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

Johan Vik Mathisen

May 23, 2024

Abstract

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

Sammendrag

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

Contents

Chapter 1

Introduction

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

1.1 Acknowledgements

Supervisors, Erlend, IDUN, UCR Archive creators

1.2 Motivation

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

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

1.3 Overview/structure

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

1.4 Research questions

Stage1

RQ1: Does VQVAE learn good representations for classification?
RQ2: Will self a supervised learning approach enhance downstream classification while simultaneously reconstruct well?
RQ3: How does augmentations influence reconstruction and downstream classification?

Stage 2

RQ4: Will more expressive representations improve prior model learning?
RQ5: Does higher downstream classification accuracy correlate with improved class conditional generation?
RQ6: How does augmentations influence prior learning and synthetic sample quality?

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

1.5 AI Declaration

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

Chapter 2

Theoretical Background

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

2.1 Notation

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

2.2 Information theoretic/basic stats used in evaluation

- maximum entropy distribution
- mutual information

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

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

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

- perplexity

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

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

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

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

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

2.3 Time Series Inference

- short-time-fourier transform etc.

2.4 Neural Network

An *artificial neural network* or simply *neural network* is a fundamental model in machine learning, and more specifically in *deep learning*. Neural networks are loosely inspired by the way neurons are assembled in the brain. The model can be traced back the year of 1943 when Warren McCulloch and Walter Pitts developed the first artificial neuron [2], which is considered to be the first neural model invented. It was first set out in the real world by Frank Rosenblatt in 1957 [3]. But not until the development of the backpropagation algorithm in its modern form in the 1980's did the model really gain traction. Neural networks have since then been the highly influential in the development of the machine learning field, with an impressive resume of applications. Some of which, if we stretch the definition a bit, include face recognition, beating humans in chess, Go and Starcraft, self-driving cars and predicting the structure of proteins. The unreasonable effectiveness of neural networks on a broad range of tasks can in part be explained by the *universal approximation theorem*, proven by Kurt Hornlik in 1989 [4], which roughly states that a neural network can approximate any (Borel measurable) function to any desired degree of accuracy.

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

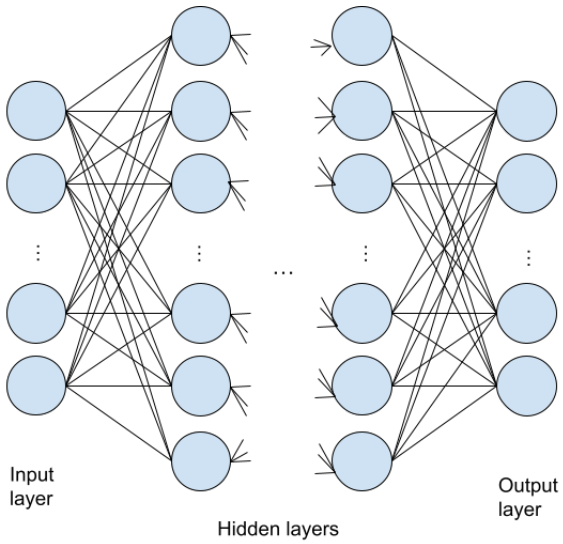


Figure 2.1: Illustration of a Neural Network model.

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

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

where

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

and f_K is the output layer, with application dependent structure.
The introduction of nonlinearity by the activation function is what enables the model to approximate nonlinear signals and differentiate itself from a linear regression model. 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 options exists.
The architecture of neural networks, and most specializations thereof, is sequential in nature. They can effectively be described as compositions of some combination of a flavour of matrix multiplication, non-linear transformation and down or upsampling.

2.4.1 Training Neural Networks

The training of a neural network is the process of finding values for the weight and bias parameters. The general idea governing neural network training is to optimize the parameters based on some distance metric between the predicted values and the target values. This distance metric is referred to as the *loss function*, and

a common choice of loss function is the mean squared error (MSE).

For a multi-variable function F which is differentiable around a point a, the direction which it decreases fastest when starting in the point 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 update

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

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

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

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

and one would iteratively update the parameters by gradient descent

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

If the dataset X is large, the gradient calculations are expensive. In these cases, which in modern machine learning is the standard scenario, an effect estimator for the true gradient is used instead. Stochastic gradient descent (SDG) is a very popular approach. Instead of computing the gradient at each datapoint every iteration, SDG updates the parameters by iterating through the dataset and using the gradient at each single datapoint. The dataset is then permuted and iterated through again until an approximate minimum is reached. Pseudocode of SDG is presented in algorithm 1.

Algorithm 1 Stochastic Gradient Descent (SDG)

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

TODO: Mini batch SDG

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

For a through introduction to the subject of neural networks and the training thereof we refer to chapter 6 and 8 of [5].

2.5 Convolutional Neural Network

This section draw inspiration on the presentation of convolutional networks in chapter 9 of [5].

A convolutional neural network (CNN) is a particular type of neural network that is developed to learn local features in the data. This local feature learning is enabled by the mathematical operation of convolution. In essence a CNN is a neural network where matrix multiplication is switched for convolution at least one of the layers [5].
Fully connected neural networks have a fundamental drawback in that their computational complexity grows intractably large when the input dimensionality is high. This makes them unsuited for high dimensional data, such as images. Convolutional neural networks directly address this issue via various downsampling techniques. Convolutional neural networks has a rich history and a long track record of success stories. They are inspired by the architecture of visual cortex cells in mammals. Inspired by the discovery of Hubel and Wiesel [6] the neocognitron was proposed in 1980[7]. The neocognitron is widely considered the predecessor of convolutional neural networks. In 1989 Yann LeCun et al. introduced the modern framework for CNNs [8] and demonstrated its effectiveness on the task of hand written digit recognition. Since then CNNs have been an indispensable part of machine learning research, especially in the computer vision domain. For an exposition on the advances on convolutional neural networks and its applications we refer to [9].

2.5.1 The convolution operation

The convolution operation is an integral transform with extensive applications. It generalizes the notion of a moving weighted average. In mathematics it is ubiquitous because of its relationship with the Fourier transform.

Let f and g be real valued functions, then their convolution is defined as

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

The mathematical nuances of the exact criteria for the above integral to exist is outside the scope of this thesis, and not particularly relevant. But if f and g are integrable (in the Riemann or Lebesgue sense) then the convolution exists. As a rule of thumb, the convolution of f and g is as "smooth" as the smoothest of f and g. It is worth mentioning that convolution is commutative, i.e that $f * g = g * f$, which can be seen by a simple change of variables.

As is typical for integral transforms, the function g is referred to as the *kernel*. In the context of convolutional networks the kernel consists of learnable parameters and the function f is the *input*. The output is referred to as the *feature map* [5]. In machine learning we handle discrete signals, represented as multidimensional arrays. As a result we must employ a discrete variation of the convolution operation. Let I be the input and K be the kernel, both discrete, then their convolution is defined as

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

In practice I and K typically has finite support, i.e they are zero for large positive and negative arguments, which circumvents any convergence problem.
Convolutions are naturally defined for higher dimensional functions by component wise extension. For a two dimensional image I and a kernel K we calculate their convolution as

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

The operation is illustrated in figure **??**.

Convolution in machine learning does not always correspond exactly to the mathematical definition of the operation, but rather to cross-correlation. The difference is just a sign flip in the kernel arguments. Operation is no longer commutative, but in practice this does not affect anything as the learned kernel parameters will be equivalent [5].

In the field of digital signal processing discrete convolutions are used extensively. Traditionally predefined kernels are used to alter a signal in predictable ways. Two well known kernels are Gaussian and edge detection kernels. Gaussian kernels are two dimensional gaussian distributions and are used for blurring. Edge detection kernels are designed in such a way that areas of similar intensity are mapped to 0 and areas of variable intensity are mapped to high values. In figure 2.3 we demonstrate the effect of two such kernels. Edge detection in particular provide a hint to how CNNs learn interesting features in the data. In a convolutional layer the kernels are learned such that the feature map is helpful for the training objective.

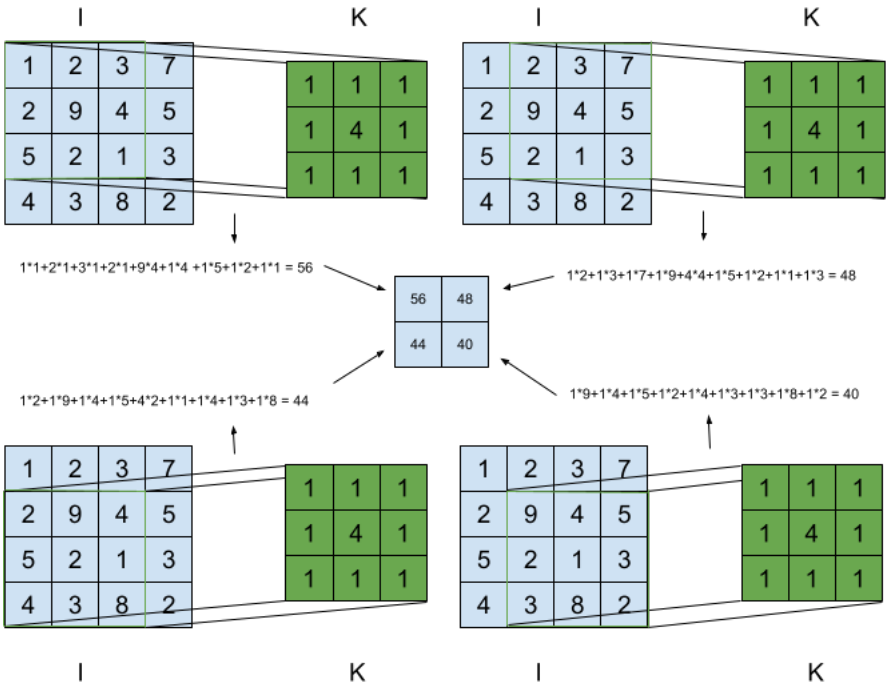


Figure 2.2: Illustration of discrete two dimensional convolution

2.5.2 Pooling

As mentioned earlier CNNs are mainly used when the data has high dimensionality. In order to reduce the dimension to a manageable level *pooling* is used. A pooling operation is applied as a down sample technique on feature maps, replacing regions of the output with summary statistics. Two of the most common are max and average pooling, which replaces the region by its maximal or average value respectively. There are two hyperparameters for any pooling operation, the filter size, which determines the region of values to calculate the summary statistic, and stride length, which determines how the filter moves across the feature map. In addition to dimension reduction, pooling assist in making the representations approximately invariant to small distortions of input. Illustrations of max pooling with different stride is presented in figure **??** and **??**, while the effect of max and average pooling on image data is illustrated in **??**.

2.5.3 Architecture

There is a large variety of specific architectures which fall in the category of a CNN, but their basic components are largely the same. They consist of convolutional layers, pooling layers and fully connected layers. In figure 2.8 an illustation of the original LeNet-5 [8] is presented as an illustration of a general CNN.

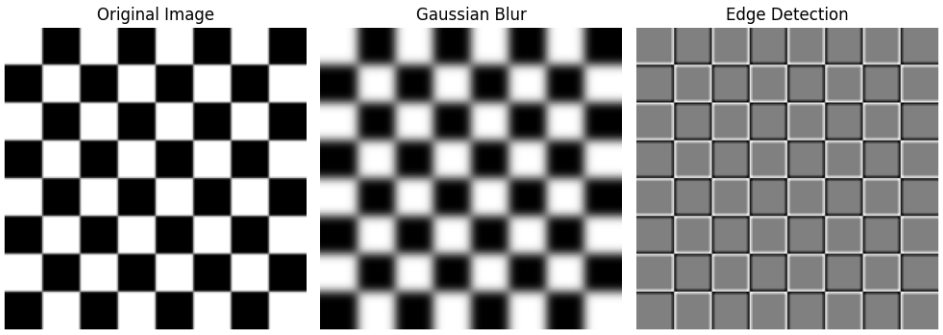


Figure 2.3: Illustration of discrete convolution applied to images. Image part of Scikit-image data package.

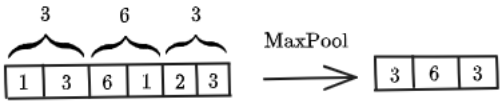


Figure 2.4: Max pooling of one dimensional array. Filter size: 2, stride: 2.

A convolutional layer consists of several kernels used to compute different *feature maps*. Each kernel convolved with entire input, and the different feature maps are produced by changing kernel. A nonlinear activation function is then applied pointwise to the feature maps. **A pooling layer** is typically placed between convolutional layers and work as a stronger downsampler which aims to enforce approximate translation invariance. A pooling operation is applied across each feature map.

The fully connected layer is just your typical hidden layer in a neural network, i.e connect every input to every node in the output. Because of computational issues mentioned earlier fully connected layers are first introduced when the input data has been sufficiently downsampled.

2.5.4 Transposed Convolutional Networks

Transposed convolution or deconvolution, also known as fractionally-strided convolution is a technique used to reverse the downsampling from convolutions. In essence it is an inpainting or upsampling technique known from digital signal processing. The flexibility of learning data dependent transposed convolutional kernels enables one to more effectively reverse the downsampling from convolutional layers. They are extensively used in combination with convolutional downsampling in *encoder-decoder architectures* presented in section 2.9.

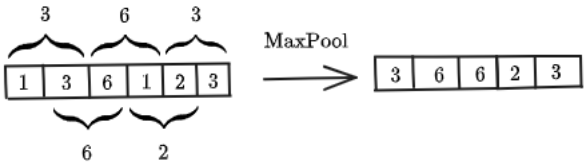


Figure 2.5: Max pooling of one dimensional array. Filter size: 2, stride: 1.

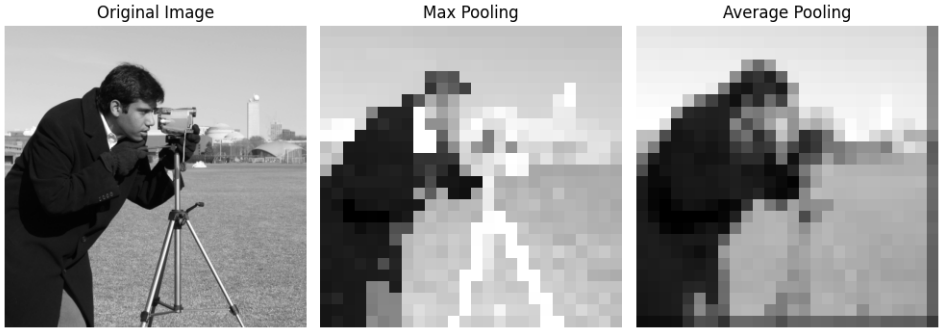


Figure 2.6: Illustration of mean and average pooling applied to images. Filter size and stride is 20×20. Original image size is 512×512, pooled images are 26×26. Image part of Scikit-image data package.

2.6 Representation Learning

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

2.6.1 What a representation?

Representation learning is a term not too easily defined, one reason being the abstraction level. It is helpful to first consider what is meant by *representation* of information. Lets begin by walking through a familiar and illustrative example. Consider the base ten integer $(4)_{10} = 4$. The number can equivalently (in terms of information content) be expressed, that is represented, in any other base. The particular base we choose depends on our intention with the number. If we want to work with digital electronics, a binary representation ($(4)_2 = 10$) is very useful, as transistors has two states. When humans do arithmetic, base ten representations of the integers are very natural, as we have ten fingers. A particular representation of information can make a task easier or harder. The information content is unchanged by a change of representation. What is changed is the easiness or difficulty of certain information processing tasks. Representation learning is then the process of learning a certain representation of information.
Representations are too highly dependent on who, or what, that will process it. An example is time. Humans have developed a standardized system for writing timestamps which works fairly well for us. But, if we want to model time depend-

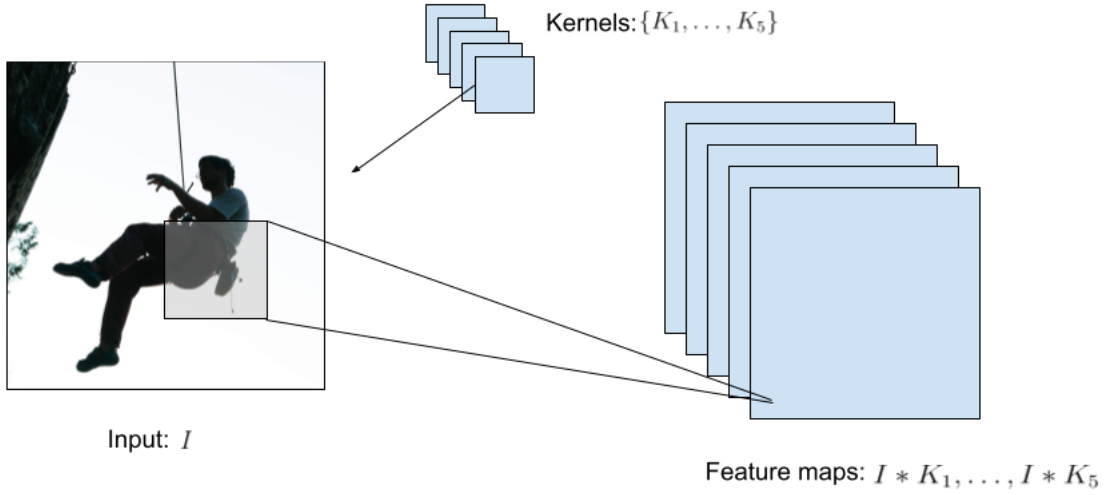


Figure 2.7: Convolutional layer.

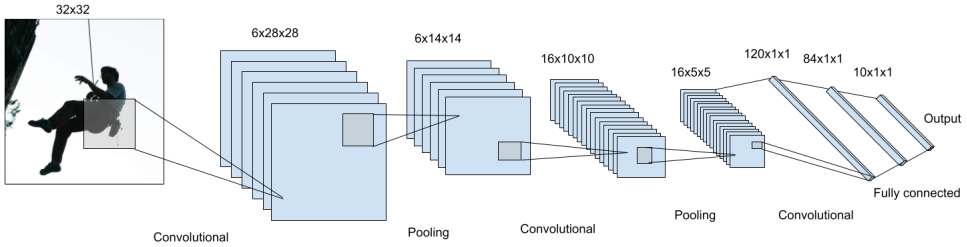


Figure 2.8: LeNet-5 network [8]

ent phenomena, say using tabular data, the DateTime representation is of very little help to a tree based model for instance. The reason being that the numerical representation of timestamps close in time is not necessarily close in numerical value. Think of 23:59 to 00:00. A possible solution is to change the representation such that the numerical values actually respect the periodic nature by mapping to the circle. The new representation is then useless for humans, but quite a lot more useful to a computer. Representation learning in machine learning can thus be thought of as algorithms which are designed to learn representations which is useful for a ML objective.

2.6.2 Why do we care about representation learning?

Those who has worked with data science or machine learning and has come across feature engineering are familiar the effect good feature engineering has on a models performance. The same people too knows the level of domain expertise, creativity and time is needed to feature engineer well. In reality much of the actual time spent in the process of deploying machine learning methods revolves around

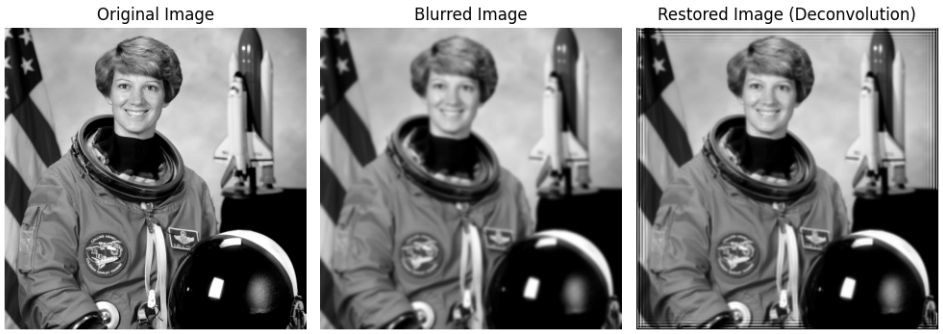


Figure 2.9: Illustration of deconvolution applied to images. Here using Gaussian blur and restoring using the Richardson Lucy algorithm with a Gaussian deconvolution kernel. Image part of Scikit-image data package.

constructing good data pipelines and applying transformations that produce representations beneficial for the algorithm at hand [10]. 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 [10].

2.6.3 What is a good representation?

The goodness of a representation is then determined by how easy it makes a subsequent task. The concept of universally good representations is ill-defined, for any representation extracted of a non-invertible function, a *downstream tasks* can be designed (in principle) to based on the lost information, hence achieve arbitrarily bad performance. There is no free lunch in representation learning either. One must specify a set of predefined downstream tasks, and evaluate according to those. Intuitively the quality of the representations are also considered higher if the the representations is able to perform well on several downstream tasks i.e when they are more general.

2.6.4 How does one evaluate representations?

There are several different evaluation protocols in representation learning. They involve training a model on a *pretext task*, which as defined in [11] is a task for a network to solve, where the goal is to learn representation. Then the learned representations are evaluated on a downstream task. In general the downstream task is solved in a supervised manner, using human annotated data.

In a N-layered network $f = f_N \circ ... \circ f_1$, the intermediate value of the data x in some layer n is what is meant by the networks learned feature representations.

When we are interested in the representations learned it thus is helpful to dissect a 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 the pattern

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

The standard evaluation protocol is to train a linear head g_D on top of the *frozen* representations in a supervised manner and evaluate this models performance. This is to say that we train $f_D = g_D \circ \widehat{h}$ by only updating the parameters of the linear model g_D, and evaluate $\widehat{f_D}$ on some 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 common alternative protocol is similar to the one above, but rather than freezing the feature extractor one let all parameters of f_D to be learnable on the downstream task. This protocol is referred to as a pretraining-finetuning.

It is of interest how f_D performs in terms of accuracy on the downstream task and training time, and too how these sensitive metrics are to training data size. The baseline comparison would then be an identical but randomly initialized model. It is considered highly advantageous if one is able to pretrain a feature extractor using cheap and abundant data in a way which ensures faster convergence on a downstream task where data is expensive or scarce.

It is with 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 we refer the reader to [12].

2.7 Transformers

Since their introduction in the seminal paper titled "Attention is all you need"[13] transformers have taken the field of machine learning and artificial intelligence by storm. It is the enabling architecture behind large language models (LLMs) such as Googles BERT, Metas Llama and OpenAIs ChatGPT. Drawing inspiration from the initial success in natural language processing vision transformers [14] were developed for computer vision applications, and similar trends are shown for other modalities such as audio [15] and time series [16].
Transformers were developed for sequential modelling and has capabilities of learning far ranging dependencies in data through the *multi-headed attention mechanism*. In the realm of sequential modelling recurrent neural networks and

cite

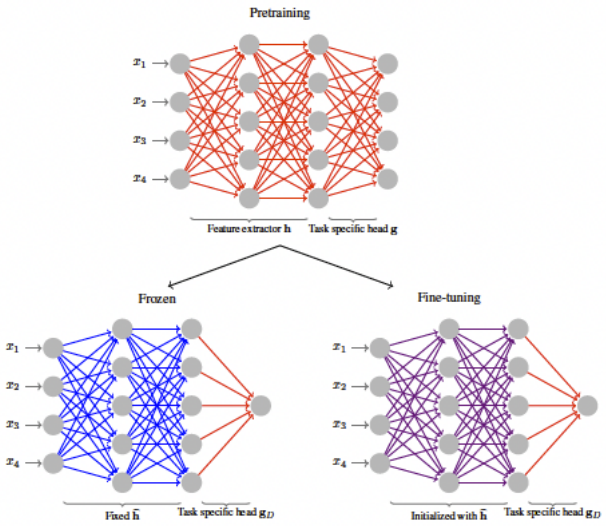


Figure 2.10: Taken from [12]

long short-term memory neural networks were the previous state of the art, but modeling long term dependencies has proven difficult [17] as one encounters the *vanishing or exploding gradient problem*. 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.

cite

The transformer takes as input a sequence of symbols. This means in particular that for a specified modality, e.g text, speech, image and time series, they must first be translated into a sequence of symbols. This is accomplished by a process called *tokenization*. The conceptually easiest word-level tokenization in NLP, where one creates a dictionary of all words in the dataset, and assigns to them an integer value. A piece of text can then with relative ease be mapped to a sequence of integers. Other modalities as speech, image and time series are usually represented as matrices and can therefore naively be modeled as a sequence numbers. Because of the high dimensionality, this would require a significant amount of computational resources. In addition the transformer would need to model incredibly far ranging dependencies and handle redundant information. Hence various tokenization methods are used to represent the data as coarser sequences.

Transformers can be adopted for generative tasks, and has done so with great success. 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 conditional on the input. With the introduction if vision transformers [14], input images were split into patches and linearly projected before being processed by the transformer. Since then *Vector-Quantized*-based tokenization introduced by [18] has emerged as a popular approach.

2.7.1 The attention mechanism

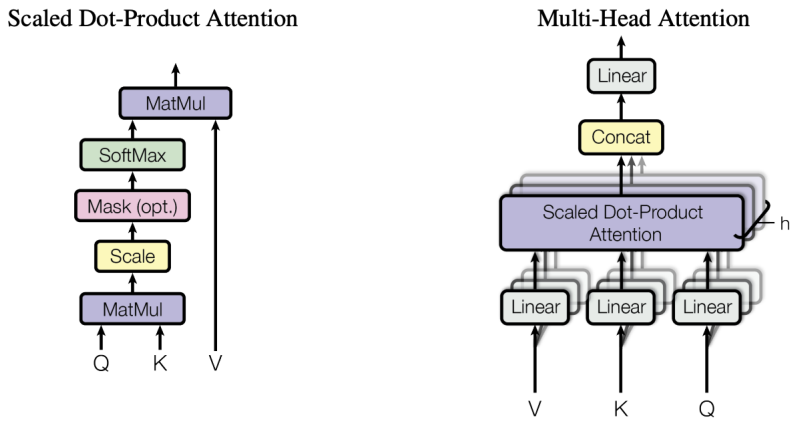


Figure 2.11: (left) Scaled dot-product attention. (right) Multi-headed attention. Taken with permission from [13]

"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". Scaled dot-product attention is computed by

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

Multi-headed attention projects the inputs Q, K, V down to h subspaces, where the scaled dot product attention is applied to each subspace. The individual attention heads are concatenated before passing through a final linear layer. That is to say

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

where

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

The multi-headed attention enables the transformer to focus on different aspects of the input sequence simultaneously.

2.7.2 Architecture

An overview of the original transformer architecture is presented in figure **??**. The architecture has several distinct components.

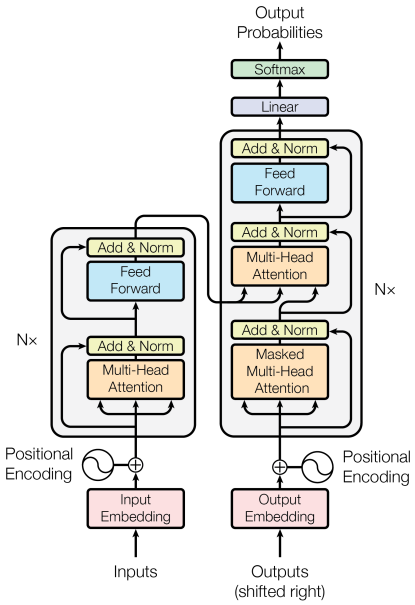


Figure 2.12: The Transformer - model architecture. Encoder (left), decoder (right). Taken with permission from [13]

Embeddings: The input embedding maps each token in an input sequence to a vector. The input embedding is essentially a lookup table.

Positional encoding: The transformer does not rely on convolution or recurrence to make use of the order of the input sequence. Instead it relies on a positional encoding, which is done by adding information about the relative position of an element in the embedded sequence. In the original paper [13] sinusoids are used to encode the position with unique values.

Encoder: The embeddings with positional encodings are passed through the transformer encoder, which consist of N identical layers. The layers consist of the multi-head attention and a fully connected neural network, with residual connections and normalization at each sublayer.

Decoder: The decoder similarly consists of N identical layers. They consist of a masked attention layer, which prevents the decoder to utilize future values in the sequence during training, a multi-head attention applied to the output of the encoder and a fully connected neural network. All sublayers employ residual connections and normalization. The output is converted to probabilities in the standard way by a linear layer followed by softmax.

Bidirectional transformer models such as BERT [19] use an encoder only architecture, which enables the attention to attend both directions. The issue of "seeing in the future" is resolved by a training technique referred to as *Masked*

modelling 2.8.2.

2.8 Self-Supervised learning

Self-supervised learning (SSL) has had great success in natural language processing and computer vision in recent years, and is quickly being applied for other modalities. In our work we leverage SSL for representation learning in the time series domain. Before we introduce SSL, it is with mentioning that machine learning models can be coarsely divided into two categories of learning, supervised and unsupervised.
Supervised learning refers to models who learn using labeled data. That is to say for a given input x we already know what the desired output y is during training, and can therefore supervise (update) our models parameters by directly comparing model output and 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 f_θ in such a way that the loss across the data is minimized

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

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

The models falling under the supervised learning category are widely deployed and has seen tremendous success. Classical statistical models, as well as support vector machines and decision tree based models are all examples of models in this learning paradigm. The main issue with supervised learning is the need for labeled data, and labeled data is in many ways scarce.

Unsupervised learning on the other hand refers to models or algorithms who learn exclusively from unlabeled data. That is to say that the models learn intrinsic patterns in the data. Examples of unsupervised learning models are clustering methods as K-means, K Nearest Neighbor and Gaussian mixture models, dimension reduction techniques as PCA and SVD and neural network architectures such as Autoencoders. Unsupervised algorithms are used extensively in exploratory data analysis, data visualization and clustering. In a world with abundant unlabeled data, good unsupervised techniques are quite valuable. Until recently such techniques have been hard to come by and typically fallen short compared to supervised approaches.

Self-supervised learning is subcategory of unsupervised learning which has shown great success within NLP and computer vision, closing in on the state of the art supervised representation learning [20] [21]. SSL refers to model who use

the data itself to generate a supervisory signal, rather than external labels as in supervised learning. Even as SSL is considered unsupervised learning, the learning formulation is quite similar to 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 f_θ in such a way that the loss across the data is minimized. In other words find

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

A pseudo label is an automatically generated label from the data. In *siamese* architectures the pseudo labels are typically some *augmentation* of the original data.

SSL has in recent times been a fruitful approach to unsupervised representation learning and has played an integral part in the boom of large language models together with the invention of the transformer architecture. With models growing ever larger and more data hungry, SSL is essential in pretraining. Randomly initialized networks are difficult to train and requires a lot of time and computational resources. SSL has shown remarkable results when used for pretraining. That is as models for learning network parameters who capture semantics of data, without the need for labels. Pretraining networks enables foundation models, which can be trained for many different tasks in a supervised fashion requiring a lot less resources.

Many successful SSL models in the computer vision domain share a similar architecture.

2.8.1 Siamese Architecture-based SSL

A siamese network architecture [22] consists of two networks, called branches, with a shared encoder on each branch. Such networks are trained to produce similar representations for different views of the same data. In models with such architecture, the existence of trivial solutions, such as both networks ignoring input and produce identical constant embeddings, is a major issue. This issue is referred to as *collapse* of the model.

In the computer vision domain several representation learning algorithms utilizing a siamese architecture have been proposed. They mainly fall under two categories: *contrastive* and *non-contrastive* SSL.

Contrastive SSL uses positive and negative samples and learns representations by pulling positive pairs closer together and negative pairs further apart. Examples include MoCo [23] and SimCLR [24]. SimCLR constructs two correlated samples by applying augmentations to an input sample, and considers these as positive

pairs. Contrastive methods requires large batch sizes as well as a large number of negative pairs compared to positive in order to learn representations effectively [25]. These obstacles lead researchers to look elsewhere.

Non-Contrastive SSL methods such as BYOL [26], BarlowTwins [21] and VIbCReg [**computer**] circumvent the need for positive and negative pairs while simultaneously avoiding model collapse by different methods. Common for all is the use of augmentations, a transformation of the input that works as a pseudolabel. The models learn to map representations of the input and augmentation closer together. In order to avoid collapse BYOL introduces asymmetry in the architecture as well as the stopgradient operation, Barlow Twins minimizes the information redundancy between the two branches and VIbCReg by maintaining the variability in each branch while decorrelating features. Common for all is the ability to learn useful representations.

2.8.2 Masked modelling

Masked modelling is a conceptually simple self-supervised learning technique for generative models. The idea is to mask or cover a portion of the data and predict the masked portions. By comparing the prediction against unmasked data the model can learn useful representations without supervision. Since the introduction of masked modelling in natural language processing by [19] it has been the de-facto standard for self-supervised pretraining of language models. BERT introduced masked modeling for language representation learning as means to use bi-directional transformers without enabling each word to "see itself".

In the computer vision domain masked modelling has too gotten much attention. Early works in masked image modelling [27] attempted to mask image patched directly with some success. But when vision transformers [14] surpassed CNN in 2021[28], CV research began to draw inspiration more heavily from [19] and began tokenizing images and pretraining transformers. In 2022 MaskGIT[29] proposed masked visual token modelling , utilizing vector-quantized-based tokenization together with a bidirectional transformer. For a thorough survey on masked modelling for SSL in the vision domain we refer to [30].

2.9 Vector Quantized Variational Autoencoder (VQVAE)

Our model is based on the Vector Quantized Variational Autoencoder (VQVAE) introduced in [18], and includes an Auto-encoder (AE) branch. Therefore it is natural to dive into the models. We first start with introducing auto-encoders, then present the variational variation VAE before presenting VQVAE.

2.9.1 Autoencoder (AE)

Consists of two neural networks, an encoder E and decoder D. They can be seen as maps between spaces X and Z, where we refer to X as the data space and Z as the latent space.

$$X \xrightarrow{E} Z \xrightarrow{D} X \tag{2.13}$$

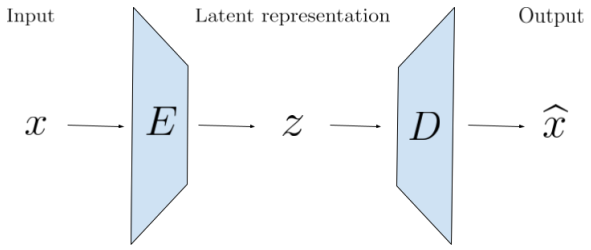


Figure 2.13: Schematics of the autoencoder architecture.

The autoencoder is trained in such a way that the composition of encoder and decoder is approximate to the identity. Typically the dimension of the latent space is much lower that that of the data space. Encoder and decoder then compresses and decompresses the data and learns efficient *latent representations*.

TODO: Information bottleneck

TODO: Issues with latent representations and the need for regularization

2.9.2 Variational Autoencoder (VAE)

Variational Autoencoders (VAE) were introduced in [31] and is a variational Bayes approach to approximate inference and learning with directed probabilistic models. The architecture of VAE is similar to AE, but the mathematical formulation is quite different. For context variational inference is a technique in statistics used to approximate complex distributions by looking for the closest approximation within a simple, 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. This is to say that we assume there is a random 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. As $\mathbf{z}$ 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 $p_\theta(x|z)$. As with any model where one wishes to employ gradient based learning, the distributions are assumed to be differentiable almost everywhere, both with respect to their parameters and argument.

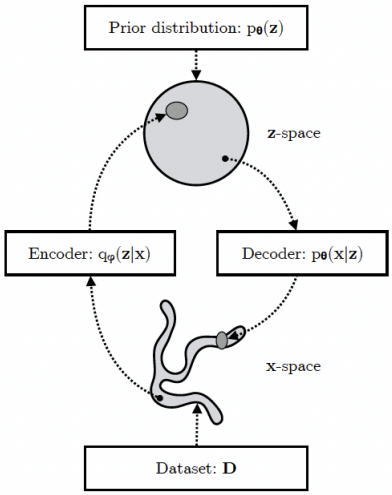


Figure 2.14: [32], need to ask for permission or make own

VAEs have two components to their architecture. The first is a probabilistic encoder, often called the inference model, $q_\phi(z|x)$ which approximates the true posterior. Secondly a probabilistic decoder, often called the generative model $p_\theta(x|z)$, which approximates the likelihood. These models are typically parameterized by some type of neural network, and in that case ϕ and θ are the weights and biases of the two networks. 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 part of the encoder and decoder is the sampling. The output for a given x or z is a distribution, and for the same input the same output distribution is calculated (as long as the network parameters are frozen). Actually x and z are mapped to the parameters of a distribution, which uniquely determines the distribution in a particular family.

The most common situation is to assume

- 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 parameters of the variational distribution, i.e the approximate posterior 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 $p(x|z)$, $z_i \mapsto D(z_i)$.

There are several reasons for choosing Gaussian distributions, one being that the Gaussian distribution a scale-location family. This enables us to circumvent the problematic random component in the VAE when using gradient based learning. As described above we sample $z \sim N(\mu_x, \Sigma_x)$ where $E(x) = (\mu_x, \Sigma_x)$. We can equivalently sample from $N(\mu_x, \Sigma_x)$ by reparameterization using the location-scale property. This means to introduce an auxillary variable $\epsilon \sim N(0, I)$ and rewriting $z = \mu_x + L_x\epsilon$, where L_x is the Cholesky decomposition of Σ_x. This factors the random part out of path of gradient flow.

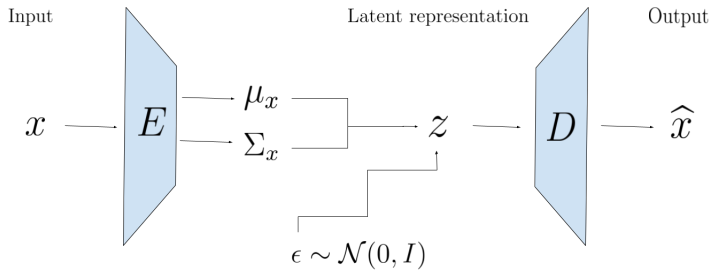


Figure 2.15: Schematics of the VAE architecture with Gaussian reparameterization.

Training objective

As with other variational methods VAEs are optimized with respect to the *evidence lower bound* or ELBO for short. Let **x** and **z** be two jointly distributed variables, with distribution p_θ. Then for any distribution q_ϕ the ELBO is defined as

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{p_\theta(x,z)}{q_\phi(z|x)}\right) \tag{2.14}$$

The ELBO can be reformulated in terms of the marginal likelihood and KL divergence of the variational to the true posterior.

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

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, which improves the generative model, as well as reducing the KL-divergence of the approximate to the true posterior, which improves the inference model [32].
An alternative reformulation gives a more evident connection to Autoencoders,

with a the prior acting as a regularizer on the posterior

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)} \log\left(p_\theta(x|z)\right) - \mathbb{E}_{q_\phi(z|x)} \log\left(\frac{q_\theta(z|x)}{p_\phi(z)}\right) \\ &= \underbrace{\mathbb{E}_{q_\phi(z|x)} \log\left(p_\theta(x|z)\right)}_{\text{Expected reconstruction log likelihood}} \underbrace{- \mathrm{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{Regularizer}}.\end{aligned} \tag{2.16}$$

As the $p_\theta(x|z)$ is typically assumed to be Gaussian we have

$$\log p_\theta(x|z) = -\frac{1}{2}\left[k\log(2\pi) + (x - D(z))^T(x - D(z))\right], \tag{2.17}$$

which is equivalent, as an optimization objective of $D(z)$, to $||x - D(z)||_2^2$. Consequently the likelihood in the loss is implemented as the MSE of the input x and the output $D(z)$.

Generative model

The role of the prior in a VAE is two fold. Firstly as a regularizing constraint for the posterior during training, and secondly as a way of obtaining new samples x via ancestral sampling. Ancestral sampling refers to **z** being the parent node of **x** in the VAE, and that we can generate samples x by drawing a sample $z \sim p(z)$ and decode $D(z)$.

TODO: Speak on limitations of VAE before introducing VQVAE

Variational autoencoders has the issue of collapsing. Variance issues.

2.9.3 Vector Quantization (VQ)

Dictionary learning model [33]

2.9.4 VQVAE

The Vector Quantized Variational AutoEncoder (VQVAE) was first introduced in [18] and presented a new way of training VAEs with discrete latent variables. It is the first discrete VAE model which has similar performance to the continuous variant. VQVAE was developed to learn useful discrete representations for generation. This is enforced by more flexible distributional assumptions, as they say "Our model, the Vector Quantised- Variational AutoEncoder (VQ-VAE), differs from VAEs in two key ways: the encoder network outputs discrete, rather than continuous, codes; and the prior is learnt rather than static."-[18].
The overall architecture of VQVAE consists of an encoder and decoder as before, together with a *codebook* which is used in the quantization process. The codebook, also referred to as the latent embedding space, $\mathcal{Z} = \{z_k\}_{i=1}^K$ is consists of K latent vectors with dimensionality D. It can be considered a lookup table of vectors that are used for vector quantization (VQ). The output of the encoder $E(x)$

is quantized by the means of nearest neighbor lookup

$$z_q = \arg\min_{z_k \in \mathcal{Z}} ||E(x) - z_k||. \tag{2.18}$$

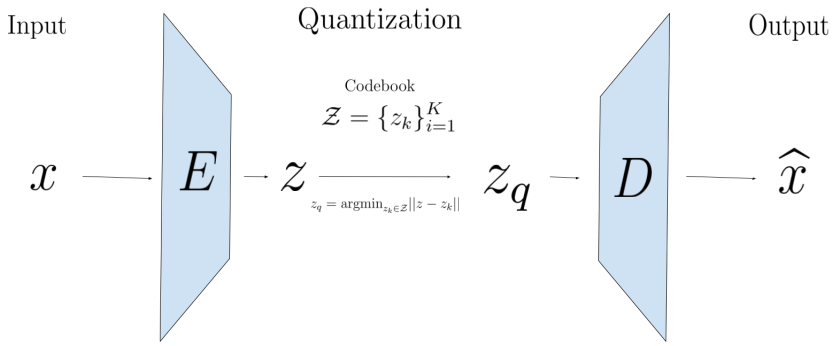


Figure 2.16: Schematics of the VQVAE architecture.

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

In the original article [18] they propose to separately learn the prior distribution, after the encoder, decoder and codebook is learned. During the initial training the prior is assumed to be uniform, which poses no constraint on the posterior learning.

In contrast to VAEs the posterior is assumed to be categorical, as opposed to normal and the probabilities are defined as

$$p(z = k|x) = \begin{cases} 1 & \text{for } k = \arg\min_j ||E(x) - z_j||_2 \\ 0 & \text{otherwise} \end{cases}, \tag{2.19}$$

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

VQVAE can be viewed as a VAE, hence we can bound the marginal likelihood with the ELBO 2.16. As the variational posterior is deterministic and the prior (during training) is uniform over $\{1, ..., K\}$ we get that the regularizing term

$$\begin{aligned} \text{KL}(q(z|x)||p(z)) &= \sum_z q(z|x) \log\left(\frac{q(z|x)}{p(z)}\right) \\ &= q(z = k|x) \log\left(\frac{q(z = k|x)}{p(z = k)}\right), \; q \text{ is deterministic} \\ &= \log\left(\frac{1}{1/K}\right), \text{ uniform prior} \\ &= \log(K), \end{aligned} \tag{2.20}$$

is constant. Consequently the ELBO reduces to the reconstruction term only.

TODO: Can think of the codes as the modes of a mixture of Gaussians, and instead of sampling from the mixtrure you just choose the closest mode.

Prior Learning

During the initial training the prior is as mentioned constant and uniform such that the posterior has more flexibility. After training an autoregressive prior is fit on z so we can sample x via ancestral sampling. The prior is learned with a generative objective, which increases the quality of generated samples. The choice of prior model depends on the particular application. In the original article PixelCNN [34] was used for image, while WaveNet [35] was used for audio. More recently in TimeVQVAE [1] utilized a bidirectional transformer [29] for prior learning in the time series domain.

Loss funciton

As we want to do gradient based learning, and the quantization process is quite far off from being continuous, measures has to be taken. The output of the encoder and the input of the decoder has the same dimension, which opens the possibility of simply copying the gradients from the decoder input to the decoder output. This is referred to as the *straight through estimator*. 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 loss function consists of two parts, the reconstruction loss, codebook loss. We already mentioned that the ELBO objective reduced to the reconstruction term only. In order to learn the codebook one uses the codebook loss, which is the euclidean distance between the encoder output and the quantized output, together with commitment term, which is introduced to keep the distance between codewords from growing arbitrarily large. To summarize the total loss is given by

$$\begin{aligned} \mathcal{L}_{\text{recons}} &= ||x - \widehat{x}||_2^2 \\ \mathcal{L}_{\text{codebook}} &= ||sg(z) - z_q||_2^2 + \beta||z - sg(z_q)||_2^2 \\ \mathcal{L}_{\text{VQ}} &= \mathcal{L}_{\text{codebook}} + \mathcal{L}_{\text{recons}} \end{aligned} \tag{2.21}$$

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

The VQ loss only affects the codebook, and can hence alternatively be updated as a function of moving averages of the encoder outputs z. This is expanded on in appendix A.1 in [18].

To illustrate the need for commitment term and what it does consider a model where the data is either 0 or 1 and we initialize the codebook with $\mathcal{Z} = \{-1, 1\}$. The range of the encoder is $\mathbb{R}$ and $E(x)$ is quantized to -1 if its negative and 1 if its positive. Assume that the encoder and decoder will try to differentiate between the two classes by pushing $E(0)$ and $E(1)$ away from each other. As the reconstruction loss only affects the encoder and decoder, and the codebook loss only affect the codebook, we get that when the encoder and decoder parameter trains faster than the codebook, then the distance between the encodings and the codewords will steadily increase and the codebook loss diverge. This behavior is observed experimentally.

2.10 Evaluation metrics

2.10.1 Tokenization model

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

Reconstruction

Downstream Classification

SVM, KNN. The difference in inductive bias for the two classifiers.

2.10.2 Generative model

Good evaluation protocols for generative models are hard to come by, and the hunt for such is an active area of research. There are data modalities for which such evaluation metrics have more established standards than others. Most notably in the computer vision domain there is a benchmark dataset (ImageNet) and classifier (Inception v3).

The typical sanity check for generative models is human visual inspection. In development of evaluation metrics for generative models, how well a proposed metric correlates with quality determined by visual inspection is standard. Most people know what cats and dogs look like and can recognize features even in

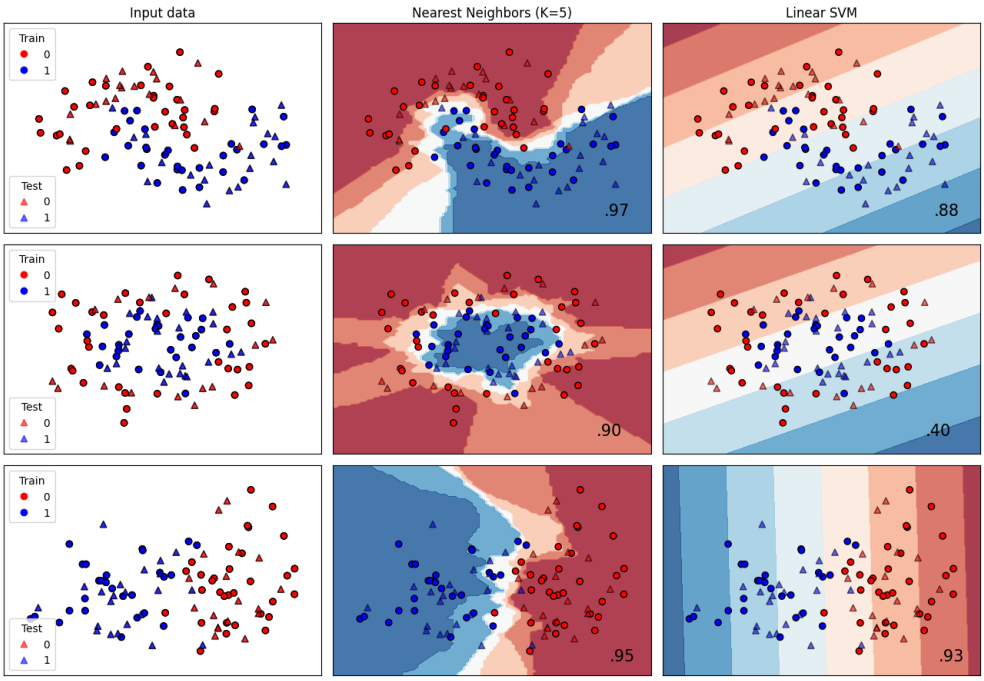


Figure 2.17: Modified example taken from `https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html`.

crudely generated images. In contrast, many time series dataset measure quantities and signals for which the untrained eye has no idea of determining the quality. 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 on time series to visually see similarities of two distributions. The major limitations of this is that the visual inspections cannot be reduced to a single score, which makes objective comparison difficult.

Following [1] we mainly consider three evaluation metrics in this thesis. *Inception Score* (IS), *Fréchet Inception Distance* (FID) and *Classification Accuracy Score* (CAS). IS measures quality of synthetic samples using the entropy of the label distribution and the evenness in prediction across labels. The *Fréchet Inception Distance* measures a distance between the generative and ground truth distribution. Finally the *Classification Accuracy Score* measures the quality of the class-conditionally generated samples by training a classifier on synthetic data and testing on real.

Common for all the above evaluation metrics is the reliance on a pre-trained classification model. In contrast to computer vision there is no widely adopted classification model for time series, but in [36] a fully convolutional network (FCN) was presented as a proposed baseline classification model for time series. As their model is not available we use the open source FCN trained on the UCR

Archive presented in [1] for our evaluations.

Inception Score (IS)

Inception Score (IS) was first introduced in [37] as an automatic evaluation method of synthetic samples.
The Inception Score works by applying a classifier to every generated sample to get the conditional label distribution $p(y|x)$. Samples containing clear characteristics should have low entropy, i.e one should be certain of the label. In addition we expect a good model to generate varied samples, so the marginal $\int p(y|x = D(z))dz$ should have high entropy. On this basis, the Inception Score is defined as

$$\mathrm{IS}(\theta) = \exp\left(\mathbb{E}(D_{\mathrm{KL}}(p_\theta(y|\mathbf{x})||p_\theta(y)))\right). \tag{2.22}$$

In contrast to computer vision there is no widely adopted classification model for Time Series. In [36] a fully convolutional network (FCN) was presented as a proposed baseline classification model, and we use the open source FCN trained on the UCR Archive presented in [1] for our evaluations.

Let $X_{\mathrm{gen}} = \{x_{i,\mathrm{gen}}\}_{i=1}^{N}$ be a set of generated samples. We use the apply a SoftMax layer to pretrained FCN representations to obtain an estimate of the conditional label distribution as follows

$$x_{i,\mathrm{gen}} \xrightarrow{\text{FCN+SoftMax}} p(y|x_{i,\mathrm{gen}}).$$

The marginal label distribution is obtained by averaging across all the synthetic data as follows

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

Issues with IS [38] (Use different network, for image classification). Important to report different metrics that indicate that the model has not overfitted. Also issues with IS [39].

Fréchet Inception Distance (FID)

As an attempt to improve on IS [40] introduced the Fréchet Inception Distance (FID). Since then FID has been the standard for assessing generative models [39]. The primary concern with IS is, as mentioned earlier, that it does not use any statistics of real world samples to compare with the statistics of the generated samples. In contrast FID relates the synthetic sample to the real world samle via

the *Fréchet distance*. For any two probability distributions, f, g over $\mathbb{R}^n$, with finite mean and variances, their distance is defined as

$$\begin{aligned} d_F(f,g) &= \left(\inf_{\gamma \in \Gamma(f,g)} \int_{\mathbb{R}^n \times \mathbb{R}^n} ||x-y||_2^2 d\gamma(x,y) \right)^{\frac{1}{2}} \\ &= \left(\inf_{\gamma \in \Gamma(f,g)} \mathbb{E}_{(x,y)\sim\gamma} ||x-y||_2^2 \right)^{\frac{1}{2}}, \end{aligned} \tag{2.23}$$

where $\Gamma(f,g)$ is the set of all *couplings* of f and g. The Fréchet distance is a special case of the Wasserstein metric and will in literature be referred to as such.

In [41] it was shown that for two Gaussian distributions the Fréchet distance is explicitly solvable as

$$d(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\mu',\Sigma'))^2 = ||\mu-\mu'||_2^2 + \mathrm{Tr}\left(\Sigma+\Sigma'-2(\Sigma\Sigma')^{\frac{1}{2}}\right) \tag{2.24}$$

The *Fréchet Inception Distance* is an application of the Fréchet distance on the representation distributions from a specified model. The models are Inception v3 in the image domain and SupervisedFCN in the time series domain.

In [40] they argue 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. The mean and covariance is estimated from the samples and the explicit formula **??** is used to calculate the FID.

FID is not without fault, and the Gaussian assumption has been shown not to hold [42]. Further, as the FID relies on estimating large covariance matrices, a large number of samples is needed to obtain a reliable estimate.

Rethinking FID: [42]

[43] FID and IS are biased.

FID score measures difference in the distribution of representations of time series. The representations capture high level semantics of the time series and analyses of these representations can provide greater insights to the realism of generated samples.

Classification Accuracy Score (CAS)

A method for evaluating the models ability to learn class conditional distributions is to train a separate classifier on the on synthetic data and test on real data

(TSTR). That is we generate N synthetic samples $\{x_i, y_i\}_{i=1}^{N}$ using class conditional sampling. We then train a classifier on this synthetic dataset, and evaluate the CAS of this classifier on a real test dataset. High CAS values indicate that the generative model produces samples similar to the real data and captures relevant class specific features.

We evaluate the CAS for TSTR by using the Supervised FCN introduced in [1] on all our models considered and compare against the baseline model to investigate the relative performance.

Chapter 3

Related Work

TODO: Introduce this section. Why are these models presented?

3.1 MaskGIT

The Masked Generative Image Transformer (MaskGIT)[29] is a generative transformer model for image synthesis developed by Google Research. The novelty of the model lies in the token generation. Unlike popular autoregressive generative transformers, who treat images as a sequence of tokens, MaskGIT introduces an image synthesis paradigm using a bi-directional transformer. This means that during training MaskGIT learns to predict tokens in all directions, an intuitively more natural way to consider images. At inference time MaskGIT starts out with a blank canvas and predicts the entire image, and iteratively keeps and conditions on the most confident pixels.

MaskGIT assumes a tokenization procedure for stage 1. In the original paper [29] VQGAN [44] was used and the actual contribution of the work revolved around improving stage 2, hence we present that part only.

3.1.1 Masked Visual Token Modeling (Prior learning)

For prior learning the codebook learned in the tokenization procedure is provided with a masking vector, which is the embedding of the special masking token, which we denote by `M`. The input embedding in the bidirectional transformer is initialized with this expanded codebook. For some 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. Such a sequence can equivalently be described as a sequence of indices $s = \{k_i\}_{i=1}^{N}$. The prior learning amounts to masking such a sequence and training the bidirectional transformer to predict the masked indices.

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

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

where $\odot$ is the Hadamard product, i.e point wise multiplication, and 1_N is a vector with the same shape as M and s.

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

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

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

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

The bidirectional transformer is used to predict the probabilities $p(s_i|s_{\text{Mask}})$ of each masked token, and $\mathcal{L}_{\text{Mask}}$ is computed as the cross entropy between the ground truth one-hot token and the predicted token probabilities.

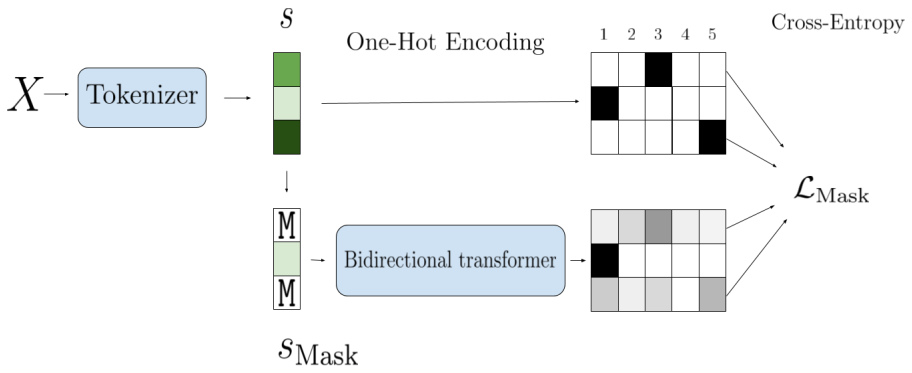


Figure 3.1: MaskGIT forward computation.

3.1.2 Iterative decoding (Image generation)

The bi-directional transformer could in principle predict all masked tokens and generate a sample in a single pass by simply sampling from the predicted probabilites $p(\widehat{s}_i|s_{\text{Mask}})$ from a forward pass of an all masked sequence. However, there

are challenges with this approach. In their original article [29] proposes a novel non-autoregressive decoding method to synthesize samples in a constant number of steps.

The decoding process goes from $t = 0$ to T. To generate a sample at inference time one starts out with an all masked sequence which we denote by $s^{(0)}_{\text{Mask}}$. At iteration t the model predicts the probabilities for all the mask tokens, $p(\widehat{s}_i | s^{(t)}_{\text{Mask}})$, in parallel. At each masked index i a token $s^{(t)}_i$ is sampled according to the predicted distribution, and the corresponding probability $c^{(t)}_i$ is used as a measure of the confidence in the sample. For the unmasked tokens a confidence of 1 is assigned to the true position. The number of $s^{(t)}_i$ with highest confidence kept for the next iteration is determined by the mask scheduling function. We mask $n = \lceil \gamma(t/T) \cdot N \rceil$ of the lower confidence tokens by calculating $M^{(t+1)}$ by

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

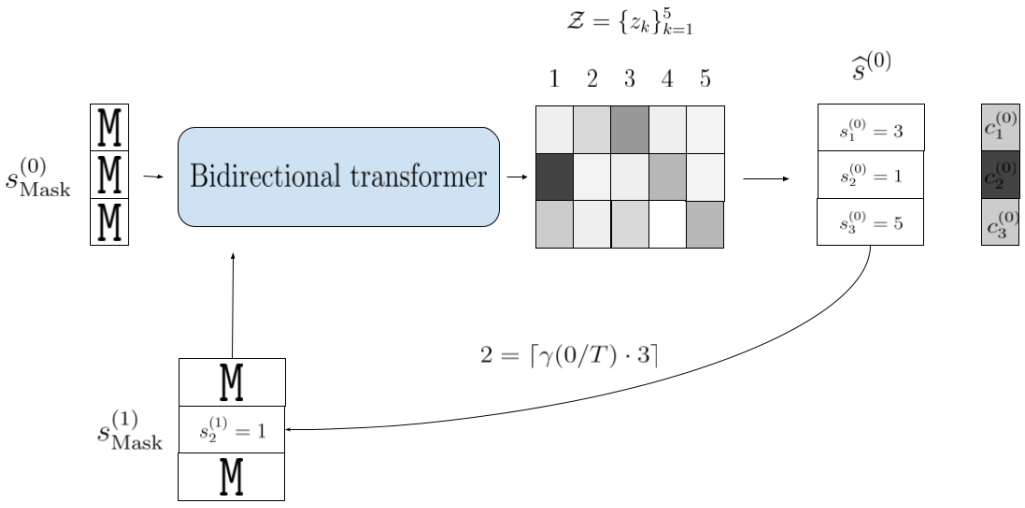


Figure 3.2: Illustration of first pass of the iterative decoding algorithm.

The algorithm synthesizes a full image in T steps. For image generation, cosine scheduling function proved best 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 TSG problem. It leverages a two stage approach similar to VQVAE and uses a bidirectional transformer akin to MaskGIT for prior learning. Additionally, the authors propose VQ modeling in time-frequency domain, separating data into high and low frequency components to better retain temporal consistencies and generate higher quality samples.

The contributions TimeVQVAE presents is, in addition to VQ-modeling in time-frequency domain, a process of sampling jointly from high and low frequency latent spaces and guided class-conditional sampling. By appending a class token, similarly to [14], the prior is learned such that the model can generate synthetic samples both conditionally and unconditionally.

Our work consists of extending a variation of the TimeVQVAE model without the high-low frequency split. This reduces the prior learning method to MaskGIT, with the addition of guided class-conditional sampling. Hence we present only the tokenization stage and refer the reader to [18] for the prior model training.

3.2.1 Tokenization

The tokenization stage is similar to VQVAE presented in section 2.9 except for the frequency split. An overview of the model is presented in figure **??**. First a time series is mapped to time-frequency domain using the Short-time Fourier Transform (STFT). Then the time-frequency representation is separated into a two branches, one zero-padding the HF region and the other zero-padding the LF region. From here the two branches follow the VQVAE architecture, with separate encoders, decoders and codebooks denoted by E_{LF}, E_{HF}, D_{HF}, D_{HF} and $\mathcal{Z}_{\text{LF}}$, $\mathcal{Z}_{\text{HF}}$ respectively. The output of the decoders are again zero-padded giving $\widehat{u}_{\text{LF}}$ and $\widehat{u}_{\text{HF}}$, before being mapped back to time domain by the Inverse Short-time Fourier Transform (ISTFT) to produce the reconstructed HF and LF components, $\widehat{x}_{\text{LF}}$ and $\widehat{x}_{\text{HF}}$, of the time series.

Loss

The codebook loss of TimeVQVAE is similar to codebook loss presented section 2.9 equation 2.21 but reflects the HF-LF split

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

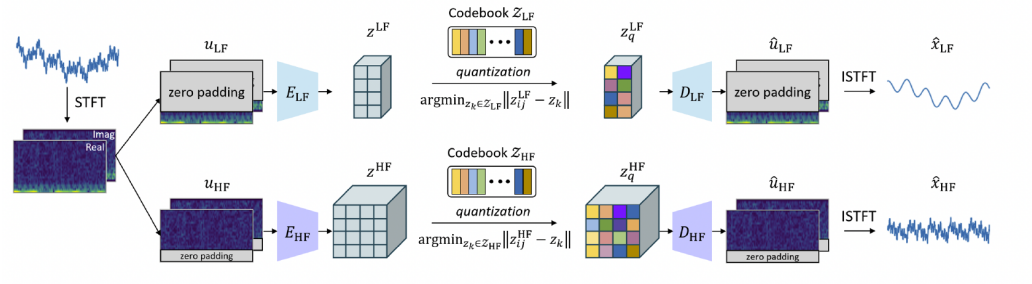


Figure 3.3: Stage 1: Tokenization. Figure taken with permission from [1]

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

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

The total loss is given by

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

In order to update the codebooks TimeVQVAE uses an exponential moving average presented in appendix A.1 of [18].

3.3 SSL

Our model leverages SSL algorithms in order to learn more expressive latent representations. Here we present the relevant algorithms for our work, Barlow Twins and VIbCReg.

3.3.1 Barlow Twins

Barlow Twins is a non-contrastive SSL method based on applying the *redundancy-reduction principle* (or efficient coding hypothesis) [45] from the neuroscientist H. Barlow to a pair of identical networks.

In essence the model wants to encourage representations of similar samples to be similar, while simultaneously reducing the amount of redundancy between the components of the vectors. This is enforced by producing two augmented views of each sample and projecting the their representations onto a vast feature space, in such a way that their cross-correlation is close to the identity.

The Barlow Twins algorithm starts out by creating two different augmented views for each datapoint in a batch D. The augmentations are selected by sampling

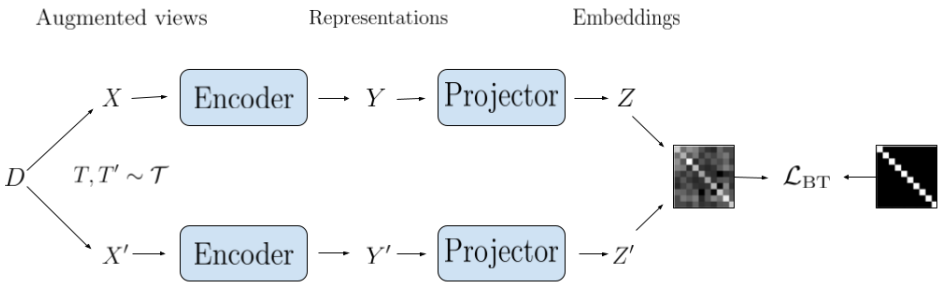


Figure 3.4: Overview of the Barlow Twins architecture. Figure inspired by [21]

from a collection of augmentations $\mathcal{T}$. We denote the batches of augmented views $T(D) = X$ and $T'(D) = X'$, for augmentations $T, T' \sim \mathcal{T}$. The batches are then passed through an encoder (give representations Y and Y') and a *projector* to produce batches of 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', and measuring its deviance from the identity. In particular the Barlow Twins loss is defined as

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

where

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

The *invariance term* assists in making the embedding invariant to the distortions introduced by the augmentations, hence pushes the representations closer together. The *redundancy reduction term* decorrelates the different vector components, which reduces the information redundancy.

3.3.2 VIbCReg

VIbCReg [**lee2024vibcreg**] is a non-contrastive SSL model with siamese architecture based on VICReg [20]. It can be seen as VICReg with better covariance regularization and IterNorm [46]. Overall the architecture is similar to Barlow Twins, but a key difference is that variance/covariance regularization is done in each branch individually.

define or elaborate on this

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

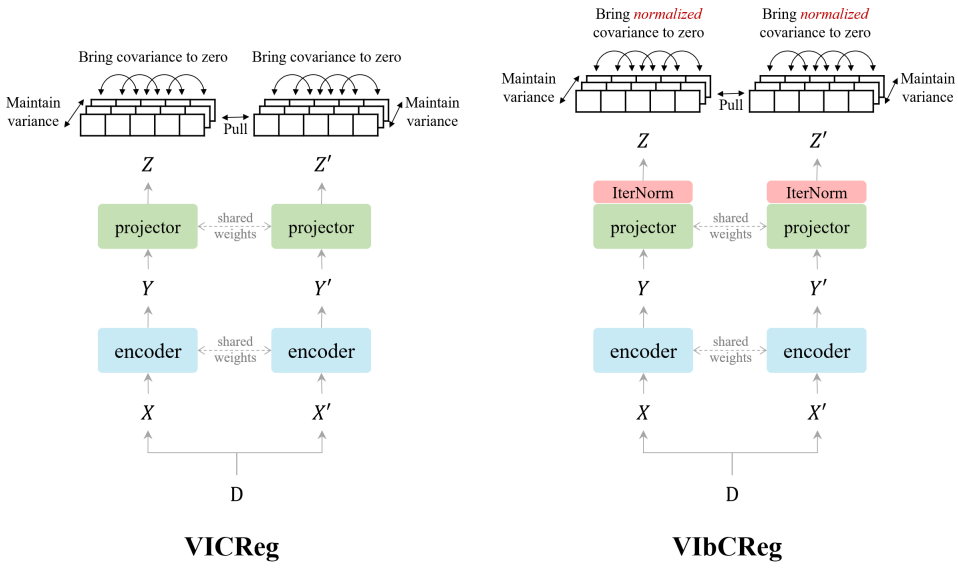


Figure 3.5: Overview of VIbCReg, and comparison with VICReg. Taken with permission from [25]

The loss consists of a similarity loss between the branches, and feature decoration (FD) loss together with a feature component expressiveness (FcE) term at each branch. Input data is processed in batches. Let $Z \in \mathbb{R}^{B\times F}$ 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 MSE of the two embeddings

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

which encourages them to be similar. The FcE term acts on each branch separately and encourages the variation across a batch to stay at a specified level γ. It is defined as

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

where Var() is a variance estimator, γ is a target value for the standard deviation, which both in VIbCReg and VICReg is set to 1. ϵ is a small scalar preventing numerical instabilities.
For the FD loss we first mean shift and normalize along the batch dimension

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

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

compute the normalized covariance matrix

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

and take the mean square across all off-diagonal elements to obtain the FD loss

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

The total loss is then given by

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

where λ, μ and ν are hyperparameters determining 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 can be seen as a tangent of 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, which reduces the model to what they refer to as naive TimeVQVAE in their paper. We expand on naive TimeVQVAE with a self-supervised extension.

A schematic figure of our proposed tokenization model is given in **??**. To improve on the reconstruction we add a regularizing term by reconstructing augmented views. We hypothesize that the model generalizes better to unseen data by letting the decoder "see" the augmented views.
To separate classes better we introduce a non contrastive self supervised loss. The intuition being that the representation of original and augmented views are pushed closer together by the SSL loss.

4.1 Proposed model: NC-VQVAE

Our model, termed NC-VQVAE, is a generative time series model which learns expressive discrete latent representations by combining VQVAE [18] with non contrastive self supervised learning algorithms. NC-VQVAE uses the two staged modelling approach presented in [1], and can be considered an extension of their "naive TimeVQVAE". Our model mainly extends the tokenization stage, where we incorporate Barlow Twins [21] and VIbCReg [25] as our non contrastive SSL, but the framework is flexible. For stage two we model the prior using a bidirectional transformer as MaskGIT [29].

4.1.1 Stage 1: Tokenization

The architecture of the tokenization model **??** consists of two branches. The top and bottom branch is referred to as the original and augmented branch respectively. The model takes a time series x as input and creates an augmented view x'.

The original branch is simply the naive TimeVQVAE from [1], while the augmented branch is an autoencoder, constructed by omitting the quantization layer. The views are passed through their respective branches, where 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 concatenate the global average and max pool of both representations individually and passing the resulting vectors through the projector.

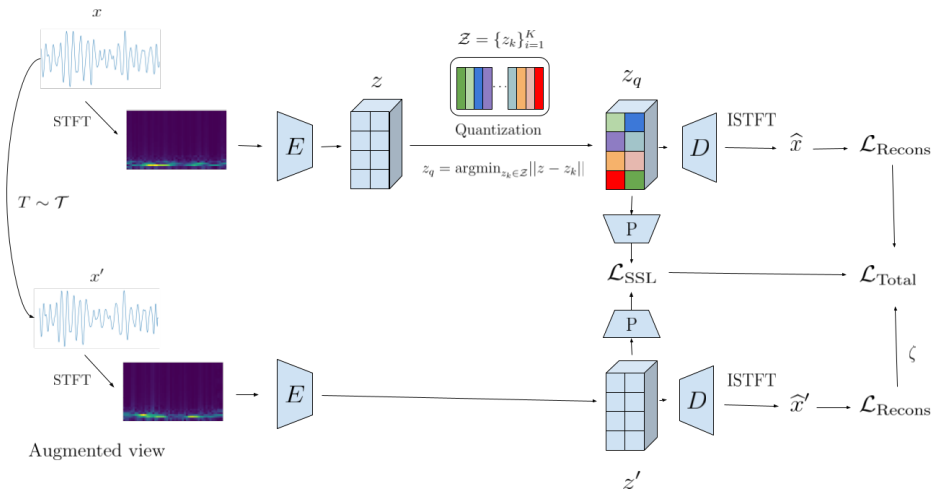


Figure 4.1: Overview of proposed model. NC-VQVAE. Fix index on Z, add weight to arrow from ssl loss to total. fix argmin index thing.

Loss

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

To refresh the readers memory the VQ loss consists of a reconstruction loss, codebook loss which is the MSE of continuous and discrete latent representations together with a commitment loss which keeps the codewords from diverging. In our setup the codebook loss reduces to

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

and the reconstruction loss to

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

Our VQ loss is too given by

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

The SSL loss takes form depending on which SSL method used. The loss is calculated on derived values from z_q and z'. In our work 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 The discrete latent representations from the original branch and the continuous latent representations from the augmented branch are pushed to similar regions of latent space.

The augmented reconstruction loss is simply given as

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

and provides the encoder and decoder with instructions to reconstruct the augmented view, which in conjunction with the SSL loss, influence the codebook to encode information regarding the augmentations. Additionally we hypothesize that it assists the encoder in not ignoring reconstruction on behalf of the SSL loss. During initial testing, omitting the augmentation reconstruction lead to severe overfitting.

The total loss is given by

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

where η and ζ are hyperparameters influencing the importance of the terms for 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 with learned structure from both reconstruction and SSL loss. Instead of introducing an additional masking vector in the embedding matrix, we rather use the codebook directly, and create a separate learnable masking vector, and mask the embedded sequences using this. The only difference between our proposal and the mechanism used in MaskGIT and TimeVQVAE is that the learning of the masked token embedding is independent of the other embeddings, i.e it is not influenced and does not influence by the learning of the finetuned codebook. The rationality behind this is to further leverage and not unnecessarily influence the learned codewords from stage 1. Except for this adjustment and the possibility of class conditional sampling from TimeVQVAE, our method is equivalent to MaskGITs.

In order to separate this masking vector, we do the embedding stage before masking, effectively factoring the embedding out of the transformer. The generation procedure is identical to the iterative decoding from MaskGIT.

Chapter 5

Experiments

5.1 Implementation details

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

Time Frequency Modelling

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

Encoder and decoder

The same encoder and decoder architecture as in [18] is used and further use the implementation from [47] with adaptations from [1].

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

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

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

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

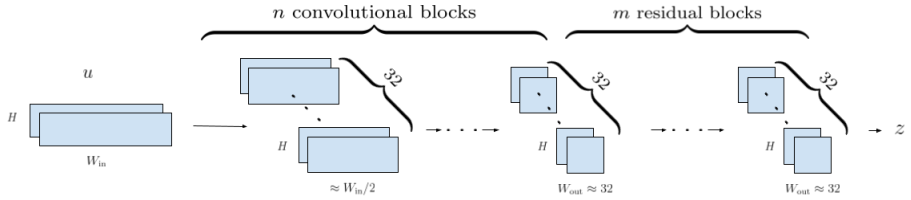


Figure 5.1: Overview of the encoder architecture. The decoder architecture is simply obtained by reversing the arrows and switching out the convolutional block for transposed convolutional blocks.

VQ

Implementation from lucidrains/vector-quantize-pytorch. https://github.com/lucidrains/vector-quantize-pytorch
Codebook size $= 32$ and dimension $= 64$.
Use exponential moving average with decay 0.9, and commitment loss with weight $\beta = 1$.

Augmentations

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

Projector

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

Prior learning

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

Optimizer

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

Evaluation

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

5.2 Initial Experimentation and Model Development

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

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

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

5.3 Main Experiments

We are primarily interested in two things. For stage 1, if NC-VQVAE learns more expressive representations, i.e are we able to reconstruct on par with VQVAE while simultaneously improve on downstream classification. For stage 2 we are interested in the effect NC-VQVAE has on prior learning and time series generation.

We evaluate our model NC-VQVAE with both Barlow Twins and VIbCReg as SSL method against the naive VQVAE as described in [1]. For each SSL method, we train three separate models which uses different sets of augmentations. Firstly we look at the tokenization models, evaluating the reconstruction capability and performance on downstream classification. Then we train a prior model on top of the different tokenization models and evaluate the performance of the generative models by IS, FID, CAS and visual inspection.

5.4 Stage 1

5.4.1 Augmentations

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

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

Amplitude Resizing + Window Warp scales in both x and y direction. The window warp has similar qualities to phase shift, but not uniformly and keeps endpoints fixed. They were considered as the observed conditional distribution in some datasets, such as ShapesALL **??**, 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 **??**.

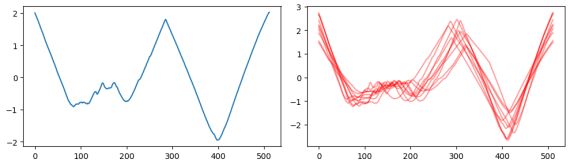


Figure 5.2: ShapesAll: Original (left), augmented (right). 15 instances of Amplitude Resizing + Window Warp applied to the original sample.

Slice and Shuffle crops the time series into four sections and permutes them. For datasets with sharp modularity and few peaks, such as ElectricDevices **??**, the augmentation provides a view with peaks occurring at timestamps not seen in the training data, which is illustrated in figure **??**. 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 **??**, the semantics of the dataset is preserved under this augmentation, despite their continuous nature.

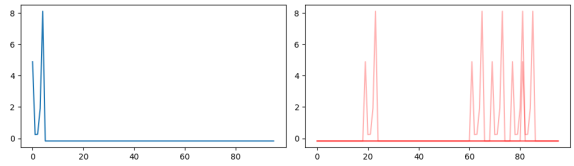


Figure 5.3: ElectricDevices: Original (left), augmented (right). 5 instances of Slice and Shuffle applied to the original sample.

Gaussian noise adds a nose $\epsilon \sim N(0, 0.05)$ to each datapoint in the time series. This introduces, in many cases, a substantial high frequency component as seen in figure **??**. 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.

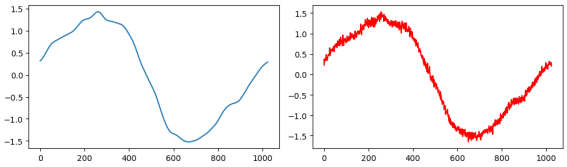


Figure 5.4: StarLightCurves: Original (left), augmented (right). One instance of Gaussian noise applied to the original sample.

5.4.2 Evaluation

Reconstruction, Classification and Visual inspection

5.5 Stage 2

5.5.1 Evaluation

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

5.6 UCR Time Series Classification Archive

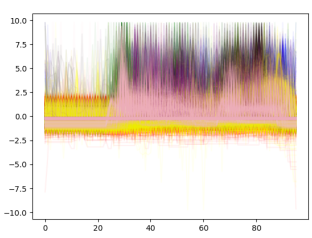
The evaluation of our model NC-VQVAE is done on a subset of the UCR Time Series Archive [48]. The UCR archive is a collection of 128 datasets of univariate time series for time series classification. The different datasets in the archive span a wide range characteristics and include among others sensor,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

Our subset of the UCR archive is

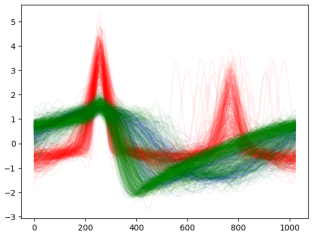
Type	Name	Train	Test	Class	Length
Device	ElectricDevices	8926	7711	7	96
Sensor	FordB	3636	810	2	500
Sensor	FordA	3601	1320	2	500
Sensor	Wafer	1000	6164	2	152
Simulated	TwoPatterns	1000	4000	4	128
Sensor	StarLightCurves	1000	8236	3	1024
Motion	UWaveGestureLibraryAll	896	3582	8	945
ECG	ECG5000	500	4500	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.

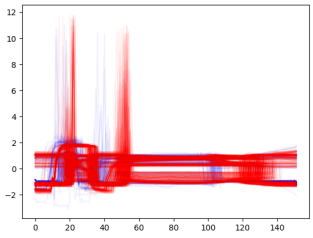
We choose to test on a subset, rather than on the entire UCR Archive, due to computational limitations as well as to more thoroughly investigate the effect of our models and the role of augmentations. The subset is chosen such that they span a wide range of train set sizes, lengths, classes and type, while the class distributions have visually different characteristics which can be seen from table 5.1 and figure **??**.



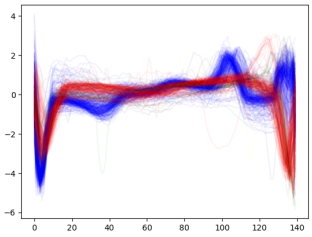
ElectricDevices



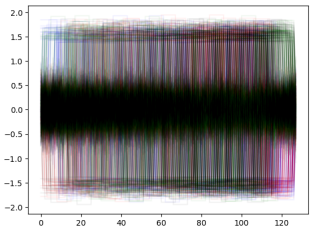
StarLightCurves



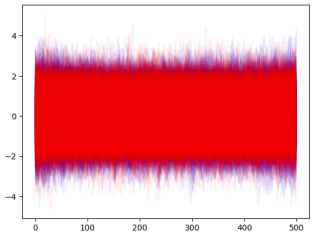
Wafer



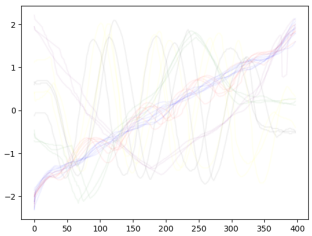
ECG5000



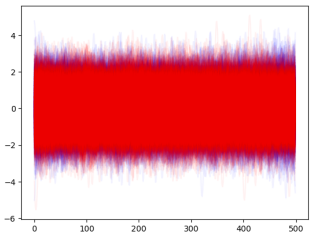
TwoPatterns



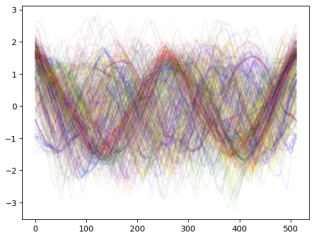
FordA



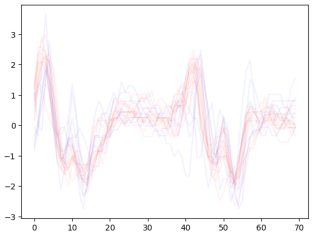
Symbols



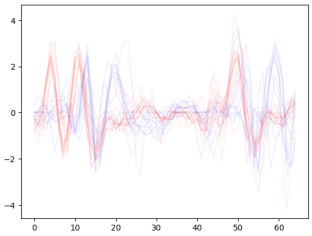
FordB



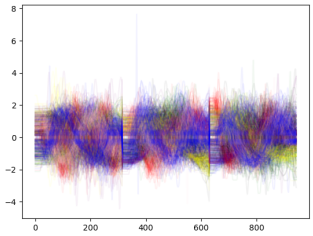
ShapesAll



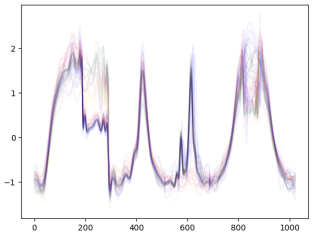
SonyAIBORobot-Surface1



SonyAIBORobot-Surface2



UWaveGestureLib-raryAll



Mallat

Chapter 6

Results and Discussion

6.1 Stage 1

6.1.1 Reconstruction

6.1.2 Classification

6.2 Stage 2

6.2.1 Thoughts

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

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

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

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

For datasets of smaller size with classes of different characteristics (a clear distributional difference in visual inspection [Sony2 and Symbols]) NC-VQVAE

seems to perform better both in terms of FID and IS.

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

6.2.2 Ablation

6.2.3 Augmentation Reconstruction Weight

Here are the results of the ablation on the effect of "Augmentation Reconstruction Weight" on Stage 1. "Augmentation Reconstruction Weight" is the weight given to the reconstruction loss on the augmented branch. Tested weights 0.05, 0.1, 0.15 and 0.2. Augmentations [Window Warp, Amplitude Resize] and [Slice and Shuffle]. The weight has little effect on linear probe accuracy across the four datasets tested, and the two sets of augmentations. The effect on Validation reconstruction loss is small for all except FordA. It seems, not very surprisingly, that the choice of augmentations are of (much) greater importance.

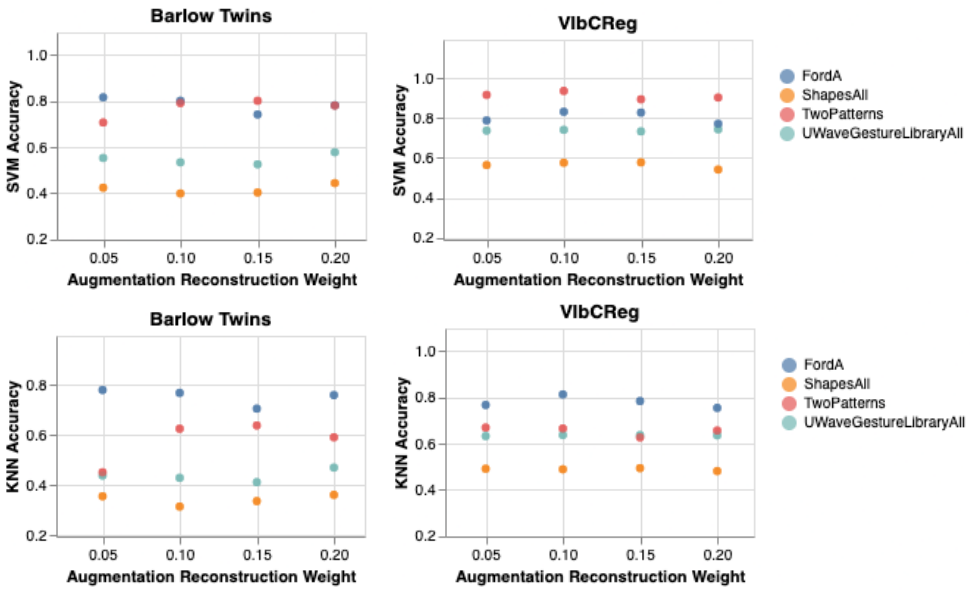


Figure 6.1: Augmentations: Window Warp and amplitude resize. Averaged across 2 runs. Trained for 250 epochs

6.2.4 Augmentation robustness

S1 - S2 - Augs: Choose datasets such that half were thought to be well fitting for slice and shuffle and half for amplitude resize + window Warp. This was after seeing FordA performing well with S and S, and looking at the augmented views.

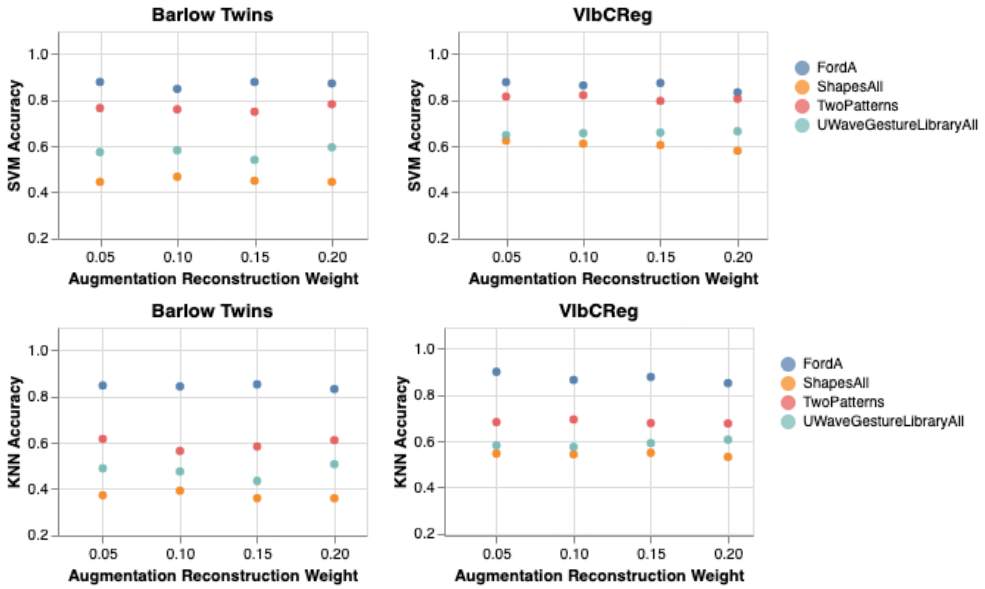


Figure 6.2: Augmentation: Slice and shuffle. Averaged across 2 runs. Trained for 250 epochs

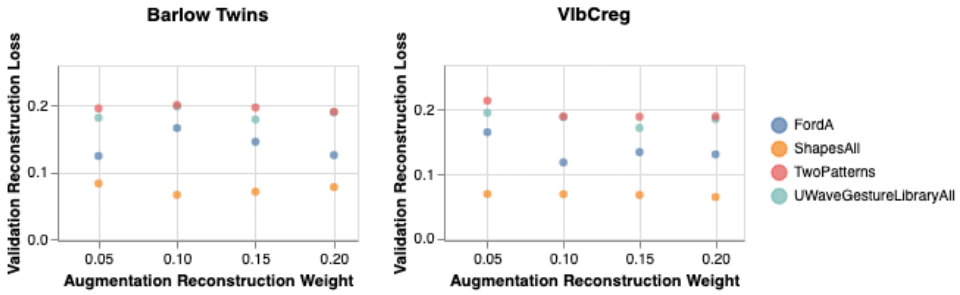


Figure 6.3: Augmentation: Window Warp and amplitude resize. Averaged across 2 runs. Trained for 250 epochs

FordA/B, Electric devices, ShapesALL for S and S
TwoP, UWave, symbols, Mallat for ampRes + winwarp

Visual inspection: Plot training samples + spectrogram, compare to augmented view. Compare these against others from other classes.
Does the resulting improvement in stage 1 transfer to stage 2?

TODO: Download the Wandb data.

Plot for each dataset and each augmentation: Color code according to SSL-model.

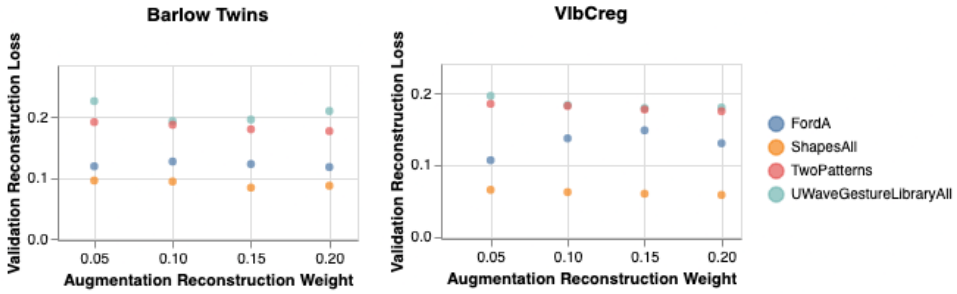


Figure 6.4: Augmentation: Slice and shuffle. Averaged across 2 runs. Trained for 250 epochs

6.3 Discussion

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

6.4 Further work

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

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

Chapter 7

Conclusion

7.1 Reconstruction

7.2 Classification

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[17] 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.

[18] 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.

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

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

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

[22] 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.

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

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

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

[26] 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].

[27] 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].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Appendix A

Additional Material

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

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

NTNU

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

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

_____

_____ (oppdragsgiver), og

_____

_____

_____ (student(er))

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): _____

Oppdragsgivers kontaktperson (navn): _____

Student(er) (signatur): _____ dato _____

_____ dato _____

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur):_____ dato _____

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

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

Instituttleder/faggruppeleder (signatur): _____ dato _____