Abstract

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

Sammendrag

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

Preface

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

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

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

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

Max Lunde Hauge
Trondheim, June 2023

'A famous person once said...'

— Max in writing this document

Contents

Chapter 1

Introduction

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

Chapter 2

Theoretical background

En ting å tenke på gjennom hele denne seksjonen er hvorfor du introduserer et konsept/en modell og hvem man skriver for.

Her snakker vi om hvorfor vi introduserer tingene vi gjør. Litt om linja vi ønsker å legge oss på, den logiske oppbygningen i seksjonen etc.

2.1 Neural Networks

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

Ikke helt fornøyd. Legge ved en kilde på hvor de ble introduser første gang. Hvem som definerte og når?

2.1.1 Structure of Neural Networks

Her må det komme en definisjon. Matriseformen er fin. Finn en god kilde på det.

Typically, NNs are structured in layers, including:

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

The three scientific works that sparked the modern era of Neural Networks (NNs) was the proposal of the single layer Neural Network by McCulloch and Pitts in 1943 [MP43], the first perceptron by Rosenblatt in 1958 [F58] and the backpropagation algorithm by Werbos in 1974 [Wer90].

Mathematically a Neural Network (NNs) is a function f that maps an input vector x to an output vector y through a series of transformations. A NN consist of a set of interconnected nodes, or neurons, that are organized in layers. Each neuron is connected to the neurons in the previous and next layer, forming a directed graph. To illustrate lets define a 2 layered Multilayer Perceptron (MLP) with N inputs, K hidden units, weights W_1 and W_2 and a single output y.

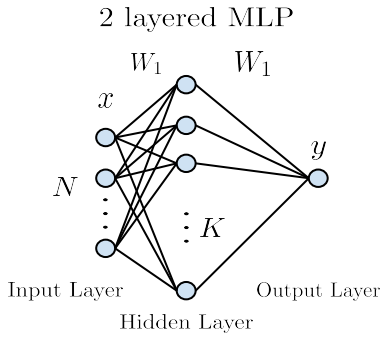


Figure 2.1: Illustration of a 2 layered MLP with N inputs, K hidden units, weights W_1 and W_2 and a single output y.

The output $y \in \mathbb{R}$ can be expressed as a function of the input $x \in \mathbb{R}^N$ and the parameters $\theta = \{W_1 \in \mathbb{R}^{K \times N}, W_2 \in \mathbb{R}^{1 \times K}, \beta \in \mathbb{R}^K\}$ as follows:

$$y = f_\theta(x) = W_2(\sigma(W_1 x + \beta)) \tag{2.1}$$

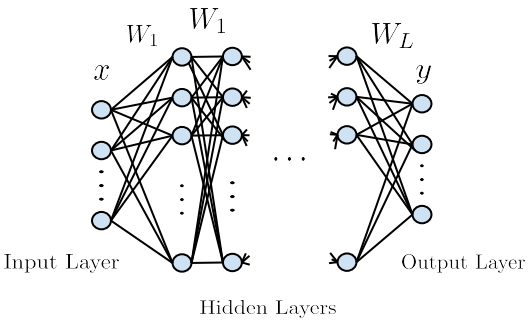
The calculation of y using the networks parameters is referd to as a feedforward operation. The activation function, denoted as σ, determines the linearity of the output y. A concatenation of linear combinations is also a linear combination, choosing a linear σ would result in a linear f_θ. This is not ideal, as most real world data is non-linear, meaning a straight line is not sufficient to represent the data. To allow the network to learn non-linear functions, non-linear activation functions are used.

Common non-linear activation functions include the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ and the

Rectified Linear Unit (ReLU) $\sigma(x) = \max(0, x)$.

Modern NNs build on the MLPs by adding more layers and dimensions, which is known as deep learning. The intuition being that more layers allows the network to learn more complex functions f and find deeper connections between features. A L-layered neural network using activation functions $\sigma^1, \sigma^2, ..., \sigma^L$, weights $W_1, W_2, ..., W_L$ and biases $\beta_1, \beta_2, ..., \beta_L$ can be expressed as follows:

$$y = f_\theta(x) = \sigma^L(W_L(\sigma^{L-1}(W_{L-1}(...\sigma^1(W_1x + \beta_1)...) + \beta_{L-1}) + \beta_L) \tag{2.2}$$



2.1.2 Learning in Neural Networks

The learning process of a NN is done by adjusting the parameters θ to minimize a loss function $\mathcal{L}$. The loss function is a measure of how well the network performs on a given task. For regression tasks the loss function is often the mean squared error (MSE)

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.3}$$

Where y_i is the true value and $\hat{y}_i \in \hat{y}$ is the predicted value, $f_\theta(x) = \hat{y}$. Learning is a optimization problem, where the goal is to find the optimal parameters θ^* that minimizes the loss function $\mathcal{L}$.

$$\theta^* = \arg\min_\theta \mathcal{L} \tag{2.4}$$

The optimization is done by using the gradient of the loss function $\nabla_\theta\mathcal{L}$ to update the parameters θ in the direction of the steepest descent. This is known as gradient descent (GD).

$$\theta_{t+1} = \theta_t - \alpha\nabla_\theta\mathcal{L} \tag{2.5}$$

Where α is the learning rate, which determines the size of the step taken in the direction of the steepest descent. Modern NNs use the Stochastic Gradient Descent (SGD) algorithm first

introduced by Herbert et al [RM51] in 1951. SGD is a variant of the GD that uses a random subset of the training data to calculate the gradient $\nabla_\theta \mathcal{L}$. This has the benefit of reducing the computational cost of calculating the gradient, as the full training set can be very large.

Gradient decent variations in addition to the technique backpropagating is typically used to train NNs. Backpropagation is a method for calculating the gradient of the loss function $\nabla_\theta \mathcal{L}$ with respect to the parameters θ. Popularized in the 1980s by Rumelhart et al [RHW86b]. It involves a forward pass to compute the loss, $\mathcal{L}$, and a backwards pass where the chain method is applied layer by layer, output to input, to calculate the contribution of each parameter to the total loss gradient.

These calculations require alot of computational power, which is why we have seen a surge in the use of GPUs for training NNs. GPUs are well suited for the task as they are designed to perform parallel calculations on large amounts of data. GPU technology has been a key factor in the recent success of deep learning. Allowing larger models to be trained on larger datasets.

2.2 Supervised and unsupervised learning

Using a machine learning model there are two key learning paradigms. Supervised and unsupervised learning. Supervised learning refers to a learning process where the network has access to pre-labelled inputs which acts as targets. For each training example there will be a set of input values alongside a designated output value. During training the model learns to mimic the labelled data by minimizing a loss function. For example the $\mathcal{L}_{\text{MSE}}$.

A simple example of a supervised learning task is linear regression. Here the model finds the best fit line for a given set of data points. The prediction $\hat{y}$ is determined using the equation for a line $y = ax + b$ and adjusting a and b to minimize $\mathcal{L}_{\text{MSE}}$. The $\mathcal{L}_{\text{MSE}}$ requires a set of labelled data points, where the true value y is known. This is the main idea behind supervised learning, and is applied to more complex models like NNs.

For a unsupervised learning the model does not have access to pre-labelled inputs. Instead the model learns to extract patterns and features from the data without any external guidance. A example of unsupervised learning is clustering. Here the model learns to group similar data points together based on some similarity metric. Other examples include dimensionality reduction, generative models and Self Supervised Learning (SSL).

2.3 Convolutional Neural Networks, CNN

The modern Convolutinal Neural Network (CNN) gained popularity in the 1990s, when LeChun et al[LeC+98] used it to classify handwritten digits. This was based on the work done by Fukushima in the 80s[Fuk80] on the Neocognitron, a hierarchical, multilayered artificial neural

network. Designed to mimic the perceptive field of the the brain and the visual cortex. The Neocognitron used a convolutional operator to extract features from the input, which was then used to classify the input. The use of a convolutional operator in Neocognitron was inspired by the work done by Hubel and Wiesel in the 60s[HW] on the visual cortex of cats and monkeys.

These works layed the foundation for modern CNNs, which are now used for a wide range of tasks including image classification, object detection, image segmentation and video analysis. The CNNs ability to extract features, make it well suited for data with a grid like topology. Examples include Images and timeseries.

2.3.1 Convolutional layer

The convolutional operator is a fundamental mathematical tool that merges two functions, x and w, to produce a third function s. The convolutional operator is defined as follows:

$$s(t) = (x * w)(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da \tag{2.6}$$

Here, $s(t)$ is the resulting function after convolution. The function x is often referred to as the input, while w is the filtering function also known as the kernel. The integral runs over the entire range of a, combining the two values $x(a)$ and $w(t-a)$ at each point t to produce $s(t)$. As the convolutional operator is commutative, the order of the input and kernel can be swapped without changing the result.

Assuming t is a integer, and using the commutative property we can express the following discrete convolution:

$$S(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a-t)w(a) \tag{2.7}$$

Here the kernel is not dependent on the position t, which is beneficial in a machine learning setting because the kernel can be reused for different positions t. By defining I as the input and K as the kernel, we can express the discrete convolution in two dimensions:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n) \tag{2.8}$$

Here the sums calculate the dot product between the kernel, with width m and height n, and the input at position i, j. This is visualized in fig 2.2. By calculating $S(i,j)$ for all i, j we calculate the activation map S corresponding to the kernel K and the input I.

$$S = \sum_{i}\sum_{j}(I * K)(i,j) \tag{2.9}$$

The activation map S is a representation of the input I that highlights the features detected by the kernel K. By letting K have learnable parameters, the network can learn to detect features in the input through the generation of activation maps.

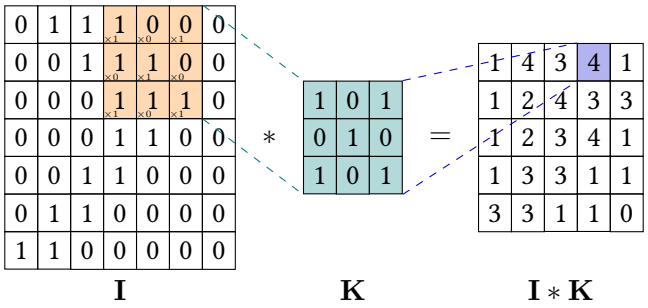


Figure 2.2: Example of convolutional operation. $\mathbf{I}$ is the input, $\mathbf{K}$ is the kernel and $\mathbf{I} * \mathbf{K}$ is the activation map. Illustration taken from the Random TikZ collection[Rie22]

The configuration of a convolutional layer involves not only the selection of the kernel size (m, n) but also the specification of the stride length and the padding. The stride defines the number of units the filter shifts over the input data for each convolution operation. A stride of 1 implies the i, j gets incremented by 1 in eq 2.9. This makes for a more fine-grained S compared to a stride of 2, which would increment i, j by 2 for each calculation. The use of a higher stride reduces the dimensionality of the output S. Which is beneficial when the task is to compress the input into a smaller activation map.

2.3.2 Pooling and Downsampling

A pooling layer, also refered to as a detector stage, is a component of CNNs that reduces the dimensionality of the input data. It works as a downsampler, reducing the spatial size of the input data by applying a filter. The filter is applied to a region of the input, and the output is a summary of the region.

The most common pooling operations are:

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

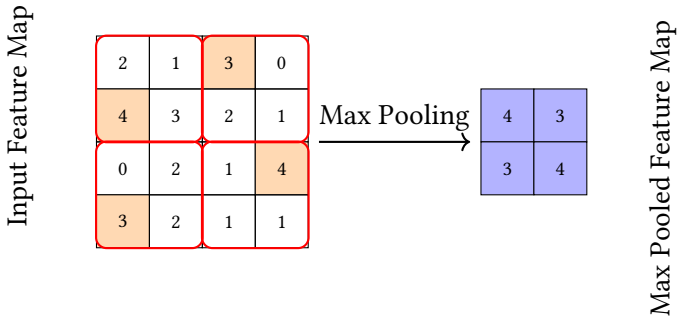


Figure 2.3: Illustration of a max pooling operation. The input feature map is reduced in size by applying a max pooling filter with size 2x2 (red boxes), which selects the maximum value in each filter region to produce the max pooled feature map.

Using a pooling layer in the context of CNNs help summarize the activation map S and reduce the dimensionality. This helps to mitigate overfitting aswell as reducing the computational cost of the network.

2.3.3 Architecture of CNNs

A typical CNN architecture consists of the following layers:

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

2.4 Residual block

The residual block, pioneered by He et al. in their paper on deep residual learning [He+15], represents a advancement in neural network arcitecture, specifically addressing the challenges of training deeper networks.

The problem at hand, as described by He et al, is that deeper networks are more difficult to train. This challenge is primarily attributed to the vanishing gradient problem. In deeper networks, the gradient of the loss function tends to diminish progressively, complicating the training process. This emerges from the key mechanism in the backpropagation algorithm, the chain rule. The chain rule, a mathmatical principle, decomposes the gradient computation into a series of smaller, more manageable calculations. In deeper networks this recursive procedure can lead to increasingly smaller gradient values due to accumulating errors in the large amount of smaller calculations.

The residual block, visualized in fig 2.4, introduces a arhitecture feature known as the "shortcut connection" that helps mitigate this issue. Mathematically, if we consider the standard layer operation as a function $F(x)$, where x is the input to the layer, the output of a typical layer would be just $F(x)$. In the residual block, we define the output to be $H(x) = F(x) + x$. This addition of x is the shortcut connection. Its a form of identity mapping ensuring that the deeper layers can, at the very least, achieve the performance of shallower networks by simply learning the identity function for the additional layers.

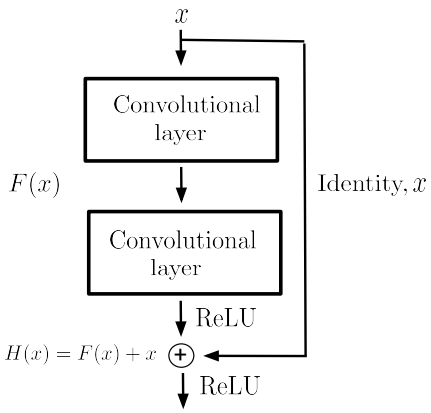


Figure 2.4: Illustration of the Resnet block from He et al's original paper[He+15]

2.5 Self Supervised Learning, SSL

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

2.5.1 Contrastive and Non Contrastive learning

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

Non-Contrastive learning methods in SSL, by contrast, does not depend on negative pairs. Instead, these approaches aim to learn representations by encouraging similarity between different augmented views of the same data point. This approach is based on the principle that different transformations of the same data should yield similar representations, thereby ensuring consistency and robustness in the learned features. Augmented versions of the same cat, should yield the same cat embedding. Notable non-contrastive methods include BYOL (Bootstrap Your Own Latent)[Gri+20] and Barlow Twins [Zbo+21].

2.5.2 The Role of Siamese Networks in SSL

Siamese networks[LST15] is a neural network architecture designed for specialized tasks that require the comparison of input pairs. The defining characteristics of these networks is the dual-branch structure, where two identical sub-networks with shared parameters process two seperate inputs. The outputs of the sub networks, often refered to as twin embeddings, are then brought togheter to evaluate the degree of similarity or difference. The similarity or difference is then used to calculate a contrastive or non contrastive loss function. The siamese architecture is illustrated in fig 2.5.

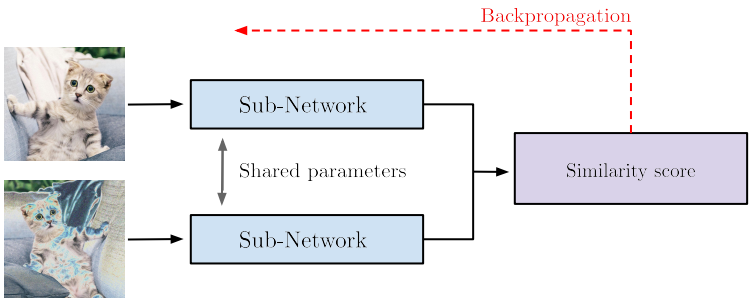


Figure 2.5: Example of a non contrastive siamese network using augmented views. Here the images are processed to calculate a similarity score, the backpropagation arrow indicates that the similarity score is then used to update the weights of the sub network. Pictures taken from `www.pexels.com`

Many modern SSL approaches base their architecture on the siamese network as it is efficient for teaching a sub network to discriminate between different classes or features in a dataset.

2.5.3 Barlow Twins

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

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

Objective function of Barlow Twins

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

$$\mathcal{L}_{BT}(\mathbf{z_1}, \mathbf{z_2}) = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2 \tag{2.10}$$

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

Siamese architecture

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

The procedure is as follows. First a batch normalization is applied to the distorted views (x_A, x_B), allowing the data to be normalized before being fed into the encoder, E. Typically E is implemented using convolutional layers and residual blocks that create non linear activation

maps with reduced dimensions, capturing the essense of the data. The latent variables are then fed into the projector, P. The projector, P, acts as a dimensionality expander with learnable parameters. Typically a deep fully connected neural network. It learns to expand the encodings into a space where the barlow twins loss can be efficiently applied. The intuition behind expanding the dimensions is that more features in the cross-correlation matrix enable a more efficient and robust comparison while also allowing the model to capture a broader range of characteristics within the encodings. Next the empirical cross correlation is compared with the Identity matrix as visualized in fig 2.6

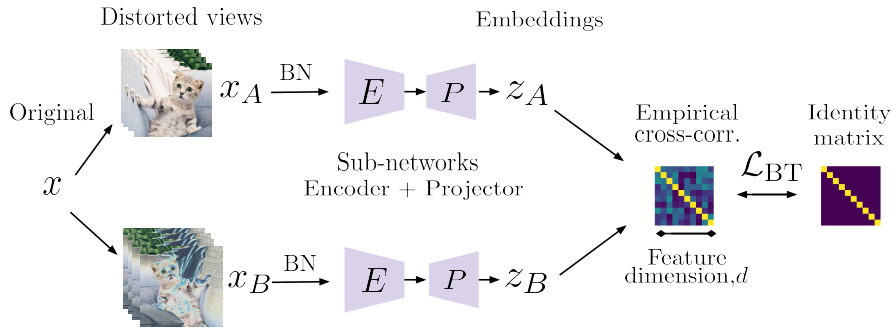


Figure 2.6: Illustration of the Barlow Twins procedure, inspired by the original paper[Zbo+21].

The optimalization problem is then to minimize the objective function $\mathcal{L}_{BT}$ with respect to the parameters of the encoder and projector. The result is parameters that produce encodings that are invariant to distortions while also being informative. Typical appliances includes using the encoder as a feature extractor for tasks like classification.

2.6 The Vector Quantized Variational Autoencoder, VQ-VAE

The Variational Autoencoder first introduced by Kingma and Welling in 2013[KW19] is a probabilistic model that learns the underlying distribution of the input data, which can then be used to generate new data points. It is a type of autoencoder, which is a neural network designed to learn representations of the input in a unsupervised manner. First introduces in the 1980s by Rumelhart et al[RHW86a]. The Vector Quantized Variational Autoencoder (VQVAE), introduced by van den Oord et al[OVK17] in 2017, is a variant of the VAE that uses discrete latent variables instead of continuous ones. This discretization of the latent space has shown to be a evolutionairy step from the VAE, improving the representations and quality of the generated data.

2.6.1 Evolution from Variational Auto-Encoder, VAE

As a fundemental principle of latent variable models, the Autoencoder explicates the relationship between the observed data and latent variables (unobserved) through the marginal distribution. This relationship is central to understanding how a autoencoder encodes and decodes data, capturing the essence of the data's underlying structure.

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

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

$$p_\theta(x) = \int_z p_\theta(x, z)dz = \int_z p_\theta(z)p_\theta(x|z)dz \tag{2.11}$$

Variational inference and optimization

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

Challenges in Maximum Likelihood Estimation

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

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log p(x_i|\theta) \tag{2.12}$$

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

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log \int_{z_i} p_\theta(x|z)p_\theta(z) \tag{2.13}$$

Variational Approximation

To circumvent the intractability of the integral, we approximate the posterior $p_\theta(z|x)$ with an encoder model $q_\phi(z|x)$, parameterized by a neural network with variational parameters ϕ, that maps the input data to the latent space. Hence giving the autoencoder the name Variational Autoencoder (VAE). The decoder neural network aims to reconstruct the input data $\mathbf{x}$, thereby approximating the conditional likelihood $p_\theta(\mathbf{x}|\mathbf{z})$.

$$\mathbf{x} \to \underset{\sim q_\phi(\mathbf{z}|\mathbf{x})}{\text{EncoderNeuralNet}_\phi(\mathbf{x})} \to \mathbf{z} \to \underset{\sim p_\theta(\mathbf{x}|\mathbf{z})}{\text{DecoderNeuralNet}_\theta(\mathbf{z})} \to \hat{\mathbf{x}} \tag{2.14}$$

Evidence Lower Bound

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

$$\mathcal{L}_{\theta,\phi}(x) = \log p_\theta(x) - \mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x)) \tag{2.15}$$

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

$$\mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x)) \tag{2.16}$$

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

By maximizing the ELBO with respect to θ and ϕ we achieve the following:

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

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

2.6.2 Transition to discrete latent variables

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

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

Vector quantization

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

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

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

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

Effect on KL divergence

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

input, the posterior assigns a probability of one to a single latent index and zero to all others.

$$q(z = z_q|x) = \begin{cases} 1 & \text{if } q = \arg\min_{z_q \in Z} ||z_{ij} - z_q|| \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

(Stemmer formuleringen her??)

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

$$\begin{aligned} \mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z)) &= \sum_i q(z = q|x) \log \frac{q(z = q|x)}{p(z = q)} \\ &= \log K \end{aligned} \tag{2.18}$$

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

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

Chapter 3

Related work / Literature review

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

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

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

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

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

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

Chapter 4

Methology

In this section, we present a detailed description of the methodology used in this thesis. We begin by introducing the Encoder-Decoder architecture, which are crucial components for the VQVAE. We then proceed to describe the VQVAE implementation for timeseries. Finally, we present our modification of the VQVAE, which incorporates the Barlow Twins method to enhance the encoder's ability to learn robust and informative latent representations.

The implementation of the models is inspired by the paper TimeVQVAE[LAM23]. Here they argue for a 2 stage approach. The first stage consists of optimizing the Encoder, Codebook and Decoder to efficiently compress the input into tokens and then reconstruct the input. This is achieved by minimizing the informational loss obtained by comparing the input x with the reconstructed $\hat{x}$. The next stage consists of training a transformer to approximate the prior. Stage 1 aligns with our objectives, as we are interested in the latent space generated by the encoder . They base their implementation on a improved approach to VQVAE, designed to extract better high frequency information. We base our implementation on the naive VQVAE as we are interested in the general case for VQVAEs on timeseries.

4.1 Encoder

The Encoder model, inspired by the fundamental properties of CNNs and the ResNet architecture introduced by He et al.[He+15], is designed to compress the input data into a compact and manageable representation. Approximating the posterior distribution $q_\phi(z|x)$, the encoder learns to map the input data to a latent space. To efficiently approximate the posterior we combine the two results: the CNNs ability to learn complex patterns and features within the input data, and the ResNet's ability to train deeper networks. This is achieved by using a block like structure, where each block is designed to process the input data sequentially.

Encoder blocks are the component responsible for giving the model a perceptual view of the input data aswell as a downsampling to a lower dimension. This is achieved through convolutional layers that strategically reduce the dimensionality of the input, while highlighting important features in non linear activation maps. Following the convolutional layer, tha activation map is normalized through batch normalization. This technique adjusts and scales the activations. Allowing the network to use higher learning rates, as it reduces the internal covariate shift which in turn makes the learning process more efficient and stable, as described by Ioffe and Szegedy[IS15].

Residual blocks introduce residual or shortcut connections, allowing the encoder to have more layers without sacrifizing efficiency in training. These are the building blocks of the original ResNet architecture.

The complete encoder arhitecture is illustrated in figure 4.1. The first encoder block is responsible for reducing the dimensionality of the input data. The following encoder blocks sequentially refines the activation maps to include more complex patterns and features. The number of encoder blocks is determined by a downsampling rate which is determined by the input size and the amount of compression configured for the model. The amount of compression in this implementation is the same as in the TimeVQVAE imlpementation.

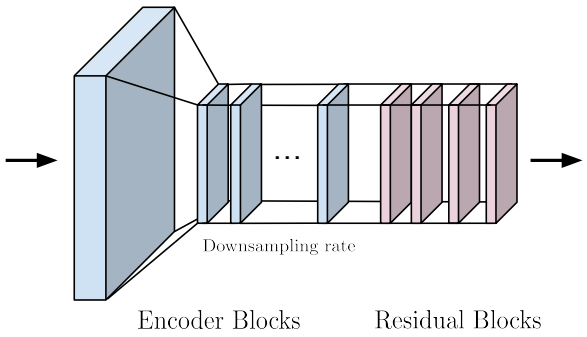


Figure 4.1: Illustration of the Encoder architecture.

4.2 Decoder

The Decoder model, also ispired by CNNs and ResNet, is designed to reconstruct the input data from the latent representation. Approximating the likelihood $p_\theta(x|z)$. Its main function is decompression. It is responsible for the upsampling of the latent representation to a higher dimension, while also refining the feature maps to recover details that were lost during encoding. Like the encoder, the decoder consists of two two types of blocks, decoder blocks and residual blocks.

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

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

4.3 VQVAE

We divide the description of the VQVAE implementation into two sections: procedure of reconstring the input $x \rightarrow \hat{x}$, and the definition of the loss function $\mathcal{L}_{\text{VQVAE}}$ and learning aspect of the VQVAE.

4.3.1 Reconstruction

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

The **STFT** converts the time-domain signal into a time-frequency representation. This transformation allows the encoder to analyze the signal in both time and frequency domains simultaneously, providing a comprehensive view of the signal's spectral content as it varies over time. The encoder thus learns convolutional filters that capture and extract frequency patterns within the input data. Reversly the Inverse Short Time Fourier Transform (**ISTFT**) converts the time-frequency representation back to a time-domain representation. Both STFT and ISTFT are computational methods grounded in a series of fourier transforms. A key parameter is n_{fft}, dictating the resolution of the frequencies in the transformed domain. As done in the TimeVQVAE implementation, we set n_{fft} to be 8.

Next the encoder compresses $\mathbf{u}$ into a continuous latent variable $\mathbf{z}$ followed by a quantization to $\mathbf{z_q}$. Transforming the continous latent space to a discrete latent space. The input is at this state represented as tokens in our codebook. The tokens or discrete latent variables is then passed through the decoder denoted as D, projecting the discrete latent space back into a time-frequency domain, $\hat{\mathbf{u}}$. Where it is then mapped back to a time-domain using ISTFT. Giving the reconstructed input $\hat{\mathbf{x}}$. This is summerized in the figure 4.2

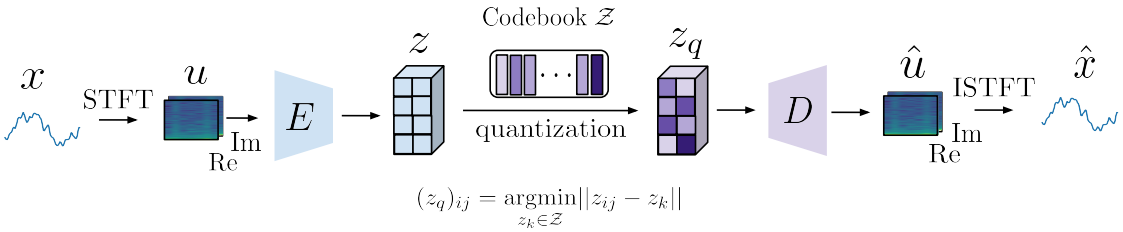


Figure 4.2: Illustration of the VQVAE. Illustration and implementation inspired by the TimeVQVAE paper[LAM23]

4.3.2 Learning

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

Based on the VQVAE paper[OVK17] paper and the ELBO formulation the total loss for our VQVAE is the sum of two parts. Reconstruction loss, $\mathcal{L}_{\text{recon}}$, and codebook loss, $\mathcal{L}_{\text{codebook}}$.

$$\mathcal{L}_{VQVAE} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{codebook}} \tag{4.1}$$

Reconstruction

By minimizing the reconstruction loss, $\mathcal{L}_{\text{recon}}$, we align with maximization of $\log p_\theta(x)$ in the ELBO formulation 2.15. We define the reconstruction loss to be:

$$\mathcal{L}_{\text{recon}} = ||x - \hat{x}||_2^2 + ||u - \hat{u}||_2^2 \tag{4.2}$$

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

By replacing the norms with MSE we get the implemented reconstruction loss:

$$\mathcal{L}_{\text{recon}} = MSE(x, \hat{x}) + MSE(u, \hat{u}) \tag{4.3}$$

Codebook-learning loss

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

The codebook-learning loss is defined as

$$\mathcal{L}_{\text{codebook}} = ||\text{sg}\left[E(\text{STFT}(x))\right] - z_q||_2^2 + \beta||E(\text{STFT}(x)) - \text{sg}\left[z_q\right]||_2^2 \tag{4.4}$$

The first term in the codebook-learning loss,

$$||\text{sg}\left[E(\text{STFT}(x))\right] - z_q||_2^2 \tag{4.5}$$

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

The second term,

$$\beta||E(\text{STFT}(x)) - \text{sg}\left[z_q\right]||_2^2 \tag{4.6}$$

encourages the encoder's output to move closer to the quantized vectors. The parameter β acts as a balanding factor, and is refered to as the commitment weigth. Consistent with the TimeVQVAE implementation, β is set to be 1. By applying the stop gradient on the quantized vector z_q, this term focuses on updating the encoder's parameters to produce outputs that are more aligned with the existing codebook vectors.

Togheter, these two terms ensure that the codebook adapts to the encoded, z. Simultaneously, the encoder adapts to the quantized z_q. As a result the Encoder codebook pair aproximate the categorical posterior, giving the benefits of the VQVAE architecture. As done in the reconstruction loss the norms, $||\cdot||_2^2$, is replaced with MSE in our loss calculation.

Codebook Update Mechanism

In the VQVAE implementation, the quantization is a non-differentiable transformations. To address this, as described by the VQVAE paper by Oord et al[OVK17], gradients from the decoder are copied to the encoder. This approach facilitates the update of the encoder's parameters in response to the loss gradient, even though the quantization step iteself does not support direct backpropagation.

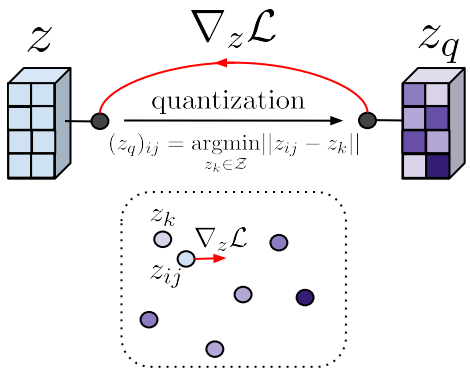


Figure 4.3: Visualization of the embedding space inspired by the original VQVAE paper[OVK17]. The continous z_{ij}, in blue, gets mapped to the nearest token z_k. The gradient $\nabla_z \mathcal{L}$ in the backpropagation is copied from the decoder to the encoder.

The gradient transfer modifier the encoder's output in subsequent forward passes. Simultaneously, the codebook loss $\mathcal{L}_{\text{codebook}}$ updates the codebook tokens to better represent the evolving outputs of the encoder. the

4.4 Modifying the VQVAE for Enhanced Self Supervised Learning

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

4.4.1 Modifying the VQVAE encoder

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

The same encoder in both branches then encodes both time-frequency views, $(\mathbf{u_A}, \mathbf{u_B})$. Producing two latent variables $(\mathbf{z_A}, \mathbf{z_B})$. We continue by calculating the barlow twins loss. First applying a batch normalization, BN, then using a projector denoted as P to expand the dimensionality of the latent variables. As done by Zbontar et al in their paper[Zbo+21]. After the projection to a higher feature space, the cross correlation matrix, C, is calculated followed by the barlow twins loss function, $\mathcal{L}_{BT}$. This is summerized in fig 4.4. As illustrated, the upper

branch is the regular VQVAE and the lower branch is the extension allowing the encoder to process two views.

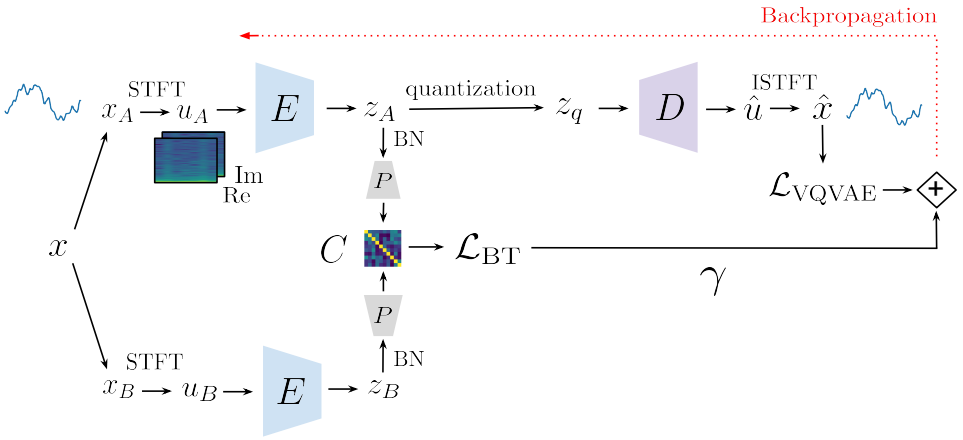


Figure 4.4: Illustration of the Barlow Twins modification to VQVAE including the loss function calculation.

4.4.2 Projector

The projector is implemented as a fully connected neural network. The input gets flattened and passed through a global max pooling layer, summarizing the latent variables, before being passed through the fully connected layers. This architecture is commonly used in SSL methods, and in our implementation it is based on the VICReg paper[BPL21] aswell as the paper by lee et al[LA21].

In the original barlow twins paper [Zbo+21] they found that the dimensionality of the projector had a substantial effect on the barlow twins loss efficiency. A higher projected feature dimensionality (output dimension), d, aswell as a high number of hidden layers in the projector gave better results. Based on this find, we expand the encoded dimension to $d = 4096$ with 4096 hidden layers as done by Lee et al[LA21].

Learning

The modified VQVAE now has a additional SSL objective. As done by Bardes et al in their paper VICReg[BPL21] we scale the barlow twins loss by the projected feature dimension, d, visualized in fig 2.6. Our Barlow Twins loss is extended as follows:

$$\mathcal{L}_{BT}(\mathbf{z_1}, \mathbf{z_2}) = \frac{1}{d}\left[\sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2\right] \tag{4.7}$$

In the original barlow twins paper[Zbo+21] they found that a low valued λ equal to 0.005 gave the optimal balance between invariance and redundancy reduction. The total loss function, $\mathcal{L}_{\text{BT-VQVAE}}$, of our modified VQVAE becomes:

$$\mathcal{L}_{\text{BT-VQVAE}} = \mathcal{L}_{\text{VQVAE}} + \gamma \cdot \mathcal{L}_{\text{BT}} \tag{4.8}$$

The loss calculation is visualized in fig 4.4. The parameter γ determines the weight of the barlow twins loss in the calculation. The $\mathcal{L}_{VQVAE}$ is calculated as done in eq 4.1 using the upper branch reconstruction and codebook.

4.4.3 Augmentation Techniques

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

Time Domain Augmentations

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

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

Time-Frequency Domain Augmentations

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

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

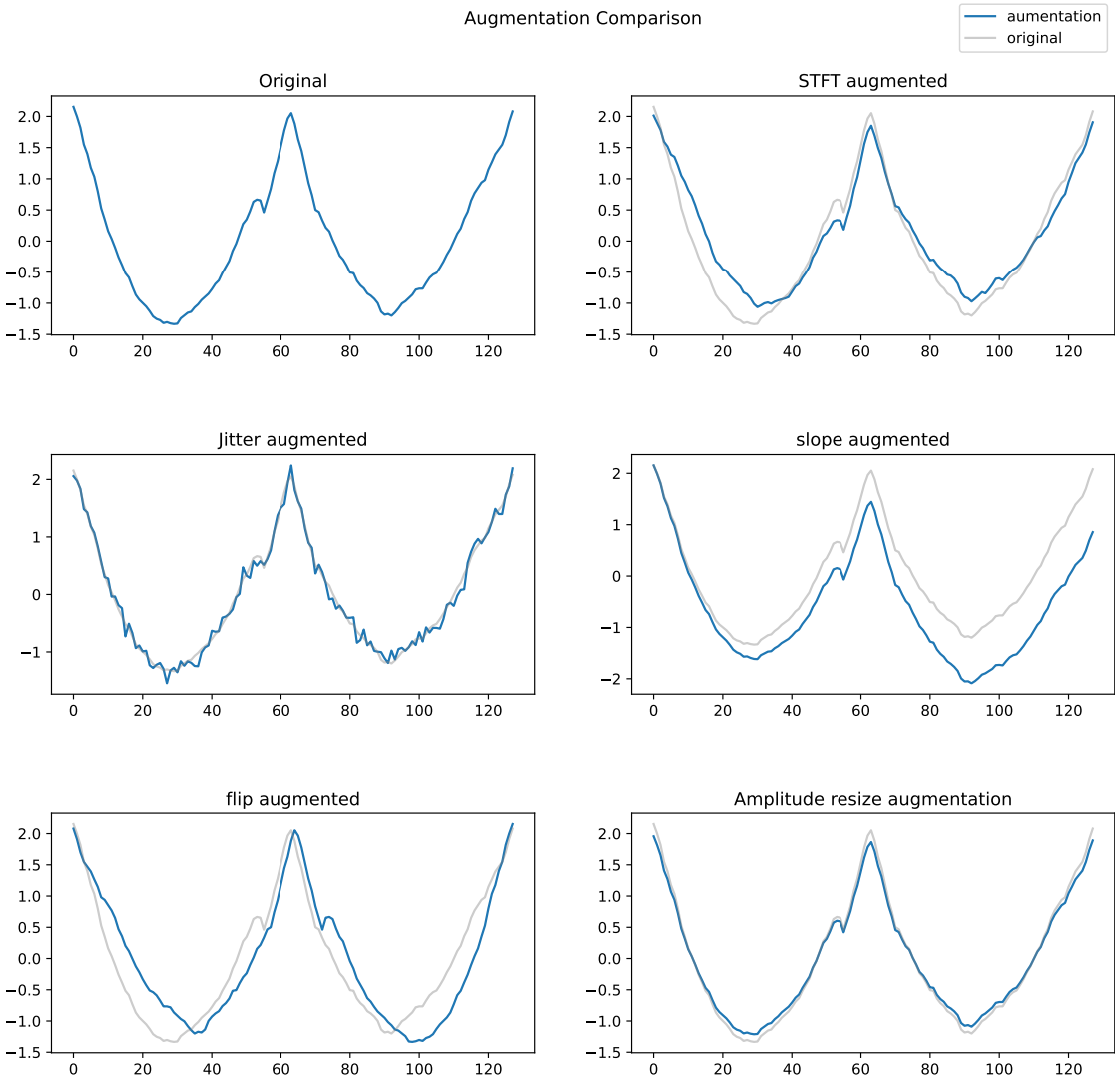


Figure 4.5: Visualization of the flip, slope and STFT augmentation aplied on a timeseries.

Chapter 5

Experimental Setup

Our experimental investigation is structured into two primary focuses:

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

As a benchmark for comparison, we establish the naive VQVAE as the baseline model. Against this baseline, we examine the performance of the Barlow Twins modified VQVAE (BT-VQVAE). We examine how the BT-VQVAE performs as a function of γ in eq 4.8. The comparison is geared towards understanding the enhancements and differences brought about by the Barlow Twins modification, particularly in terms of the quality of the latent representations and the reconstruction accuracy.

5.1 UCR Archive

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

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

Dataset Name	#Samples	#Classes	Length
Electric Devices	16637	7	46
StarLightCurves	9236	3	1024
Wafer	7164	2	152
ECG5000	5000	5	140
TwoPatterns	5000	4	128
FordA	4921	2	500
UWaveGestureLibraryAll	4478	8	945
FordB	4446	2	500
ChlorineConcentration	4307	3	166
ShapesAll	1200	60	512

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

5.2 Reconstruction evaluation

We evaluate the reconstruction accuracy by calculating $\mathcal{L}_{\text{recon}}$ in eq 4.3 using the validation set. Giving $\mathcal{L}_{\text{val-recon}}$. This gives us a score on how well the models generalize on the validation set.

5.3 Downstream evaluation

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

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

5.3.1 Extracting Discrete latent variables

The UCR archive include labels Y alongside the set of timeseries X. Splitted in training and validation datasets respectively. The process of extracting the latent representations is by propagating X_{train} and X_{test} through the encoder and quantization model seperatively. This gives us the models discrete latent representations, Z_{train} and Z_{test}, as illustrated in fig 5.1. The concatenation of Z_{train} and Z_{test} is denoted as Z.

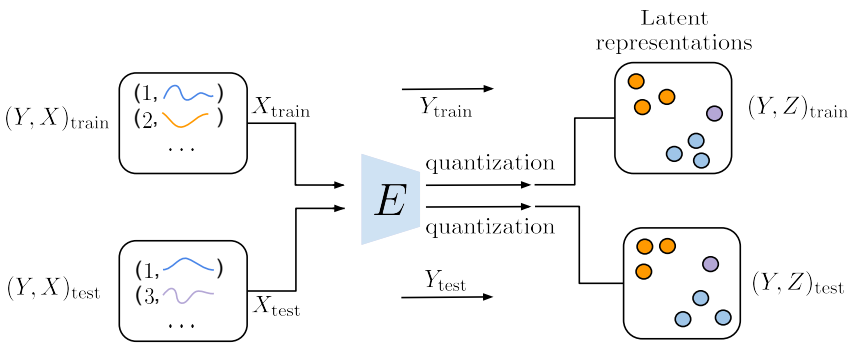


Figure 5.1: Illustration showing the processing of training and validation datasets to latent representations.

During training of the models, we can now perform downstream tests on Z, Z_{train}, Z_{test}.

5.3.2 Downstream tests

The **intrinstic dimension** measure identifies the most straight forward yet comprehensive set of features required to grasp our data's structure. We approximate the intrinstic dimension by applying a Principal Component Analysis (PCA) on Z and pinpoint the number of principal components that account for 95% of the variance. This provides a measure of the complexity of the discrete latent variables. Other metrics like entropy and perplexity is also calculated, giving us additional comparison material.

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

Additionally to the richness of information related to the labels, we are also investigating if

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

Chapter 6

Results and discussion

Chapter 7

Conclusion

Bibliography

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

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

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

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

[F58] Rosenblatt F. 'THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION'. In: *Psychological Review* 65.6 (1958).

[Fuk80] Kunihiko Fukushima. 'Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position'. In: (1980).

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

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

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

[HW] David H. Hubel and Torsten N. Wiesel. 'RECEPTIVE FIELDS AND FUNCTIONAL ARCHITECTURE OF MONKEY STRIATE CORTEX'. In: ().

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

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

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

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

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

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner. 'Gradient-based learning applied to document recognition'. In: (1998).

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

[MP43] W. S. McCulloch and W. H. Pitts. 'A logical calculus of the ideas immanent in nervous activity'. In: *Bulletin of Mathematical Biophysics* (1943).

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

[RHW86a] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. 'Learning internal representations by error propagation'. In: (1986).

[RHW86b] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. 'Learning representations by back-propagating errors'. In: (1986).

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

[RM51] Herbert Robbins and Sutton Monro. 'A Stochastic Approximation Method'. In: (1951).

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

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

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

[Wer90] Paul J. Werbos. 'Backpropagation Through Time: What It Does and How to Do It'. In: (1990).

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

Appendix

Derivation of the Evidence Lower Bound (ELBO)

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

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

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