Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Data-driven Preprocessing of EEG Data

Master's Thesis

Adrian Hoffmann

adriahof@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Ard Kastrati, Martyna Beata Płomecka, Benjamin Estermann
Prof. Dr. Roger Wattenhofer

January 24, 2023

Acknowledgements

I would like to extend my gratitude to all people who helped me during my thesis. Of course, a special thanks to my supervisors. To Ard Kastrati whose advice helped me keep on track and produce a far better and more well-rounded work than would have been possible otherwise. To Martyna Beata Płomecka who was always ready to help me whenever I had trouble with or questions about EEG data or her preprocessing in EEGEyeNet. And to Benjamin Estermann whose knowledge about Variational Autoencoders I drew on countless times. Furthermore, I would like to thank Prof. Dr. Nicolas Langer for his help during my Thesis and Prof. Dr. Roger Wattenhofer for the opportunity to do my Thesis in his group.

Abstract

In this work we investigated ways to preprocess data from Electroencephalography (EEG) in a fully data-driven manner. We surveyed different ways of preprocessing the data and evaluated how well they aid β-TCVAEs and annealed β-TCVAEs to split said data. We succeeded in applying one of the preprocessing schemes to the data from the EEGEyeNet dataset [1]. The resulting models split the data into uncorrelated latent dimensions while keeping the reconstructions at a decent level. We tested the information content of the resulting latent representation by using the trained encoders as the preprocessing step for the raw EEG data in the Amplitude, Angle, and Position task from EEGEyeNet. This led to decent performance on both the Amplitude and Angle task using the raw data (compared to the results from Kastrati et al. on preprocessed data). Furthermore, we investigated what latent dimensions are important for the aforementioned tasks and which less so giving us insights into what characteristics of the EEG data matter to the regression models.

Contents

Acknowledgements **i**

Abstract **ii**

1 Introduction **1**

1.1 Motivation and Goal 1

2 Background **3**

2.1 Independent Component Analysis (ICA) 3

2.2 Variational Autoencoders 4

2.2.1 Basics 4

2.2.2 Variants 5

2.3 Fourier Transform 6

2.4 Skip Connections 7

2.5 Permutation Feature Importance 7

3 Terminology **8**

4 Dataset and Preprocessing **10**

4.1 The EEGEyeNet Dataset 10

4.1.1 Data Acquisition 10

4.1.2 Minimal Preprocessing 11

4.1.3 Cutting the Data 11

4.1.4 Tasks in EEGEyeNet 13

4.1.5 Maximal Preprocessing 13

4.2 Machine Learning Preprocessing Schemes 14

4.2.1 Preprocessing Schemes 14

4.2.2 Advantages and Disadvantages of Common Preprocessing Schemes 15

5 Methods and Modeling **18**
5.1 Discovered Potential . . . 18
5.1.1 Variational Autoencoders . . . 18
5.1.2 VAEs Show Potential . . . 19
5.2 Preparing the Data for the VAEs . . . 19
5.2.1 The Giant Span . . . 19
5.2.2 High Frequencies . . . 20
5.2.3 Line Noise . . . 21
5.2.4 Line Noise and Normalization . . . 21
5.3 Training the VAE . . . 22
5.3.1 Signal vs Sample . . . 22
5.3.2 The Full Pipeline . . . 23
5.3.3 Standard Deviations of the Frequencies . . . 23
5.4 Uncovered Latent Representation . . . 24
5.4.1 Learnt Latent Representation . . . 24
5.4.2 Information Content . . . 24

6 Experiments **26**
6.1 Sanity Checks . . . 26
6.2 Full Dataset . . . 27
6.3 Feature Statistics . . . 28
6.4 Downstream Tasks . . . 28
6.5 Perturbation Importance . . . 29
6.6 Angle and Amplitude from Line Noise . . . 30

7 Results **31**
7.1 Sanity Checks . . . 31
7.2 Full Dataset . . . 31
7.3 Feature Statistics . . . 32
7.4 Downstream Tasks . . . 33
7.5 Perturbation Importance . . . 34
7.6 Angle and Amplitude from Line Noise . . . 35

8 Conclusion **36**

Bibliography **37**

A Latent Dataset **A-1**

B Splitting the Data for the Downstream Tasks **B-1**

C Permuting Electrodes or Latent Dimensions in Latent Datasets **C-1**

D Visualisations of Latent Dimensions **D-1**

E Root Mean Squared Error Versions **E-1**

F Plots Sanity Check Reconstructions **F-1**

F.1 Standardize per Electrode in Time Domain F-2

F.2 Normalize per Electrode in Time Domain F-3

F.3 Standardize per Electrode in Frequency Domain F-4

F.4 Normalize per Electrode in Frequency Domain F-5

F.5 Standardize per Split Frequency in Frequency Domain F-6

F.6 Normalize per Split Frequency in Frequency Domain F-7

F.7 Standardize per Electrode in Frequency Domain (with Logarithm) F-8

G Full Dataset Reconstructions **G-1**

G.1 TCVAE with $\mu = 400$ G-2

G.2 TCVAE with $\mu = 800$ G-3

G.3 Annealed β-TCVAE G-4

H Full Dataset Correlation Matrices **H-1**

H.1 TCVAE with $\mu = 400$ H-2

H.2 TCVAE with $\mu = 800$ H-3

H.3 Annealed β-TCVAE H-4

I Feature Importance Results **I-1**

I.1 Amplitude Task I-2

I.1.1 Latent Dimensions I-2

I.1.2 Electrodes I-3
I.2 Angle Task I-7
I.2.1 Latent Dimensions I-7
I.2.2 Electrodes I-8
I.3 Annealed β-TCVAE I-12
I.3.1 Time Domain I-12
I.3.2 Frequency Domain I-13
I.4 Rank Differences I-13
I.4.1 Amplitude Task I-14
I.4.2 Angle Task I-15

J Feature Statistics **J-1**
J.1 Sanity Checks J-1
J.1.1 Standardize per Electrode in Time Domain J-1
J.1.2 Normalize per Electrode in Time Domain J-2
J.1.3 Standardize per Electrode in Frequency Domain J-3
J.1.4 Normalize per Electrode in Frequency Domain J-4
J.1.5 Standardize per Split Frequency in Frequency Domain . . J-4
J.1.6 Normalize per Split Frequency in Frequency Domain . . . J-5
J.1.7 Standardize per Electrode in Frequency Domain (with Logarithm) J-6
J.2 Full Dataset - Raw J-7
J.2.1 Standardize per Electrode in Time Domain J-7
J.2.2 Normalize per Electrode in Time Dimension J-8
J.2.3 Standardize per Electrode in Frequency Domain J-9
J.2.4 Normalize per Electrode in Frequency Domain J-10
J.2.5 Standardize per Split Frequency in Frequency Domain . . J-10
J.2.6 Normalize per Split Frequency in Frequency Domain . . . J-11
J.2.7 Normalize (-0.5 to 0.5) per Signal in Time Domain J-12
J.3 Full Dataset - Minimally Preprocessed J-13
J.3.1 Standardize per Electrode in Frequency Domain J-13
J.3.2 Standardize per Split Frequency in Frequency Domain . . J-13

Chapter 1

Introduction

Electroencephalography (EEG) is a non-invasive, minimally restrictive, and relatively low-cost measure of mesoscale brain dynamics [1]. Furthermore, EEG is capable of delivering a high temporal resolution. However, EEG also offers some challenges. The main one being the level of noise and amount of artefacts that it usually contains. Among the most prominent are line noise (in Europe this is a 50Hz noise signal) and the artefacts from the eyes moving (a main artefact for the electrodes around the eyes). It is hence not surprising that this is an area of research where the neuroscience community has developed impressive tools to preprocess EEG data, like for example in [2].

1.1 Motivation and Goal

In our work we aimed at investigating if we can use deep learning based approaches to preprocess raw EEG signals. The results could pave the way for a more (or one day maybe even completely) data-driven option for preprocessing EEG data. Such an undertaking consists of two main parts. First, a way has to be devised to train a model with which one can split an EEG signal or at the very least remove unwanted parts from a signal. Second, it has to be ensured that the approach retains important information in the signals.
We mainly see three opportunities in a pipeline that is fully data-driven. First, every system crafted by humans has the potential to introduce human biases. A fully data-driven approach might be able to remove those or shed light on potential biases in today's pipelines. Second, preprocessing as, for example, used in Kastrati et al. [1] is time-intensive - a factor of four was not uncommon in their work. Meaning that five minutes of EEG data could take upwards of 20 minutes of postprocessing (or preprocessing if we take the point of view from a downstream task). However, this can be a problem for a few applications where speed is key, for example live eye-tracking from EEG data which EEGEyeNet [1] is a step towards. Systems based on deep learning have the potential for better performance. Third, solutions based on learning have the added advantage that they can easily make use of additional (training) data which likely yields perfor-

mance improvements. If the approach is self-supervised then this effect is even greater since (unlike with supervised learning) there is no need for labels.
This work is a first step into this direction. We investigated several common preprocessing methods and what their advantages and disadvantages are on EEG data. Furthermore, we successfully trained a VAE on raw EEG data and used its encoder successfully as a preprocessing step for the Amplitude and Angle task from EEGEyeNet.

Chapter 2

Background

2.1 Independent Component Analysis (ICA)

The ICA is an algorithm for the blind source separation problem wherein one tries to recover the independent source signals from only sensor observations [3]. To add to the difficulty, not only the source signals but also the way they were mixed are unknown. In the ICA the problem is modelled as a matrix multiplication of the source signals $\mathbf{S}$ with shape $M \times T$, the full rank mixing matrix $\mathbf{A}$ with shape $N \times M$, and the sensor observations $\mathbf{X}$ with shape $N \times T$. M is the number of source signals, N is the number of sensors, and T is the number of time steps. These are combined into the following model:

$$\mathbf{X} = \mathbf{AS}$$

Further assumptions taken by the ICA are:

1. There are more sensors than sources, i.e. $N \geq M$
2. The sources are mutually independent at each time instant
3. At most one source is normally distributed
4. The sensor noise must be additive and low (or non-existent) - however, noise can also be interpreted as another source signal itself to which the above assumptions need to apply

If one had the mixing matrix $\mathbf{A}$, then it would be easy to compute $\mathbf{S}$ from $\mathbf{X}$ since one only needs to compute the inverse of $\mathbf{A}$ (called the unmixing matrix $\mathbf{W}$) and then compute $\mathbf{WX} = \mathbf{S}$ yielding the source signals. However, computing $\mathbf{A}$ or $\mathbf{W}$ is not easy (it is even ill-defined) and since it does not add to the material in the rest of this work, we do not dive into the theory of optimizing this problem.

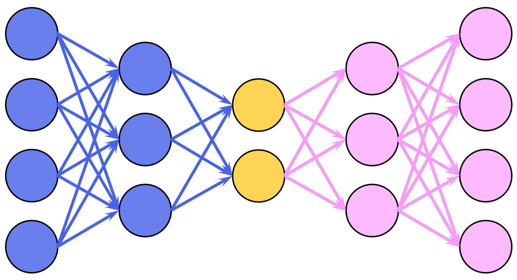


Figure 2.1: Visualisation of an autoencoder. The blue parts are the encoder and the pink ones are the decoder. The yellow neurons form the bottleneck which means they are simultaneously the output neurons for the encoder and the input neurons for the decoder.

2.2 Variational Autoencoders

We assume that the reader has a basic understanding of deep learning. Terms like dense layer, non-linearity, neurons, or loss function in this context should be familiar.

2.2.1 Basics

Autoencoder: The idea behind an autoencoder is that the model implements the identity function, i.e. that its output is the same as its input. This operation, however, becomes more interesting once one introduces constraints like a bottleneck layer (see Fig. 2.1 for a visualisation). The autoencoder hence has to compress the input (for example, an image) such that the neurons in the bottleneck can capture its information - we denote this vector as z. This representation z is called the *latent representation* of the input. The training is very straightforward in the basic case since one only needs to define a differentiable distance between the input and the output of the model (e.g. the mean squared error) and can then apply the optimizer of their choice.
Variational Autoencoder (VAE): There is nothing that prevents (or discourages) the activations of neurons in the bottleneck layer of an autoencoder to be correlated. This is fine if one is mainly focused on the quality of the reconstructions. However, interpreting the latent representations can be difficult since multiple neurons might have intricate correlations. The goal of VAEs is to move a bit away from this by introducing a prior for the latent dimensions and encouraging them to stay close to the prior [4, 5]. This is achieved by changing the bottleneck slightly and adding a term to the loss. In a VAE the encoder outputs the parameters of a distribution which is typically a multivariate Gaussian with a diagonal covariance matrix (see Fig. 2.2 for a visualisation). Let us denote the vector of predicted means with $\mu = [\mu_1, \mu_2, ..., \mu_k]^T$ and the matrix with the

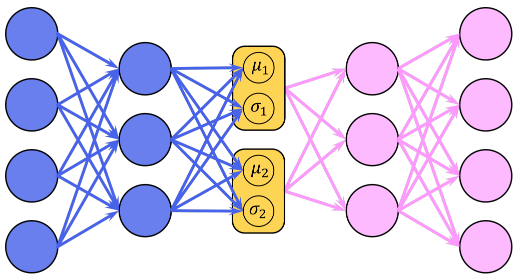


Figure 2.2: Visualisation of a VAE. The encoder (blue) computes statistics of a distribution (here multivariate Gaussian with diagonal covariance matrix) which are used to draw from said distribution yielding the input to the decoder (pink).

predicted standard deviations on the diagonal as $\sigma = diag(\sigma_1, \sigma_2, ..., \sigma_k)$ where k is the dimension of the modeled multivariate Gaussian. To get from μ and σ to the input for the decoder one draws a sample s from a multivariate normal distribution and computes $z = \sigma * s + \mu$, where $*$ is matrix multiplication. z is then fed into the decoder. Note that sampling is only necessary during training. One can simply take $z = \mu$ once the VAE is trained.
This allows us to add an additional term to the loss, which is the Kullback-Leibler-Divergence (KLD) between the variational distribution (defined through μ and σ) and the multivariate normal distribution (our prior). Thus, the loss has a reconstruction term and a KLD term. The latter of which can be understood as a regularizer that encourages the model (specifically the encoder) to predict (for a given) input a variational distribution that is close to a multivariate normal distribution. And with that comes that the individual dimensions are uncorrelated.
Visualising latent codes: Since the different dimensions of the latent representations are uncorrelated in a well-trained VAE, we can gain insight into what concepts different latent dimensions might encode. For that, we focus on the decoder of a trained VAE. One would usually start to take the zero vector and feed it into the decoder. Afterwards, one would input a series of vectors whose entries are all 0 but for one dimension. For that specific dimension, one would vary the values slightly and observe the changes in the output of the decoder. Whatever changes that are observed in the output seem to be encoded in the dimension whose values were changed. This can be done for every dimension. However, one has to be careful: while the latent dimensions are uncorrelated that does not mean that the decoder does not combine the information from different dimensions for its output.

2.2.2 Variants

β-VAE: Higgins et al. [6] showed the benefits of introducing a tunable hyperparameter β to lower or heighten the relative weight the KLD term has over the

reconstruction term in the VAE loss. We use this additional hyperparameter throughout our work.
β-TCVAE: Chen et al. [7] decoupled the KLD term in the loss of the VAE and split it into three parts which allowed them to introduce further hyperparameters to more precisely weight different parts of the loss, like the correlation between different latent dimensions. We used this split but rearranged the terms slightly such that we ended up with a loss of the shape $\mu *$ reconstruction $+ (\beta - 1) *$ total correlation $+$ KLD term, where μ and β are tunable hyperparameters.
Annealed VAE: Burgess et al. [8] introduced the annealed VAE which replaces the KLD term in the VAE loss with an absolute difference between the KLD term and C. During training C is slowly increased until some maximum value is reached. The aim of this is to steer the KLD term in the loss and only let it slowly go up, which was shown to lead to a learning behaviour where different concepts were learned one at a time instead of all at once. The absolute difference is weighted by an additional hyperparameter γ. The maximum value of C, the slope of the increase of C during training, and γ are all tunable hyperparameters. We deploy this in some of our experiments where we replace the KLD term in our rearranged β-TCVAE loss with the just described absolute difference.

2.3 Fourier Transform

A common tool when working with signals is the Fourier transform with which one can decompose a signal (possibly complex) into its different frequencies. We made use of this transformation in our work but since the EEG data is a discrete signal, we did not use the Fourier transform itself, but its discrete version the Discrete Fourier transform (DFT), of which the formula is given below:

$$S_k = \sum_{n=0}^{N-1} s_n \cdot \left[\cos\left(\frac{2\pi}{N} kn\right) - i \cdot \sin\left(\frac{2\pi}{N} kn\right) \right]$$

where $s = [s_0, s_1, ..., s_{N-1}]$ is the signal in time domain and $S = [S_0, S_1, ..., S_{N-1}]$ the signal in frequency domain. N will be 500 in all our cases. Hence, the Nyquist frequency (the highest frequency after which aliasing happens) with our sampling rate of 500 Hz is 250 Hz. Furthermore, since we have a real-valued input (EEG data) $S_{499}, S_{498}, ..., S_{251}$ are simply the complex conjugate of $S_1, S_2, ..., S_{249}$ respectively. Hence, we only need to compute $S_0, S_1, ..., S_{250}$ to capture all information. Lastly, in our work we split the vector $[S_0, S_1, ..., S_{250}]$ into $[S_0^r, S_1^r, ..., S_{250}^r, S_0^i, S_1^i, ..., S_{250}^i]$ where S_k^r denotes the real part of S_k and S_k^i denotes the imaginary part of S_k. We did this to get a real-valued vector again.

2.4 Skip Connections

For our models, we utilised skip connections from the ResNet paper by He et al. [9]. However, we do not use convolutions but rather dense networks. For our purposes, one *ResNet block* consisted of first a fully connected layer with ReLU and batch normalization. This fully connected layer was used to steer the number of neurons in hidden layers (in He et al. [9] this was achieved with larger strides or pooling). The first layer was followed by two more sets of a linear layer followed by ReLU and batch normalization, each keeping their input dimension. Afterwards, we utilised the skip connection by adding the resulting vector and the vector from after the first set of operations together. The block ends with passing the result from the addition through another ReLU and batch normalization layer. We term the *dimension of the block* to be the dimension of the output vector - which is also the dimension of the vectors being added.

2.5 Permutation Feature Importance

To gain more insights into our models we made use of the permutation feature importance. The concept was first introduced by Breiman [10] for random forests. The basic idea is that if a feature is insignificant for a model then we can permute the values for that feature over all the samples in a dataset and have the outputs for the dataset change little or not at all. It is obvious that the idea can be adapted to many situations. We used it to get an idea about the importance of different latent dimensions and different electrodes for our models.
This method is unfortunately not perfect. As Molnar [11] writes, the results derived from this method can vary greatly because of the randomness involved in the permutation. This can be mitigated to a degree by running multiple rounds and taking the mean. However, the problem might not completely vanish and the computation time increases. Another problem is that unrealistic data is likely generated by permuting the values of a feature, which is an especially pressing issue if features are correlated.

Chapter 3

Terminology

We introduce some terminology that allows us to elucidate our work precisely in a much more readable and concise way:

- **VAE**: We use this as an umbrella term for all variations of VAEs that we introduced in Section 2.2.
- **Sample**: This is one unit from the EEGEyeNet dataset, i.e. a matrix with 129 rows (= the different electrodes) and 500 columns (= a one-second window) containing the EEG data from a measurement. Samples come in different flavours based on the level of preprocessing Kastrati et al. [1] applied. There are raw samples, minimally preprocessed samples, and maximally preprocessed samples.
- **Signal**: General term for one of the rows of a sample, i.e. a vector with 500 entries. Hence, a signal always has an electrode associated with it.
- **Latent Signal**: A signal in a latent representation. We usually created a latent sample when we passed a signal through the encoder of a VAE. The computed means (or a subset thereof) are then the latent signal.
- **Latent Sample**: A sample for which all 129 signals were replaced with their respective latent representations.
- **Electrode**: Can either be the physical electrode that is used to measure the EEG data or refer to the rows of samples. Note that in the latter case that includes the reference signal from the EEG (the 129th electrode).
- **Channel**: This is used as a synonym for electrode.
- **Split representation**: The representation we introduced in Section 2.3 where we used $[v_0^r, ..., v_{n-1}^r, v_0^i, ..., v_{n-1}^i]$ to represent an array of complex numbers $[v_0, ..., v_{n-1}]$.
- **Frequency**: This either refers to a frequency in Hertz or the response value from the DFT for a specific frequency.

- **Split frequency**: Refers to the real *or* imaginary part of a response value from the DFT for some frequency. This term is used in conjunction with the term ‘split representation’ since we represent the complex output array of the DFT of some signal in the split representation.

Chapter 4

Dataset and Preprocessing

At the center of this thesis is the EEGEyeNet dataset from Kastrati et al. [1]. Therefore, it makes sense to further elaborate on this dataset and the preprocessing that they performed.

4.1 The EEGEyeNet Dataset

Kastrati et al. [1] published a paper accompanying the EEGEyeNet dataset at NeurIPS 2021. We will not repeat all the information in said paper but rather only some select parts that are relevant for this work. Thus, there will be details and parts that we intentionally keep high level or non-specific to keep the description concise. If the reader wants more information we refer them to [1].

4.1.1 Data Acquisition

EEGEyeNet is a dataset that was introduced by Kastrati et al. [1] in their NeurIPS 2021 paper. Each participant who contributed to the dataset was placed in front of a monitor using a chin rest to ensure a stable position. They were then presented with one of several tasks. The one we are interested in is based on the "Large Grid" paradigm where the participants (the version of the dataset we used contained data from 72 participants) were asked to fixate on 25 different dots which were displayed sequentially. Two modalities were recorded while the participants were performing these tasks. One is 128 channel, high-density EEG data at a sampling rate of 500 Hz and the other is the participant's eye position (i.e. where the participant looks at on the monitor).
For the full setup we refer you to the paragraph "Large Grid" in subsection 3.4 in [1].

4.1.2 Minimal Preprocessing

This work is mainly concerned with the raw data meaning the steps in this subsection were not applied to the data used in this work. However, the works of other people used a preprocessed version of the data and also draw on the preprocessed data from time to time.
The preprocessing consists of a hand full of steps which are visualised in Fig. 4.1. First, bad channels were detected and removed from the data (resulting in the second subplot of Fig. 4.1). A channel was deemed bad if it contained a lot of line noise, it contained too little signal variation (for example, if an electrode detached), or it contained a lot of other noise. Next, the Zapline toolkit [12] was applied to the remaining channels to remove line noise artifacts in them (which in Europe is an easy to spot 50 Hz artefact). Also, high-pass filtering with a cut-off at 0.5 Hz was applied to the remaining data which primarily affected the spans of the signals. The result after filtering (Zapline and high-pass) is visualised in the third subplot of Fig. 4.1. As the last step, each of the removed channels was replaced with a signal that was interpolated from the electrode's neighbours (resulting in the fourth subplot of Fig. 4.1). All these operations were performed per recording using Automagic [2]. A recording lasted for longer than the two seconds visualised in Fig. 4.1.
We call data which was preprocessed as described in this subsection to be **minimally preprocessed**.

4.1.3 Cutting the Data

At this point the data was still a stream of long recordings, not particularly suited for machine learning. Hence, there was one final step: cutting the data. After this, two (sub)datasets emerged each with samples of the shape (129, 500) as the input features. We will consider the output labels in the next subsection.
129 electrodes: Although there were 128 physical electrodes on a participants head measuring the EEG data, a sample in EEGEyeNet has 129 electrodes. Along with the 128 physical electrodes there is also a reference electrode. The values for a physical electrode stored in a sample are the difference between the measured voltage on the participants head for that electrode and the reference electrode.
500 timesteps: As mentioned above, the EEG data was measured at 500 Hz. Hence, each sample in the dataset spans one second of a signal.
(Sub)datasets: For one dataset the EEG data is cut such that a saccade (= a movement of the eye) happens in the middle of the sample and in the other dataset each sample is a one second window of a fixation (= no movement of the eye). We call the former set the **direction dataset** and the latter the **position dataset**.

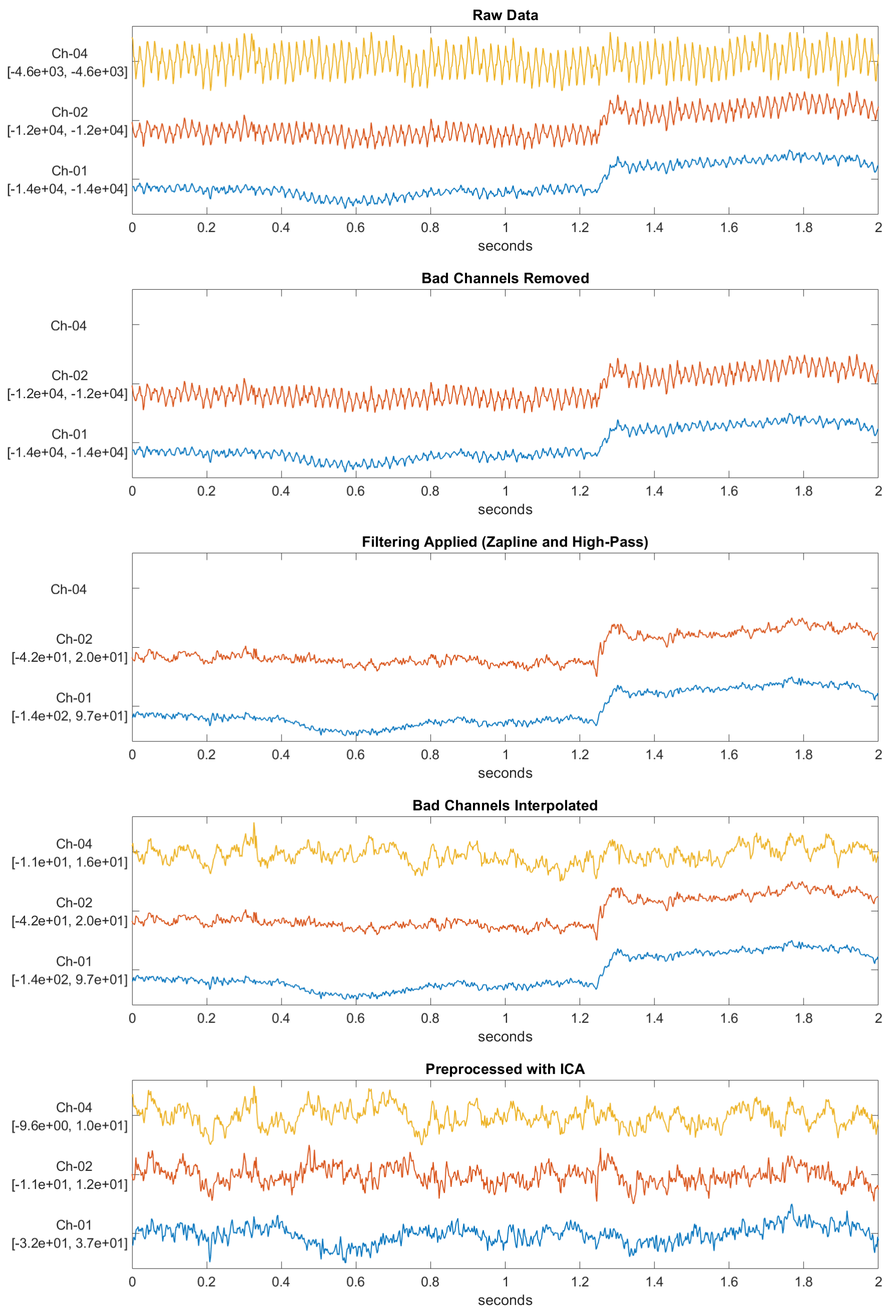


Figure 4.1: Three channels from real EEG data at different steps of the preprocessing pipeline from Kastrati et al. [1]. We only show two seconds here but the preprocessing is applied to the whole recording. The smallest and largest millivolt values of each displayed signal are noted below the y-axis tick labels (rounded to 1 decimal). The second to last subplot is the minimally preprocessed data. The last one is the maximally preprocessed data.

4.1.4 Tasks in EEGEyeNet

Kastrati et al. [1] proposed three different tasks that accompany the EEGEyeNet dataset. These tasks were termed Left-Right, Angle/Amplitude, and Absolute Position. For each of them there is a separate dataset in EEGEyeNet (i.e. three datasets). The Angle/Amplitude task is sometimes also counted as two tasks which we do as well in this work. The Amplitude, Angle, and Absolute Position tasks were of interest for us.
A model has to predict the distance the gaze of a subject travels on the monitor based on the EEG data in the Amplitude task. In the Angle task, models have to predict the Angle at which this movement happens. The distance is measured in pixels and the angle in radian. Both tasks are based on the direction dataset and hence the dataset contains the amplitude and angle of the saccade as the labels for each sample.
The Absolute Position task on the other hand asks of a model to predict the pixel that a subject is fixating on based on the EEG data of a sample. The position dataset hence has the x and y coordinates of the pixel a subject is fixating on during the sample its label.
Furthermore, each sample has a subject-ID associated with it so that we can distinguish between the samples of different participants. This holds for both datasets, direction and position.
Note that the focus of our work was not to improve the the current best scores on these tasks. However, they were of great convenience to test our approach (see more in Chapter 6).

4.1.5 Maximal Preprocessing

In Subsection 4.1.2 we already saw what Kastrati et al. [1] applied as their minimal preprocessing to the raw EEG data. While they used the minimally preprocessed data for their benchmarks they also produced a maximally preprocessed version of the data. This maximal preprocessing is built on top of the minimally preprocessed data. ICA was applied to the minimally preprocessed data as the next step. Specifically, on a per recording level (imagine a five minute chunk of data of a subject) the minimally preprocessed EEG data was used as the sensor observations $\mathbf{X}$. Once the source signal matrix $\mathbf{S}$ and hence the independent components (individual rows of $\mathbf{S}$) were computed they used ICLabel [13] to classify each of the components. ICLabel is a classifier trained to classify independent components from EEG data into the 7 categories Brain, Muscle, Eye, Heart, Line Noise, Channel Noise, and Other. Based on these labels, all independent components that were classified Eye, Heart, Line Noise, or Channel Noise (with at least 80% certainty) were removed from $\mathbf{S}$ (and the according columns from $\mathbf{A}$) and these filtered $\mathbf{A}$ and $\mathbf{S}$ were then used to generate a filtered version of the EEG data $\mathbf{X}$ (resulting in the fifth subplot of Fig. 4.1). We show in Fig. 4.2 some

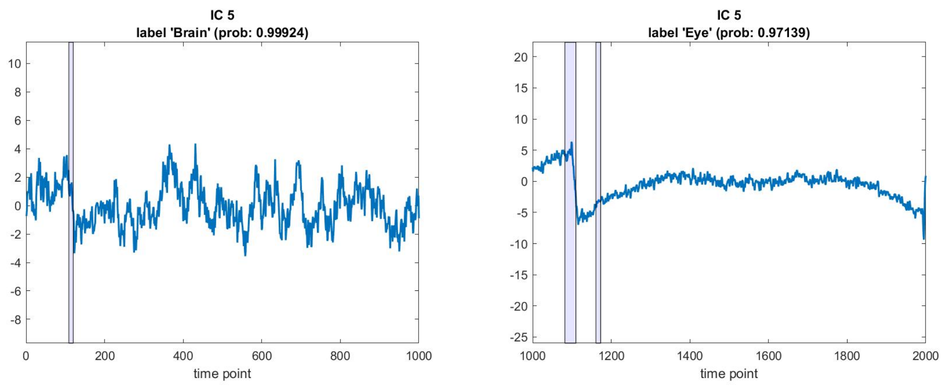


Figure 4.2: Two windows (each of two seconds length) of independent components from two different recordings. The ICLabel classification is in their titles along with the probability for the class. The small shadowed areas towards the beginning of the plots show when the subjects made a saccade (eye movement). The unit along the y-axis is millivolt. However, the values have limited meaning since the ICA is ill-defined.

examples of independent components and their classification. Automagic [2] was also used to perform the ICA step of the preprocessing.

4.2 Machine Learning Preprocessing Schemes

In this section we describe preprocessing schemes that are commonly used in machine learning, as well as what their strengths and weaknesses are for EEG data.

4.2.1 Preprocessing Schemes

In machine learning it is generally a good idea to apply some level of preprocessing to ones data as models have proven to perform better and learn more quickly with proper preprocessing. At the same time we needed to be mindful as we do not want to introduce too much human bias into the system.
Two schemes that fit this description were *normalization* and *standardization*. In standardization one removes the mean from data and divides it by its standard deviation. This both helps to reduce the spread of data and makes the absolute values of the numbers smaller. Normalization takes this a step further and subtracts the smallest value in data from data and then divides the result by the span of data (the largest value in data minus the smallest value). The result is that data is between zero and one. The only question that is left to the users is what "data" exactly means in their case. We elaborate in Subsection 4.2.2.

Another common trick that is used in machine learning is to take the logarithm of data. This helps to make large numbers a lot smaller while keeping some relative information (if $x > y$ then also $log(x) > log(y)$).

4.2.2 Advantages and Disadvantages of Common Preprocessing Schemes

A general advantage is that these schemes are simple and that they have proven their worth in many domains. Furthermore, they help with the extreme spread of values the raw EEG data exhibits.
The question remains for standardization and normalization over what portions of the data we compute their statistics (mean, standard deviation and minimum, span). To discuss this, we introduce a very simplified dataset which only contains two samples which themselves are simplified too (they only have 3 electrodes and 4 timepoints instead of 129 and 500 respectively):

$$s_1 = \begin{pmatrix} 1 & 3 & 4 & 18 \\ 2 & -3 & 2 & -3 \\ 4 & 0 & -2 & 1 \end{pmatrix} \quad s_2 = \begin{pmatrix} 2 & 2 & 0 & -1 \\ 3 & 4 & -1 & 2 \\ 1 & -1 & -13 & -1 \end{pmatrix} \tag{4.1}$$

Floating point numbers are rounded to two digits after the comma in the following and we only note down the two matrices themselves (i.e. we do not add the variable names) so that we can fit them side by side. In the following we go through some options of scopes over which the statistics can be computed.
Per dataset: Computing the statistics for normalization over the whole dataset $(\text{min} = -13, \text{span} = 31)$ is very susceptible to outliers which leads to the bulk of the values being squeezed into a small range as can be seen by the normalized versions of our example:

$$\begin{pmatrix} 0.45 & 0.52 & 0.55 & 1.0 \\ 0.48 & 0.32 & 0.48 & 0.32 \\ 0.55 & 0.42 & 0.35 & 0.45 \end{pmatrix} \begin{pmatrix} 0.48 & 0.48 & 0.42 & 0.39 \\ 0.52 & 0.55 & 0.39 & 0.48 \\ 0.45 & 0.39 & 0.0 & 0.39 \end{pmatrix} \tag{4.2}$$

A similar pattern can also occur if the data has some electrodes whose values are all much closer to 0 than for most other electrodes. Those values would then also be squeezed into a small range - no matter the noise level. These two problems are important in our case because the signals in the raw EEG data can have very large absolute values.
Standardization also has this problem but to a lesser extent since individual outliers affect the mean and the standard deviation not quite as much as they can the minimum value and the span of the data. In our case (calculating the statistics over the whole dataset) gives $\mu = 1, \sigma = 4.93$ and standardized data:

$$\begin{pmatrix} 0.0 & 0.41 & 0.61 & 3.45 \\ 0.2 & -0.81 & 0.2 & -0.81 \\ 0.61 & -0.2 & -0.61 & 0.0 \end{pmatrix} \begin{pmatrix} 0.2 & 0.2 & -0.2 & -0.41 \\ 0.41 & 0.61 & -0.41 & 0.2 \\ 0.0 & -0.41 & -2.84 & -0.41 \end{pmatrix} \tag{4.3}$$

Per electrode: In this version the statistics are computed for each electrode independently. In our examples this would mean that we compute 3 means and standard deviations for standardization or 3 minimums and spans for normalization. This already lessens some of the problems that we discussed above. The effect of outliers are now confined to the electrodes they are in. For example, the three minimums for our dataset are $(-1, -3, -13)$ and the spans are $(19, 7, 17)$ which leads to values that are not as squeezed:

$$\begin{pmatrix} 0.11 & 0.21 & 0.26 & 1.0 \\ 0.71 & 0.0 & 0.71 & 0.0 \\ 1.0 & 0.76 & 0.65 & 0.82 \end{pmatrix} \begin{pmatrix} 0.16 & 0.16 & 0.05 & 0.0 \\ 0.86 & 1.0 & 0.29 & 0.71 \\ 0.82 & 0.71 & 0.0 & 0.71 \end{pmatrix} \tag{4.4}$$

However, there is some relative information removed. For example, the means of the different electrodes in the original data in 4.1 are $3.62, 0.75, -1.38$. In 4.4 this changes to $0.24, 0.54, 0.68$ which clearly shows that the relationships between the means have been broken. For standardization this is even more obvious since the mean is removed for each electrode individually and hence every electrode's mean is zero after standardization. Importantly, removing this relative information also happens in a similar manner in the preprocessing pipelines discussed in Section 4.1 with the high-pass filtering effectively removing the mean and it is still assumed that the resulting data provides enough information to solve the different tasks presented in Kastrati et al. [1].

Per timepoint or split frequency: For the schemes so far we did not think about the machine learning model that will receive the data. The VAEs we used in our work took individual signals as their input, meaning that every time point or every split frequency (depending on whether the DFT is applied to the signal or not) can be considered one feature. Consequently, it is tempting to normalize/standardize per input feature as models (and hence also VAEs) tend to do better if their input features all have about the same mean and standard deviations. However, in our case, normalizing/standardizing per feature removes information between timepoints which is very important information. For example, say we standardize per time point in our example dataset. The means for the four timepoints are $2.17, 0.83, -1.67, 2.67$ and the standard deviations are $1.07, 2.41, 5.44, 7.04$. Hence, the standardized data is:

$$\begin{pmatrix} -1.09 & 0.9 & 1.04 & 2.18 \\ -0.16 & -1.59 & 0.67 & -0.8 \\ 1.72 & -0.35 & -0.06 & -0.24 \end{pmatrix} \begin{pmatrix} -0.16 & 0.48 & 0.31 & -0.52 \\ 0.78 & 1.31 & 0.12 & -0.09 \\ -1.09 & -0.76 & -2.08 & -0.52 \end{pmatrix} \tag{4.5}$$

The oscillating signal from the middle row in s_1 suddenly has a much more irregular behavior. It is again a good moment to repeat that we are interested in the latent representations the VAEs produce. If we were only interested in the reconstructions then preprocessing would not be a problem since we could simply apply the reverse to the output of the VAE. However, the latent representation of a VAE can only capture the information that is fed into it. Hence, one has

to be very careful with changing the shape of the signal during preprocessing as this is vital information.
The story is similar if we first apply the DFT to all signals and put the results into the split representation. Then one feature is a split frequency. Normalizing (or standardizing) per split frequency breaks the relative information between the different frequencies that are contained in the original signal as well as the relative information between the real and imaginary part of the DFT value of each frequency. This hence also changes the shape of the signal.
Per signal: This lessens the problem of outliers even more since an outlier would only affect the signal it is in. Since we have 6 signals we would compute 6 minimums and 6 spans, here depicted as 4 vectors (the subscript shows whether it was derived from s_1 or s_2):

$$\text{min}_1 = \begin{pmatrix} 1 \\ -3 \\ -2 \end{pmatrix} \quad \text{min}_2 = \begin{pmatrix} -1 \\ -1 \\ -13 \end{pmatrix} \quad \text{span}_1 = \begin{pmatrix} 17 \\ 5 \\ 6 \end{pmatrix} \quad \text{span}_2 = \begin{pmatrix} 3 \\ 5 \\ 14 \end{pmatrix}$$

Using these values to normalize the data per signal yields:

$$\begin{pmatrix} 0.0 & 0.12 & 0.18 & 1.0 \\ 1.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.33 & 0.0 & 0.5 \end{pmatrix} \quad \begin{pmatrix} 1.0 & 1.0 & 0.33 & 0.0 \\ 0.8 & 1.0 & 0.0 & 0.6 \\ 1.0 & 0.86 & 0.0 & 0.86 \end{pmatrix} \tag{4.6}$$

As we can see, the values are very well behaved, the outliers (the 18 and -13) only influence their respective signals, and the shape of the signals is preserved. However, information between different signals has been removed and hence even the relative information between signals *within one sample* has been tampered with. For example, the means for the three electrodes in s_1 are 6.5, -0.5, and 0.75 but in 4.6 the means are $0.32, 0.5$, and 0.46 - the order has flipped. Furthermore, this preprocessing scheme can leak some information about some frequencies into the values of other frequency components in the DFT. But more on this later.
Logarithm: Models can struggle if one's data has some values or features that are much larger than others. We have already seen some preprocessing options now. Another one to help make the data better for your model is the logarithm since it lowers very high values much more than smaller ones. Yet, the logarithm has also its disadvantages. For one, the logarithm is not defined for negative numbers in the real space. Meaning that one has to venture into the complex numbers for such cases - which was not a problem for us since we were working with the DFT anyways. Second, it creates very negative numbers if the input is close to zero. And zero itself is an invalid input.

Chapter 5

Methods and Modeling

In this section, we share the conceptual insights we have garnered during this work. We set out by motivating our choice of model and goal in this work. We transition to discussing the data and its particularities. This is followed up by a part about Variational Autoencoders and their latent representations of the EEG data. We close out by sharing insights about using a VAE as a preprocessing step for tasks in EEGEyeNet.

5.1 Discovered Potential

Learning projects on EEG data these days usually start by cleaning the data with tools such as the ones used by Kastrati et al. [1]. Afterwards, a model is trained on the cleaned data. This approach has many advantages. The tools used for cleaning are well established and we understand what they do. Machine learning models generally perform better when the data they receive is freed from unnecessary information in the data - such as noise or unwanted artefacts. But there are also disadvantages, one being speed. The minimal preprocessing used by Kastrati et al. [1] typically takes more time to process a recording than the recording itself captures. This is a problem for real-time applications. Also, these pipelines are not particularly simple.
Hence, a question worth asking is whether there are other options to prepare data for downstream tasks with different advantages and disadvantages. Work in this direction will at the very least leave us with a better understanding of the domain itself.

5.1.1 Variational Autoencoders

Models that introduce new advantages are VAEs. As a start, they do not require labelled data but only the data to be reconstructed, thus cutting the need for expensive labelling. We also can cut down on other complex preprocessing steps by using a VAE as the centrepiece for preprocessing raw EEG data. This also

allows for potentially much faster preprocessing overall. Lastly, the latent space of VAEs is particularly nice since the latent dimensions are (in the optimal case) independent of each other giving a level of interpretability to its latent space.
This also offers the opportunity to make the decision processes of models that build on this latent representation more interpretable. One can, for example, train a classifier on the latent representation and then employ some feature importance method to it. This allows one to better isolate what concept of the latent space is important to the classifier. Something that is much more difficult with, for example, the models and preprocessing from Kastrati et al. [1].
However, VAEs also have their downsides. To begin with, they themselves are models and have difficulty with the (raw) EEG data. Furthermore, it might not be evident what concept a latent dimension captures. And even if it is, there is usually also some noise encoded in the latent dimension.

5.1.2 VAEs Show Potential

There has not been work into using VAEs to preprocess EEG data (or reconstruct it) to the best of our knowledge. However, several simple preprocessing schemes on a subset of the minimally preprocessed dataset allowed us to train VAEs that gave good reconstructions of the EEG data. We hence proceed to share our insights into VAEs and the raw data.

5.2 Preparing the Data for the VAEs

There is very little chance that one could train a model (VAE in our case) directly on the untreated, raw EEG data because it simply has too many problems (for example the giant ranges the values span or the prevalence of so many kinds of noises and artefacts like channel noise, line noise, or muscle artefacts). Thus, a pipeline is necessary to bring the data into a format that a VAE can handle.
There are a few circumstances where today's deep models perform better. One of them is that data should live in a preferably small space which the training data nicely covers. Another one is that the different features all have about the same mean (preferably around zero) and a reasonable standard deviation. Furthermore, they like large amounts of training data.
In the pursuit of these, we now go through the key issues in the EEG data we targeted.

5.2.1 The Giant Span

One core problem of the EEG data is that it covers a giant range of values. To give an impression: the smallest standard deviation is a little above 12'000 millivolts if we compute the per electrode standard deviations on the raw direction

dataset.
Normalizing per signal combines some major advantages when it comes to this problem. For one, outliers have a very localised impact (a big shortcoming of normalizing per electrode). Furthermore, it forces each signal into a small range between -0.5 and 0.5 so the model does not have to prepare for very large or small values in the input. This preprocessing also makes it easier for the model to concentrate on just the shapes of the signals. What do we mean by that? A lot of signals share similar shapes when plotted. However, for some these patterns happen with very large millivolt values, for others with very low millivolt values, for some the shape spreads over a large span, and for others over a small one. Normalizing can remove all this variability. Thus, the model can directly learn the shapes of the signals and does not need to abstract away the mentioned variability that has nothing to do with the shapes themselves. Lastly, we normalize between -0.5 and 0.5 so that the mean is closer to zero than if we normalized between zero and one.
The minimal preprocessing pipeline used by Kastrati et al. [1] also addresses this issue. First of all, it is quite likely that bad channels are a source of outliers. Furthermore, they can easily be removed since they anyways carry limited information. This is also exactly what Kastrati et al. [1] did. The second thing they did that helped solve the problem of giant spans was to do high-pass filtering with a cutoff at 0.5 Hz per electrode per recording. This helps since removing the 0 Hz component from a signal is the same as removing its mean. Their approach has the added advantage that far less relative information gets destroyed (compared to our normalization per signal).

5.2.2 High Frequencies

Another problem with EEG data is the high-frequency noise. What is even more problematic for a VAE is that this noise does not follow any patterns. Fortunately, the high-frequencies do not contain any information that is of use to us. We determined that micro saccades are still clearly visible even when we removed the frequencies above 85 Hz.
To take advantage of this insight we used the DFT to move the signals from the time to the frequency domain. This allowed us to single out these higher frequencies during training. We adapted the reconstruction loss which was a simple mean squared error (MSE) between the input signal in split representation and the reconstruction. Anew we computed the same MSE for the lower frequencies (up to the cutoff) but then compute the MSE between the reconstructed higher frequencies and the zero vector. This way we forced the VAE to learn and remove the higher unnecessary higher frequencies.
Curiously, the minimal preprocessing pipeline used by Kastrati et al. [1] did not have a step that is specifically geared towards high frequencies. A channel was removed if the noise got too much. Hence, excessive high-frequency noise can

definitely be a cause for that. But other than that there was not any part of the minimal preprocessing pipeline that would target high-frequency noise. The maximal preprocessing however was able to find the noise. And if ICLabel classified it as such then it would be removed during the maximal preprocessing.

5.2.3 Line Noise

Line noise is very easy to spot in a signal because it is a very nice 50 Hz wave. However, even though it is a very easy concept, it is very difficult to reproduce perfectly in the time domain. For that to happen, a VAE would have to learn to perfectly decide on the output value at 500 different timepoints (= the number of timepoints in one signal in EEGEyeNet). At the same time, this whole noise is captured by two numbers in the frequency domain (the real and imaginary part of the 50 component of the DFT). Consequently, while we moved from the time domain to the frequency domain to better handle the high-frequency noise, it also had the very welcome side effect that the extremely prevalent line noise can be modelled more easily.
Removing line noise is definitely a stronghold of the preprocessing pipelines used by Kastrati et al. [1], for both minimal and maximal preprocessing. First, a channel would be removed if the line noise surpassed a certain level in it. Then they applied the Zapline toolkit - a state-of-the-art tool - to remove the line noise in the remaining electrodes. And lastly, independent components that resemble line noise would be removed if ICLabel classified them as such with high enough confidence.
The example in Fig. 4.1 shows this very nicely. All three depicted channels have line noise. Channel 4 is removed in the first step and then in the second step, we can perfectly see how the very rhythmic 50 Hz noise disappears.

5.2.4 Line Noise and Normalization

One shortcoming of our decisions so far (to use normalization in time domain and to move to the frequency domain then) is that information about other frequencies can leak into the 50 Hz component of the DFT.
For that let us consider an artificial example. Say we have the two signals in the top row of Fig. 5.1. In the frequency domain, they have exactly the same 50 Hz component. If we then normalize them each between -0.5 and 0.5 (bottom row of Fig. 5.1) the amplitudes of the 50 Hz waves change differently for the two. And by extension, the 50 Hz components are no longer the same.
This mechanism theoretically has the power to give the 50 Hz component from the DFT some predictive power that one might not expect at first.

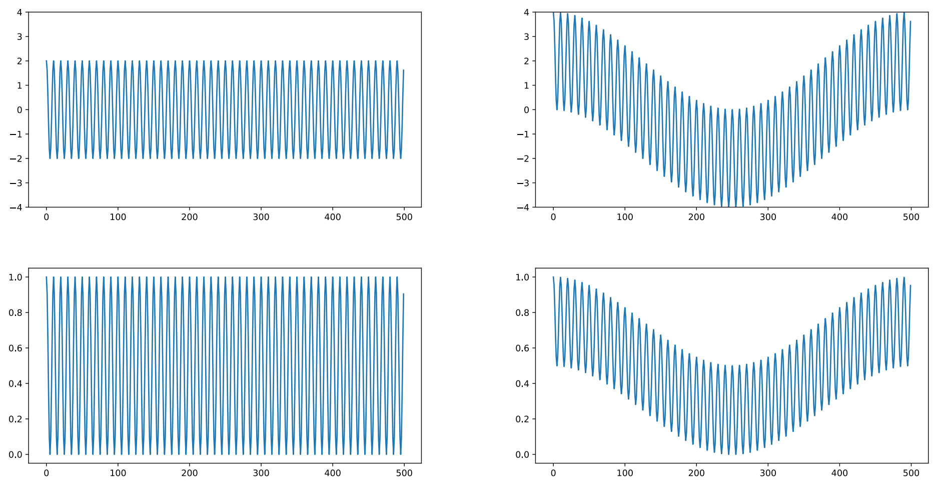


Figure 5.1: Visualisation of how normalization can leak one frequency's info into another one. The 50 Hz component is the same for both signals in the top row. But not anymore after normalization (bottom row).

5.3 Training the VAE

Now we turn our attention to the actual training of the VAEs and what we learned there.

5.3.1 Signal vs Sample

The EEGEyeNet datasets consist of one-second samples of the 128 physical and the one reference electrode, as already mentioned in Subsection 4.1.3. The obvious question was: should a VAE take a flattened sample or individual signals as inputs? Our decision fell on the latter because of several reasons:

1. The input layer is smaller and hence the computational burden can be reduced

2. Reconstructing a signal is a sub-problem of reconstructing a full sample and consequently should be easier

3. The shapes of signals are generally the same across different electrodes. Hence it would make sense to at least share weights for the different signals in a sample. But the even easier option is to use a model that only takes individual signals as input.

4. There are 129 times as many signals as there are samples. Hence, the dataset is larger. Obviously, it is not quite this simple. As mentioned in

the previous point there would be some sort of weight sharing (e.g. via convolutions) if one were to use full samples as individual training points. Even in a model that uses some weight sharing, there are always layers which do not share weights among different parts of the input.

5. Visualising the latent dimensions is easier to do for one signal compared to an entire sample

5.3.2 The Full Pipeline

With the information from the last subsection, we can now give a summary of our full pipeline for the VAE. Each signal is:

1. normalized between -0.5 and 0.5
2. the DFT is applied
3. the result is put into its split representation

And that is it. A very simple pipeline indeed.

5.3.3 Standard Deviations of the Frequencies

We use the MSE for the reconstruction term in the loss of our VAEs. This unfortunately creates a problem with the EEG data. The components of the lower frequencies in the DFT have a much larger standard deviation than the medium and higher ones. This means that the contributions of the lower frequencies to the MSE can dominate the whole error. And hence the VAEs are constantly concentrating on getting those few low frequencies right and forget about the medium ones.
Note that a large mean for a split frequency per se does not yield this problem. Imagine a split frequency (say the real part for the 70 Hz component) had an enormous mean over the dataset but also a tiny standard deviation. The VAEs would simply learn to always approximately predict the mean for that split frequency making its contribution to the MSE small.
Obviously, the choice of dataset and more importantly the preprocessing can exacerbate or mitigate this problem. The normalisation per signal does a comparatively very good job in this. We hypothesise that this is because of the *per signal* part of the preprocessing scheme (as opposed to *per electrode*). For ease of reading, we here discuss standardization but the argumentation can be adapted to normalization.
Removing the mean over a whole electrode will still leave many signals markedly above and below zero. At the same time, the movement within individual signals is comparatively low (there are no or few signals which go several times from

very negative to very positive values). But if the means of many signals can be markedly away from zero while the movement within the signals being comparatively low, then the standard deviation for low frequencies will be very large. And the medium and high frequencies will give components that are often close to zero since there is comparatively little medium and high-frequency movement within the samples.
The normalization per signal makes sure that no individual sample gets comparatively far away from zero in our case. This already mitigates the problem of the large standard deviations of the low frequencies between signals a lot. Furthermore, the normalization stretches signals that are quite flat into the range between -0.5 and 0.5. Likely amplifying the medium and high frequencies in them and hence resulting in larger standard deviations among medium and high frequencies.
However, solely concentrating on the differences in standard deviations among the split frequencies is maybe not the full story. Standardizing per split frequency gives every split frequency the same standard deviation. Yet we had trouble with this scheme. But, this trouble could also be because standardizing per split frequency breaks some of the information between the frequencies. And hence maybe removes patterns that facilitate learning for the VAEs.

5.4 Uncovered Latent Representation

In this section, we take a look at the latent representation and what it allows us to do.

5.4.1 Learnt Latent Representation

Judging from the visualisations of the latent dimensions (see Fig. 5.2 for two examples and see Appendix D for all visualisations) we can conclude that the VAEs learned some nice concepts. For example, every decent VAE always had two latent dimensions that learned the line noise (with a phase shift between the two dimensions).
However, unfortunately, the latent dimensions are not that clean. For example, most of them seem to also encode a little bit of line noise as a side product. Or even the dimensions encoding the line noise have some other information in them (they do not only model the 50 Hz component).

5.4.2 Information Content

We repeat: preprocessing has two goals, (1) to make data easier to work with and (2) to keep important information in the data. Hence, we should evaluate our normalization and VAEs against these requirements if we want to use them as a

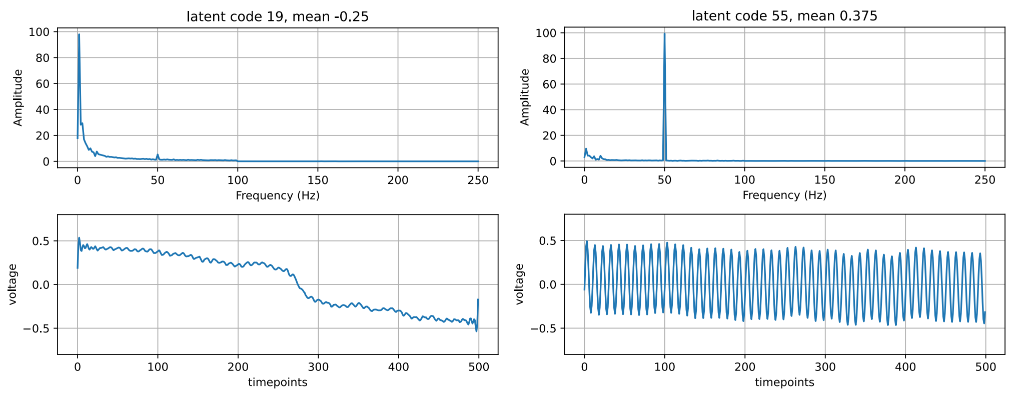


Figure 5.2: Visualisations of two latent dimensions from the annealed β-TCVAE. The dimension on the left is important for eye movement tasks while the right one seems to encode line noise.

preprocessing scheme for raw EEG data. In short, our approach transforms the raw EEG data in a format that even small models can work with but it removes important information for some tasks.

Using our pipeline and the encoder of the trained VAE as preprocessing allowed us to use the raw data and a tiny ResNet to train on tasks where before only the minimally preprocessed data and much larger models were used. However, this only worked when the task did not require relative information between the different signals within a sample. Yet, this was expected because the preprocessing per signal has exactly said drawback.

Chapter 6

Experiments

6.1 Sanity Checks

The goal of these experiments was to make a sanity check for different preprocessing schemes by letting the VAEs overfit on a small training dataset. The desired takeaways were which preprocessing schemes make it easier for the VAEs to learn.
In the experiment, we sampled ten samples (yielding 1290 signals) from the minimally preprocessed direction dataset. 10% of the signals were used as a validations set. We used seven kinds of preprocessing:

1. Standardize per electrode in time domain
2. Normalize per electrode in time domain
3. Standardize per electrode in frequency domain
4. Normalize per electrode in frequency domain
5. Standardize per split frequency in frequency domain
6. Normalize per split frequency in frequency domain
7. Standardize per electrode in frequency domain with complex logarithm applied to DFT responses (i.e. before standardizing)

If the preprocessing happens in the time domain we applied the DFT to the signals and put them in the split representation. Otherwise, the DFT was applied first, followed by putting the signals into the split representation, and then the preprocessing (the obvious exception is the complex logarithm from preprocessing number seven). All statistics (means, standard deviations, minimums, spans) were computed using only the training set.
For each of the seven schemes, we trained a β-TCVAE with 70 latent dimensions. The encoder consisted of 3 ResNet blocks (with dimensions 400, 300, and 200 respectively) with two linear layers to produce the latent statistics from the output

of the last block and a decoder which consisted of 3 ResNet blocks (with dimensions 200, 300, 400) with a final linear layer to get back to the input dimension of the VAE. The VAEs had 502 input neurons which equated to one signal in split representation. The weight for the reconstruction term was set very high ($\mu = 1'000'000$) and the one for the total correlation low ($\beta = 3$).
We used the ADAM optimizer [14] with a learning rate of 0.002 (the other parameters were set to the defaults from PyTorch version 1.11.0) and batch size 128. All but the first two schemes were trained for 25 epochs. The two first schemes were trained for 100 epochs (25 epochs were too few for their loss curves to flatten out). The reconstruction loss was the mean squared error (averaged within a batch). For the split frequencies from 100 Hz on we set the target to zero such that the VAE learned to not reconstruct the higher frequencies.

6.2 Full Dataset

The goal of these experiments was to tune β-TCVAEs to reconstruct raw signals well while keeping the latent dimensions uncorrelated.
We used the raw direction and raw position datasets and combined them into one large dataset. Thereof we randomly selected 10% of the signals as a validation set. The different preprocessing schemes we tried are:

1. Standardize per electrode in time domain
2. Normalize per electrode in time domain
3. Standardize per electrode in frequency domain
4. Normalize per electrode in frequency domain
5. Standardize per split frequency in frequency domain
6. Normalize per split frequency in frequency domain
7. Normalize (between -0.5 and 0.5) per signal in time domain

For the standardize per electrode in frequency domain and standardize per frequency in frequency domain schemes we also tried the same setup but with the minimally preprocessed version of the data. Hence, in total, we had nine combinations of data and preprocessing schemes to tune.
The remainder of the setup (order of DFT and preprocessing, architecture, optimizer, hyperparameters, and so on) are the same as for the Sanity Checks (Subsection 6.1) with a few exceptions:

1. For the last preprocessing scheme we used $\mu = 400$ and $\beta = 10$, as well as $\mu = 800$ and $\beta = 10$. For the others we used $\mu = 400$ and $\beta = 10$ as starting points.

2. We trained the models for six to eight epochs, except for the models for the last preprocessing (normalize per signal) where we increased the number of epochs up to 15 in some cases.

3. For all but the last preprocessing we trained four to five generations. We increased the weight μ in the next generation whenever the reconstructions of a configuration in one generation did not look satisfactory. To further put importance on the reconstruction we would also lower the β in many cases.

Additionally, for the normalization per signal scheme, we also trained annealed β-TCVAE. The hyperparameters for the loss were $\beta = 3, \mu = 177.76, \gamma = 5'000, C = -0.2$ at the start, $C = 1.25$ at the end with this value being reached after 60% of the training steps.

6.3 Feature Statistics

The goal of this experiment was to investigate the means and standard deviations of the different features that were fed to the VAEs in Sections 6.1 and 6.2. Regarding Section 6.1 this meant preprocessing the signals of the ten samples with the seven schemes presented and then computing the means and standard deviations of each of the split frequencies. The parallel experiments were also made for Section 6.2 over the combined raw datasets.

6.4 Downstream Tasks

The goal of these experiments was to see if a successfully trained VAE (on the raw dataset) could provide a latent representation of the raw signals that would allow us to achieve decent results on the Angle, Amplitude, and Absolute Position tasks from Kastrati et al. [1].
For that, we focused on the encoder of the trained annealed β-TCVAE introduced in Section 6.2. We used said encoder to put the raw direction and raw position datasets into their respective latent representations (we term them **latent direction dataset** and **latent position dataset**). A sample in these two sets had the shape 129×26. Refer to Appendix A for more information on how exactly the latent representation was computed.
For all experiments in this section, we split the data along subjects into 70% training data, 15% validation data, and 15% test data. We created five different splits for each of the two datasets. For more details refer to Appendix B.
The models themselves would receive samples in their latent representations flattened to a vector (hence, there are 3354 input neurons). One single ResNet block with dimension 1'500 would follow before a final linear layer with two output

neurons. For the Amplitude task only the first output neuron would be utilized, the Angle task only used the second one, and the Absolute Position task used both (to predict the x and y coordinates of the pixel the subject was fixating on). The loss was the mean squared error for all three tasks[1]. We used the ADAM optimizer [14] with a learning rate of 0.01 (the other parameters were set to the defaults from PyTorch version 1.11.0), the batch size was 256, and we trained each model for 50 epochs on the training dataset.
We trained a total of 15 models, five models for the Angle task (1 per latent direction dataset split), five models for the Amplitude task (1 per latent direction dataset split), and five models for the Absolute Position task (1 per latent position dataset split). For each of these 15 runs, we saved the version of the model which had the lowest validation loss (the validation loss was measured at the end of each epoch).

6.5 Perturbation Importance

The goal of this set of experiments was to gain insight into what latent dimensions and electrodes are important for the regression models, what latent dimensions are important for the VAEs, and how these compare. We explain each in their own paragraph.
Regression tasks: We take two trained models from the experiments in Section 6.4. One each for the Angle and Amplitude task. Since there were five models for each task to choose from we decided to take the ones with the median test scores for their respective tasks. We then used their respective test data (in latent representation) to compute the perturbation importance for the 129 electrodes and the 26 latent dimensions each. The change in metrics proposed by Kastrati et al. [1] was used to quantify importance (see also Section 7.4). We furthermore ran each experiment ten times and averaged their results since the results between different runs of the perturbation feature importance method can vary greatly. There are more details on the permutation in Appendix C. Note: we left the Absolute Position task out of this consideration since its performance was bad.
VAEs: We used the trained annealed β-TCVAE from Section 6.2 and computed the importance of the different latent dimensions. Specifically, we measured how the mean squared error of the reconstructions changed (both in frequency and time domain). To do so, we combined the raw direction and raw position datasets into one large dataset and applied the same preprocessing to the data we did before training the VAE. All signals in the dataset were put into their latent representations using the encoder of the VAE (again removing all the latent di-

[1]Note that this was a bit of an odd choice for the Angle task since an angle of 0.1 radians and 6.2 radians are very close on the unit circle but would have a large squared error. We also tried the benchmark metric introduced by Kastrati et al. [1] as a loss but this yielded slightly worse results so we stuck with the mean squared error.

mensions that do not carry information, see Appendix A for more information). The data was now in a state where it could be fed to the decoder and so we could apply textbook feature permutation importance using the just described data and the decoder. As a metric, we chose the mean squared error between the preprocessed signals and the reconstructions. Since the VAE was trained to not reconstruct frequencies starting from 100 Hz we set those to zero in the normalized signals before computing the mean squared error. Because of the computational burden we only ran and averaged the results over three runs.
Comparing importance: The goal here was to compare the importance of different latent dimensions for the VAE and the regression models. For that, we reused the values computed for the regression models but reran the VAE importance on the raw direction data (since the Amplitude and Angle tasks trained on that). Once computed, we sorted the different latent dimensions by importance for each Amplitude, Angle, and VAE. Afterwards, we computed for each latent dimension the difference between the ranks for the pairings Amplitude with VAE and Angle with VAE. So say latent dimension 21 was deemed to be the third most important for the Amplitude model but only the 13th most important for the VAE then its rank difference would be 10. Latent dimensions with very positive values would be much more important for a regression task than the VAE and vice versa.

6.6 Angle and Amplitude from Line Noise

In this experiment, we wanted to check if the latent dimensions encoding the line noise (according to the visualisations of the latent dimensions) carried some information that is usable for the Angle or Amplitude task. To test this we used the exact same setup as in Section 6.4 but instead of only removing the latent dimensions that do not carry any information we removed all latent dimensions but the two which encode the line noise.

Chapter 7

Results

7.1 Sanity Checks

The following preprocessings produced models that picked up some patterns but the reconstruction quality is a bit under the level expected for a sanity check:

- Standardize per electrode in time domain
- Normalize per electrode in time domain
- Standardize per electrode in frequency domain
- Normalize per split frequency in frequency domain
- Standardize per electrode in frequency domain with complex logarithm applied to DFT responses (i.e. before standardizing)

Remember that we only looked at training samples here.
The normalization per electrode in frequency domain scheme yielded very poor results (see example in Fig. 7.1 left). And the standardizing per split frequency in frequency domain yielded the nicest reconstructions (see example in Fig. 7.1 right). We provide more plots in Appendix F.

7.2 Full Dataset

The following preprocessing schemes always yielded bad reconstructions no matter the μ we chose (no matter the dataset considered):

- Standardize per electrode in time domain
- Normalize per electrode in time domain
- Standardize per electrode in frequency domain

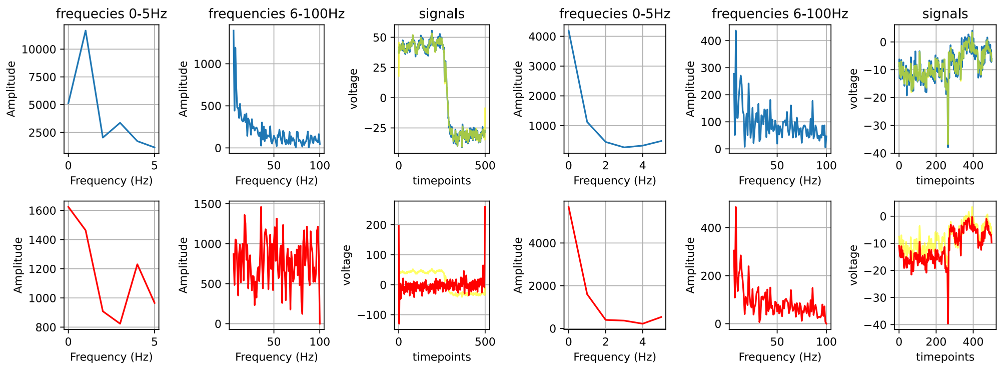


Figure 7.1: The left is a reconstruction from a VAE that was trained on data preprocessed with normalizing per electrode in frequency domain and on the right is a reconstruction from a VAE based on standardizing per split frequency in frequency domain. The blue curves are the actual input samples and the red curves are the reconstructions by the respective VAE. The yellow curve shows the input signal with all frequencies from 100 Hz on removed. We show the signals and reconstructions in the frequency domain (the left and middle subplots in the two plots) and in the time domain. The preprocessings were reverted for the time domain plots.

- Normalize per electrode in frequency domain
- Standardize per split frequency in frequency domain
- Normalize per split frequency in frequency domain

The smallest μ we had in a final generation loss was 200'000, so a lot larger than for the β-TCVAE and annealed β-TCVAE with normalization per signal which yielded models with decent reconstructions (see Fig. 7.2 left). The annealed TCVAE and the TCVAE with $\mu = 400$ also have nicely uncorrelated latent dimensions (see Fig. 7.2 left) while the TCVAE with $\mu = 800$ starts to show first higher correlations between latent dimensions.
We show more plots with reconstructions in Appendix G and more correlation plots in Appendix H.

7.3 Feature Statistics

A special case is the standardization per split frequency in frequency domain. Obviously, that scheme generated very nice input statistics. Setting this aside, one eye-catching result is that among the schemes applied to the raw dataset only the normalization per signal produced a standard deviation plot that is similar to the ones on the sanity check data (see Fig. 7.3). The other schemes

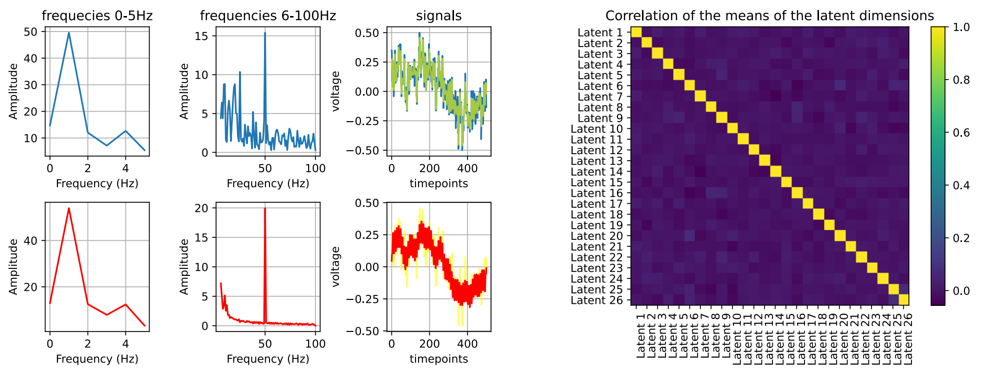


Figure 7.2: **Left**: a reconstruction from the annealed β-TCVAE. The blue curves are the actual input sample, and the red curve is the reconstruction. The yellow curve shows the input signal with all frequencies from 100 Hz on removed. We show the signal and reconstruction in the frequency domain (the left and middle subplots) and in the time domain (the right subplots). The preprocessings were not reverted for the plots. **Right**: correlation matrix of the combined latent dataset of raw direction and position datasets.

produced (besides the obvious exception) all very aggressive hockey stick plots for the standard deviation, to the point where many of the plots look like they have a 90-degree angle right at the beginning.

7.4 Downstream Tasks

As a metric for the models on their respective test sets, we used the same versions of the root mean squared error as Kastrati et al. [1] did. We add our implementations in Appendix E. The results for the tasks can be found in Table 7.1.

	RMSE (Kastrati et al.)	RMSE (ours)
Amplitude	61.4	71.38 ± 4.54
Angle	0.33	0.49 ± 0.06
Abs. Position	140.4	236.02 ± 3.55

Table 7.1: The test set scores averaged over the five models (one per data split) for each task with standard deviations after the $\pm$. The units for the Amplitude and Absolute Position tasks are pixels (divide by two for millimetres) and for the Angle task, it is radians. For comparison, we included the **best** scores for the different tasks from the baselines presented in Kastrati et al. [1]. Note that their results were obtained on the minimally preprocessed data while ours came from the raw data.

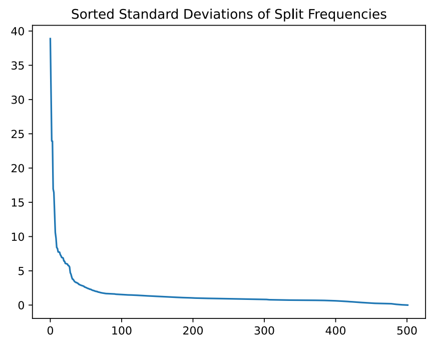


Figure 7.3: The sorted standard deviations of the split frequencies after applying our normalization per signal in time domain scheme to the combined raw direction and raw position datasets.

7.5 Perturbation Importance

Amplitude Task: The sixth latent dimension (among the dimensions that carried information) was clearly the most important latent dimension for the model predictions according to the experiment (see Fig. 7.4 for visualisations). The RMSE metric went up by an average of 121.73 pixels compared to the baseline (which was 70.59 pixels). The most important electrode is the 126th with an increase in RMSE of 7.5. Other electrodes were close behind.

Angle Task: The sixth latent dimension was again the most important one according to the experiment. Permuting its values add 0.46 radian on average to the baseline RMSE of 0.46 radian. The 127th electrode was deemed the most important one (permuting it added an average of 0.05 radian to the baseline). However, all electrodes had more or less the same score in this experiment so it is a bit difficult to say which are more important than others.

VAE: The sixth latent dimension is also the most important one for VAE reconstructions. Perturbing it added 0.03 to the MSE baseline of 0.25 for the time domain and 7.86 to the 1.09 baseline in the frequency domain. The ranking for the time domain and the ranking for the frequency domain generally agree. Meaning that a latent dimension that is very important for the reconstruction in the frequency domain is also very important for the reconstruction in the time domain, et vice versa.

Rank Difference: For the VAE reconstruction and the Amplitude task there is more or less a consensus about which latent dimensions are the most important ones. However, there is quite a curious outlier for the Angle task where the 21st latent (among the ones carrying information) was quite important for the VAE but is the least important for the Angle task. You find the full results in Appendix I.4. The visualisation of this curious 21st latent can be found in Appendix D.

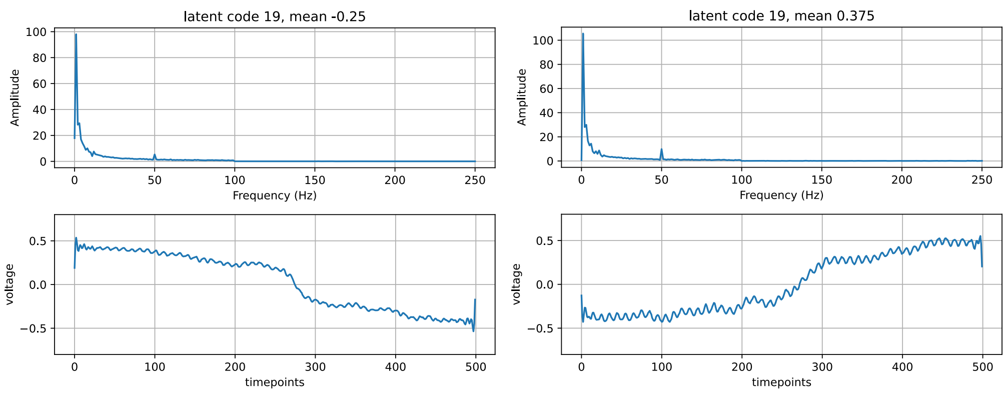


Figure 7.4: Two visualisations of the sixth latent dimension (i.e. index five) that carries information from the annealed β-TCVAE.

The full tables take up a lot of space and hence we moved them to Appendix I.

7.6 Angle and Amplitude from Line Noise

We used the same test scores as in Section 7.4. Table 7.2 shows the results.

	RMSE (Kastrati et al. - naive)	RMSE (ours - line noise)
Amplitude	149.4	162.03 ± 24.07
Angle	1.90	1.67 ± 0.03
Abs. Position	246.6	244.77 ± 0.58

Table 7.2: The test set scores averaged over the five data splits for each task with standard deviations after the $\pm$. The units for the Amplitude and Absolute Position tasks are pixels (divide by two for millimetres) and for the Angle task, it is radians. For comparison, we included the **naive** baselines for the different tasks from the baselines presented in Kastrati et al. [1].

Chapter 8

Conclusion

In this work we demonstrated potential for a data-driven preprocessing of raw EEG data. We evaluated different preprocessing schemes for the raw EEG data and reflected on their advantages and disadvantages. The majority of them could be used to train VAEs to produce good reconstructions on training data in a sanity check. However, only one scheme could perform well on the full raw dataset, the normalization per signal. We investigated the feature statistics the different preprocessing schemes produced and showed that normalizing per signal produces the best standard deviation statistics among the considered schemes.
We used normalization per signal to successfully train a pair of VAEs that not only produced decent reconstructions but also kept their latent dimensions uncorrelated. Investigating the latent space showed the capability of the VAE to learn simple concepts from EEG data such as the line noise. At the same time, we also noticed that these concepts were not clean. The latent dimensions encoding the line noise, for example, also contained other movement. And other latent dimensions usually showed also some line noise in their visualisations.
We used the annealed β-TCVAE's encoder as the preprocessing step for the raw EEG data from the EEGEyeNet dataset. The resulting data could then be used to achieve comparatively decent results on the Amplitude and Angle task. These tasks have never before been successfully attempted with the raw EEG data. However, the performance on the Absolute Position task was poor. The cause of this is likely that normalizing per signal removes important information between the different signal of a sample; information that is important for the Absolute Position task.

Bibliography

[1] A. Kastrati, M. Plomecka, D. Pascual Ortiz, L. Wolf, V. Gillioz, R. Wattenhofer, and N. Langer, "Eegeyenet: a simultaneous electroencephalography and eye-tracking dataset and benchmark for eye movement prediction," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung, Eds., vol. 1, 2021. [Online]. Available: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/a3c65c2974270fd093ee8a9bf8ae7d0b-Paper-round1.pdf

[2] A. Pedroni, A. Bahreini, and N. Langer, "Automagic: Standardized preprocessing of big eeg data," *NeuroImage*, vol. 200, pp. 460–473, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811919305439

[3] T.-W. Lee, *Independent Component Analysis*, 1998, pp. 27–66.

[4] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," 2014. [Online]. Available: https://arxiv.org/abs/1401.4082

[5] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013. [Online]. Available: https://arxiv.org/abs/1312.6114

[6] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=Sy2fzU9gl

[7] R. T. Q. Chen, X. Li, R. Grosse, and D. Duvenaud, "Isolating sources of disentanglement in variational autoencoders," 2018. [Online]. Available: https://arxiv.org/abs/1802.04942

[8] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in -vae," 2018. [Online]. Available: https://arxiv.org/abs/1804.03599

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[10] L. Breiman, “Random forests,” *Machine Learning*, vol. 25, pp. 5–32, 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[11] C. Molnar, *Interpretable Machine Learning*, 2nd ed., 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book

[12] A. de Cheveigné, “Zapline: A simple and effective method to remove power line artifacts,” *NeuroImage*, vol. 207, p. 116356, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811919309474

[13] L. Pion-Tonachini, K. Kreutz-Delgado, and S. Makeig, “Iclabel: An automated electroencephalographic independent component classifier, dataset, and website,” *NeuroImage*, vol. 198, pp. 181–197, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811919304185

[14] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

Appendix A

Latent Dataset

The latent representation of a dataset in our context is (in short) the latent representation of the signals in a dataset induced by a VAE where only latent dimensions are kept which pass a certain threshold (= at least 10% of the standard deviations for said latent dimension are below 0.5 on a random batch with 512 signals).
What follows is a more thorough explanation. Remember a sample consists of 129 signals, which in turn are vectors with 500 entries (the EEG data).To produce a latent dataset, we want to use a trained VAE to take each signal and replace it with the latent representation the VAE's encoder computes (i.e. the means it computes). However, there are two further complications.
First, as we can see in Fig. A.1, a VAE's encoder might compute the same values for some latent dimensions even for different inputs. Hence, that latent dimension does not encode any variability in the data. As a criterion to filter these dimensions we used the standard deviation the encoder computed for each signal. Specifically, for each latent dimensions, we checked if more than 10% of the computed standard deviations (over a random batch of 512 signals) were below 0.5. If a latent dimension met this criterion we kept it otherwise we removed said latent dimension. For example, in Fig. A.1 we would have removed the latent dimension with index zero but would have kept the latent dimension with index four.
Second, we had problems with the batch normalization layers in our VAEs. If a PyTorch module contains such layers then the its behaviour is different in training and evaluation mode. In the former, the batch normalization mode actually computes the batch's means and standard deviations. In the latter, this process is replaced by statistics computed during training. This led to problems so we decided to keep the trained VAEs in training mode. But we needed to be careful since the behaviour of batch normalization layers now depended on the batches themselves. We paid attention to two main issues:

1. Signals from different sets (training, validation, and test) cannot be in the same batch as this could leak information between sets.
2. Batches should have a nice variety of signals. We would run the risk that

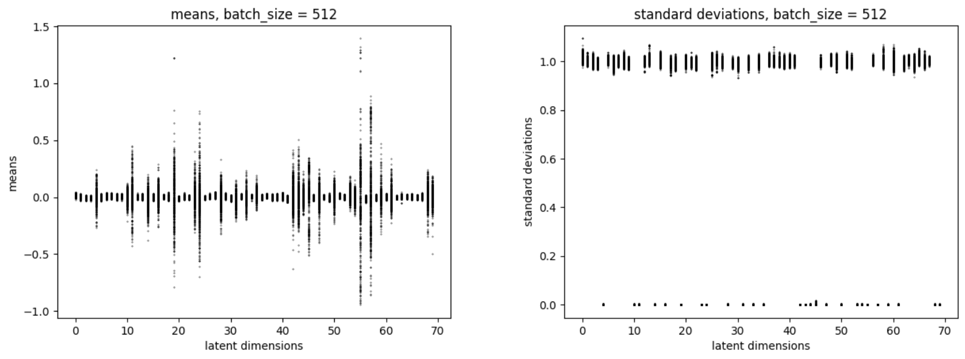


Figure A.1: The means and standard deviations computed by the encoder of the annealed β-VAE trained for Section 6.2 for a batch with 512 random signals from the combined raw direction and raw position datasets.

very similar samples would be in the same batches if we fed the signals into the VAEs in the same order as we read them from the datasets. Hence, the statistics in the normalization layers would not be representative of the dynamics encountered during the training of the VAE, where the data was randomized. The obvious solution is to also shuffle the signals when creating the latent datasets.

Appendix B

Splitting the Data for the Downstream Tasks

In Section 6.4 we split the data along subjects. Unfortunately the number of samples per subject is not balanced in both the direction and the position dataset. This made a clean 70-15-15 split between the training, validation, and test sets impossible. Hence, instead of splitting the samples along this ratio, we randomly split the subjects - which also induces a splitting of the samples (if a subject is in the test set then all his/her samples are also in the test set). We then tallied up how many samples were in the different sets and how much this differs from the perfect 70-15-15 split. Specifically, for each set we take the absolute difference and add these three numbers up. For example, say we had 1'000 samples in our dataset. A perfect split would then be 700-150-150. However, say by randomly splitting the individuals it so happened that some of the subjects with a more than average number of samples land in the validation set. The three sets might then have something like 679, 206, and 115 samples. Giving a difference of 112 $(= |700 - 679| + |150 - 206| + |150 - 115|)$.
For a "split" as mentioned in Section 6.4 we made 20 random splittings of subjects and chose the one with the smallest difference to the perfect split as the actually used split.

Appendix C

Permuting Electrodes or Latent Dimensions in Latent Datasets

There are two important parts to understanding how we permuted the latent data for the permutation feature importance in the regression models.
First, remember the shape of a sample in the latent dataset we used. It was 129×26. Meaning the 129 signals of the original sample were condensed into 26 values using a VAE. Hence, each column of such a sample corresponds to the outputs of one latent dimension of the encoder of the VAE. Now, the regression models take the flattened version of such a sample as input which means that the different values stemming from the same latent dimension are suddenly no longer a nice column of a matrix but spread over a long vector. Hence, if we wanted to assess the importance of a latent dimension then we need to make sure to permute all features in said vector that originate from the same latent dimension. The same concept holds for the electrodes.
Second, we only permuted values along the combination of electrode and latent dimension. This is most easily explained with a plot which you find in Fig. C.1. We only permuted values within each of the different shades of blue. Although all the squares shaded in blueish colors visualise a value that came from the same latent dimension of a VAE, we did not permute across the different columns (= different shades). This decision was taken for ease of implementation. The analog also holds for permutations per electrode. There the different shades of blue would also be along the vertical but all the columns between two bold lines would be blue.

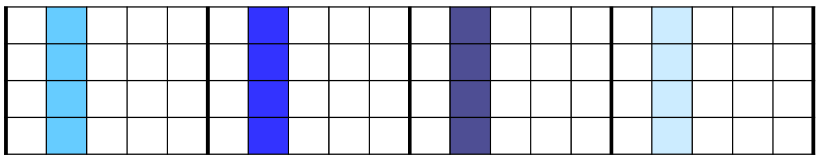


Figure C.1: In this toy example we have four samples (= rows), four electrodes (electrodes are separated by the bold vertical lines), and the encoder used to create these latent samples had five latent dimensions that carry information (visualised by the 5 squares per electrode). The blueish colors visualise the values that come from the same latent dimension from the VAE - in this particular case from the second latent dimension.

Appendix D

Visualisations of Latent Dimensions

Throughout this work we commonly used the annealed β-TCVAE described in Section 6.2. Hence, we show here all the visualisations of the latent dimensions of said VAE. Section 2.2 introduces how to visualise a latent dimension. We used the values $-0.5, -0.375, -0.25, -0.125, 0., 0.125, 0.25, 0.375, 0.5$ for the means of the latent dimension to visualise in this chapter. Note, we only visualise the latent dimensions that actually carry information. Each subplot consists of the reconstruction by the decoder in frequency domain (the amplitudes are plotted) and in time domain (note that the normalization per signal between -0.5 and 0.5 is not reverted).

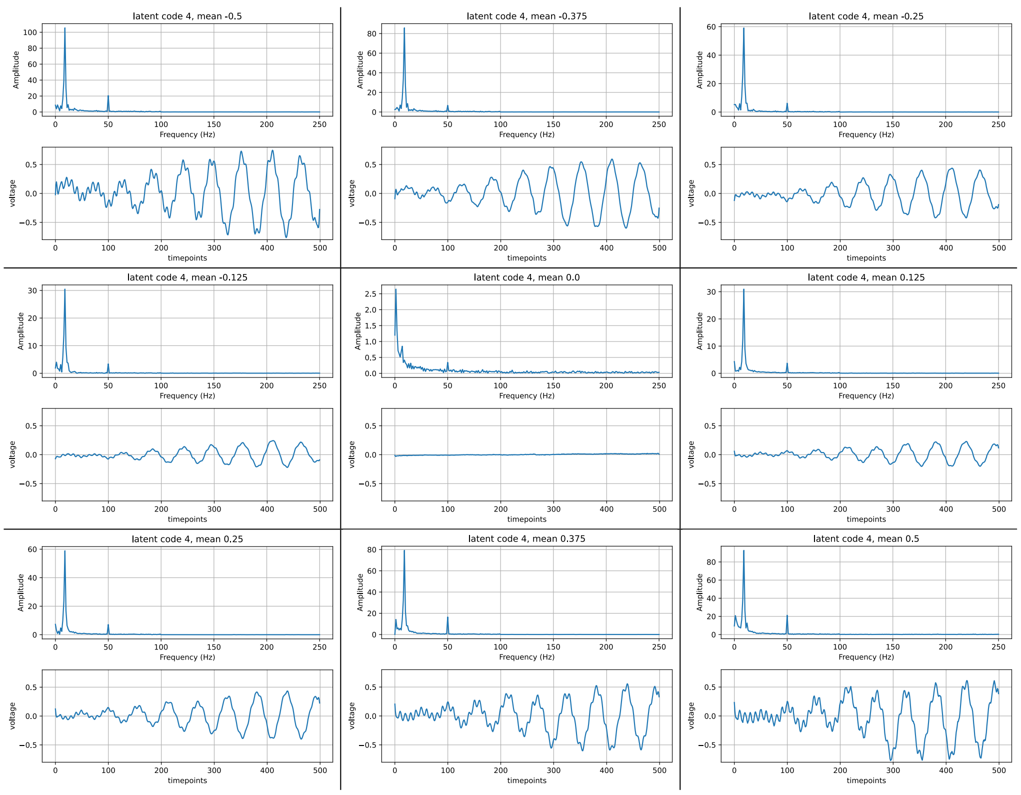


Figure D.1: Visualisation of the latent with index 4 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 0 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

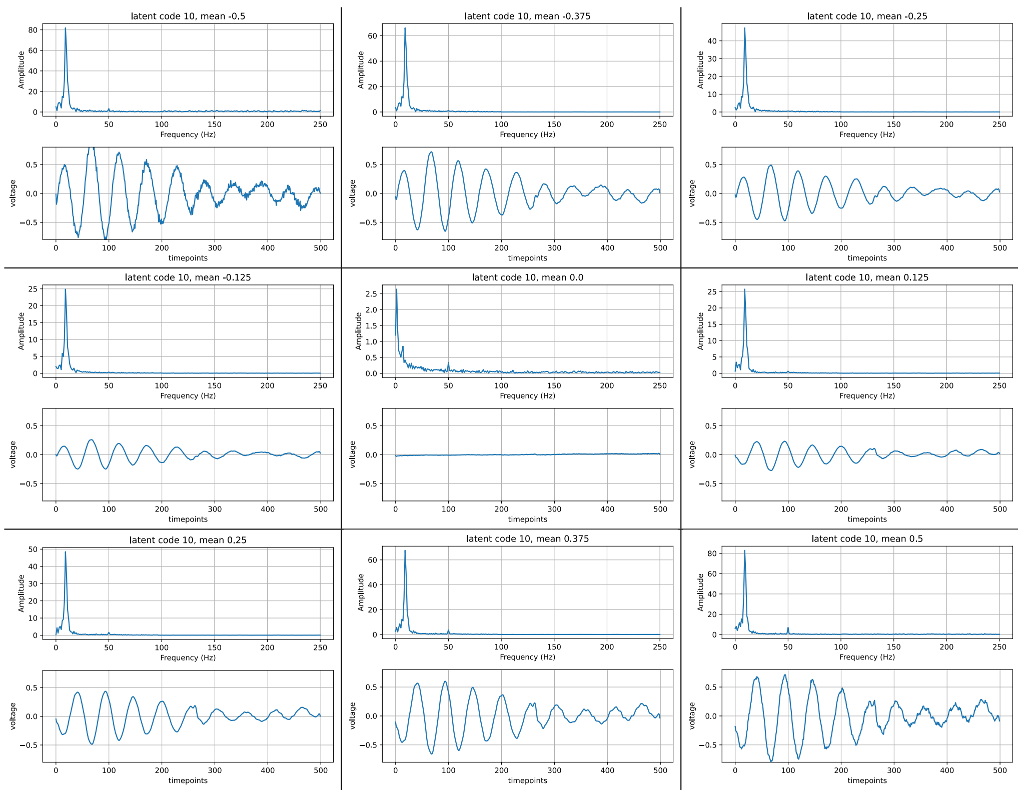


Figure D.2: Visualisation of the latent with index 10 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 1 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

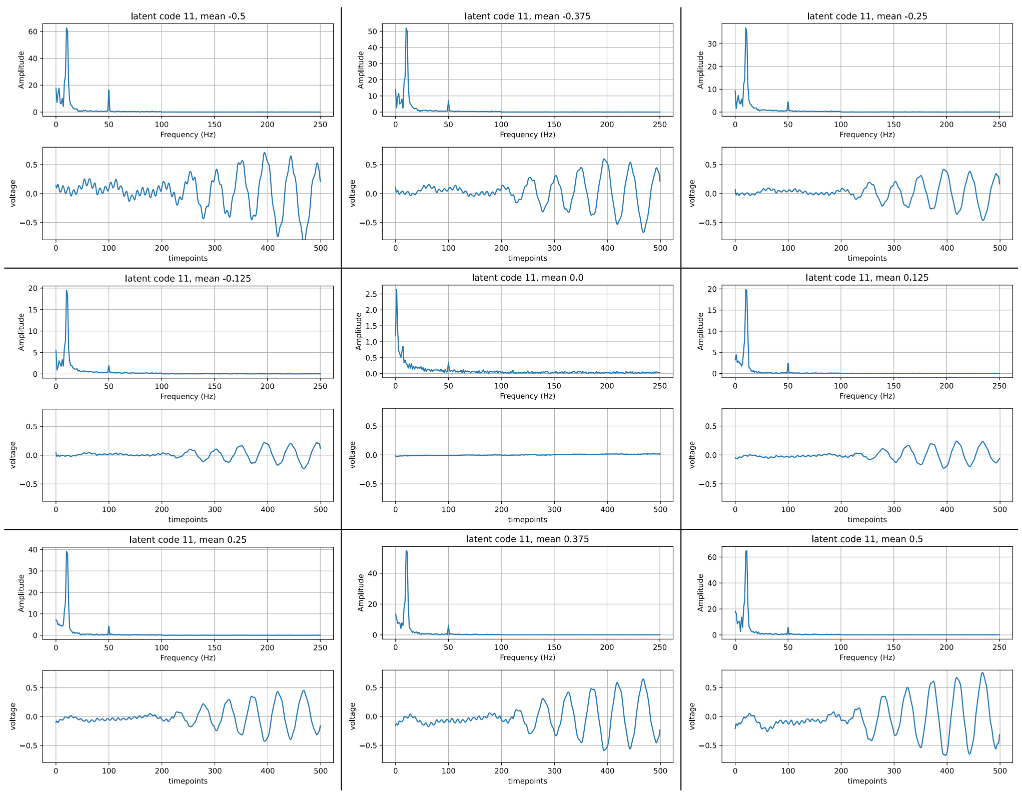


Figure D.3: Visualisation of the latent with index 11 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 2 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

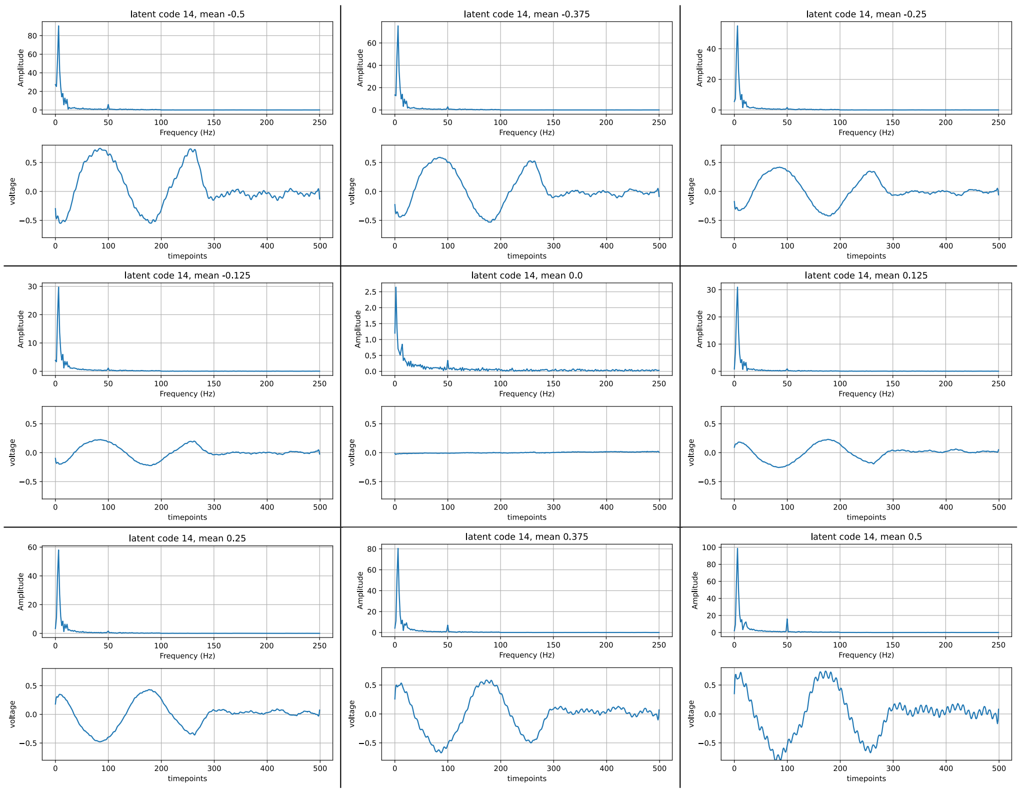


Figure D.4: Visualisation of the latent with index 14 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 3 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

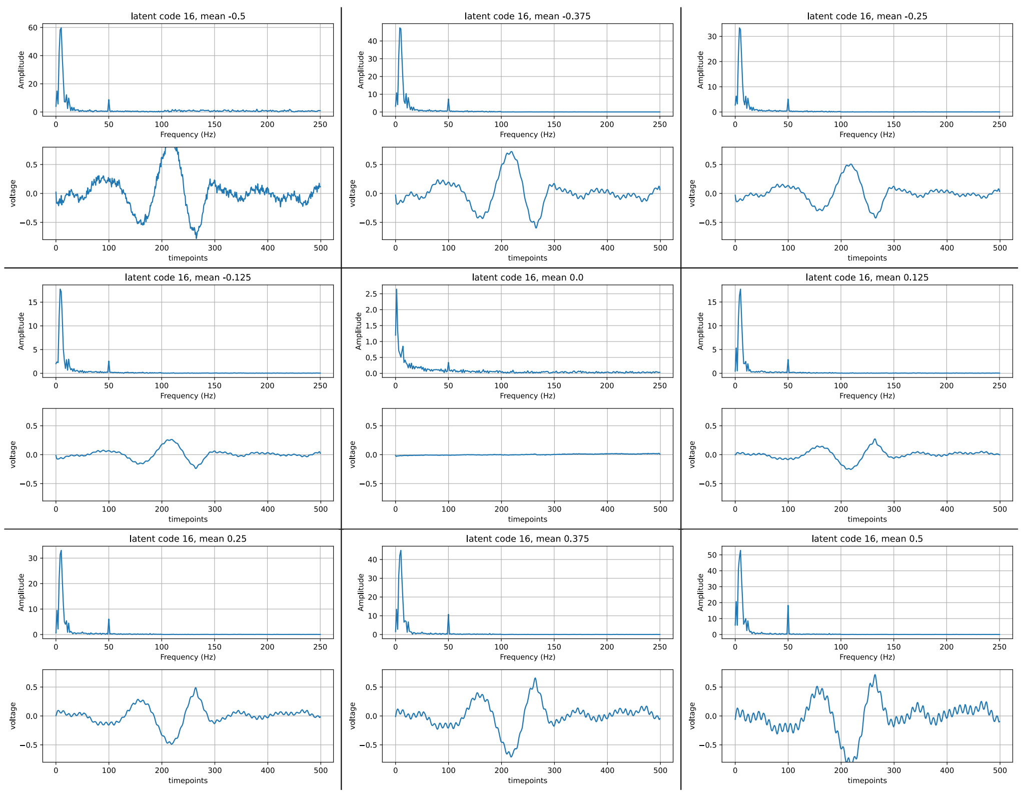


Figure D.5: Visualisation of the latent with index 16 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 4 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

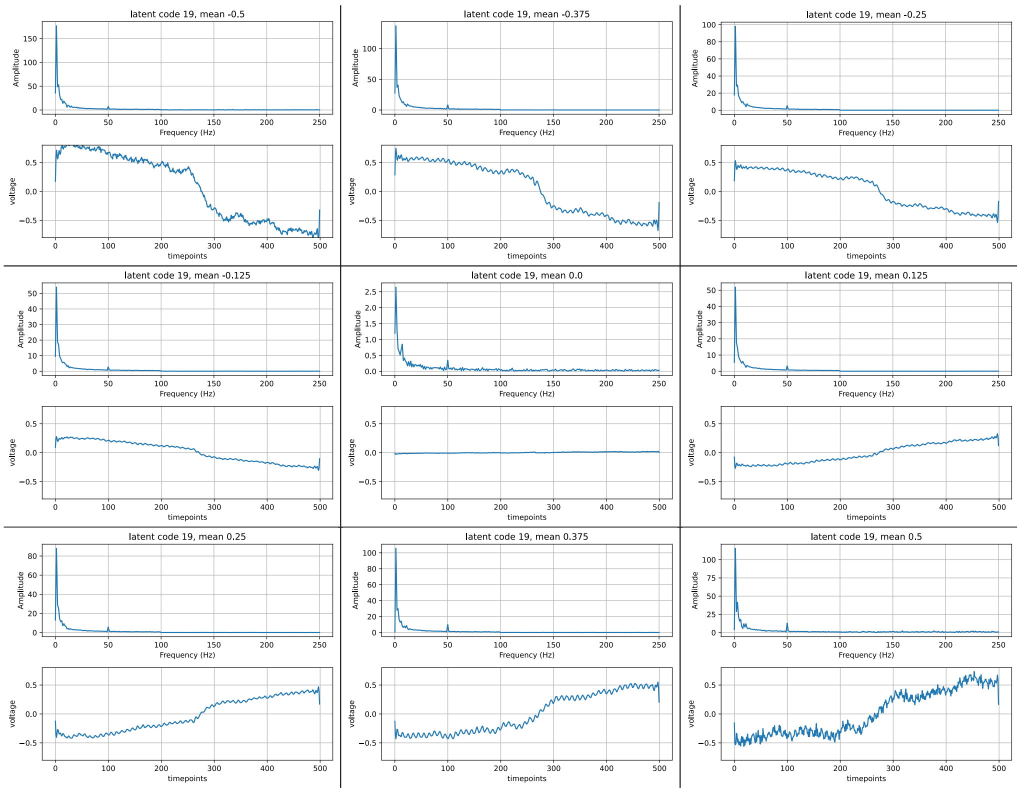


Figure D.6: Visualisation of the latent with index 19 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 5 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

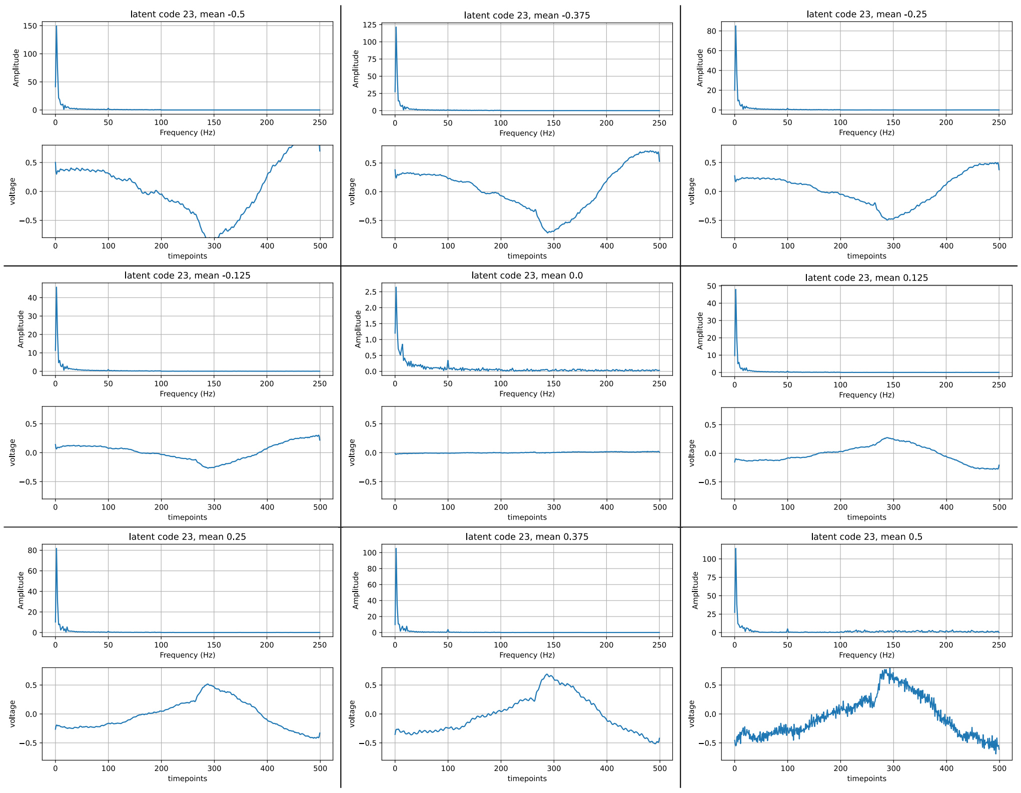


Figure D.7: Visualisation of the latent with index 23 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 6 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

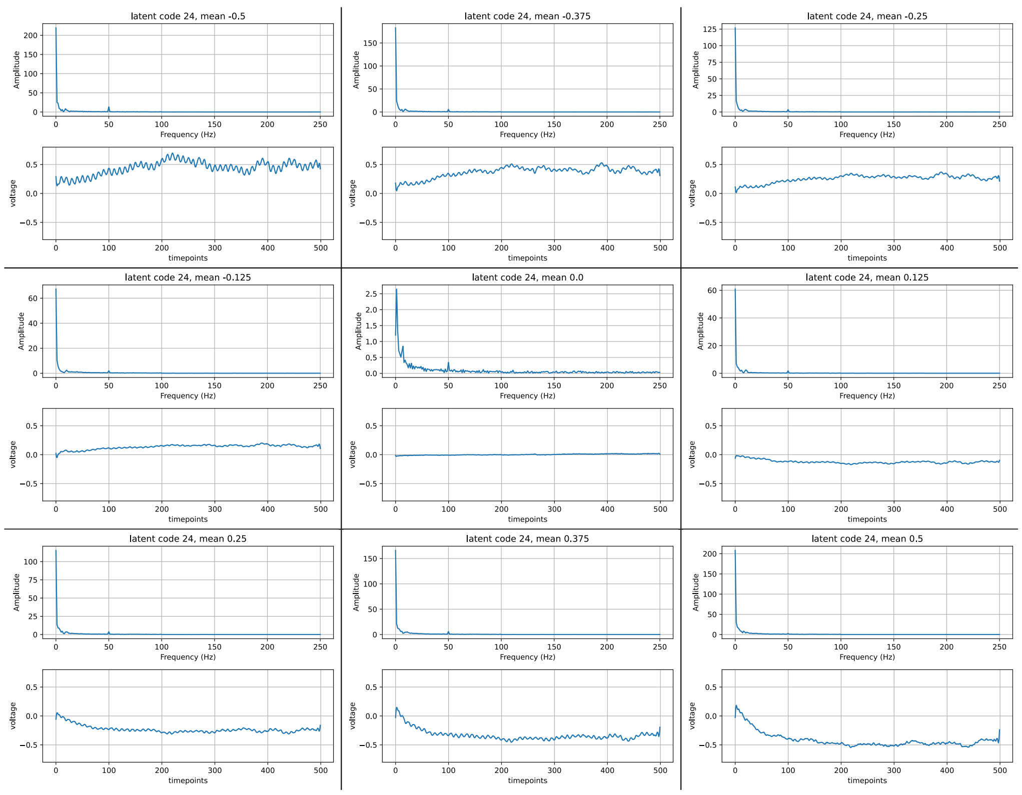


Figure D.8: Visualisation of the latent with index 24 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 7 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

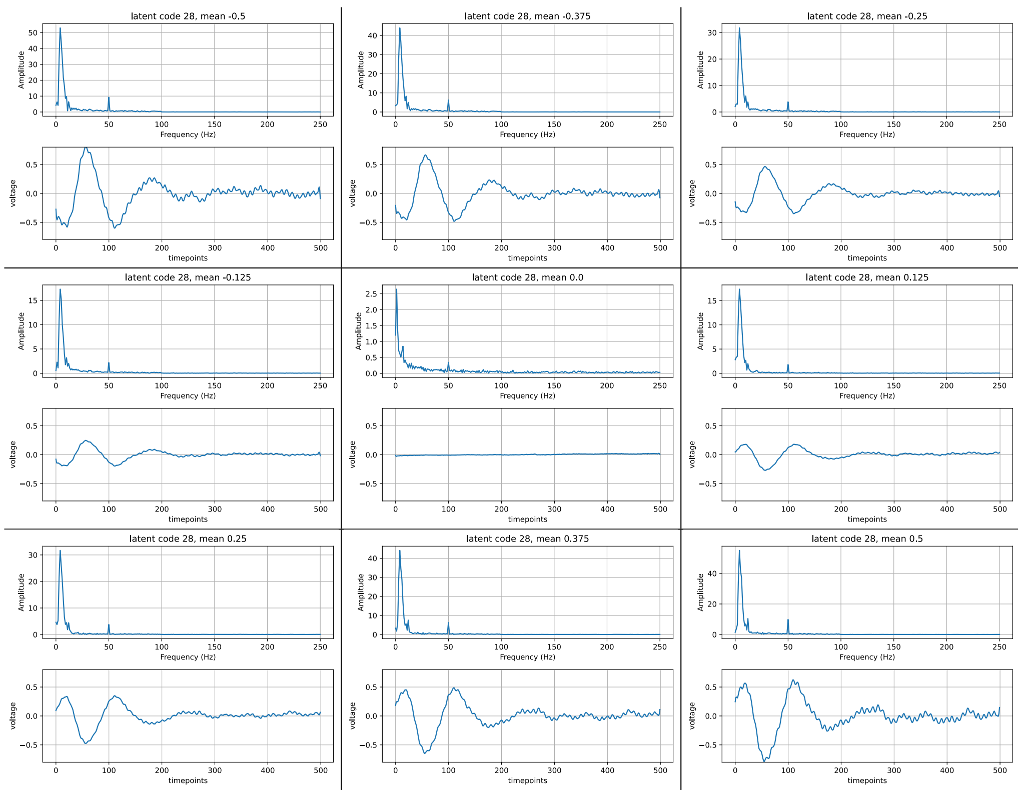


Figure D.9: Visualisation of the latent with index 28 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 8 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

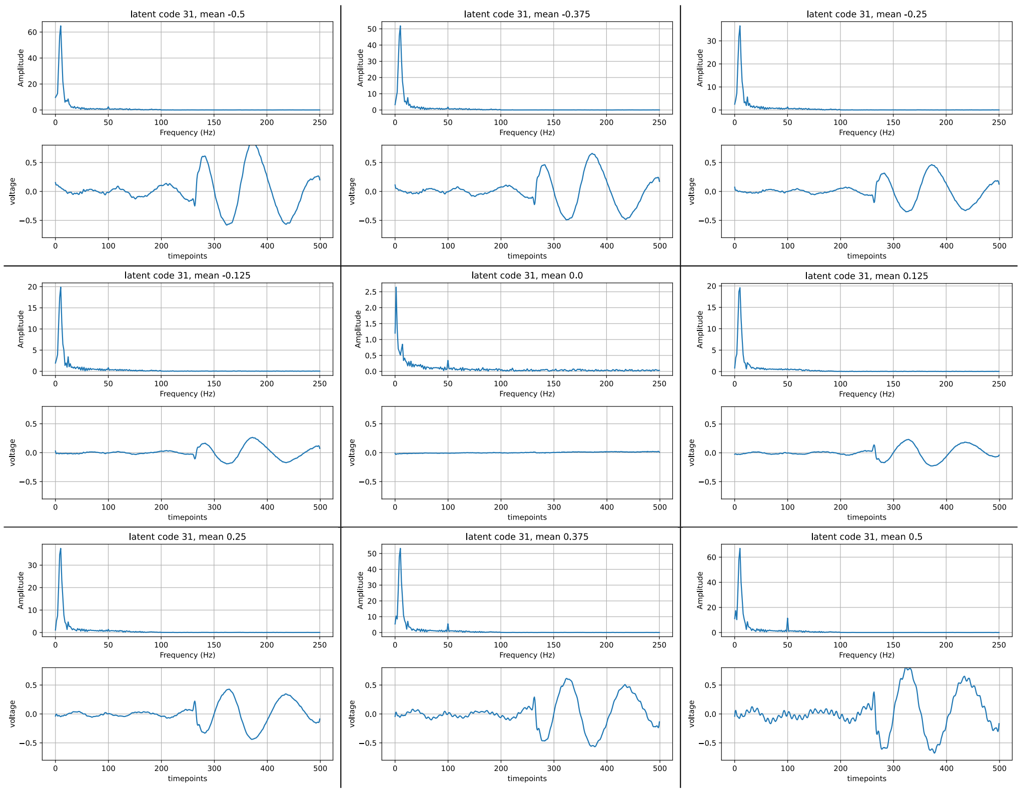


Figure D.10: Visualisation of the latent with index 31 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 9 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

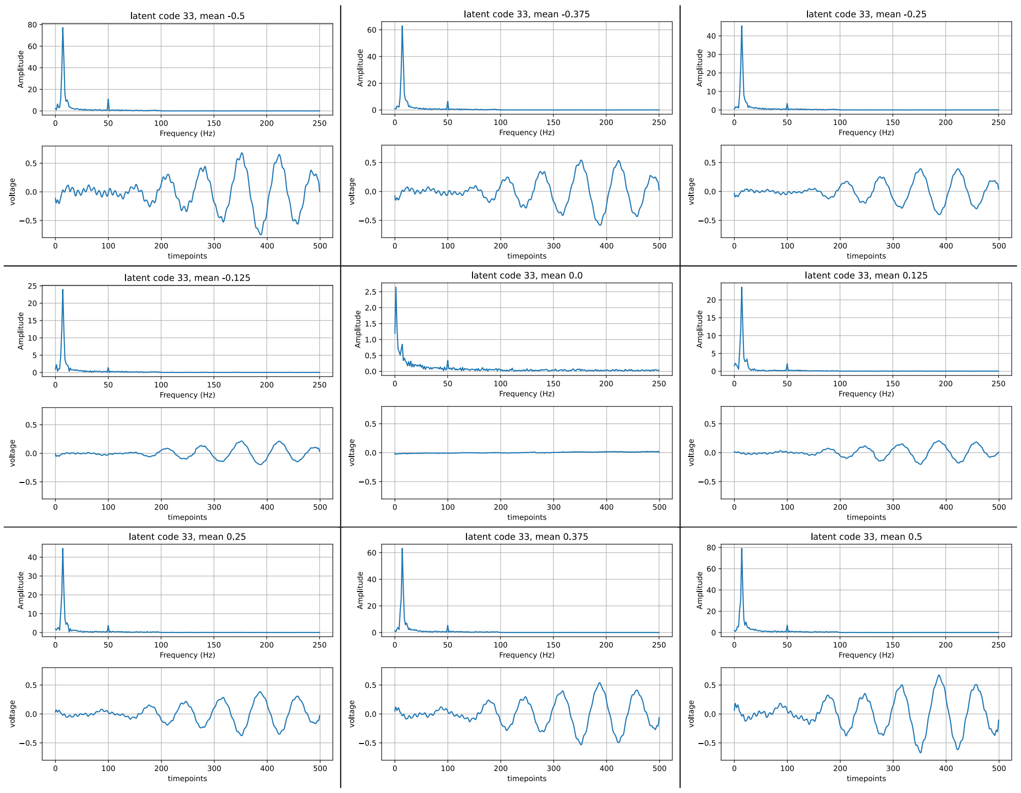


Figure D.11: Visualisation of the latent with index 33 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 10 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

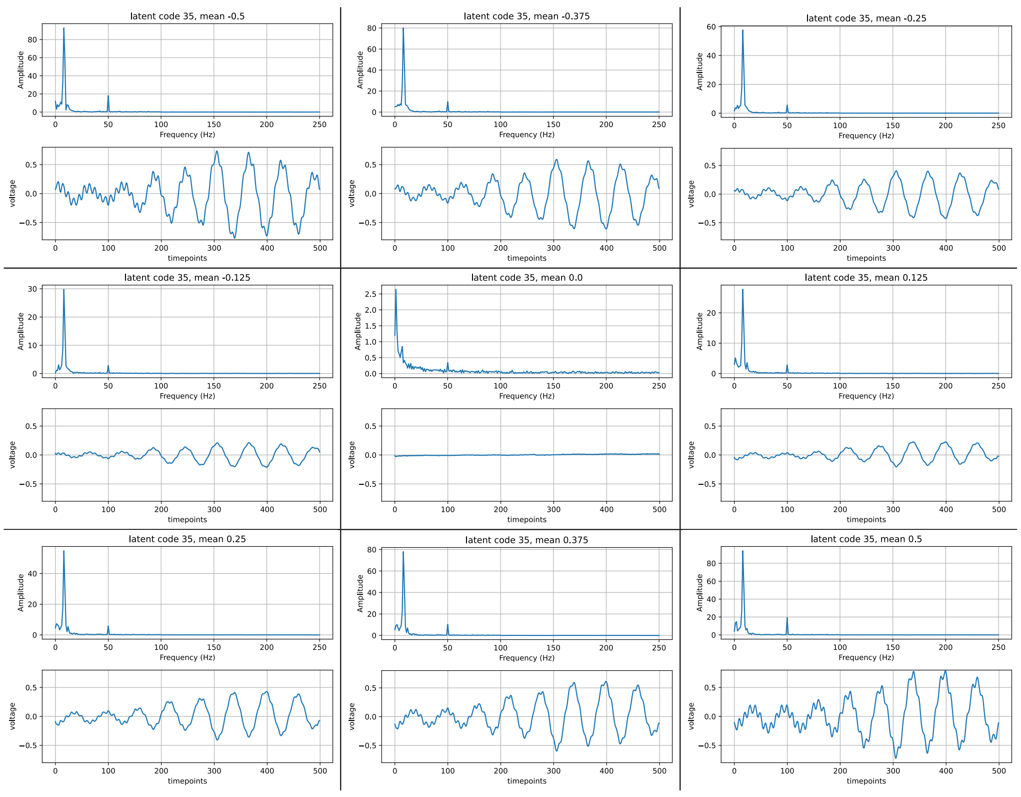


Figure D.12: Visualisation of the latent with index 35 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 11 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

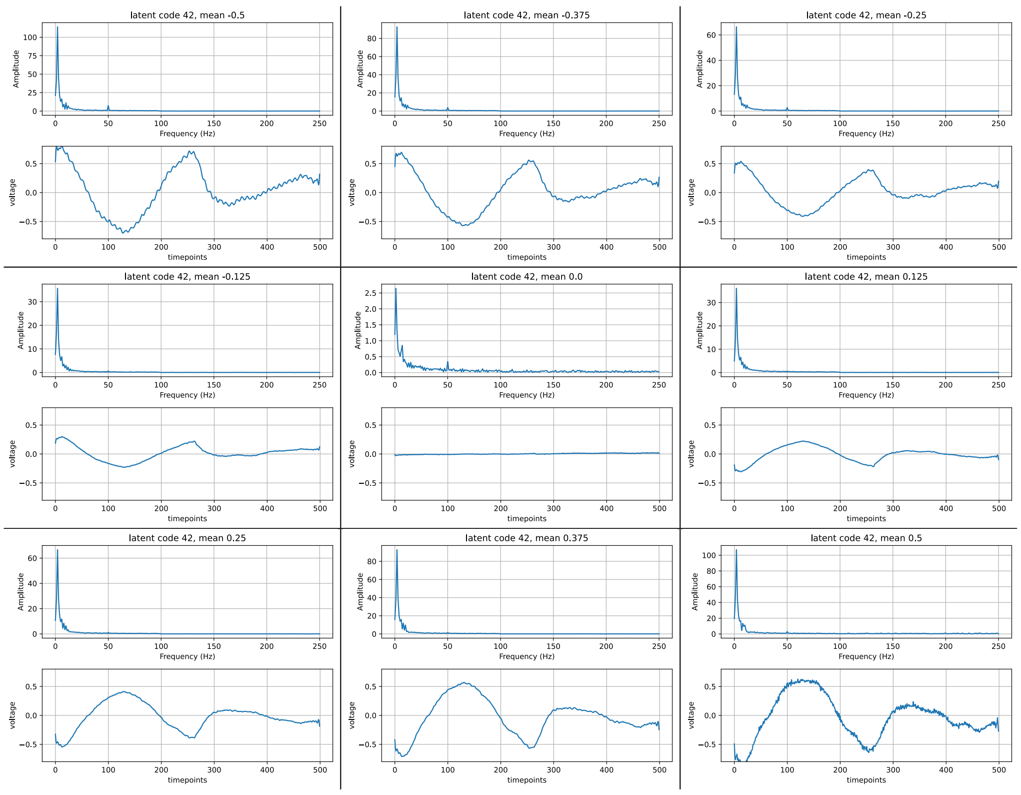


Figure D.13: Visualisation of the latent with index 42 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 12 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

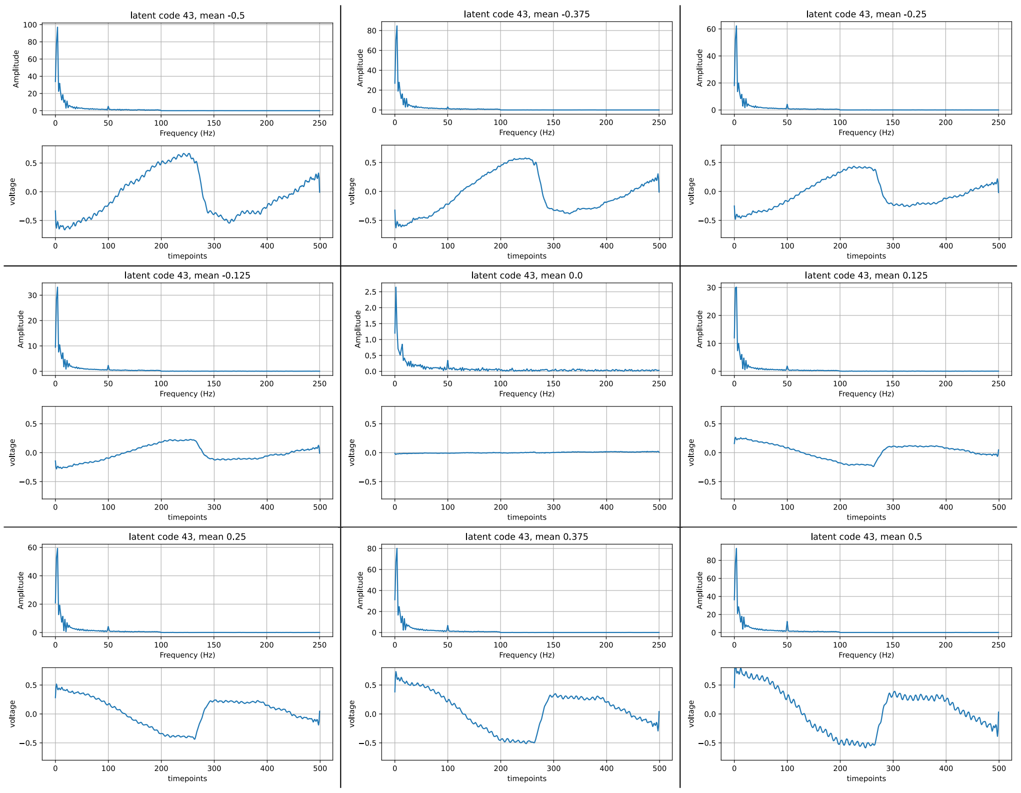


Figure D.14: Visualisation of the latent with index 43 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 13 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

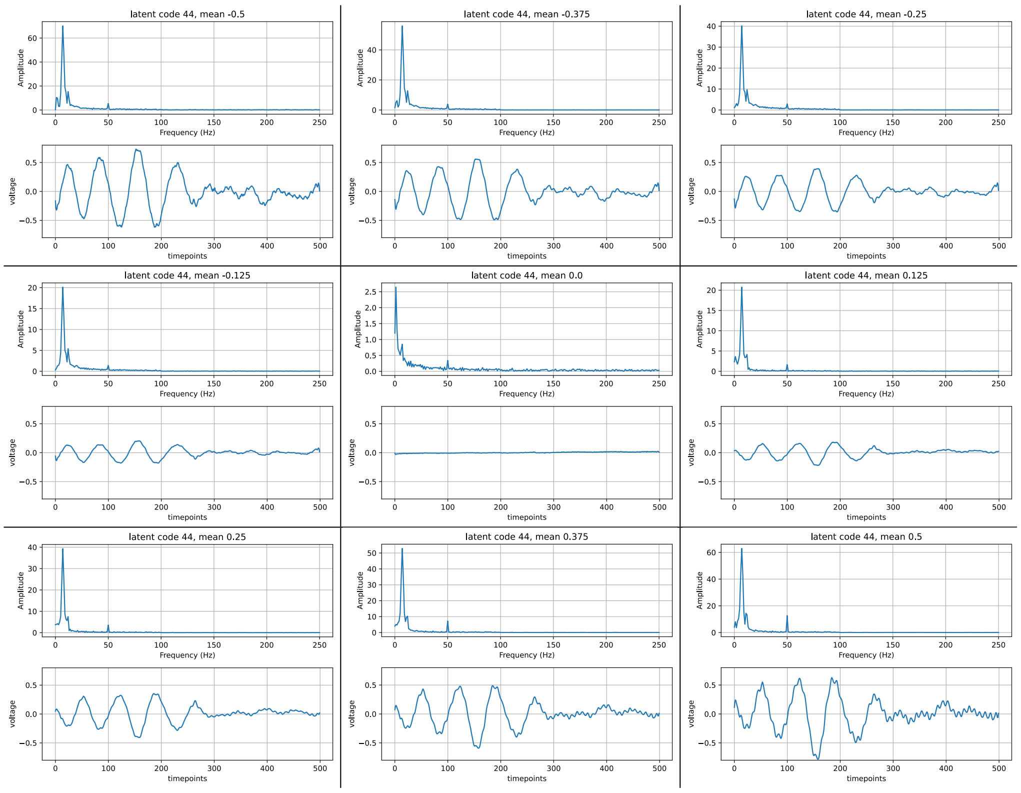


Figure D.15: Visualisation of the latent with index 44 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 14 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

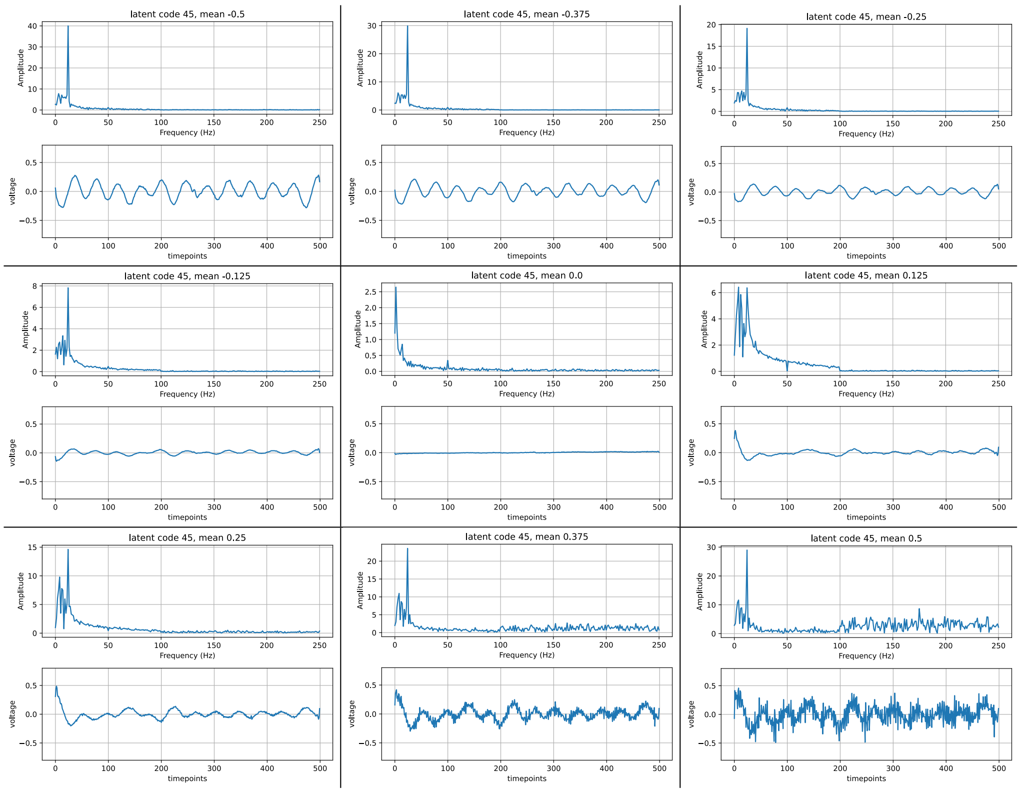


Figure D.16: Visualisation of the latent with index 45 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 15 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

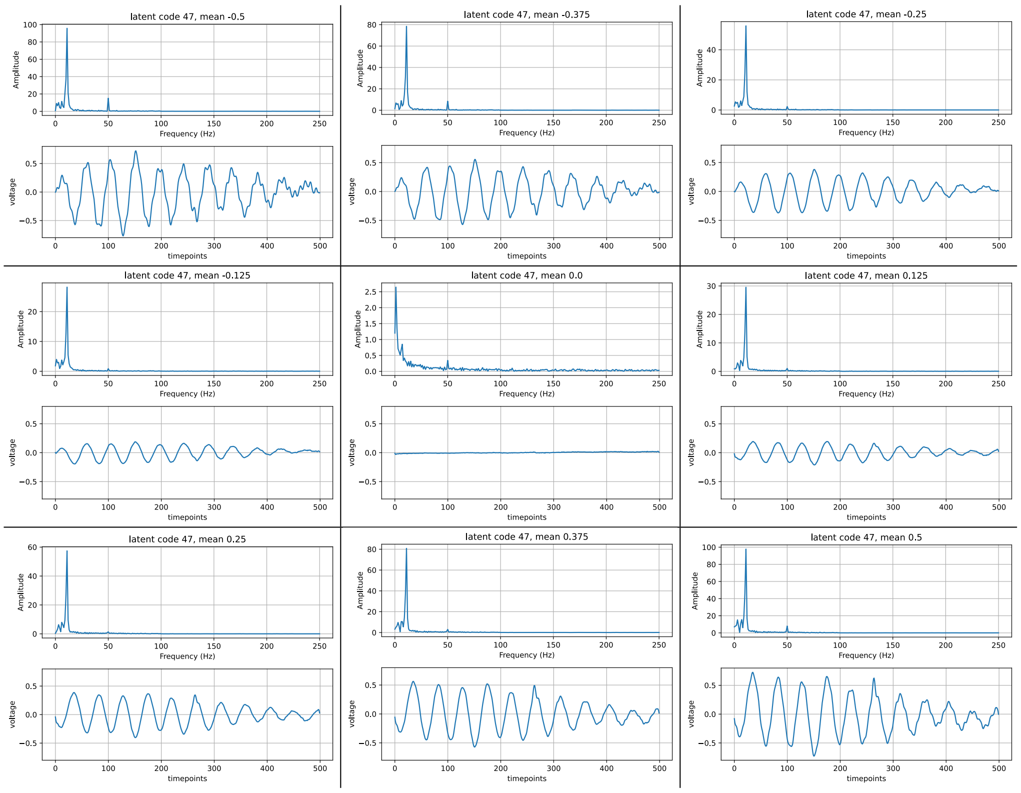


Figure D.17: Visualisation of the latent with index 47 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 16 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

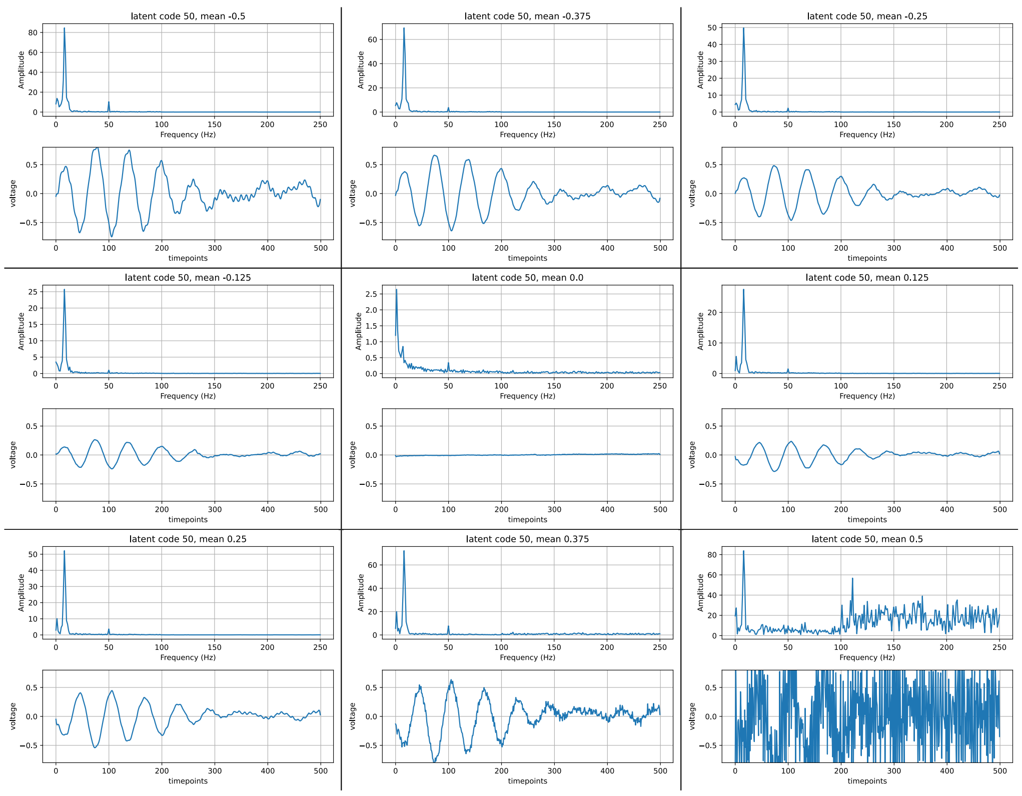


Figure D.18: Visualisation of the latent with index 50 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 17 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

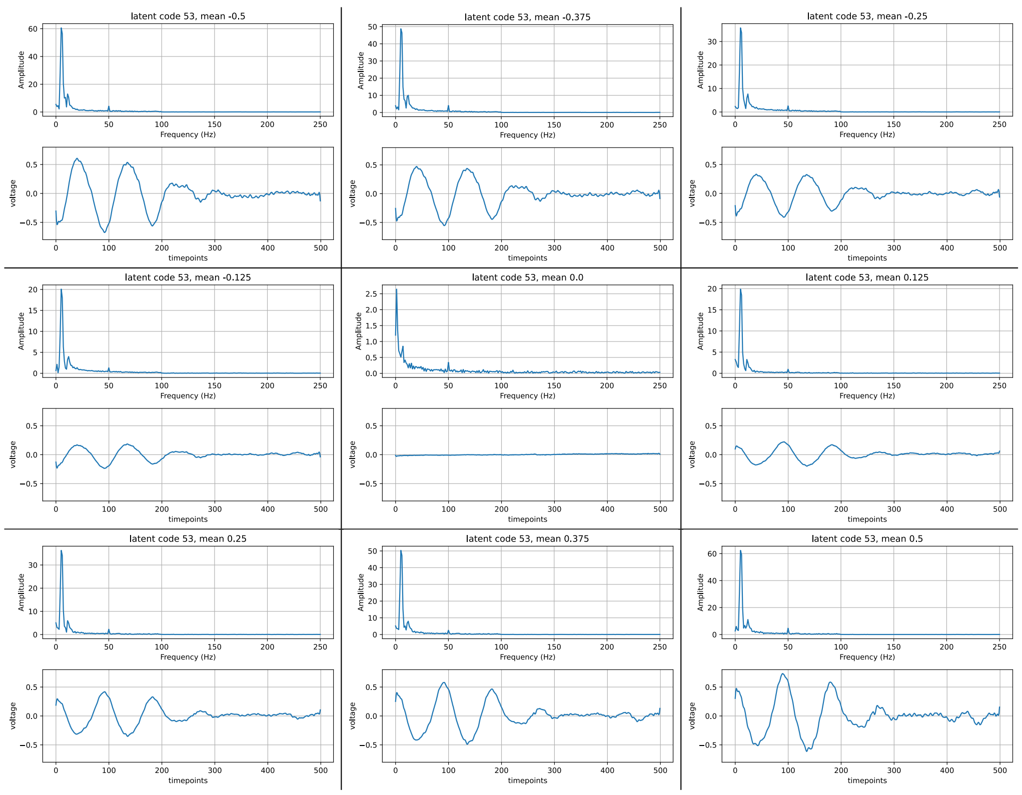


Figure D.19: Visualisation of the latent with index 53 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 18 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

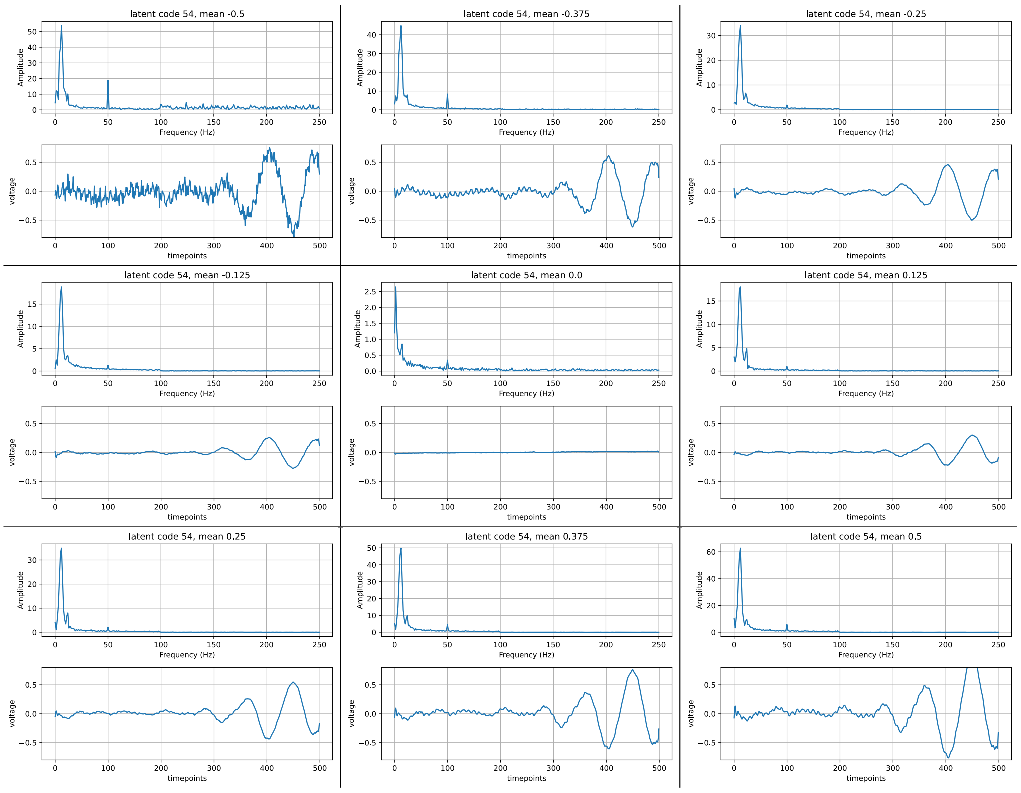


Figure D.20: Visualisation of the latent with index 54 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 19 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

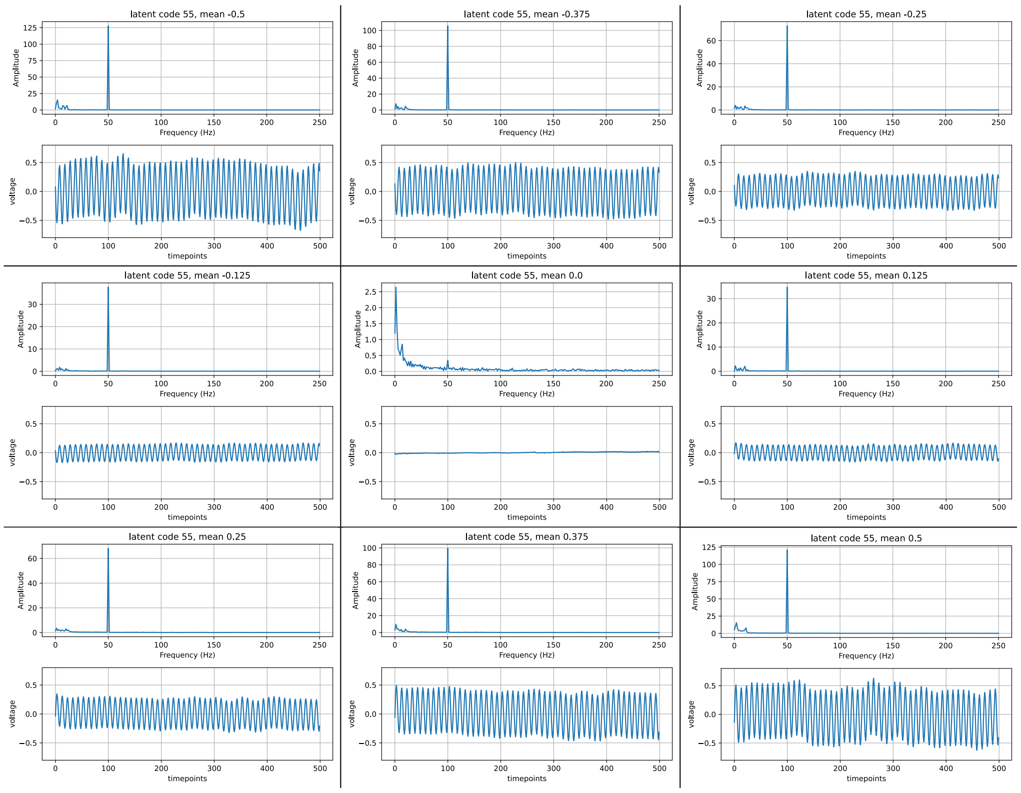


Figure D.21: Visualisation of the latent with index 55 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 20 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

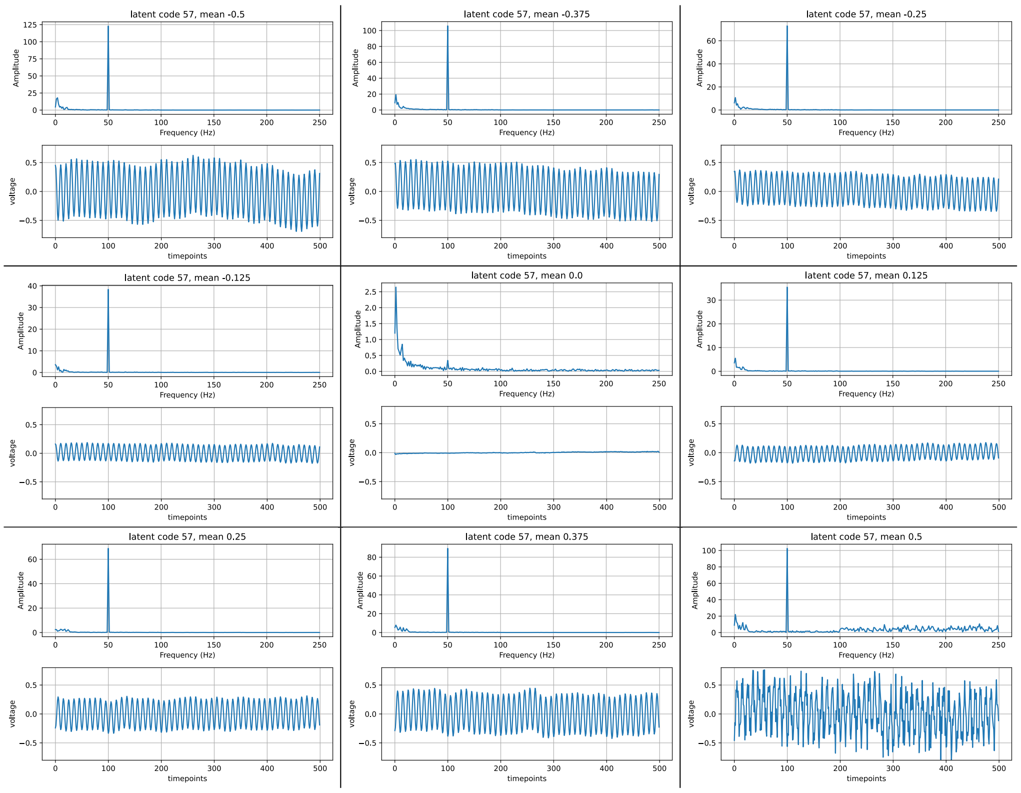


Figure D.22: Visualisation of the latent with index 57 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 21 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

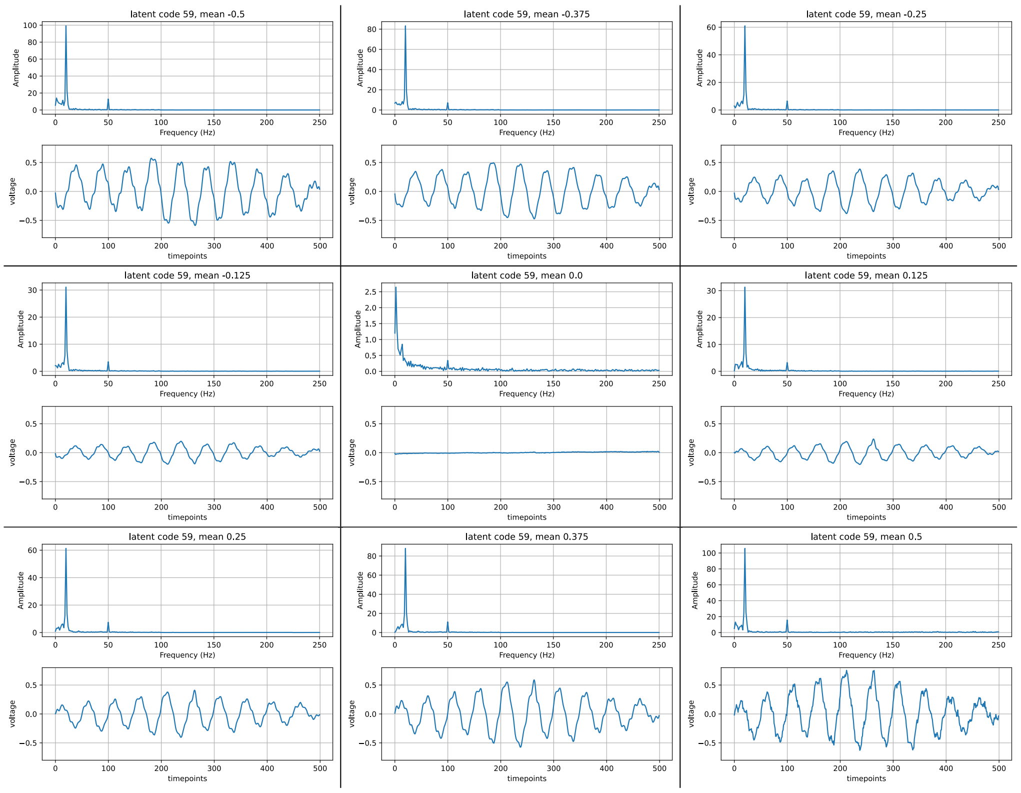


Figure D.23: Visualisation of the latent with index 59 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 22 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

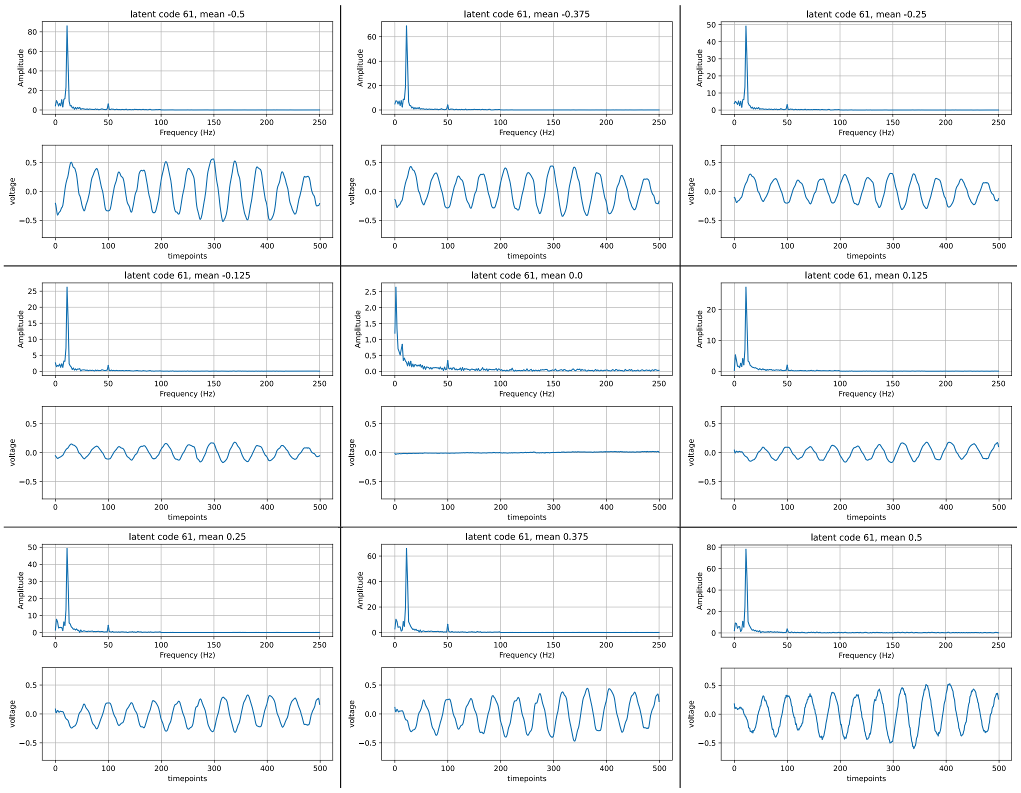


Figure D.24: Visualisation of the latent with index 61 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 23 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

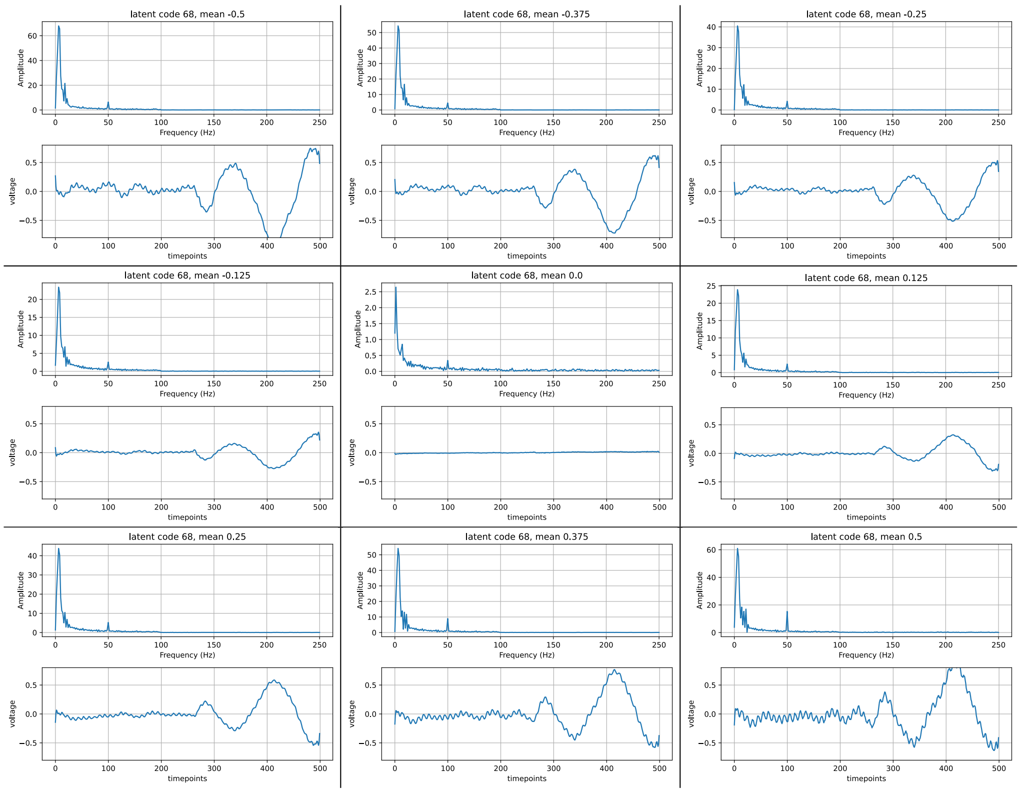


Figure D.25: Visualisation of the latent with index 68 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 24 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

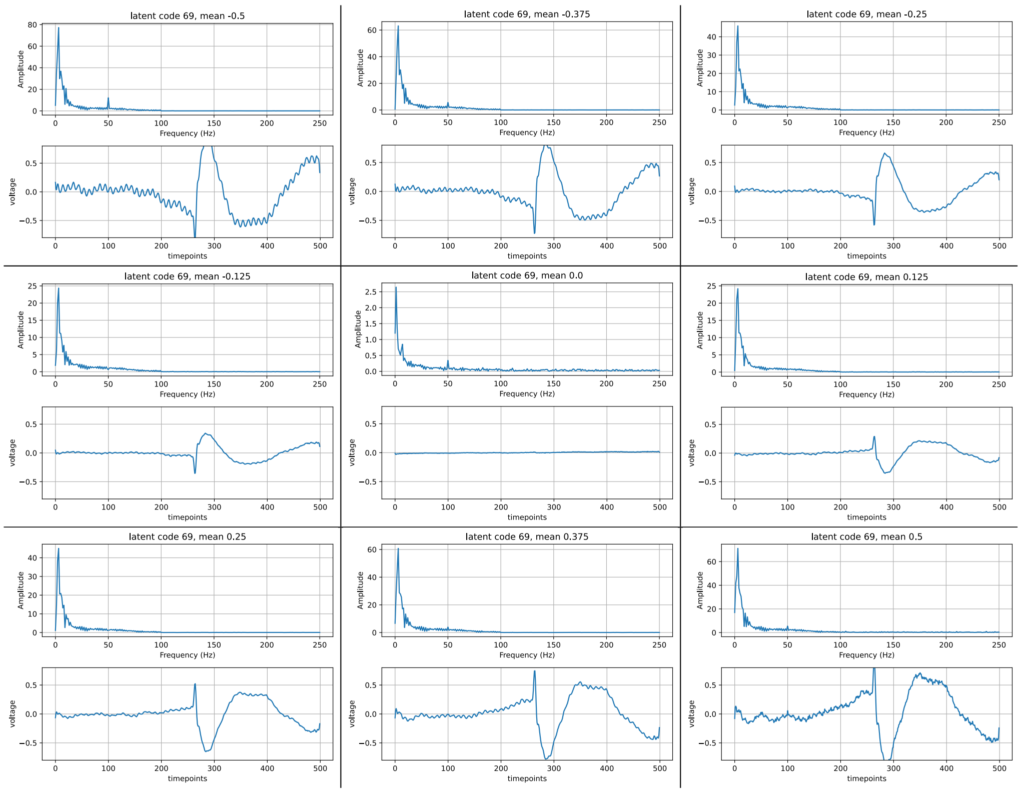


Figure D.26: Visualisation of the latent with index 69 (of the 70 latent dimensions the annealed β-TCVAE had). This latent has the index 25 among the latent that actually carry information (there were 26 latent that carried information). The figure should be read from left to right and from top to bottom. Each subplot visualises the latent for a different latent value (see titles of subplots).

Appendix E

Root Mean Squared Error Versions

For the Amplitude task we use the following implementation of the RMSE metric as in Kastrati et al. [1]:

```
numpy.sqrt(sklearn.metrics.mean_squared_error(y, y_pred))
```

For the Angle task we use

```
s = numpy.sin(y - y_pred)
c = numpy.cos(y - y_pred)
a = numpy.arctan2(s, c)
numpy.sqrt(numpy.mean(numpy.square(a)))
```

In both cases the y and y_pred are a 1D array with the true values and the predicted values respectively.
For the Absolute Position task we used:

```
numpy.linalg.norm(y - y_pred, axis=1).mean()
```

where y and y_pred both matrices with shape $n \times 2$ where n denotes the number of samples the two stands for the x and y values. y and y_pred are respectively the true values and the predicted values.
We used numpy version 1.21.2 and sklearn version 1.1.2.

Appendix F

Plots Sanity Check Reconstructions

In this chapter we added some of the reconstructions of signals that were in the training set for the different sanity check experiments. Each plot was made with the model after the last epoch of training. Important: preprocessing in the frequency domain is always reversed for the plots. Since the low frequencies were usually dominant, we split their amplitude plot away from the rest.

F.1 Standardize per Electrode in Time Domain

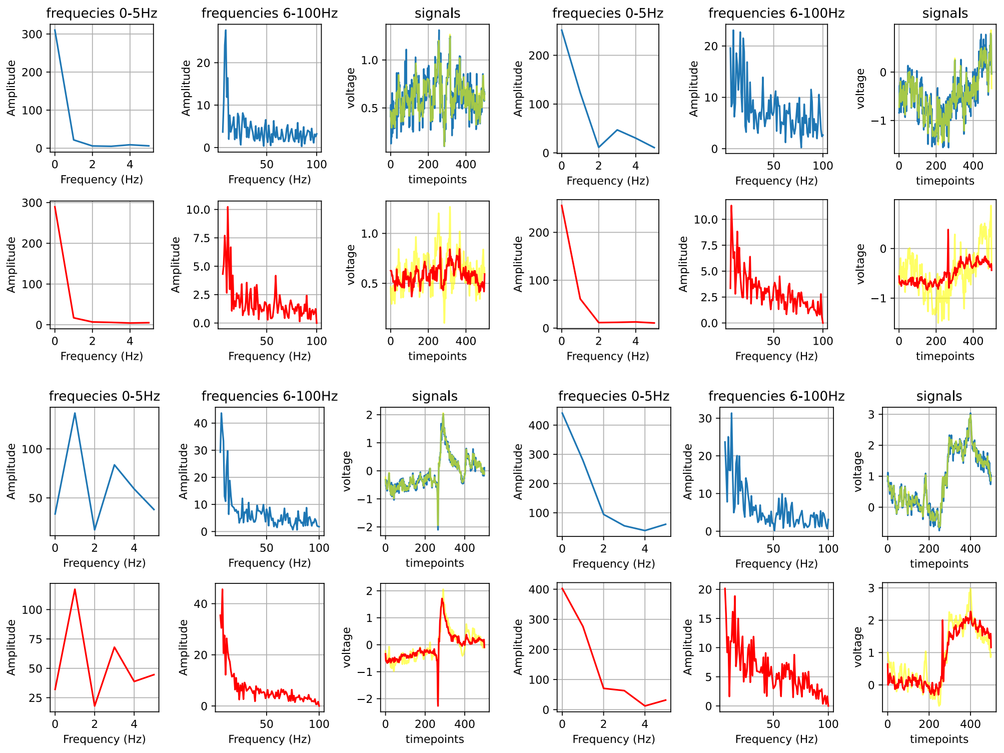


Figure F.1: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 100 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

F.2 Normalize per Electrode in Time Domain

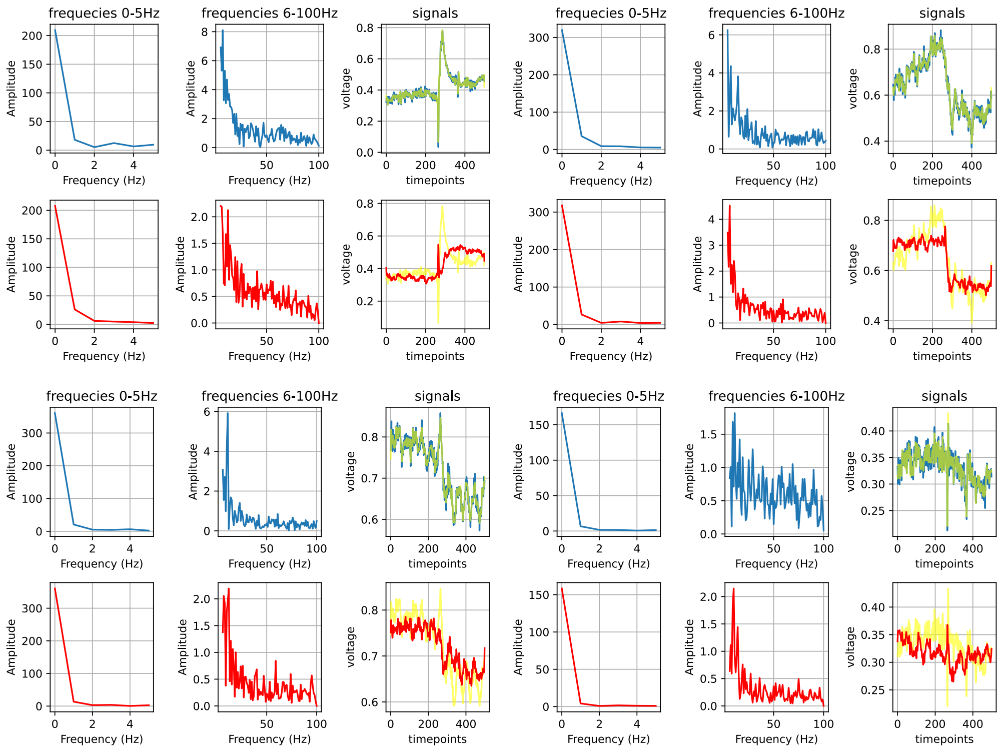


Figure F.2: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 100 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

F.3 Standardize per Electrode in Frequency Domain

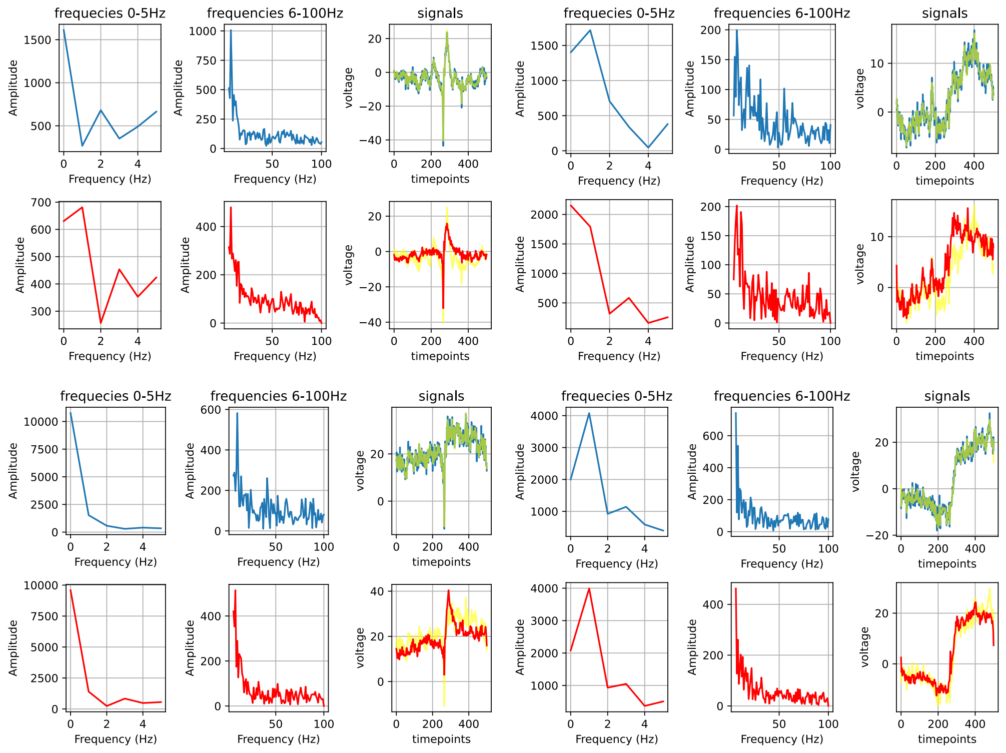


Figure F.3: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 100 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

F.4 Normalize per Electrode in Frequency Domain

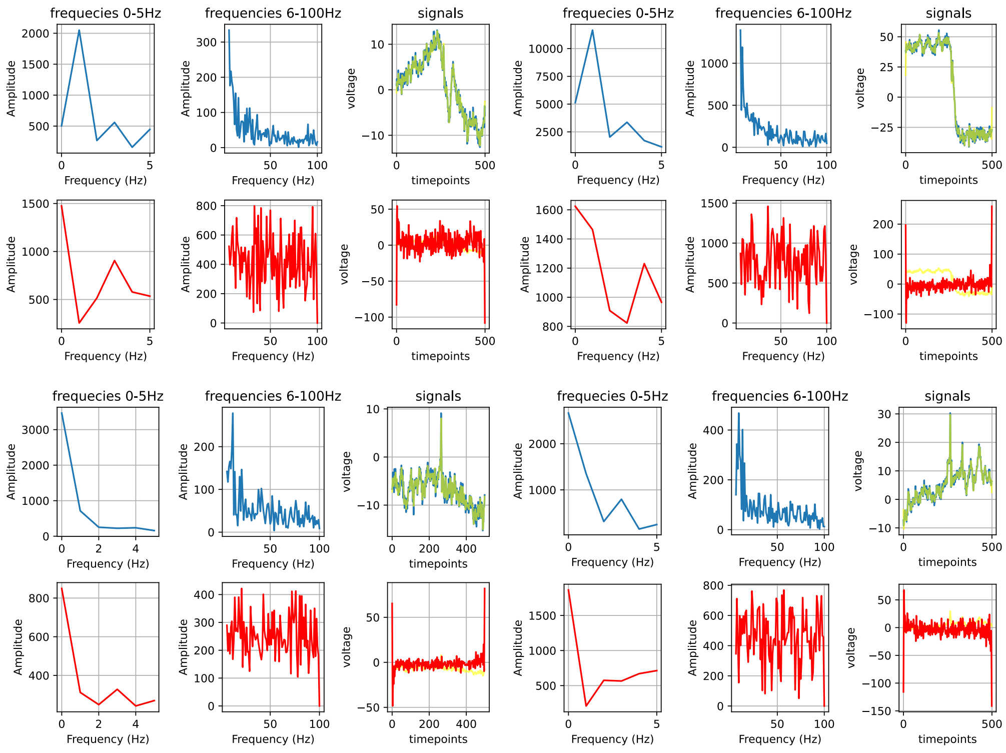


Figure F.4: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 100 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

F.5 Standardize per Split Frequency in Frequency Domain

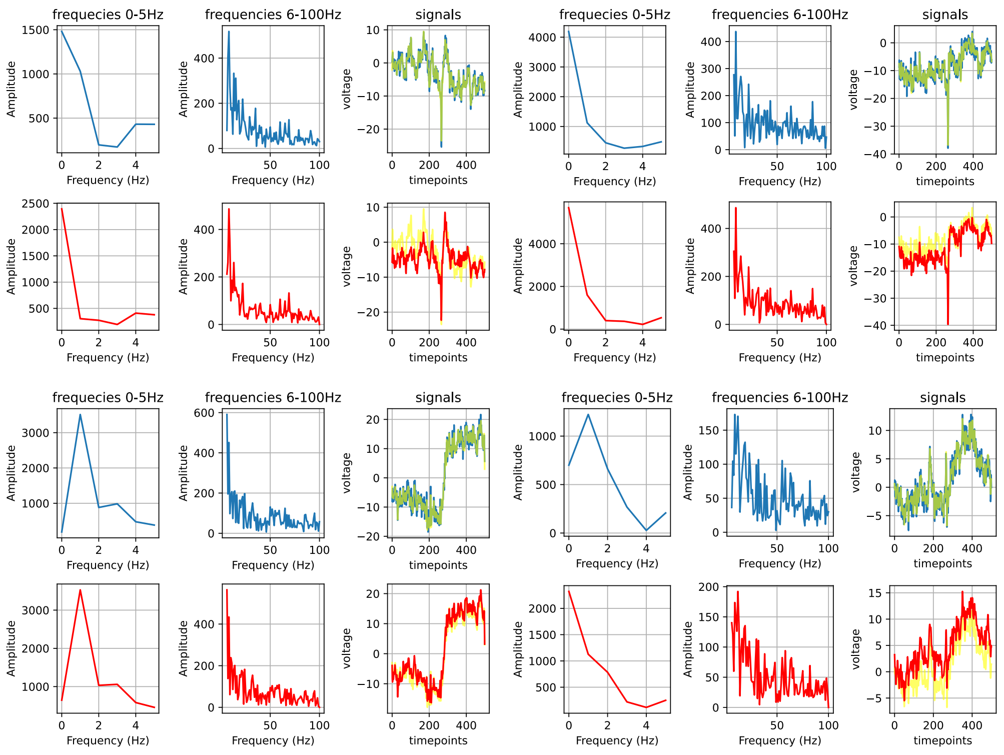


Figure F.5: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 100 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

F.6 Normalize per Split Frequency in Frequency Domain

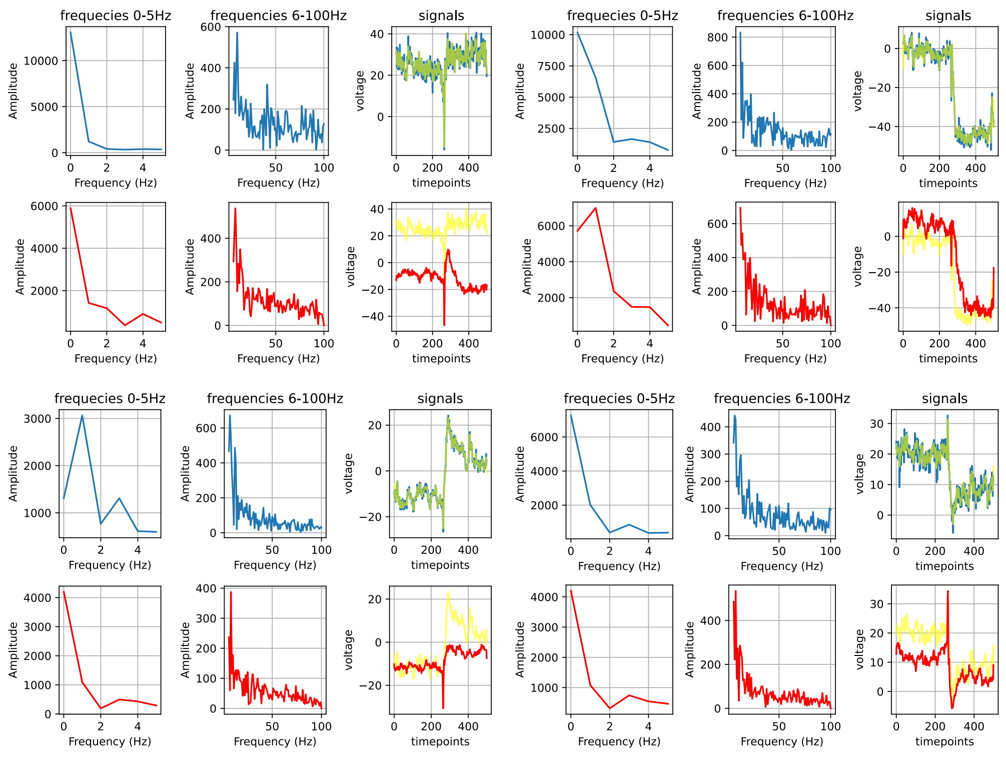


Figure F.6: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 100 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

F.7 Standardize per Electrode in Frequency Domain (with Logarithm)

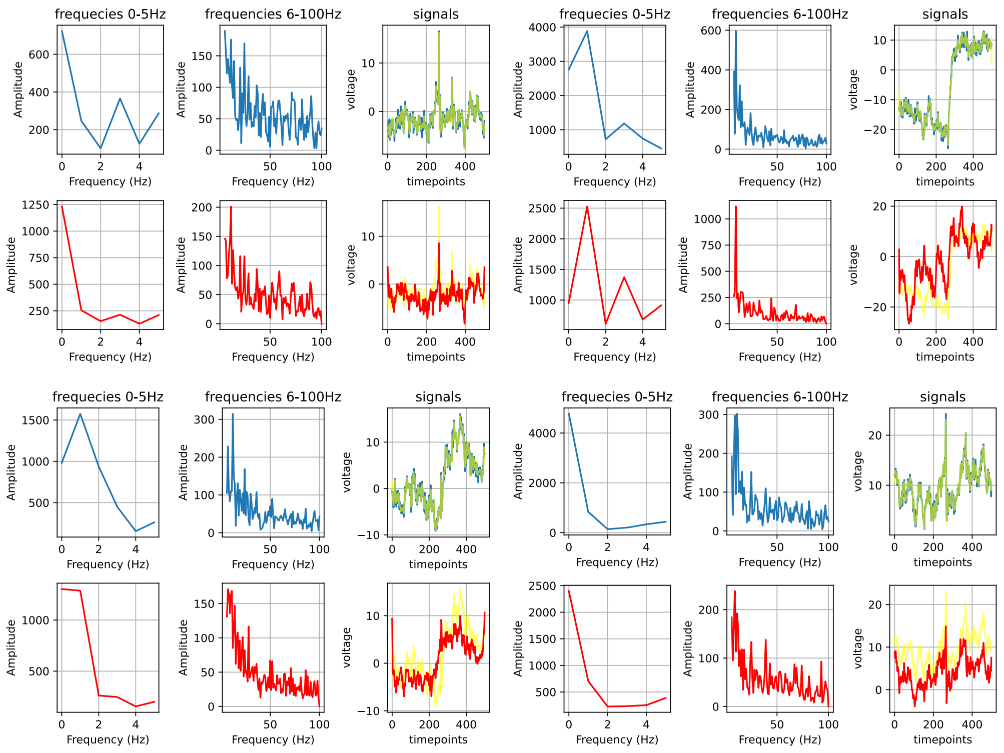


Figure F.7: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 100 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

Appendix G

Full Dataset Reconstructions

Here we show some reconstructions from the validation sets of each of the three VAEs that yielded nice reconstructions. The VAEs in each case were the versions that gave the best validation loss (the validation loss was calculated at the end of every epoch). The configurations of the three models were provided in Section 6.2. Each of the three VAEs were trained on raw data normalized between -0.5 and 0.5 per signal. This preprocessing was applied before the plots were created and not reverted. Since the low frequencies were usually dominant, we split their amplitude plot away from the rest.

G.1 TCVAE with $\mu = 400$

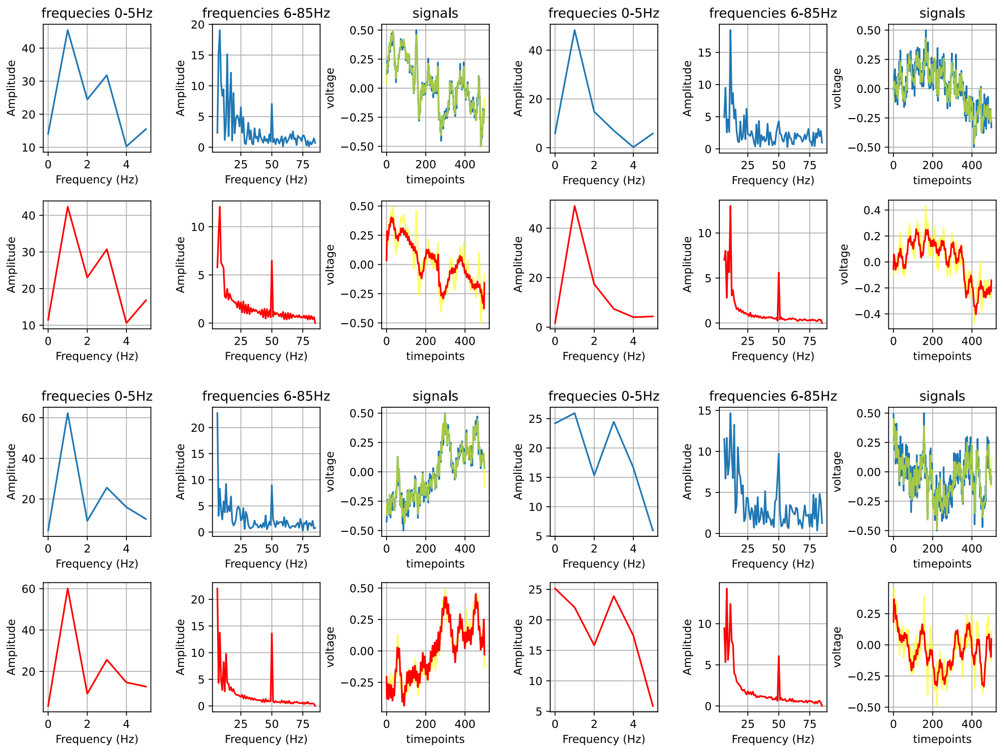


Figure G.1: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 85 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

G.2 TCVAE with $\mu = 800$

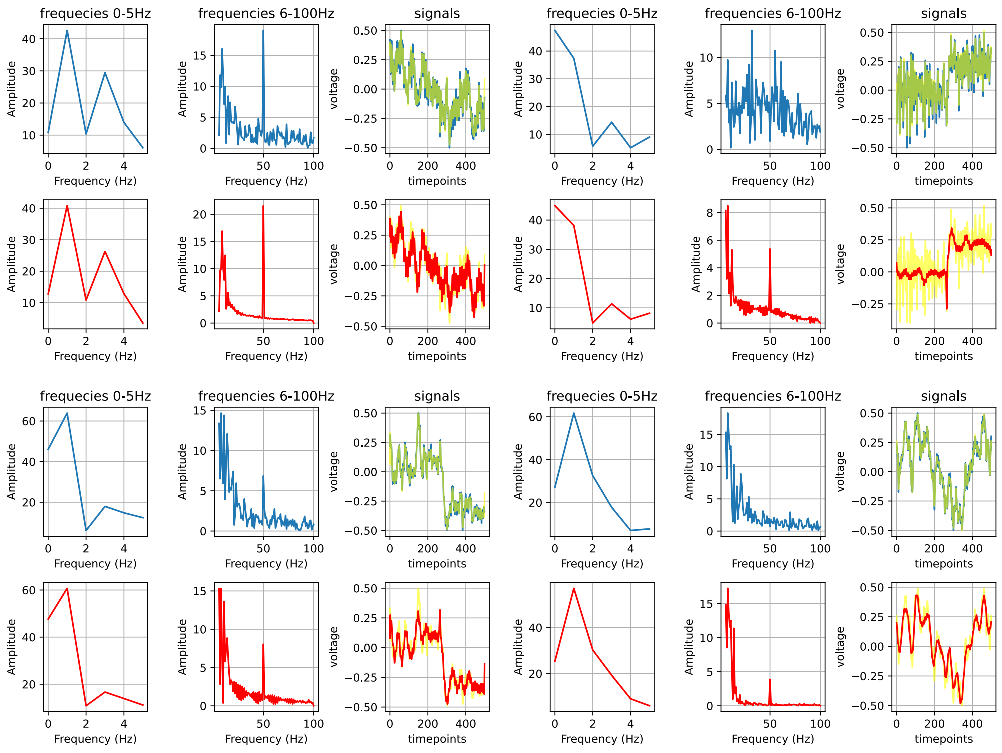


Figure G.2: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 100 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

G.3 Annealed β-TCVAE

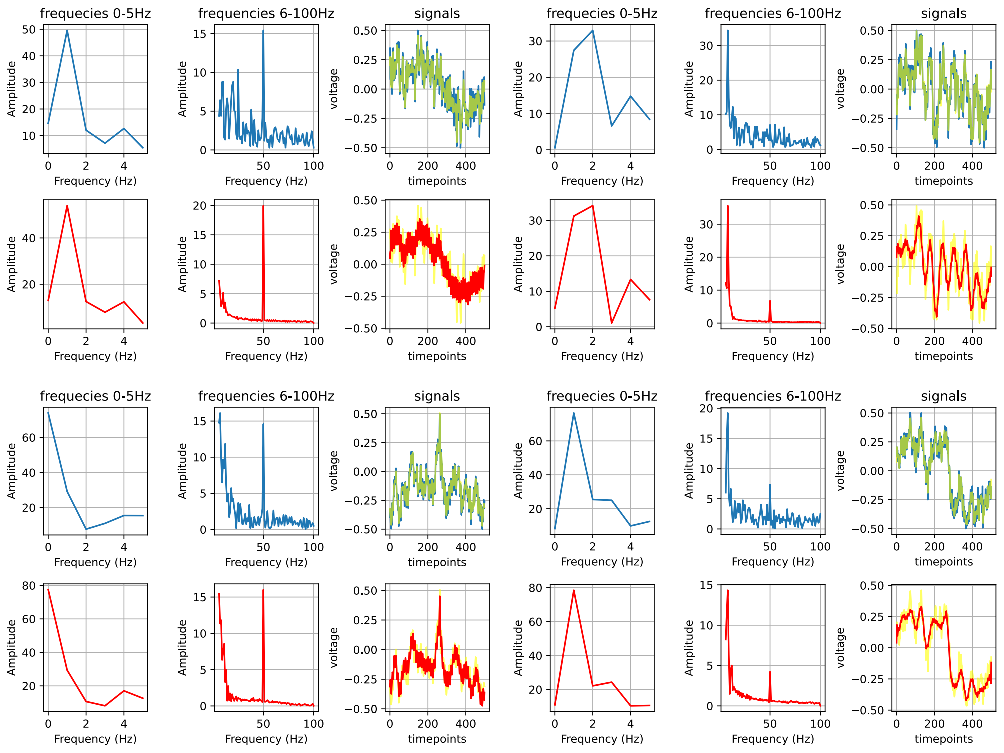


Figure G.3: Four plots (each consisting of 6 subplots) of signal reconstructions. The blue line is always the signal after preprocessing. The red line is always the reconstruction based on the VAE. Yellow is the preprocessed, true signal but with all frequencies from 100 Hz and up removed (were set to 0 for the loss). The subplots from left to right in a plot show the amplitudes of the low frequencies of the signal, the amplitudes of the higher frequencies, and the signal in time domain.

Appendix H

Full Dataset Correlation Matrices

In this chapter we add the correlation matrices of the latent dimensions carrying information from the three models that yielded decent reconstructions. The correlation was calculated among the means the respective encoders output on the combined full data (i.e. the raw direction and raw position datasets combined).

H.1 TCVAE with $\mu = 400$

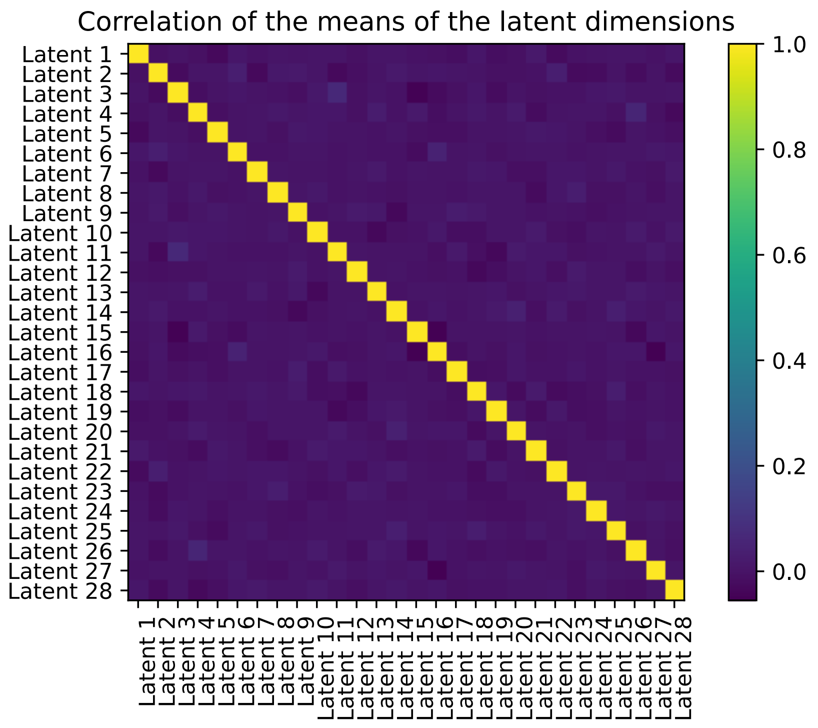


Figure H.1: Correlation matrix for the latent dimensions that carry information. The largest correlation (not along the diagonal) is 0.07 and the smallest is -0.06.

H.2 TCVAE with $\mu = 800$

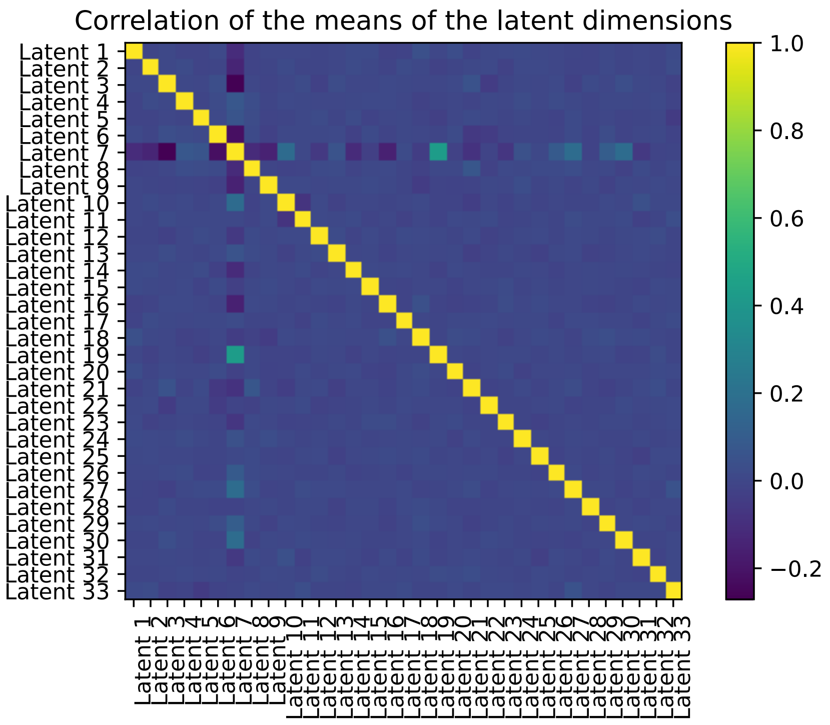


Figure H.2: Correlation matrix for the latent dimensions that carry information. The largest correlation(not along the diagonal) is 0.43 and the smallest is -0.27.

H.3 Annealed β-TCVAE

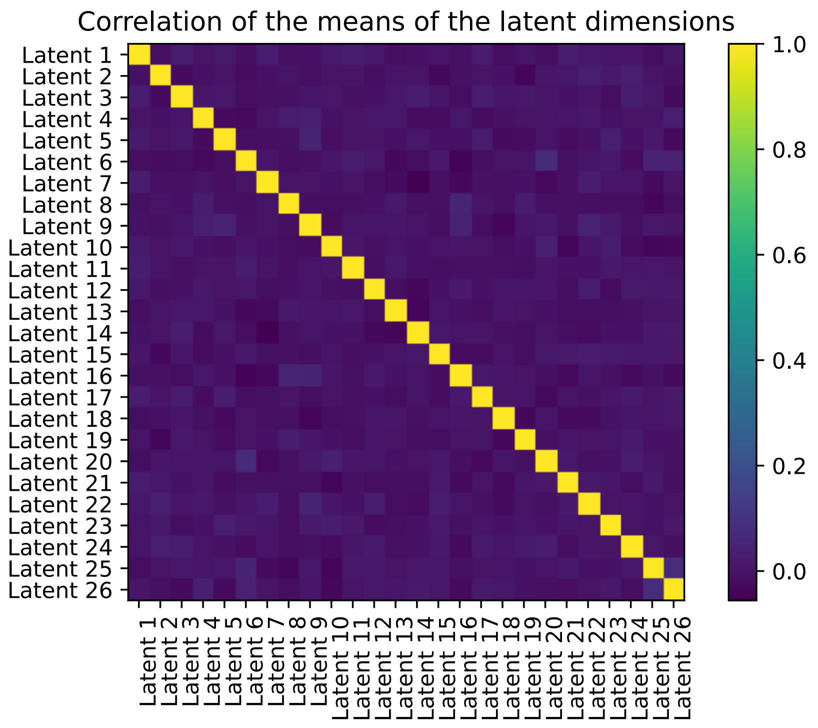


Figure H.3: Correlation matrix for the latent dimensions that carry information. The largest correlation (not along the diagonal) is 0.08 and the smallest is -0.06.

Appendix I

Feature Importance Results

I.1 Amplitude Task

I.1.1 Latent Dimensions

RMSE ± std	Diff. to Baseline	Latent Index
70.59 ± 0.00	0.00	baseline
192.32 ± 0.57	121.73	5
98.33 ± 0.29	27.74	13
77.87 ± 0.18	7.27	20
75.72 ± 0.07	5.13	21
73.78 ± 0.30	3.18	7
72.83 ± 0.12	2.23	25
72.60 ± 0.09	2.01	4
72.60 ± 0.08	2.00	10
72.56 ± 0.14	1.96	12
71.97 ± 0.20	1.38	6
71.96 ± 0.13	1.36	9
71.74 ± 0.09	1.15	11
71.54 ± 0.08	0.95	3
71.41 ± 0.10	0.82	23
71.16 ± 0.10	0.56	0
71.00 ± 0.06	0.40	16
70.98 ± 0.13	0.39	24
70.97 ± 0.05	0.38	2
70.90 ± 0.17	0.31	15
70.74 ± 0.07	0.14	8
70.73 ± 0.08	0.14	1
70.62 ± 0.09	0.03	14
70.56 ± 0.05	-0.03	17
70.55 ± 0.09	-0.05	19
70.54 ± 0.08	-0.05	18
70.52 ± 0.08	-0.08	22

Table I.1: Root mean squared error is in pixels and the standard deviation is computed over the ten runs of the perturbation importance. For a specific row the values for "Latent Index" were perturbed. Note that indexing starts at zero and we only count the 26 latent dimensions that actually carried information among the 70 latent dimensions the VAE had.

I.1.2 Electrodes

RMSE $\pm$ std	Diff. to Baseline	Electrode Index
70.59 ± 0.00	0.00	baseline
78.09 ± 0.21	7.50	125
75.81 ± 0.16	5.22	31
75.45 ± 0.27	4.86	126
74.36 ± 0.20	3.76	0
73.99 ± 0.20	3.40	16
73.85 ± 0.28	3.25	42
73.82 ± 0.18	3.22	124
73.81 ± 0.24	3.22	127
73.22 ± 0.18	2.63	1
72.80 ± 0.11	2.21	37
72.72 ± 0.13	2.12	120
72.58 ± 0.13	1.99	24
72.50 ± 0.08	1.91	25
72.41 ± 0.19	1.82	20
72.15 ± 0.12	1.55	119
72.10 ± 0.09	1.51	43
72.04 ± 0.06	1.45	13
71.98 ± 0.10	1.39	7
71.85 ± 0.13	1.26	112
71.77 ± 0.07	1.18	2
71.76 ± 0.05	1.16	22
71.64 ± 0.07	1.04	21
71.62 ± 0.11	1.03	108
71.57 ± 0.06	0.97	73
71.56 ± 0.09	0.97	14
71.48 ± 0.08	0.89	86
71.47 ± 0.07	0.88	79
71.39 ± 0.06	0.80	68
71.39 ± 0.08	0.79	26
71.38 ± 0.14	0.78	32
71.33 ± 0.08	0.73	17
71.33 ± 0.05	0.73	111
71.28 ± 0.10	0.69	8
71.28 ± 0.10	0.68	39
71.27 ± 0.12	0.68	122
71.23 ± 0.11	0.63	56
71.21 ± 0.08	0.61	64
71.19 ± 0.07	0.59	15
71.18 ± 0.11	0.59	113

71.18 ± 0.08	0.58	44
71.16 ± 0.05	0.56	3
71.14 ± 0.10	0.55	62
71.14 ± 0.04	0.54	97
71.13 ± 0.06	0.54	89
71.09 ± 0.06	0.50	92
71.09 ± 0.06	0.49	93
71.07 ± 0.06	0.47	63
71.05 ± 0.05	0.46	121
71.05 ± 0.06	0.45	74
71.02 ± 0.03	0.43	105
71.02 ± 0.04	0.42	71
71.01 ± 0.13	0.41	47
71.00 ± 0.06	0.40	115
70.98 ± 0.05	0.39	87
70.98 ± 0.07	0.38	83
70.97 ± 0.04	0.38	28
70.95 ± 0.04	0.36	88
70.95 ± 0.06	0.36	30
70.95 ± 0.07	0.35	95
70.95 ± 0.06	0.35	103
70.94 ± 0.04	0.34	29
70.92 ± 0.06	0.33	57
70.91 ± 0.05	0.31	70
70.89 ± 0.05	0.30	6
70.89 ± 0.08	0.29	35
70.89 ± 0.05	0.29	38
70.88 ± 0.07	0.29	94
70.87 ± 0.04	0.28	104
70.87 ± 0.08	0.28	58
70.86 ± 0.11	0.26	36
70.86 ± 0.09	0.26	65
70.85 ± 0.05	0.26	72
70.85 ± 0.05	0.26	45
70.84 ± 0.05	0.25	41
70.83 ± 0.06	0.24	123
70.82 ± 0.06	0.23	23
70.82 ± 0.05	0.23	49
70.81 ± 0.09	0.22	33
70.80 ± 0.06	0.21	59
70.80 ± 0.07	0.21	75
70.80 ± 0.05	0.21	4
70.80 ± 0.04	0.20	9
70.76 ± 0.07	0.17	27

70.76 ± 0.04	0.17	10
70.76 ± 0.05	0.16	81
70.76 ± 0.04	0.16	109
70.76 ± 0.04	0.16	90
70.75 ± 0.08	0.15	55
70.75 ± 0.05	0.15	61
70.73 ± 0.05	0.14	99
70.72 ± 0.06	0.12	100
70.71 ± 0.04	0.12	12
70.71 ± 0.05	0.12	107
70.71 ± 0.06	0.11	96
70.70 ± 0.03	0.10	51
70.69 ± 0.04	0.10	106
70.69 ± 0.07	0.09	69
70.68 ± 0.08	0.09	5
70.66 ± 0.09	0.06	114
70.65 ± 0.06	0.06	67
70.65 ± 0.04	0.05	102
70.64 ± 0.03	0.05	50
70.61 ± 0.01	0.01	128
70.58 ± 0.05	-0.02	40
70.57 ± 0.08	-0.02	53
70.57 ± 0.04	-0.02	60
70.57 ± 0.05	-0.02	116
70.54 ± 0.07	-0.05	84
70.51 ± 0.06	-0.08	34
70.51 ± 0.05	-0.08	18
70.48 ± 0.04	-0.11	98
70.47 ± 0.06	-0.12	80
70.46 ± 0.05	-0.14	101
70.46 ± 0.05	-0.14	19
70.45 ± 0.04	-0.14	117
70.44 ± 0.06	-0.16	91
70.43 ± 0.03	-0.16	46
70.43 ± 0.06	-0.17	52
70.42 ± 0.11	-0.18	54
70.40 ± 0.07	-0.19	110
70.40 ± 0.04	-0.20	82
70.38 ± 0.09	-0.22	11
70.34 ± 0.05	-0.26	78
70.27 ± 0.06	-0.32	118
70.25 ± 0.05	-0.34	66
70.24 ± 0.08	-0.35	77
70.19 ± 0.09	-0.40	76

70.14 ± 0.10	-0.46	48
69.96 ± 0.07	-0.63	85

Table I.2: Root mean squared error is in pixels and the standard deviation is computed over the ten runs of the perturbation importance. For a specific row the values for "Electrode Index" were perturbed. Note that indexing starts at zero.

I.2 Angle Task

I.2.1 Latent Dimensions

RMSE $\pm$ std	Diff. to Baseline	Latent Index
0.46 ± 0.00	0.00	baseline
0.92 ± 0.01	0.46	5
0.59 ± 0.01	0.14	13
0.50 ± 0.01	0.04	25
0.48 ± 0.01	0.02	6
0.47 ± 0.00	0.01	2
0.47 ± 0.01	0.01	12
0.46 ± 0.01	0.01	15
0.46 ± 0.01	0.01	4
0.46 ± 0.00	0.01	1
0.46 ± 0.01	0.01	7
0.46 ± 0.00	0.00	22
0.46 ± 0.00	0.00	18
0.46 ± 0.00	0.00	0
0.46 ± 0.00	0.00	9
0.46 ± 0.00	0.00	3
0.46 ± 0.01	0.00	21
0.46 ± 0.01	0.00	16
0.46 ± 0.00	0.00	10
0.46 ± 0.01	-0.00	14
0.46 ± 0.00	-0.00	23
0.46 ± 0.00	-0.00	19
0.46 ± 0.00	-0.00	24
0.45 ± 0.01	-0.00	11
0.45 ± 0.00	-0.01	8
0.45 ± 0.00	-0.01	17
0.45 ± 0.01	-0.01	20

Table I.3: Root mean squared error is in pixels and the standard deviation is computed over the ten runs of the perturbation importance. For a specific row the values for “Latent Index” were perturbed. Note that indexing starts at zero and we only count the 26 latent dimensions that actually carried information among the 70 latent dimensions the VAE had.

I.2.2 Electrodes

RMSE ± std	Diff. to Baseline	Electrode Index
0.46 ± 0.00	0.00	baseline
0.51 ± 0.01	0.05	126
0.50 ± 0.01	0.04	125
0.48 ± 0.00	0.03	20
0.48 ± 0.00	0.02	0
0.48 ± 0.01	0.02	13
0.48 ± 0.01	0.02	24
0.48 ± 0.00	0.02	7
0.47 ± 0.00	0.02	31
0.47 ± 0.01	0.01	127
0.47 ± 0.01	0.01	124
0.47 ± 0.00	0.01	120
0.47 ± 0.00	0.01	16
0.47 ± 0.01	0.01	118
0.47 ± 0.00	0.01	8
0.47 ± 0.00	0.01	89
0.47 ± 0.00	0.01	17
0.47 ± 0.00	0.01	35
0.47 ± 0.00	0.01	67
0.47 ± 0.00	0.01	12
0.47 ± 0.01	0.01	42
0.46 ± 0.01	0.01	21
0.46 ± 0.00	0.01	26
0.46 ± 0.00	0.01	1
0.46 ± 0.00	0.01	119
0.46 ± 0.00	0.01	40
0.46 ± 0.00	0.01	39
0.46 ± 0.00	0.01	9
0.46 ± 0.01	0.01	14
0.46 ± 0.00	0.01	2
0.46 ± 0.00	0.01	60
0.46 ± 0.00	0.01	55
0.46 ± 0.01	0.01	48
0.46 ± 0.00	0.01	57
0.46 ± 0.00	0.01	37
0.46 ± 0.00	0.01	11
0.46 ± 0.00	0.00	113
0.46 ± 0.00	0.00	61
0.46 ± 0.00	0.00	108
0.46 ± 0.00	0.00	47

0.46 ± 0.00	0.00	66
0.46 ± 0.00	0.00	22
0.46 ± 0.01	0.00	43
0.46 ± 0.00	0.00	6
0.46 ± 0.00	0.00	102
0.46 ± 0.00	0.00	101
0.46 ± 0.00	0.00	112
0.46 ± 0.00	0.00	32
0.46 ± 0.01	0.00	15
0.46 ± 0.00	0.00	98
0.46 ± 0.00	0.00	100
0.46 ± 0.00	0.00	73
0.46 ± 0.01	0.00	62
0.46 ± 0.00	0.00	38
0.46 ± 0.00	0.00	92
0.46 ± 0.00	0.00	79
0.46 ± 0.00	0.00	85
0.46 ± 0.00	0.00	25
0.46 ± 0.00	0.00	56
0.46 ± 0.00	0.00	93
0.46 ± 0.00	0.00	33
0.46 ± 0.00	0.00	83
0.46 ± 0.00	0.00	105
0.46 ± 0.00	0.00	121
0.46 ± 0.00	0.00	116
0.46 ± 0.00	0.00	4
0.46 ± 0.00	0.00	46
0.46 ± 0.00	0.00	122
0.46 ± 0.00	0.00	49
0.46 ± 0.00	0.00	81
0.46 ± 0.00	0.00	5
0.46 ± 0.00	0.00	51
0.46 ± 0.00	0.00	99
0.46 ± 0.00	0.00	68
0.46 ± 0.00	0.00	91
0.46 ± 0.00	0.00	10
0.46 ± 0.00	0.00	70
0.46 ± 0.00	0.00	34
0.46 ± 0.00	0.00	63
0.46 ± 0.00	0.00	117
0.46 ± 0.00	0.00	82
0.46 ± 0.00	0.00	19
0.46 ± 0.00	0.00	94
0.46 ± 0.00	0.00	107

0.46 ± 0.00	0.00	128
0.46 ± 0.00	-0.00	78
0.46 ± 0.00	-0.00	114
0.46 ± 0.00	-0.00	74
0.46 ± 0.00	-0.00	111
0.46 ± 0.00	-0.00	65
0.46 ± 0.00	-0.00	44
0.46 ± 0.00	-0.00	18
0.46 ± 0.00	-0.00	87
0.46 ± 0.00	-0.00	96
0.46 ± 0.00	-0.00	115
0.46 ± 0.00	-0.00	54
0.46 ± 0.00	-0.00	104
0.46 ± 0.00	-0.00	3
0.46 ± 0.00	-0.00	110
0.46 ± 0.00	-0.00	72
0.46 ± 0.00	-0.00	123
0.46 ± 0.00	-0.00	45
0.46 ± 0.00	-0.00	103
0.46 ± 0.00	-0.00	90
0.46 ± 0.00	-0.00	29
0.46 ± 0.00	-0.00	36
0.46 ± 0.00	-0.00	58
0.46 ± 0.00	-0.00	106
0.46 ± 0.00	-0.00	95
0.46 ± 0.00	-0.00	86
0.46 ± 0.00	-0.00	28
0.46 ± 0.00	-0.00	23
0.46 ± 0.00	-0.00	50
0.46 ± 0.00	-0.00	71
0.46 ± 0.00	-0.00	84
0.46 ± 0.00	-0.00	109
0.46 ± 0.00	-0.00	52
0.46 ± 0.00	-0.00	64
0.45 ± 0.00	-0.00	77
0.45 ± 0.01	-0.00	97
0.45 ± 0.00	-0.00	69
0.45 ± 0.00	-0.00	41
0.45 ± 0.00	-0.00	88
0.45 ± 0.00	-0.00	80
0.45 ± 0.00	-0.00	75
0.45 ± 0.00	-0.00	76
0.45 ± 0.00	-0.00	59
0.45 ± 0.00	-0.01	53

0.45 ± 0.00	-0.01	27
0.45 ± 0.00	-0.01	30

Table I.4: Root mean squared error is in pixels and the standard deviation is computed over the ten runs of the perturbation importance. For a specific row the values for "Electrode Index" were perturbed. Note that indexing starts at zero.

I.3 Annealed β-TCVAE

I.3.1 Time Domain

MSE $\pm$ std	Diff. to Baseline	Latent Index
0.25 ± 0.00	0.00	baseline
0.29 ± 0.00	0.03	5
0.26 ± 0.00	0.01	20
0.26 ± 0.00	0.01	21
0.26 ± 0.00	0.01	7
0.26 ± 0.00	0.00	13
0.26 ± 0.00	0.00	6
0.26 ± 0.00	0.00	12
0.26 ± 0.00	0.00	25
0.26 ± 0.00	0.00	3
0.25 ± 0.00	0.00	24
0.25 ± 0.00	0.00	15
0.25 ± 0.00	0.00	8
0.25 ± 0.00	0.00	9
0.25 ± 0.00	0.00	4
0.25 ± 0.00	0.00	22
0.25 ± 0.00	0.00	0
0.25 ± 0.00	0.00	11
0.25 ± 0.00	0.00	1
0.25 ± 0.00	0.00	16
0.25 ± 0.00	0.00	2
0.25 ± 0.00	0.00	17
0.25 ± 0.00	0.00	18
0.25 ± 0.00	0.00	19
0.25 ± 0.00	0.00	23
0.25 ± 0.00	0.00	10
0.25 ± 0.00	0.00	14

Table I.5: Mean squared error was computed between preprocessed signals and the reconstructions from the VAE. For a specific row the values for "Latent Index" were perturbed. Note that indexing starts at zero and we only count the 26 latent dimensions that actually carried information among the 70 latent dimensions the VAE had.

I.3.2 Frequency Domain

MSE $\pm$ std	Diff. to Baseline	Latent Index
1.09 ± 0.00	0.00	baseline
8.95 ± 0.00	7.86	5
5.18 ± 0.00	4.09	7
3.52 ± 0.00	2.43	20
3.51 ± 0.00	2.41	21
2.07 ± 0.00	0.97	13
1.92 ± 0.00	0.82	6
1.48 ± 0.00	0.38	12
1.39 ± 0.00	0.30	25
1.33 ± 0.00	0.23	3
1.29 ± 0.00	0.20	24
1.26 ± 0.00	0.16	15
1.24 ± 0.00	0.15	8
1.23 ± 0.00	0.14	9
1.23 ± 0.00	0.14	4
1.22 ± 0.00	0.13	22
1.22 ± 0.00	0.13	0
1.22 ± 0.00	0.13	11
1.22 ± 0.00	0.13	1
1.22 ± 0.00	0.13	16
1.22 ± 0.00	0.12	2
1.22 ± 0.00	0.12	17
1.21 ± 0.00	0.11	18
1.20 ± 0.00	0.11	19
1.20 ± 0.00	0.11	23
1.20 ± 0.00	0.11	10
1.20 ± 0.00	0.11	14

Table I.6: Mean squared error was computed between preprocessed signals and the reconstructions from the VAE, both were in frequency domain and put in split representation. For a specific row the values for “Latent Index” were perturbed. Note that indexing starts at zero and we only count the 26 latent dimensions that actually carried information among the 70 latent dimensions the VAE had.

I.4 Rank Differences

For the upcoming results we considered the perturbation importance results from Subsection I.1.1, Subsection I.2.1, and Subsection I.3.2. We compared the two results from the tasks each to the result from the annealed β-TCVAE.

I.4.1 Amplitude Task

Latent Index	Rank VAE	Rank Amplitude	Rank Diff.
10	25	8	17
23	24	14	10
4	14	7	7
11	17	12	5
14	26	22	4
13	5	2	3
16	19	16	3
25	8	6	2
9	13	11	2
2	20	18	2
0	16	15	1
5	1	1	0
20	3	3	0
21	4	4	0
19	23	24	-1
12	7	9	-2
17	21	23	-2
7	2	5	-3
1	18	21	-3
18	22	25	-3
6	6	10	-4
3	9	13	-4
24	10	17	-7
15	11	19	-8
8	12	20	-8
22	15	26	-11

Table I.7: The differences in rank between the results for the latent dimensions permutation importance experiment for the Amplitude task and the Annealed β-TCVAE. The table is sorted by the rank difference. A high value means the latent is (relatively) more important according to the perturbation experiment done for the Amplitude task than for the Annealed VAE, et vice versa.

I.4.2 Angle Task

Latent Index	Rank VAE	Rank Angle	Rank Diff.
2	20	5	15
18	22	12	10
1	18	9	9
10	25	18	7
14	26	19	7
4	14	8	6
25	8	3	5
15	11	7	4
22	15	11	4
23	24	20	4
13	5	2	3
0	16	13	3
6	6	4	2
16	19	17	2
19	23	21	2
12	7	6	1
5	1	1	0
9	13	14	-1
17	21	25	-4
3	9	15	-6
11	17	23	-6
7	2	10	-8
21	4	16	-12
24	10	22	-12
8	12	24	-12
20	3	26	-23

Table I.8: The differences in rank between the results for the latent dimensions permutation importance experiment for the Angle task and the Annealed β-TCVAE. The table is sorted by the rank difference. A high value means the latent is (relatively) more important according to the perturbation experiment done for the Angle task than for the Annealed VAE, et vice versa.

Appendix J

Feature Statistics

Each subsection of this chapter corresponds to one experiment from the Sections 6.1 and 6.2. Check the section and subsection titles to see which table and figure belongs to which experiment. The tables all have the same setup: the left column shows the highest five means of all split frequencies (i.e. the individual features that were fed into the VAE) above the three dots and then the lowest five after the three dots. The right column is the same but for the standard deviation. The numbers before the colon denote the index of the feature. Meaning that the values with indices from 0 to 250 were derived from the real parts of the frequency components for frequencies 0 to 250 Hz and the indices from 251 to 501 from their respective imaginary parts. The values after the colon are the means or standard deviations of said feature.

J.1 Sanity Checks

J.1.1 Standardize per Electrode in Time Domain

mean	std
252: 19.03371	0: 344.93307
1: 5.63791	252: 195.65072
8: 3.7012	1: 62.59731
260: 3.36475	254: 57.50859
258: 3.25281	2: 43.166
...	...
261: -2.47707	498: 0.06582
259: -3.05653	499: 0.04297
7: -3.30788	500: 0.03482
5: -3.9845	501: 0.0
2: -5.89568	251: 0.0

Table J.1: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

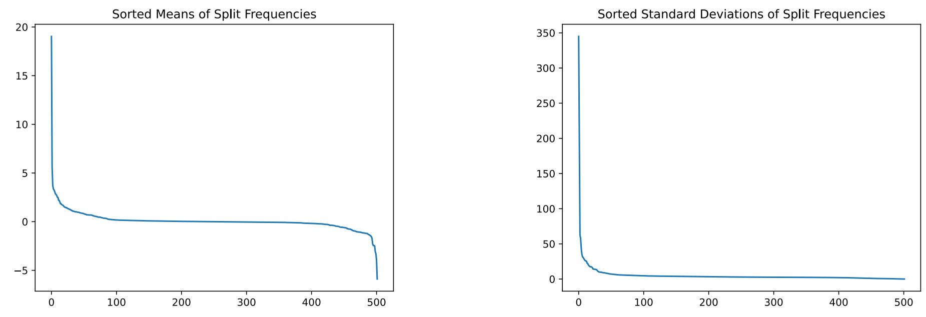


Figure J.1: Sorted means and standard deviations of the split frequencies.

J.1.2 Normalize per Electrode in Time Domain

mean	std
0: 240.26649	0: 71.87544
252: 2.50149	252: 34.62553
1: 0.92663	1: 10.69696
8: 0.5486	254: 10.19734
260: 0.51118	2: 7.39952
...	...
261: -0.40866	498: 0.01088
259: -0.45606	499: 0.00725
7: -0.51297	500: 0.00587
5: -0.60545	501: 0.0
2: -0.82523	251: 0.0

Table J.2: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

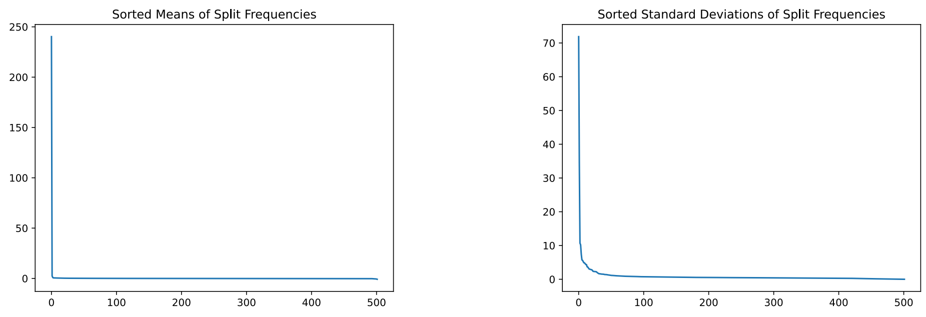


Figure J.2: Sorted means and standard deviations of the split frequencies.

J.1.3 Standardize per Electrode in Frequency Domain

mean	std
252: 0.93598	0: 17.84963
1: 0.27493	252: 10.38945
8: 0.17789	1: 3.30257
260: 0.16128	254: 3.04335
258: 0.15523	2: 2.28409
...	...
16: -0.12459	499: 0.01667
259: -0.15164	501: 0.01635
7: -0.16546	251: 0.01635
5: -0.1984	500: 0.01633
2: -0.29206	497: 0.01599

Table J.3: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

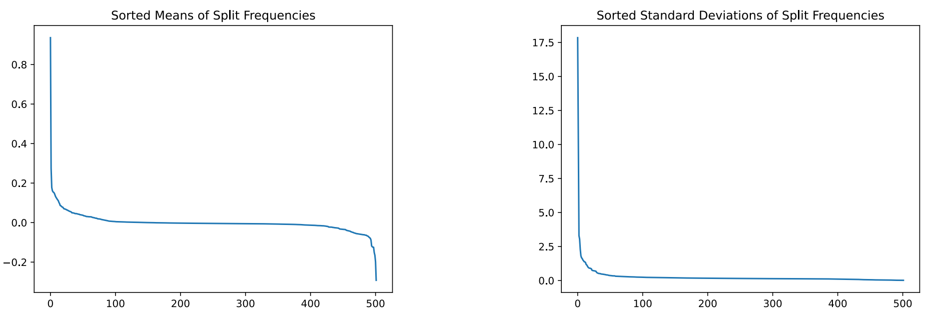


Figure J.3: Sorted means and standard deviations of the split frequencies.

J.1.4 Normalize per Electrode in Frequency Domain

mean	std
252: 0.51918	0: 0.31641
1: 0.50921	252: 0.24089
0: 0.50883	254: 0.15272
8: 0.50802	1: 0.15254
260: 0.50776	253: 0.14641
...	...
261: 0.50358	291: 0.14126
259: 0.50331	41: 0.14124
7: 0.50302	280: 0.14122
5: 0.50253	7: 0.14099
2: 0.50127	18: 0.14098

Table J.4: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

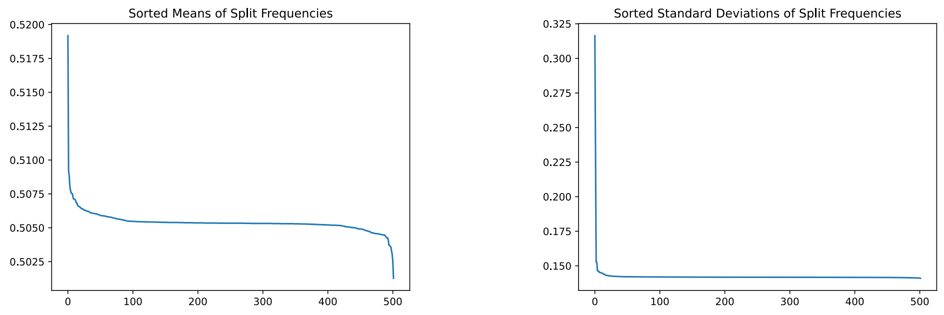


Figure J.4: Sorted means and standard deviations of the split frequencies.

J.1.5 Standardize per Split Frequency in Frequency Domain

Having a table and plots here would be silly since the whole point of the preprocessing is to set the split frequencies' means to zero and standard deviations to one.

J.1.6 Normalize per Split Frequency in Frequency Domain

mean	std
15: 0.80219	19: 0.15922
292: 0.73465	39: 0.15382
294: 0.72492	51: 0.15061
53: 0.70765	30: 0.15021
433: 0.70744	341: 0.14926
...	...
24: 0.27849	437: 0.06634
82: 0.25319	483: 0.06631
52: 0.22888	0: 0.06476
501: 0.0	501: 0.0
251: 0.0	251: 0.0

Table J.5: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

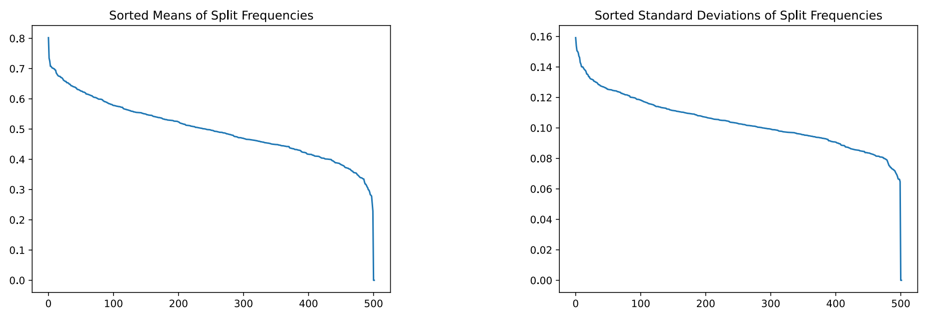


Figure J.5: Sorted means and standard deviations of the split frequencies.

J.1.7 Standardize per Electrode in Frequency Domain (with Logarithm)

mean	std
0: 2.56436	500: 0.95373
1: 2.39493	499: 0.93617
3: 1.92652	497: 0.9353
2: 1.89029	498: 0.92435
4: 1.76381	496: 0.91735
...	...
295: -0.92468	96: 0.25242
314: -0.92871	16: 0.24825
329: -0.94092	73: 0.24766
297: -0.94484	67: 0.24553
263: -0.97532	123: 0.24486

Table J.6: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

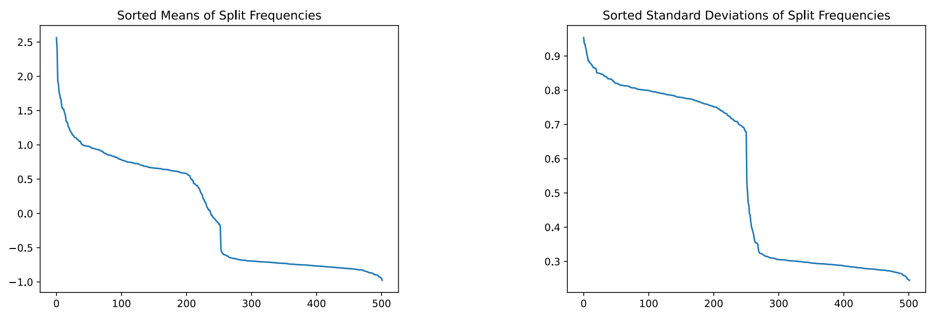


Figure J.6: Sorted means and standard deviations of the split frequencies.

J.2 Full Dataset - Raw

J.2.1 Standardize per Electrode in Time Domain

mean	std
252: 0.00577	0: 499.99457
253: 0.00255	252: 1.08218
254: 0.00188	1: 0.66405
255: 0.00146	253: 0.54654
256: 0.00113	254: 0.3559
...	...
27: -4e-05	498: 0.0002
9: -4e-05	499: 0.00018
24: -5e-05	500: 0.00018
50: -6e-05	501: 0.0
6: -7e-05	251: 0.0

Table J.7: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

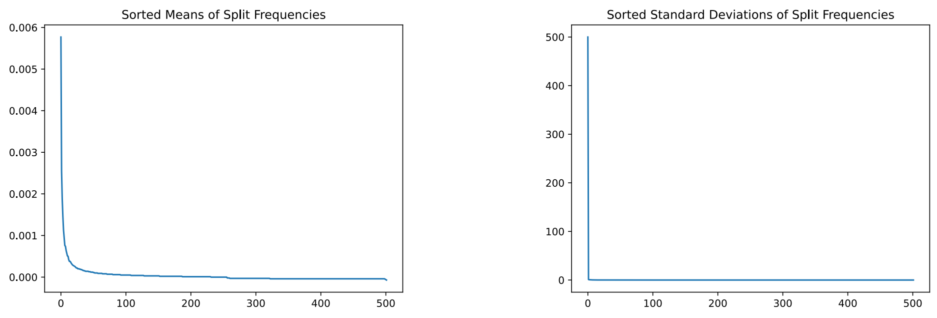


Figure J.7: Sorted means and standard deviations of the split frequencies.

J.2.2 Normalize per Electrode in Time Dimension

mean	std
0: 266.64764	0: 119.34609
252: 0.00089	252: 0.15946
253: 0.00037	1: 0.09671
254: 0.0003	253: 0.08094
255: 0.00021	254: 0.05503
...	...
13: -1e-05	498: 3e-05
21: -1e-05	500: 3e-05
1: -1e-05	499: 3e-05
6: -2e-05	501: 0.0
50: -2e-05	251: 0.0

Table J.8: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

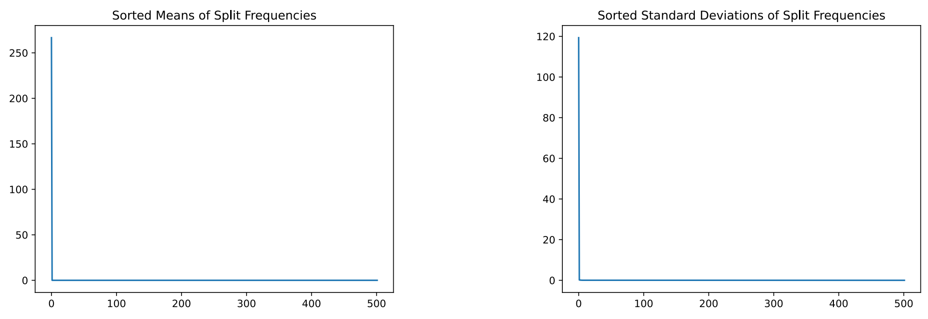


Figure J.8: Sorted means and standard deviations of the split frequencies.

J.2.3 Standardize per Electrode in Frequency Domain

mean	std
0: 2.92194	0: 22.20834
252: -0.00564	252: 0.04827
253: -0.00575	1: 0.03454
254: -0.00577	253: 0.03054
255: -0.00578	254: 0.02556
...	...
17: -0.00584	331: 0.02145
21: -0.00584	329: 0.02145
27: -0.00584	316: 0.02145
50: -0.00584	310: 0.02145
6: -0.00584	312: 0.02145

Table J.9: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

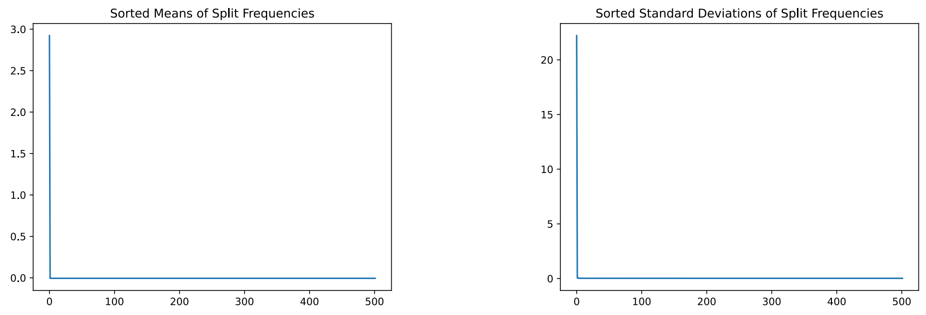


Figure J.9: Sorted means and standard deviations of the split frequencies.

J.2.4 Normalize per Electrode in Frequency Domain

mean	std
0: 0.53592	0: 0.23644
252: 0.50915	1: 0.23234
253: 0.50914	6: 0.23234
254: 0.50914	50: 0.23234
255: 0.50914	9: 0.23234
...	...
13: 0.50914	256: 0.23234
21: 0.50914	255: 0.23234
1: 0.50914	254: 0.23234
6: 0.50914	253: 0.23234
50: 0.50914	252: 0.23234

Table J.10: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

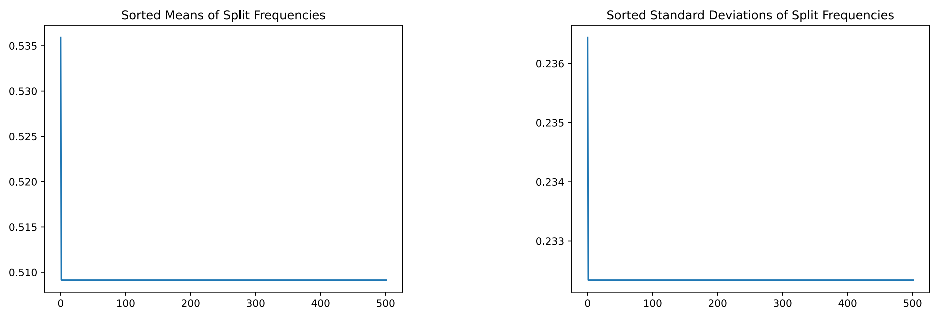


Figure J.10: Sorted means and standard deviations of the split frequencies.

J.2.5 Standardize per Split Frequency in Frequency Domain

Having a table and plots here would be silly since the whole point of the preprocessing is to set the split frequencies' means to zero and standard deviations to one.

J.2.6 Normalize per Split Frequency in Frequency Domain

mean	std
9: 0.76098	0: 0.03582
36: 0.74935	401: 0.00765
33: 0.72436	50: 0.00753
66: 0.68407	301: 0.00717
39: 0.67309	150: 0.00582
...	...
51: 0.23667	9: 0.00036
21: 0.21351	18: 0.00036
0: 0.01259	6: 0.00033
501: 0.0	501: 0.0
251: 0.0	251: 0.0

Table J.11: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

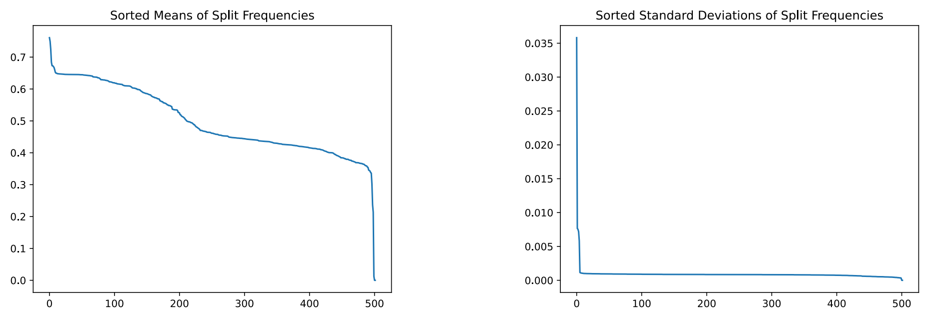
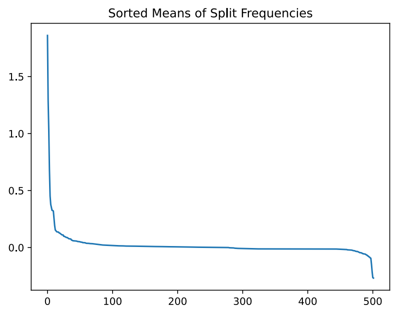


Figure J.11: Sorted means and standard deviations of the split frequencies.

J.2.7 Normalize (-0.5 to 0.5) per Signal in Time Domain

mean	std
252: 1.86118	252: 38.88895
0: 1.29288	0: 31.35796
253: 1.03197	50: 23.9414
255: 0.68354	301: 23.93475
254: 0.45213	253: 16.95273
...	...
9: -0.0936	498: 0.01799
3: -0.14883	499: 0.01402
7: -0.21361	500: 0.01194
5: -0.26417	501: 0.0
2: -0.26796	251: 0.0

Table J.12: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.



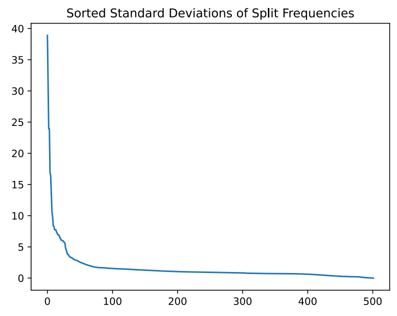


Figure J.12: Sorted means and standard deviations of the split frequencies.

J.3 Full Dataset - Minimally Preprocessed

J.3.1 Standardize per Electrode in Frequency Domain

mean	std
0: 0.039	0: 17.93165
260: 0.0063	1: 6.5147
255: 0.00426	252: 6.15574
1: 0.00404	253: 5.54902
266: 0.00368	254: 3.90495
...	...
257: -0.00546	499: 0.00545
261: -0.00608	498: 0.00507
7: -0.00747	500: 0.00501
253: -0.01052	501: 0.00164
254: -0.01735	251: 0.00164

Table J.13: Top 5 and Bottom 5 mean and standard deviation values of the split frequencies.

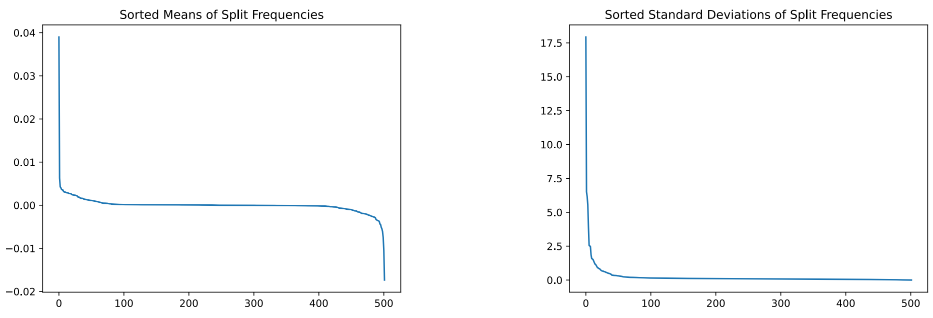


Figure J.13: Sorted means and standard deviations of the split frequencies.

J.3.2 Standardize per Split Frequency in Frequency Domain

Having a table and plots here would be silly since the whole point of the preprocessing is to set the split frequencies' means to zero and standard deviations to one.