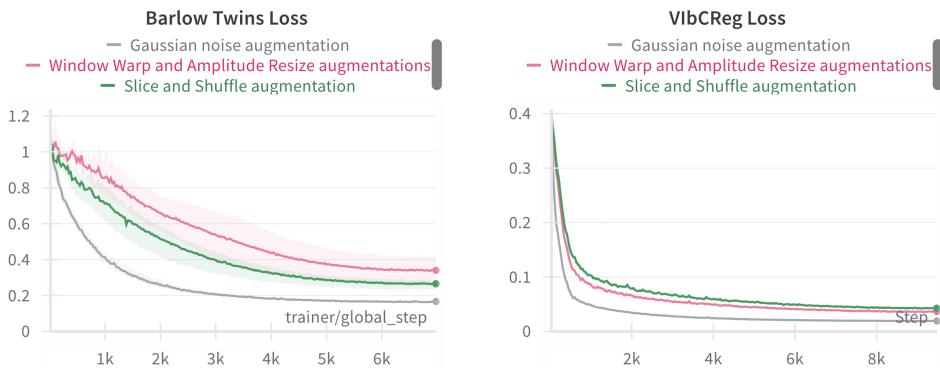


Figure 6.6: SSL loss during training on UWAVEGESTURELIBRARYALL. Averaged across four runs.

We see that the Gaussian augmentation results in a SSL loss which is easier to minimize, which might be attributed to the fact that it affects the samples in a more predictable way. We too see that the VIBCReg loss decreases more rapidly than the Barlow Twins loss. Both observations are seen across datasets.

Previously in figure 6.3 we have seen that the gaussian augmentation often resulted in lower probe accuracy than the other two. In figure 6.7 we see that, on the datasets with a significant increase in probe accuracy, the augmentations that result in a more challenging SSL loss typically has higher downstream classification accuracy. We also see that for a specific augmentation, the pattern is rather flat, indicating that the particular augmentation plays the most prominent role in probe accuracy. The SSL loss for a specific augmentation varies very little compared to probe accuracy.

The training reconstruction losses are heavily minimized, both across models and augmentations. The only consistent noticeable difference is the augmented reconstruction loss, where models using Slice and Shuffle have a slightly higher loss. The differences in VQ loss for the different models is mainly due to the code-

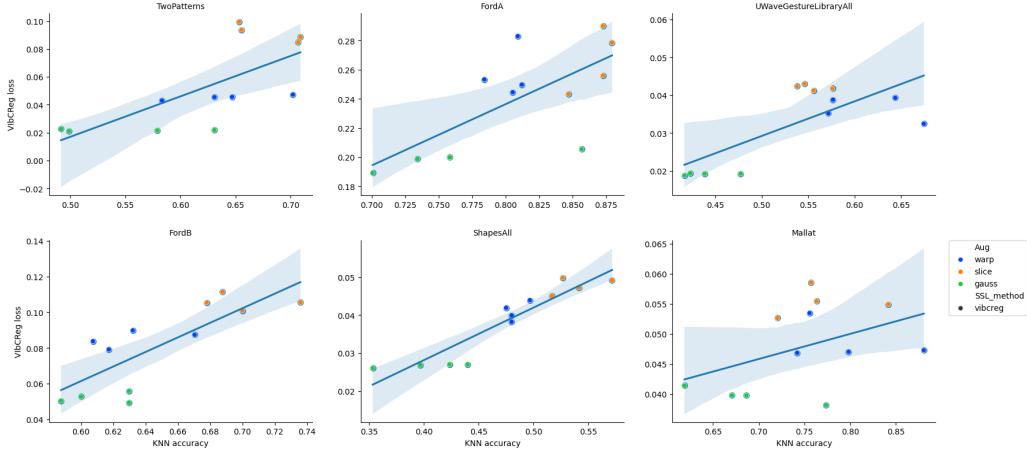


Figure 6.7: KNN accuracy plotted against VIBCReg loss. Each point correspond to a single run of the model. Similar tendency is shown for Barlow Twins.

book, where we too observe that VIBCReg minimizes more effectively than Barlow Twins, and again that gaussian augmentations results in the hardest minimization followed by Window Warp and then Slice and Shuffle.

Both VIBCReg and Barlow Twins with Gaussian augmentation routinely perform on par with naive VQVAE in terms of VQ loss during training. The minimization of the codebook loss indicates that the encoder is properly aligned with the discrete latent codes. We hypothesize that when the SSL loss is not properly minimized, the encoder must adjust its weights more throughout training which keeps the encoder outputs and the discrete codes from aligning completely.

6.2 Stage 2

The generative quality of our models are evaluated according to FID, IS and CAS. All results will be presented in this section. For datasets with very few samples, or very few per class, the generative scores must be taken with a grain of salt. Both the classifier, and the evaluation metrics is dependent on a certain number of samples to be considered reliable. We rather look more closely on the visual inspection for these.

6.2.1 FID and IS

We present the top 1 and mean score across the four runs for both FID and IS in table 6.5 and 6.6. From the tables we see that our model produces better IS score for 12 out of 13 datasets, and better FID for 10 out of 13.

Top 1 FID and IS

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

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

Mean FID and IS

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

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

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

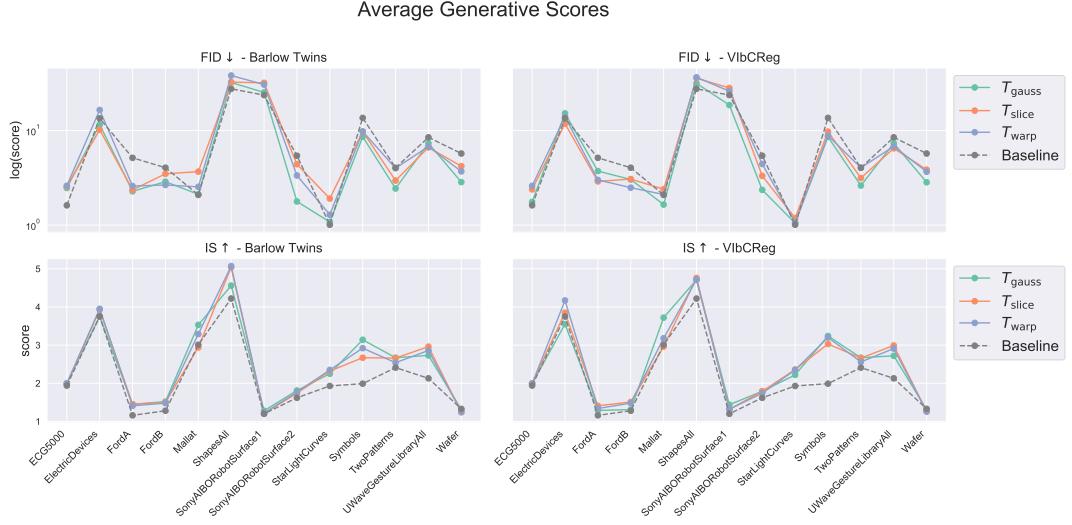


Figure 6.8: Mean FID and IS scores for Barlow Twins and ViLBReg VQVAE. FID is plotted on a log scale because of the large difference in values across datasets.

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

6.2.2 CAS

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

Mean CAS

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

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

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

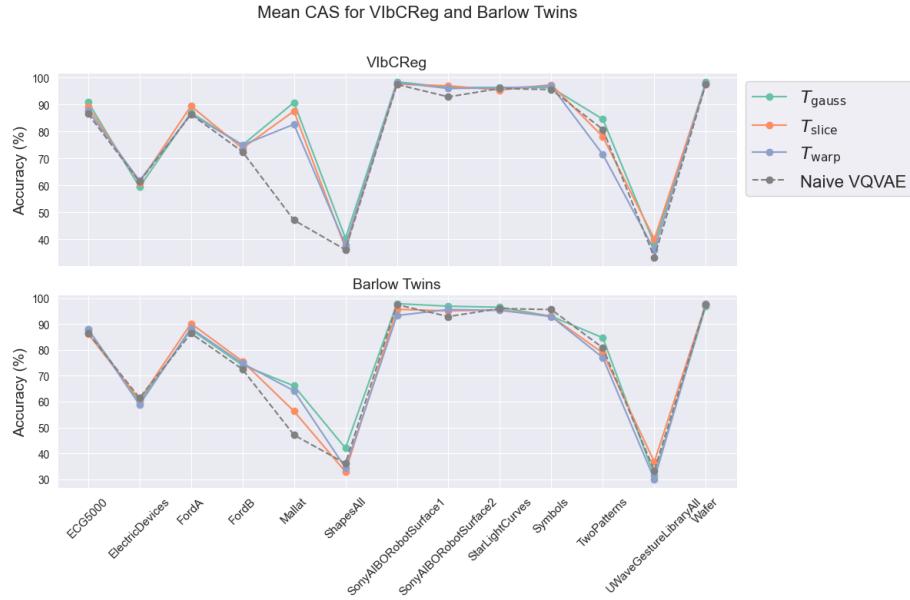


Figure 6.9: Mean CAS across all datasets.

6.2.3 Prior loss

TODO: Under construction

Naive VQVAE outperforms NC-VQVAE in terms of validation prior loss across datasets. There is the occasional dataset where a model with gaussian augmentation performs equally well. The minimization of the validation prior loss does therefore not correspond to improved FID and IS in general.

6.2.4 The influence of stage 1 on stage 2

We further attempt to address research question 3 and 4, how expressive representations influence synthetic sample quality and the role of augmentations. For this we inspect relationship between probe accuracy and FID and IS. We consider only the datasets where NC-VQVAE provided a prominent increase in probe accuracy, which are FordA, FordB, Mallat, ShapesAll, TwoPatterns and UWaveGestureLibraryAll.

In Figure 6.10 and 6.11, we show scatterplots of KNN against FID and IS with the least square regression line. The corresponding plots with SVM accuracy show similar trends. From Figure 6.10 we see a trend, with higher probe accuracy correlating 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 seems to be that augmentations that result in higher KNN accuracy tends to high IS scores, though the pattern is not consistent across all datasets. From Figure 6.11 we see that the specific augmentation is a better indicator of FID score than the KNN accuracy. Both figures too gives an indication of the model performance sensitivity to initialization, especially in terms of probe accuracy.

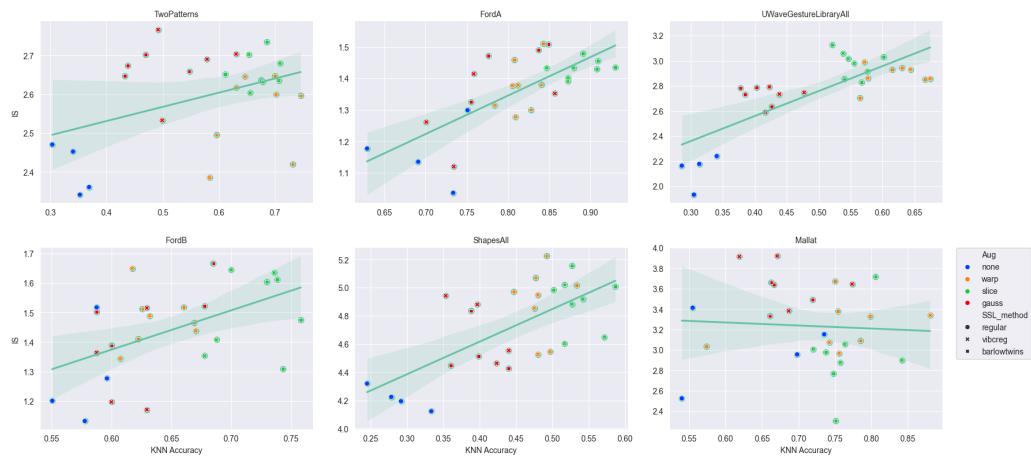


Figure 6.10: KNN plotted against Inception Score on the subset of datasets with significant improvement in probe accuracy.

6.2.5 Visual inspection

In the following we present generated samples from naive VQVAE and NC-VQVAE for some selected dataset. 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 B, 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 selected 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

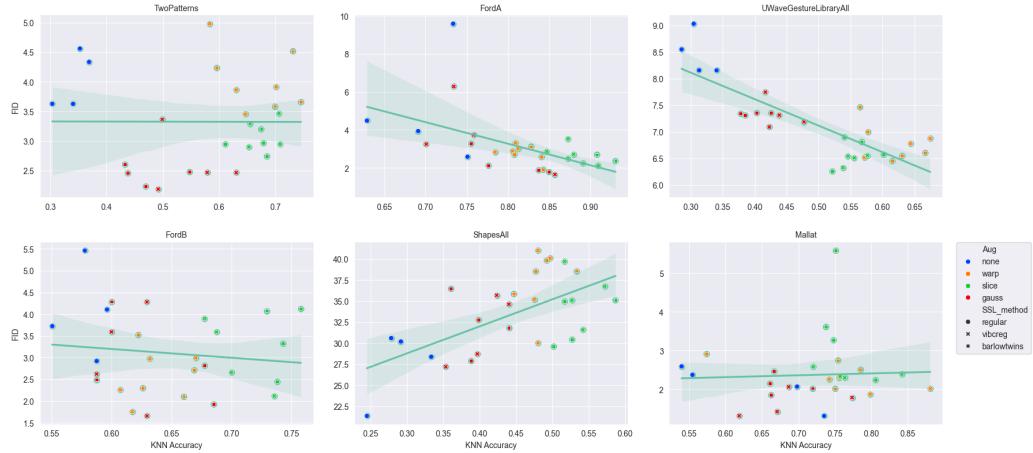


Figure 6.11: KNN plotted against Fréchet Inception Distance on the subset of datasets with significant improvement in probe accuracy.

surprisingly only observe minor differences in the generated samples from NC-VQVAE trained with different augmentations.

Typically naive VQVAE has more trouble with capturing the global consistency of the samples when 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. This is likely because some classes only have 2-5 sample in certain datasets.

ECG5000

In Figure 6.12 we present generated samples from naive VQVAE and NC-VQVAE trained with Window Warp and Amplitude Resize augmentations on ECG5000.

We see some evidence that VibCReg maintains more variability than Barlow Twins, while both has 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.

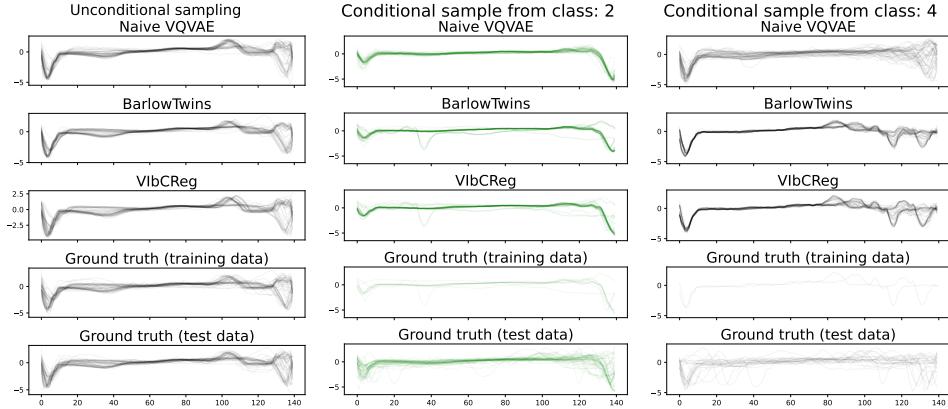


Figure 6.12: Dataset: ECG5000. Barlow and VibCReg both trained with window warp and amplitude resize augmentations. 50 samples from each model.

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 6.13. 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 6.9, particularly for VibCReg. It too explains why VibCReg with gaussian augmentation both increases IS and reduces FID.

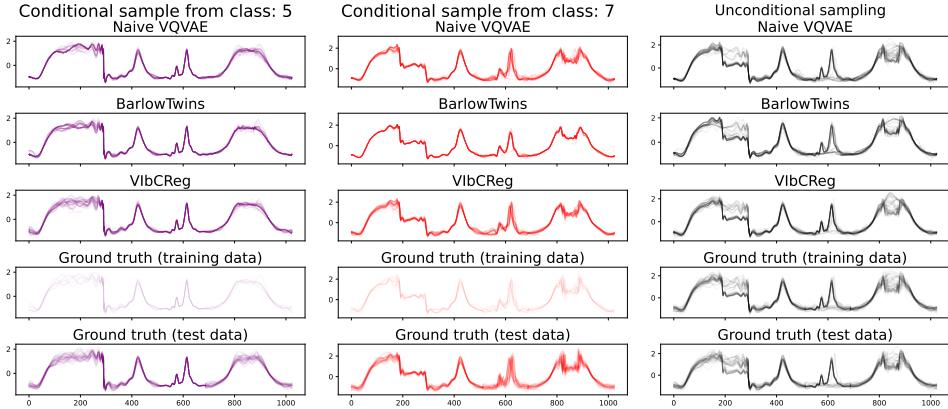


Figure 6.13: Class conditional distribution for some selected classes of Mallat, in addition to unconditional samples. Barlow and VibCReg both trained with gaussian augmentation.

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 6.15, 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.

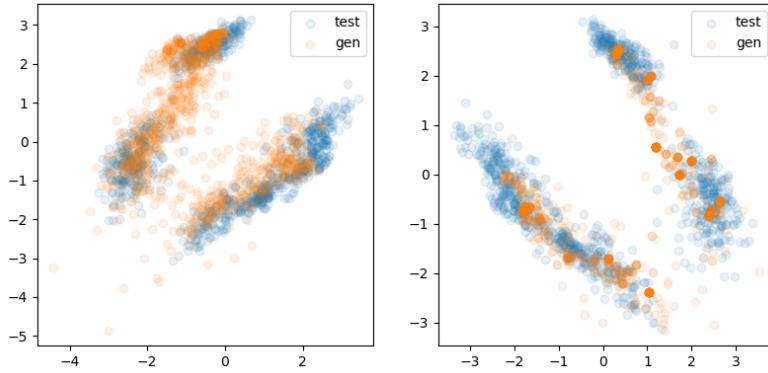


Figure 6.14: PCA of discrete latent representation from Mallat. Both VIBCReg (left) and Barlow Twins (right) are trained with gaussian augmentation.

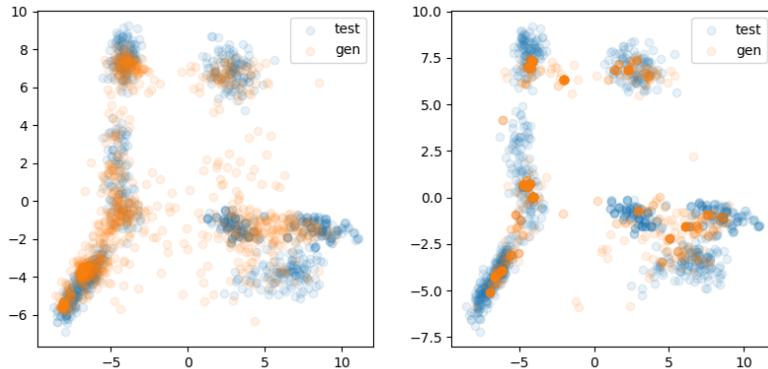


Figure 6.15: PCA of generated time series from Mallat. Both VIBCReg (left) and Barlow Twins (right) are trained with gaussian augmentation.

Symbols

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

In Figure 6.16, 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, VIbCReg to a slightly higher degree than Barlow Twins. 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 Barlow Twins, 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.

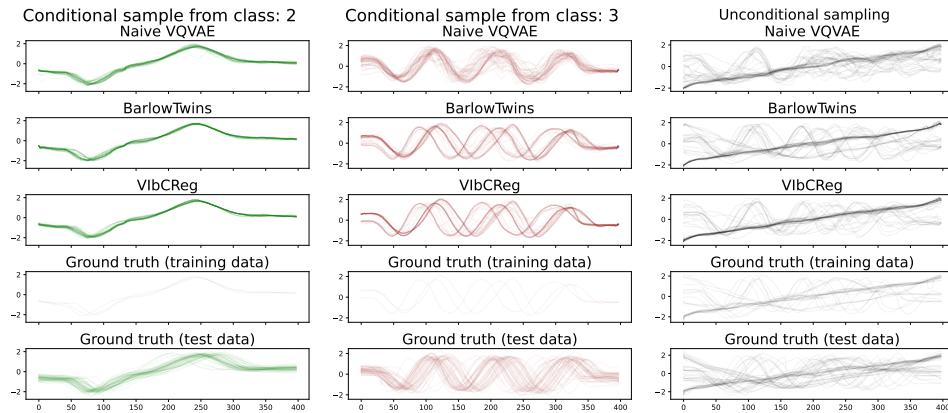


Figure 6.16: Class conditional distribution for some selected classes of Symbols.
Barlow and VIbCReg both trained with gaussian augmentation.

ShapesALL

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

In Figure 6.17, 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. As we have seen in the other datasets with few samples for a specific class, NC-VQVAE has a tendency to overfit.

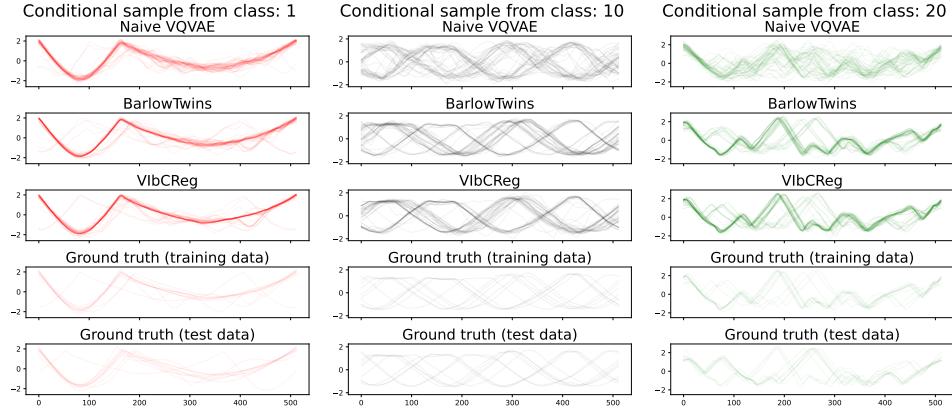


Figure 6.17: Class conditional distribution for some selected classes of ShapesAll. Barlow and VlbCReg both trained with gaussian augmentation.

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 6.18 a selected subset of classes are illustrated. We observe upon close inspection that VlbCReg maintains variability in the samples to a greater degree than Barlow Twins, as well as slightly better capturing the "dead spots" following the discontinuities.

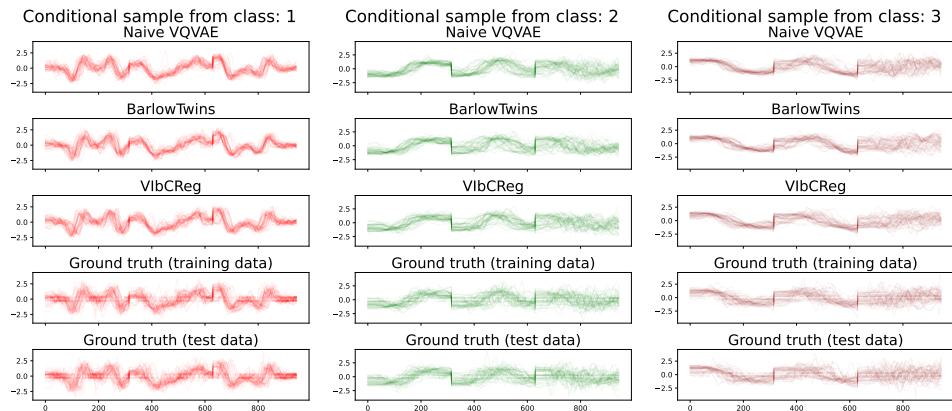


Figure 6.18: Class conditional distribution for some selected classes of UWaveGestureLibraryAll. Barlow and VlbCReg both trained with window warp and amplitude resize augmentations.

6.2.6 Augmentation Reconstruction Weight

TODO: Should this be included at all?

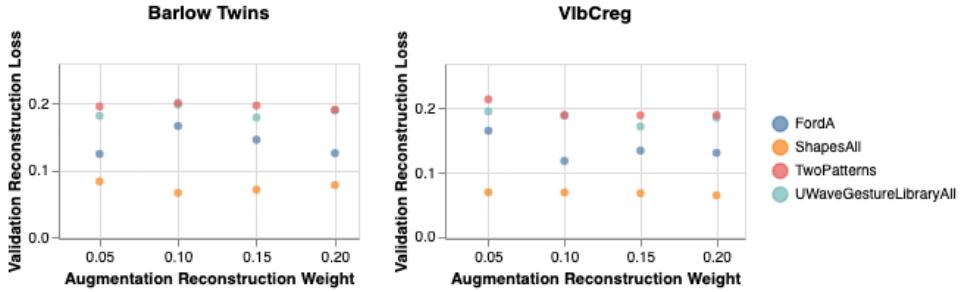


Figure 6.19: Augmentation: Window Warp and Amplitude Resize. Averaged across 2 runs. Trained for 250 epochs

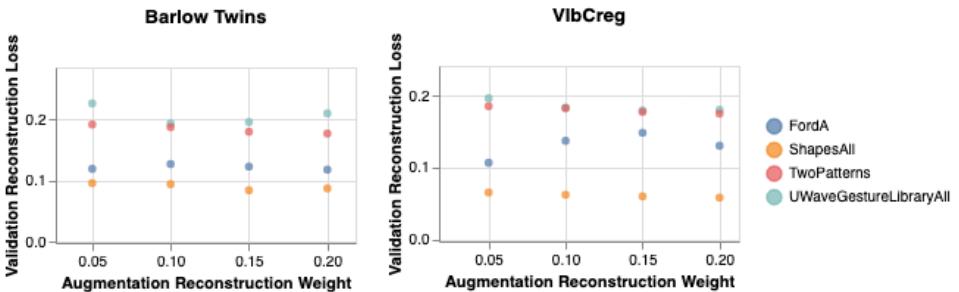


Figure 6.20: Augmentation: Slice and Shuffle. Averaged across 2 runs. Trained for 250 epochs

6.3 Discussion

6.3.1 Differences in Barlow Twins and ViBcReg

ViBcReg seems to keep the variability in the conditional distribution 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.

6.3.2 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. We have not employed any form of dropout or other regularization techniques when training on small datasets.

6.3.3 Temporal vs frequency influence of augmentations

We assessed and chose our different sets of augmentations based on their effect on the temporal representation on 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 sense puts more emphasis in the frequency components rather than the exact temporal structure.

The gaussian augmentation introduces HF component, though its influence in the time domain is visually clear, its affect on the spectrograms is minor, as the LF components typically has much larger magnitude. Window Warp and Amplitude Resize changes LF, 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 is a significant HF component.

Chapter 7

Conclusion

7.1 Conclusion

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

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

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

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

Even though there are issues, we believe our model is a step in the right direction. The representations learned by NC-VQVAE are more expressive than naive VQVAE, demonstrating that we can optimize more than one objective, without sacrificing the reconstruction capability.

The representations enable easier learning of the semantics of the conditional distributions, to such a degree that one has to take measures not to overfit.

The representations makes it easier to capture the gobal consistency of the samples, which tha naive VQVAE has large issues with. This without the HF-LF split.

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

7.2 Further work

[50] suggest that focus on augmentations is of great importance. The hunt for good augmentations in the time series domain is ongoing and should probably get more attention.

HF-LF split - augmentations tailored for HF and LF, as they often have quite different characteristics.

Wavelet transform to improve HF-LF split.

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

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

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

Investigation of the attention maps. Could one find if there is a HF direction, translation direction etc. similarly to how one in NLP can find gender direction, nationality direction etc?

7.3 Ethical Considerations

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] 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>.
- [3] 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](https://doi.org/10.1016/S0092-8240(05)80006-0). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0092824005800060>.
- [4] 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_XGPgAACAJ.
- [5] 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](https://doi.org/10.1016/0893-6080(89)90020-8). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0893608089900208>.
- [6] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [7] 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>.
- [8] 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>.

- [9] 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](https://doi.org/10.1162/neco.1989.1.4.541).
- [10] 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](https://arxiv.org/abs/1512.07108) [cs.CV].
- [11] K. He, X. Zhang, S. Ren and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: [1512.03385](https://arxiv.org/abs/1512.03385) [cs.CV].
- [12] 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](https://doi.org/10.1109/72.279181).
- [13] Y. Bengio, A. Courville and P. Vincent, *Representation learning: A review and new perspectives*, 2014. arXiv: [1206.5538](https://arxiv.org/abs/1206.5538) [cs.LG].
- [14] L. Jing and Y. Tian, *Self-supervised visual feature learning with deep neural networks: A survey*, 2019. arXiv: [1902.06162](https://arxiv.org/abs/1902.06162) [cs.CV].
- [15] K. Nozawa and I. Sato, *Empirical evaluation and theoretical analysis for representation learning: A survey*, 2022. arXiv: [2204.08226](https://arxiv.org/abs/2204.08226) [cs.LG].
- [16] 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](https://arxiv.org/abs/1706.03762) [cs.CL].
- [17] 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](https://arxiv.org/abs/2010.11929) [cs.CV].
- [18] S. Latif, A. Zaidi, H. Cuayahuitl, F. Shamshad, M. Shoukat and J. Qadir, *Transformers in speech processing: A survey*, 2023. arXiv: [2303.11607](https://arxiv.org/abs/2303.11607) [cs.CL].
- [19] 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](https://arxiv.org/abs/2202.07125) [cs.LG].
- [20] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: [1810.04805](https://arxiv.org/abs/1810.04805) [cs.CL].
- [21] A. Bardes, J. Ponce and Y. LeCun, *Vicreg: Variance-invariance-covariance regularization for self-supervised learning*, 2022. arXiv: [2105.04906](https://arxiv.org/abs/2105.04906) [cs.CV].
- [22] J. Zbontar, L. Jing, I. Misra, Y. LeCun and S. Deny, *Barlow twins: Self-supervised learning via redundancy reduction*, 2021. arXiv: [2103.03230](https://arxiv.org/abs/2103.03230) [cs.CV].
- [23] J. Bromley, I. Guyon, Y. Lecun, E. Säckinger and R. Shah, ‘Signature verification using a siamese time delay neural network.,’ vol. 7, Jan. 1993, pp. 737–744.

- [24] K. He, H. Fan, Y. Wu, S. Xie and R. Girshick, *Momentum contrast for unsupervised visual representation learning*, 2020. arXiv: 1911.05722 [cs.CV].
- [25] T. Chen, S. Kornblith, M. Norouzi and G. Hinton, *A simple framework for contrastive learning of visual representations*, 2020. arXiv: 2002.05709 [cs.LG].
- [26] D. Lee and E. Aune, *Computer vision self-supervised learning methods on time series*, 2024. arXiv: 2109.00783 [cs.LG].
- [27] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos and M. Valko, *Bootstrap your own latent: A new approach to self-supervised learning*, 2020. arXiv: 2006.07733 [cs.LG].
- [28] K. 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].
- [29] G. Mialon, R. Balestriero and Y. LeCun, *Variance covariance regularization enforces pairwise independence in self-supervised representations*, 2024. arXiv: 2209.14905 [cs.LG].
- [30] 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].
- [31] 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].
- [32] H. Chang, H. Zhang, L. Jiang, C. Liu and W. T. Freeman, *Maskgit: Masked generative image transformer*, 2022. arXiv: 2202.04200 [cs.CV].
- [33] 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].
- [34] D. P Kingma and M. Welling, *Auto-encoding variational bayes*, 2022. arXiv: 1312.6114 [stat.ML].
- [35] 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>.
- [36] A. van den Oord, N. Kalchbrenner and K. Kavukcuoglu, *Pixel recurrent neural networks*, 2016. arXiv: 1601.06759 [cs.CV].
- [37] 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].
- [38] 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].

- [39] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, *Improved techniques for training gans*, 2016. arXiv: 1606.03498 [cs.LG].
- [40] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, *Gans trained by a two time-scale update rule converge to a local nash equilibrium*, 2018. arXiv: 1706.08500 [cs.LG].
- [41] A. Borji, *Pros and cons of gan evaluation measures: New developments*, 2021. arXiv: 2103.09396 [cs.LG].
- [42] 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](https://doi.org/10.1016/0047-259X(82)90077-X). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0047259X8290077X>.
- [43] 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].
- [44] M. J. Chong and D. Forsyth, *Effectively unbiased fid and inception score and where to find them*, 2020. arXiv: 1911.07023 [cs.CV].
- [45] P. Esser, R. Rombach and B. Ommer, *Taming transformers for high-resolution image synthesis*, 2021. arXiv: 2012.09841 [cs.CV].
- [46] H. Barlow, ‘Possible principles underlying the transformations of sensory messages,’ *Sensory Communication*, vol. 1, Jan. 1961. doi: 10.7551/mitpress/9780262518420.003.0013.
- [47] L. Huang, Y. Zhou, F. Zhu, L. Liu and L. Shao, *Iterative normalization: Beyond standardization towards efficient whitening*, 2019. arXiv: 1904.03441 [cs.CV].
- [48] Y.-H. Huang, K. Huang and P. Fernández, *Vq-vae*, <https://github.com/nadavbh12/VQ-VAE>, 2021.
- [49] 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.
- [50] W. Morningstar, A. Bijamov, C. Duvarney, L. Friedman, N. Kalibhat, L. Liu, P. Mansfield, R. Rojas-Gomez, K. Singhal, B. Green and S. Prakash, *Augmentations vs algorithms: What works in self-supervised learning*, 2024. arXiv: 2403.05726 [cs.LG].

Appendix A

Additional Material

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

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

Prosjektavtale

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

_____ (oppdragsgiver), og

_____ (student(er))

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

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

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

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:

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

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

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

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

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

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.
6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligg i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.
7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.
8. Denne avtalen utfordiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.
9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.
10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.
11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): _____

Oppdragsgivers kontaktperson (navn): _____

Student(er) (signatur): _____ dato _____

_____ dato _____

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): _____ dato _____

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.

Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.

Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____