Machine learning and deep learning approaches for multivariate time series prediction and anomaly detection

Thill, M.

Chapter 7

The Temporal Convolutional Autoencoder TCN-AE

7.1 Introduction

In this chapter, we present TCN-AE, a **t**emporal **c**onvolutional **n**etwork **a**utoencoder based on dilated convolutions. Similar to the other anomaly detection algorithms discussed in this thesis, TCN-AE trains completely unsupervised. In contrast to SORAD and LSTM-AD, TCN-AE is a reconstruction-based algorithm instead of a prediction-based one.

In the first part of the chapter, we describe a relatively simple baseline version of the algorithm (baseline TCN-AE) and demonstrate its capabilities by comparing it to other state-of-the-art algorithms on a Mackey-Glass (MG) anomaly benchmark (MGAB). Furthermore, we will see that our autoencoder is capable of learning interesting representations in latent space.

In the second part of this chapter, we analyze the architecture of the baseline TCN-AE, and we propose several enhancements that lead to TCN-AE in its current form. The final algorithm shows its efficacy on the real-world MIT-BIH [52, 112, 113] data of patients with cardiac arrhythmia, which we previously used in Chapter 6. Contrary to Chapter 6, we now consider the data of 25 patients (ECG-25) instead of 13 (ECG-13), almost doubling the amount of the benchmark data. TCN-AE also significantly outperforms several other unsupervised state-of-the-art anomaly detection algorithms on this comprehensive anomaly benchmark. Moreover, we investigate the contribution of the individual enhancements and show that each new ingredient improves the overall performance on the investigated benchmark.

As we have already seen in the previous chapter, it is rather challenging to learn the underlying structure of a system's normal behavior, especially if one has to deal with periodic or quasi-periodic signals with complex temporal patterns. In such environments, anomalies may be hard-to-detect deviations from the regular recurring pattern. Especially, for prediction-based algorithms it is challenging to predict small, but random variations in frequency/periodicity in quasi-periodic time series. ECG signals are a good example for such time series. Due to the heart rate variability (HRV), which is the variance in time between two heartbeats, it is rather difficult for a prediction-based algorithm to accu-

rately predict the exact position (in time) of, for example, the next R-peak (typically, the most characteristic peak in the signal). Even small shifts in predicted and actual R-peak cause large prediction errors which might falsely be interpreted as anomalous. To account for this variability in ECG signals, we introduced the window-based error correction for LSTM-AD in the previous chapter. Another way to approach this problem could be to use reconstruction-based (using an encoder and decoder architecture) anomaly detection algorithms instead of prediction-based ones. The advantage of reconstruction-based algorithms is that they do not require a time series to be predictable (in the sense of forecastable). Instead, they have a bottleneck which forces them to learn the underlying patterns of (nominal) time series. Most approaches found in the literature (see Chapter 3) are based on (temporal) autoencoders which take short sub-sequences from a time series, encode them into a latent-space vector, and attempt to reconstruct the sub-sequence based on the latent vector. However, these kind of reconstruction-based algorithms have the disadvantage that they are commonly limited to detecting anomalies in relatively short sub-sequences.

In this chapter, we propose a novel autoencoder architecture for sequences (time series), called TCN-AE, which is inspired by temporal convolutional networks [13] and shows its efficacy in unsupervised learning tasks. Contrary to other approaches, it does not encode and reconstruct short sub-sequences. Instead, it can compress a whole time series into a significantly shorter one, before reconstructing the original time series again. TCN-AE uses so-called dilated convolutional layers to naturally create a large receptive field and process a time series signal at different time scales. It consists of two parts, an encoder, and a decoder, which are both trained simultaneously and learn to find a compressed representation of the input time series (encoder) and reconstruct the original input again (decoder). Initially, we study a baseline version of TCN-AE. Our experiments show that the baseline architecture can learn interesting representations of sequences in latent space. When trained on (mostly) normal data, the approach can also be used for anomaly detection tasks by using reconstruction errors to predict anomaly scores. Only a small fraction of labeled data is needed to find a suitable threshold for the anomaly score. This can also be fine-tuned in operation with an already trained model.

For the initial benchmarking and comparison of our baseline algorithm, we use a synthetic benchmark based on Mackey-Glass (MG) time series [106]. In its current form, the Mackey-Glass Anomaly Benchmark (MGAB) consists of 10 MG time series in which anomalies were inserted using a clearly defined procedure. Although the anomalies are inserted synthetically, spotting them is rather difficult for the human eye. Due to the structured insertion process and the clear labeling of nominal and anomalous data, no domain knowledge is required to label the data correctly.

In the second part of the chapter we propose several enhancements for the baseline TCN-AE architecture. Then, we test our model on the more challenging ECG-25 dataset (introduced in Section 2.3.3), an anomaly benchmark consisting of 25 electrocardiogram time series with a length of half an hour.

We formulate the following research questions for this chapter:

- Can *unsupervised* deep learning models learn to detect anomalies?
- Which models are best to process the complex and long-range temporal patterns observed in periodic or quasiperiodic time series data?

The key findings of the research described in this chapter can be formulated as follows:

- Under certain (mild) assumptions, it is possible to train unsupervised Deep Learning (DL) models for anomaly detection. The novel autoencoder approach is essential for achieving this.
- It is essential to process the data on different time scales (like TCN and wavelets) and utilize the information from different time scales in the anomaly detection process.
- TCN-AE outperforms all other considered state-of-the-art algorithms by more than 10% on the ECG-25 benchmark (Table 7.6).

Several earlier works inspired the TCN-AE architecture that is presented in this chapter: While Holschneider et al. applied dilated convolutions in their "algorithme à trous" algorithm in the field of wavelet decomposition already in 1990 [65], more recently, they have also been applied to deep learning architectures, where the parallels to the non-decimating/stationary discrete wavelet transform (DWT) are still apparent: van der Oord et al. [124] introduced the WaveNet architecture, which uses dilated convolutions for the generation of raw audio. Yu & Koltun [182] successfully employed dilated convolutions to the task of semantic image segmentation. Later, Bai et al. [13] proposed a more general temporal architecture for sequence modeling, which they named temporal convolutional network (TCN). Our work is built upon the work of Bai et al. To the best of our knowledge, there is no earlier work that employs TCNs in an autoencoder-like architecture. We only found one approach for time series anomaly detection that is based on TCNs [61]. However, it does not use autoencoders. Its general idea is more similar to [107] and [161], which use forecasting errors as an indication for anomalous behavior. Further related work concerened with electrocardiography and anomaly detection in ECG signals is described in the previous Chapter 6.

The rest of this Chapter is organized as follows: Section 7.2 introduces the dilated convolution operation while Section 7.3 presents the baseline TCN-AE architecture and describes several experiments with Mackey-Glass time series. Section 7.4 proposes several enhancements for TCN-AE and discusses the results for extensive experiments on the ECG-25 dataset. Finally, we conclude this work in Section 7.5.

7.2 Methods

In the following, we introduce the Temporal Convolutional Network Autoencoder (TCN-AE), describe its main components, and discuss a few of its properties and application areas for time series analysis. We will start with a baseline architecture and then later successively

add several enhancements to this architecture. As the name suggests, TCN-AE is a convolutional neural network architecture. Convolutional neural networks (CNNs) are broadly and with great success used in computer vision applications, where other fully connected/dense architectures commonly suffer from the curse of dimensionality. Convolutional nets have several useful properties such as translation invariance, weight (parameter) sharing, and computational efficiency, making them especially beneficial for computer vision tasks such as image recognition, segmentation, or object detection. Their properties are also helpful for time series processing, where typically 1D-convolutions are employed. Several architectures, such as WaveNet [124], or temporal convolutional networks [13] take advantage of the convolutional approaches developed for computer vision and adopt some ideas into the time domain.

7.2.1 Intuition

TCN-AE is designed to learn how to encode or compress a sequence into a significantly shorter sequence (using an encoder network) and subsequently reconstruct the original sequence from the compressed representation (using a decoder network).

The central idea is to create a bottleneck in the architecture that forces the network to identify and capture the most useful (temporal) patterns in the raw input data and translate them into efficient encodings. The encoded data should contain all the essential information, allowing an accurate reconstruction of the original input. Ideally, the autoencoder learns to ignore signal noise, redundancies, and other irrelevant information.

Conceptually, TCN-AE is similar to other classical (deep) autoencoder architectures. The most common autoencoder architectures encode fixed-sized inputs into a latent space representation and then use the latent variables to reconstruct the original input. Similarly, the TCN-AE encodes sequences along the temporal axis into representation and then attempts to reconstruct the original sequence. However, it differs from regular autoencoders in so far in that it replaces the fully connected/dense layers with dilated 1D-convolutional layers. Thus, the network can consider temporal relationships in the data more naturally and flexibly regarding variable-size inputs. Furthermore, the temporal receptive field of TCN-AE can be easily scaled and grows exponentially with an only linear increase in the number of weights, which is especially important for time series containing long intricate temporal patterns. Another advantage over other autoencoders is that TCN-AE (due to the shared weights) potentially has fewer weights than dense AE architectures.

This idea can be used for several applications. The applications of (temporal) autoencoders are very diverse. In this work, we will focus on anomaly detection in time series. Other applications could be time series (sequence) compression or representation learning [166], as we will investigate in Section 7.3.2.2.

7.2.2 Dilated Convolutions

Convolutional layers in neural networks comprise digital filters, which remove or amplify individual components (frequencies) in a presented signal (for example, an image or a time series). Formally, the filtering process can be described by the convolution operation. For a one-dimensional signal $x[n]$ ($x[n]$ being the n-th element in the signal), $x : \mathbb{T} \to \mathbb{R}$, where $\mathbb{T} = \{0, 1, \ldots, T-1\}$, the convolution with a (finite impulse response) filter $h[n]$, $h : \{0, 1, \ldots, k-1\} \to \mathbb{R}$ is usually defined as:

$$y[n] = (x * h)[n] = \sum_{i=0}^{k-1} h[i] \cdot x[n-i], \tag{7.1}$$

where $y[n] \in \mathbb{R}$ is the output of the filter, $h[i] \in \mathbb{R}$ is the i-th filter weight and k specifies the length of the filter. The convolution operation can be thought of as sliding a window of length k, which contains the filter weights $h[i]$, over the input sequence $x[n]$ and computing a weighted average of $x[n]$ with the weights $h[i]$ in each time step. The resulting output signal is one-dimensional and of length $T - k + 1$. In order to obtain an output signal of the same length, the input sequence is usually padded with zeros before applying the filter. Since the filter is only slid along the time axis, the operation is usually referred to as one-dimensional convolution. The behavior of the filter is determined by $h[n]$ (e.g., low-pass or high-pass characteristics), and the central idea of convolutional neural networks is not to pre-determine $h[n]$ but rather to learn suitable filter weights based on the learning task.

Convolutional layers in neural networks usually deal with multivariate input signals $\mathbf{x}[n]$ of dimension d, with $\mathbf{x} : \mathbb{T} \to \mathbb{R}^d$. In this case, each dimension $\mathbf{x}_j[n]$ is convolved separately with its own sub-filter $\mathbf{h}_j[n]$, $\mathbf{h} : \{0, 1, \ldots, k-1\} \to \mathbb{R}^d$, and $y[n]$ (remaining one-dimensional) is a dot product:

$$y[n] = (\mathbf{x} * \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \cdot \mathbf{x}[n-i]. \tag{7.2}$$

In contrast to the regular convolution operation (as specified above), the dilated convolution [182] has an additional parameter, the so called dilation rate $q \in \mathbb{N}$. It defines how many elements in the input signal $\mathbf{x}[n]$ are skipped between filter tap $\mathbf{h}[i]$ and filter tap $\mathbf{h}[i+1]$. The dilated convolution is written as:

$$y[n] = (\mathbf{x} *_q \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \cdot \mathbf{x}[n - q \cdot i]. \tag{7.3}$$

For $q = 1$ the original convolution operation is obtained.

In many applications also acausal convolutions are used. In this case, future values of a sequence $x[n]$ will be processed to generate output $y[n]$:

$$y[n] = (\mathbf{x} *_q \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \cdot \mathbf{x}\big[n - q \cdot \lfloor i - k/2 \rfloor\big] \tag{7.4}$$

In this work, we experimented with causal and acausal convolutions for TCN-AE and found acausal convolutions to produce slightly better results for the investigated anomaly detection tasks (MGAB & ECG-25). Note that this comes at the cost of slight delays in online settings.

When using dilated convolutions, one has to consider that a few properties of the frequency response of the system also change: the system function of a regular convolution ($q = 1$) for a one-dimensional input signal is given as

$$H(z) = \sum_{m=0}^{k-1} h[m] \cdot z^{-m}.$$

If we evaluate the system function $H(z)$ on the unit circle described by $z = \mathrm{e}^{j\hat{\omega}}$, we get the frequency response of the system. The parameter $\hat{\omega} = \frac{\omega}{f_s}$ is the so-called normalized radian frequency, which is obtained by dividing the radian frequency ω by the sampling frequency f_s. In this case, the Nyquist frequency is π. For the dilated convolution, the system function becomes:

$$H(z^q) = \sum_{m=0}^{k-1} h[m] \cdot z^{-q \cdot m}. \tag{7.5}$$

This means that a dilation rate $q > 1$ implicitly increases the filter order and adds extra poles and zeroes to the system. For example, if $q = 2$, then the filter order is doubled from $M = k - 1$ to $M = 2(k - 1)$ and at the same time $2(k - 1)$ zeroes/poles are added to the system. For $q = 4$, the filter order is multiplied by a factor of 4 and so on. Effectively, the frequency response becomes "sharper" for larger q and the filter is more sensitive to small changes in the frequency. In fact, the frequency response of the filter has a periodicity of $\frac{2\pi}{q}$. This can be easily verified if we insert $z = \mathrm{e}^{j(\hat{\omega} + \frac{2\pi}{q})}$ into Eq. (7.5):

$$\begin{aligned} H(z^q)\Big|_{z=\mathrm{e}^{j(\hat{\omega}+\frac{2\pi}{q})}} &= \sum_{m=0}^{k-1} h[m] \cdot \left(\mathrm{e}^{j(\hat{\omega}+\frac{2\pi}{q})}\right)^{-q \cdot m} \\ &= \sum_{m=0}^{k-1} h[m] \cdot \mathrm{e}^{-j\hat{\omega}qm} \mathrm{e}^{-j(2\pi)m} \end{aligned}$$

$$= \sum_{m=0}^{k-1} h[m] \cdot \mathrm{e}^{-j\hat{\omega}qm}$$
$$= H(z^q)\Big|_{z=\mathrm{e}^{j\hat{\omega}}}$$

Figure 7.1 illustrates the discussed properties of dilated convolutions for several filters that TCN-AE (described below) learned during its training. One can clearly see how the frequency response evolves for increasing q. Note that the periodicity of the frequency responses, as shown in Figure 7.1, are solely defined by the dilation rate q. For example, this means that for $q = 8$ we will – independently from the filter length and the filter weights – always have a periodicity of $\frac{\pi}{4}$ for the frequency response, resulting in a "rougher" landscape than for $q = 1$. This realization was one of the reasons for introducing skip connections in the architecture in order to reuse the outputs of different dilated convolutional layers (as discussed in more detail in Section 7.4.1.1).

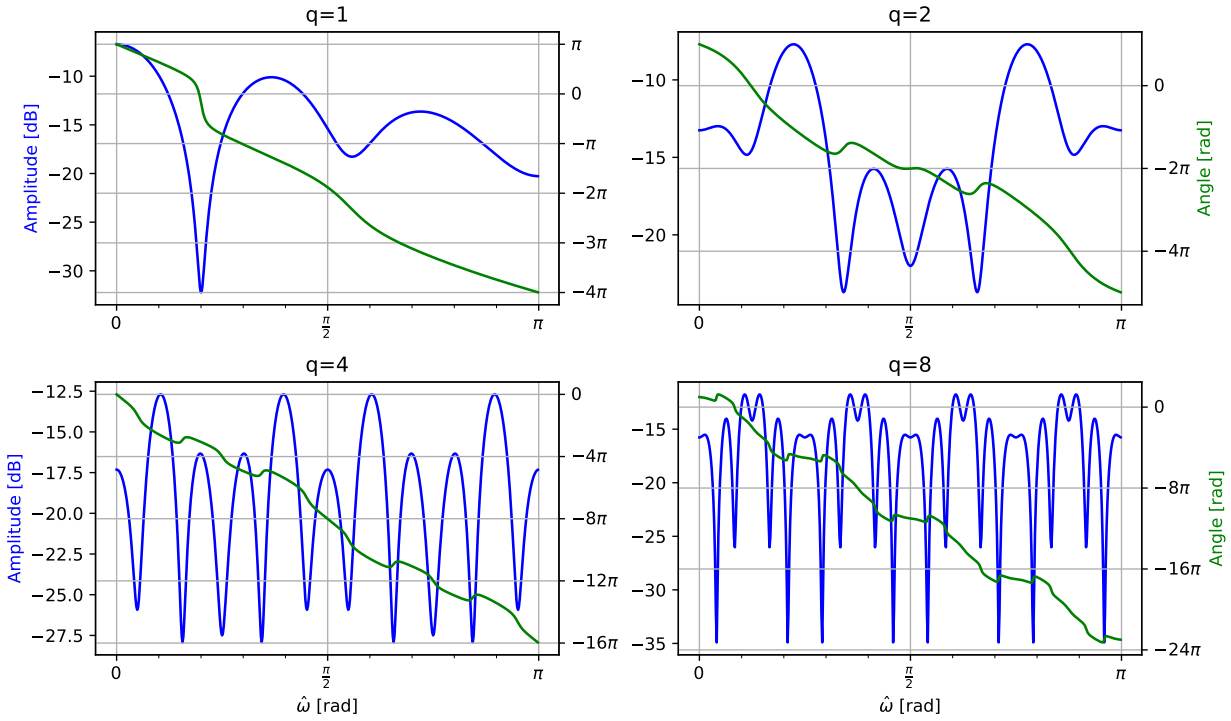

Figure 7.1: Illustration of the frequency responses of several filters taken from the first four dilated convolutional layers of TCN-AE (which is described in more detail below). Several properties, mentioned in Section 7.2.2, can be observed: For example, with increasing dilation rate q, the amplitude of the frequency response becomes "sharper" and the filter is more sensitive to small changes in the frequency of the input signal. This is because dilation rates larger than $q > 1$ implicitly increase the order of the filter, and additional zeros/poles are introduced in the complex frequency-domain representation of the filter. Also, the phase (angle) of the displayed filters is as expected not linear since the symmetry of the filter weights is generally not given in a dilated convolutional layer.

7.2.3 Dilated Convolutional Layers in Neural Networks

The previous section described how a one-dimensional output signal $y[n]$ is computed using a filter. In practice, a convolutional layer is typically comprised of many discrete filters, and the individual outputs $y[n]$ are stacked into a so-called feature map. If a signal $\mathbf{x}[n]$ of length T_{train} is passed through a convolutional layer with n_{filters} filters, the resulting feature map has the dimension $T_{\text{train}} \times n_{\text{filters}}$ (for a padded signal). The weights $\mathbf{h}[i]$ of each filter are considered learnable parameters, commonly trained using variants of the back-propagation algorithm.

Many neural network architectures for sequence modeling (e.g., [13, 124]) utilize dilated convolutions to create a hierarchical temporal model with a large receptive field, capable of learning long-term temporal patterns in the input data. The main idea is to build a stack of dilated convolutional layers, where the dilation rate increases with each added layer. A common choice is to start with a dilation rate of $q = 1$ for the first layer of the network and double q with every new layer. With this approach, we can increase the receptive field of the model exponentially. In general, the receptive field r for the causal and acausal case is given by:

$$r_{causal} = k \cdot 2^{L-1}, \tag{7.6}$$

$$r_{acausal} = \lfloor k/2 \rfloor \cdot (2^{L+1} - 2) + 1, \tag{7.7}$$

where $L > 0$ is the number of layers. If, for example, we build a stack of $L = 5$ dilated convolutional layers with a kernel size of $k = 3$, the receptive field's size will be $3 \cdot 2^4 = 48$ for the causal case and $2^6 - 1 = 63$ for the acausal setting. The size of the receptive field should be considered when choosing the length of the training sequences. For example, the receptive field should not be larger than the length of the training sequences.

In summary, a convolutional layer can be mainly described by three parameters: The dilation rate q, the number of filters n_{filters}, and the kernel size (filter length) k. A convolutional layer maps an input sequence $\mathbf{x} : \mathbb{T} \to \mathbb{R}^d$ to an output sequence $\mathbf{y} : \mathbb{T} \to \mathbb{R}^{n_{\text{filters}}}$. Note, that the shape of the output does not depend on k.

Relation between Dilated Convolutions and the DWT The non-decimating discrete wavelet transform (DWT) is, in some sense, related to dilated convolutional neural network architectures. The regular DWT decomposes a time series into so-called approximation and detail coefficients. By repeated filtering of the input with low-pass and high-pass filters, one obtains a hierarchical representation of the original signal on different frequency scales.

While the regular DWT downsamples (decimates) the signal after every low-pass filter by a factor of two, the non-decimating DWT removes all downsampling units. In turn, the filters have to be dilated. The dilation rate (which is a power of two) specifies the gaps between the filter taps. The non-decimating DWT is usually used in applications where one wants to achieve translation invariance (at the cost of redundancy). Holschneider et al. [65]

proposed an efficient algorithm for computing the non-decimating DWT using dilated convolutions. Current deep learning architectures [124, 182, 13] based on dilated convolutional layers are inspired by the earlier work of Holschneider et al. Dilated convolutional nets also repeatedly filter a signal (e.g., time series or image) in a stack of convolutional layers. The dilation rate q is usually doubled with every further layer.

There are also apparent differences: (a) The DWT filter weights depend on the mother-wavelet choice and are fixed, while the weights of convolutional layers are learnable parameters. (b) The DWT does not use non-linear activation functions such as rectified linear units (ReLU).

The baseline TCN-AE consists of two temporal convolutional neural networks (TCNs) [13], one for encoding and one for decoding. Additionally, a downsampling and upsampling layer are used in the encoder and decoder, respectively. The individual components will be described in more detail in the following.

7.2.4 Temporal Convolutional Networks

The temporal convolutional network (TCN) [13] is inspired by several convolutional architectures [36, 48, 73, 124], but differs from these approaches insofar as it combines simplicity, auto-regressive prediction, residual blocks, and very long memory. Essentially, a TCN is a stack of n residual blocks. Each block consists of two serial sub-blocks, and each sub-block is comprised of the following layers: a dilated convolutional layer, followed by a weight normalization layer [145], a ReLu activation function [116], and a spatial dropout layer [152]. Furthermore, a skip (residual) connection [60] bypasses the residual block and is added to the residual block's output. A TCN is mainly described by three parameters: a list of dilation rates $(q_1, q_2, \ldots, q_L)$, the number of filters n_{filters}, and the kernel size k. The output of each residual block and the final output is a sequence $\mathbf{y} : \mathbb{T} \to \mathbb{R}^{n_{\text{filters}}}$. A full description of TCN would be out of scope for this chapter. The reader is referred to [13] for the details.

7.2.5 Unsupervised Anomaly Detection with TCN-AE

A natural application of TCN-AE is the anomaly detection in time series. Although we have not yet discussed the architecture of TCN-AE in detail, we can already describe how its reconstruction errors are used to detect anomalous patterns. Due to the bottleneck in the architecture, the training procedure forces TCN-AE to learn compressed encodings of the input sequences, which capture the underlying structure of the data and allow accurate reconstruction. Intuitively, we expect that TCN-AE reconstructs recurring nominal patterns in a time series with only small errors. It focuses on minimizing the reconstruction error of the nominal data that are in the vast majority during training. On the other hand, when TCN-AE observes patterns that significantly differ from the norm, we expect higher reconstruction errors.

To discover abnormal behavior, we slide a window of length ℓ over the reconstruction error and collect the ℓ-dimensional vectors in an error matrix $\mathbf{E}$. The purpose of the sliding

window is to smoothen noisy events that might occasionally appear. The error matrix $\mathbf{E}$ is passed to the outlier detection algorithm, which identifies unusual points in the ℓ-dimensional space. The outlier detection algorithm outputs an anomaly score, which is later thresholded. We experimented with different outlier detection algorithms and found that a simple approach based on the Mahalanobis distance (line 15 in Algorithm 8) delivers the best results. An advantage of the Mahalanobis distance is that it is parameter-free and does not require any particular assumptions about the data distribution (such as normality). The Mahalanobis distance only requires the invertibility of the covariance matrix. However, in practice, there are rarely situations where the covariance matrix is non-invertible. We summarize the anomaly detection algorithm for TCN-AE in Algorithm 8.

Note that although we train TCN-AE with the complete time series, the overall anomaly detection algorithm consisting of TCN-AE and Mahalanobis distance calculation is entirely unsupervised. The training procedure does not pass anomaly labels to the algorithm at any time. Only for selecting an appropriate anomaly threshold on the Mahalanobis distance, we permit all algorithms to use 10% of the anomaly labels, as described later in Sec. 7.4.2.2.

7.3 A Baseline Version of TCN-AE

7.3.1 The Baseline TCN-AE Architecture

For our initial experiments, we use TCN as a building block for a baseline temporal autoencoder, referred to as baseline TCN-AE. In later sections, we will modify the baseline TCN-AE, add further enhancements to the architecture, and analyze their contribution to the final TCN-AE architecture. The baseline TCN-AE consists of an encoder network $\text{enc}(\cdot)$ and a decoder network $\text{dec}(\cdot)$.

The encoder $\text{enc}(\cdot)$, shown in Fig. 7.2, left, attempts to generate a compact representation that captures the main characteristics of the input sequences and allows a reasonably good reconstruction in later steps. In order to learn the important features in a sequence, it is necessary to identify short-term as well as long-term patterns. The encoder takes an input sequence, passes it through a TCN network, reduces the dimension of the feature map by applying a 1×1 convolutional layer[1] [96, 156] and finally down-samples the series along the time axis by a specified factor using an average-pooling layer. It does so by averaging groups of size s along the time axis. The number of filters c in the 1×1 convolution layer specifies the dimension of the encoded representation and the sample rate s determines the factor, by which the length T of the series is reduced. Hence, the original input $\mathbf{x}[n]$ will be compressed into an encoded representation $\mathbf{H}[n] = \text{enc}(\mathbf{x}[n])$, where $\mathbf{H} : \{0, 1, \ldots, T/s - 1\} \to \mathbb{R}^c$.

The decoder $\text{dec}(\cdot)$, shown in Fig. 7.2, right, attempts to reconstruct the original input sequence, using the output of the encoder as input. First, the length of the original series has to be restored. We use a simple sample-and-hold interpolation for this purpose, which

[1] A 1×1 convolution is a weighted average over all feature maps, taken at every time point. The weights are learnable parameters.

Algorithm 8 General anomaly detection algorithm using the TCN-AE architecture. The estimation of $\bar{\mathbf{x}}$ and $\mathbf{\Sigma}$ might also have to be computed in batches according to the method described in Sec. B.2.1.

Adjustable parameters:
  $\mathcal{M}_\tau$: anomaly threshold (see Sec. 7.4.2.1), ℓ: error window length
  T_{train}: length of training sub-sequences, B: batch size

function ANOMALYDETECT($\mathbf{x}[n]$) ▷ $\mathbf{x} : \mathbb{T} \to \mathbb{R}^d$, $\mathbb{T} = \{0, 1, \ldots, T-1\}$
  Construct model TCNAE() and **Initialize** the trainable parameters
  $\mathbf{X}_{\text{train}} \leftarrow$ { Sub-sequences of length T_{train} taken from $\mathbf{x}[n]$ }
  for $\{1 \ldots n_{\text{epochs}}\}$ **do**
    TRAIN(TCNAE, $\mathbf{X}_{\text{train}}$) ▷ Train with random mini-batches of size B
  $\hat{\mathbf{x}}[n] \leftarrow$ TCNAE($\mathbf{x}[n]$) ▷ Encode and reconstruct $\mathbf{x}[n]$
  $\mathbf{e}[n] \leftarrow \mathbf{x}[n] - \hat{\mathbf{x}}[n]$ ▷ reconstr. error $\mathbf{e} : \mathbb{T} \to \mathbb{R}^d$
  $\mathbf{E}[n] \leftarrow$ SLIDINGWINDOW($\mathbf{e}[n]$, ℓ) ▷ $\mathbf{E} : \mathbb{T} \to \mathbb{R}^{\ell \times d}$
  $\mathbf{E}'[n] \leftarrow$ RESHAPE($\mathbf{E}[n]$) ▷ $\mathbf{E}' : \mathbb{T} \to \mathbb{R}^{\ell \cdot d}$
  $\boldsymbol{\mu}, \mathbf{\Sigma} \leftarrow$ ESTIMATE($\mathbf{E}'[n]$) ▷ Mean $\boldsymbol{\mu} \in \mathbb{R}^{\ell \cdot d}$, Cov. Mat. $\mathbf{\Sigma} \in \mathbb{R}^{\ell \cdot d \times \ell \cdot d}$
  $M[n] \leftarrow (\mathbf{E}'[n] - \boldsymbol{\mu})^\intercal \mathbf{\Sigma}^{-1} (\mathbf{E}'[n] - \boldsymbol{\mu})$ ▷ Mahalanobis distance
  $a[n] \leftarrow \begin{cases} 0 & \text{if } M[n] < \mathcal{M}_\tau \\ 1 & \text{else} \end{cases}$ ▷ Binary anomaly flags
  return $a[n]$ ▷ Return anomaly flag for each time series point

duplicates each point in the series s times. Subsequently, the upsampled sequence is passed through a second TCN block, which has the same structure as the TCN block in the encoder (but untied/independent weights). Finally, to restore the original dimension d, another 1×1-convolutional layer with d filters is used to obtain the reconstruction (the output) $\hat{\mathbf{x}}[n] = \text{dec}(\mathbf{H}[n])$, $\hat{\mathbf{x}} : \mathbb{T} \to \mathbb{R}^d$. The architecture of the baseline TCN-AE is depicted in Figure 7.2. Once TCN-AE is trained, the input sequence and its reconstruction will be used for detecting anomalies, as described in the next section.

7.3.2 Initial Experiments

7.3.2.1 Experimental Setup

Anomaly Detection Algorithms As before, all training algorithms are unsupervised, i. e. they do not need the true anomaly labels during the training process. Only in order

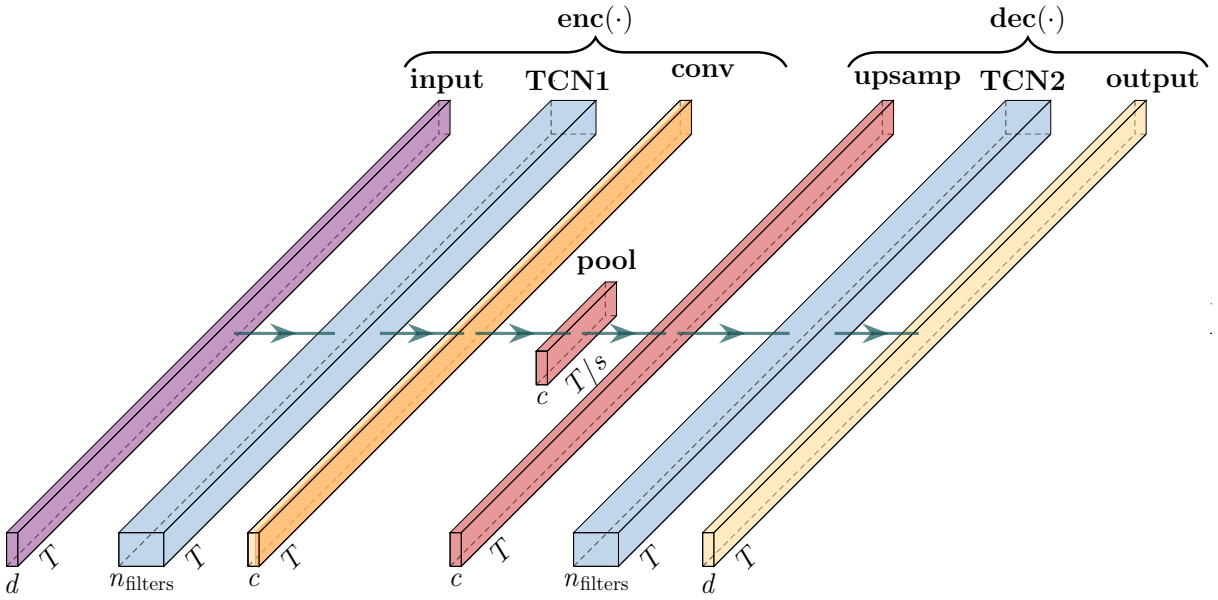

Figure 7.2: Architecture of the baseline TCN-AE as described in Section 7.3.1. The input of TCN-AE is a sequence $\mathbf{x}[n]$ with length T and dimensionality d. The layers "conv" and "output" are 1×1 convolutions with c and d filters, respectively. The TCNs have the dilation rates q . The layer "pool" downsamples the sequence by a factor s. *Configuration for MGAB:* $d = 1$, $c = 8$, $n_{\text{filters}} = 20$, and $s = 42$. *Configuration for ECG-25 benchmark:* $d = 2$, $c = 4$, $n_{\text{filters}} = 32$, and $s = 32$.

to find a suitable anomaly threshold, a small fraction of labels is used, as described below. Otherwise, the anomaly labels are only used at test time to evaluate the performance of the individual algorithms. In one run, each algorithm is trained for ten rounds: in the i-th round, the algorithms are trained on the i-th time series and evaluated on the time series $\{1, \ldots, 10\} \setminus \{i\}$. In total, we perform ten runs with different random seeds. In order to find suitable hyper-parameters for each algorithm, we use the HYPEROPT library [14] and optimize the F_1-score on a separate MG time series. For all neural networks, we use the Adam optimizer [82] to train the weights by minimizing the MSE loss. Additionally, all time series (having a dimension of $d = 1$) are standardized to zero mean and unit variance.

DNN-AE [46]: we use a PyTorch [130] implementation for the anomaly detection algorithm based on a deep autoencoder [58]. The algorithm requires several parameters, which we choose as follows: batch size $B = 100$, number of training epochs $n_{\text{epochs}} = 40$, sequence length $T_{\text{train}} = 150$ and a hidden size of $h = 10$ for the bottle neck (which results in a compression factor of $T_{\text{train}}/h = 15$ for each sequence). Finally, we set $\%_{Gaussian} = 1\%$, which specifies that 99% of the data is used to estimate a Gaussian distribution for the anomaly detection task.

LSTM-ED [108] is also implemented using PyTorch and uses the following parameter setting: batch size $B = 100$, number of training epochs $n_{\text{epochs}} = 20$, sequence length

$T_{\text{train}} = 300$, hidden size $h = 100$ and $\%_{Gaussian} = 1\%$. Both, encoder and decoder use a stacked LSTM network with two layers.

NuPIC [160]: Numenta's anomaly detection algorithm has a large range of hyper-parameters which have to be set. We use the parameters recommended by the authors in [89]. It is possible to tune the parameters with an internal swarming tool [3]. However, this is a time-expensive process which is not feasible for the large MGAB dataset.

LSTM-AD [161]: here we select the following parameters: batch size $B = 1024$, number of training epochs $n_{\text{epochs}} = 30$, and sequence length $T_{\text{train}} = 128$. A 2-layer LSTM network with 256 units in the first layer and 128 units in the second layer is used. The target horizons are chosen to be $H = (1, 3, \ldots, 51)$.

TCN-AE (baseline): The main TCN-AE parameters are given in Fig. 7.2. Additionally we use the sequence length $T_{\text{train}} = 1050$, batch size $B = 32$ and $n_{\text{epochs}} = 40$. For baseline TCN-AE, we use an existing TCN implementation in Keras [140]. The dilation rates are $q = (1, 2, \ldots, 16)$ and the kernel size of the TCNs is set to $k = 20$.

We determine this threshold for all algorithms as follows: A sub-sequence containing 10% of the data is taken, and the anomaly threshold is optimized on this short sequence, such that the F_1-score is maximized. The optimal threshold is then fixed for the complete time series, and the overall results are obtained. Since the results can vary depending on which sub-sequence is used for the threshold adjustment, we repeat the above procedure, similarly to k-fold cross validation, for ten different 10% sub-sequences of the considered time series and record the results for the ten different sub-sequences.

7.3.2.2 Learning Time Series Representations

In our first experiment, we want to assess the capabilities of the TCN-AE architecture to learn representations of time series. For this purpose, we train a TCN-AE model using many different MG time series with a varying time delay parameter τ. Ideally, TCN-AE should learn the main characteristics of the individual time series and find suitable compressed representations. In our experiment, we use TCN-AE on 10^5 different Mackey-Glass time series (10^4 for each τ in the range of $\tau = 11 \ldots 20$). Each time series of length 256 is encoded into a 2-dimensional compressed representation. The algorithm is trained in an unsupervised manner. Hence, τ is not passed to the algorithm at any time. Surprisingly, even with this large compression rate of 128, TCN-AE can find an interesting embedding for the MG time series, as depicted in Fig. 7.3 (top). For a certain τ, all samples are placed in *only one* connected cluster (except for a few satellites), and these clusters are mostly – with a few small exceptions – *non-overlapping*.

For comparison, we repeated the same experiment with the popular t-SNE [105] clustering algorithm. We executed t-SNE on a GPU with the help of a certain CUDA implementation [25]. We tried different parameter settings and finally fixed the perplexity parameter to 200, the learning rate to 10, and the number of iterations to 10^4. The results for t-SNE in Fig. 7.3 (bottom) indicate that it is not a trivial task to find suitable representations

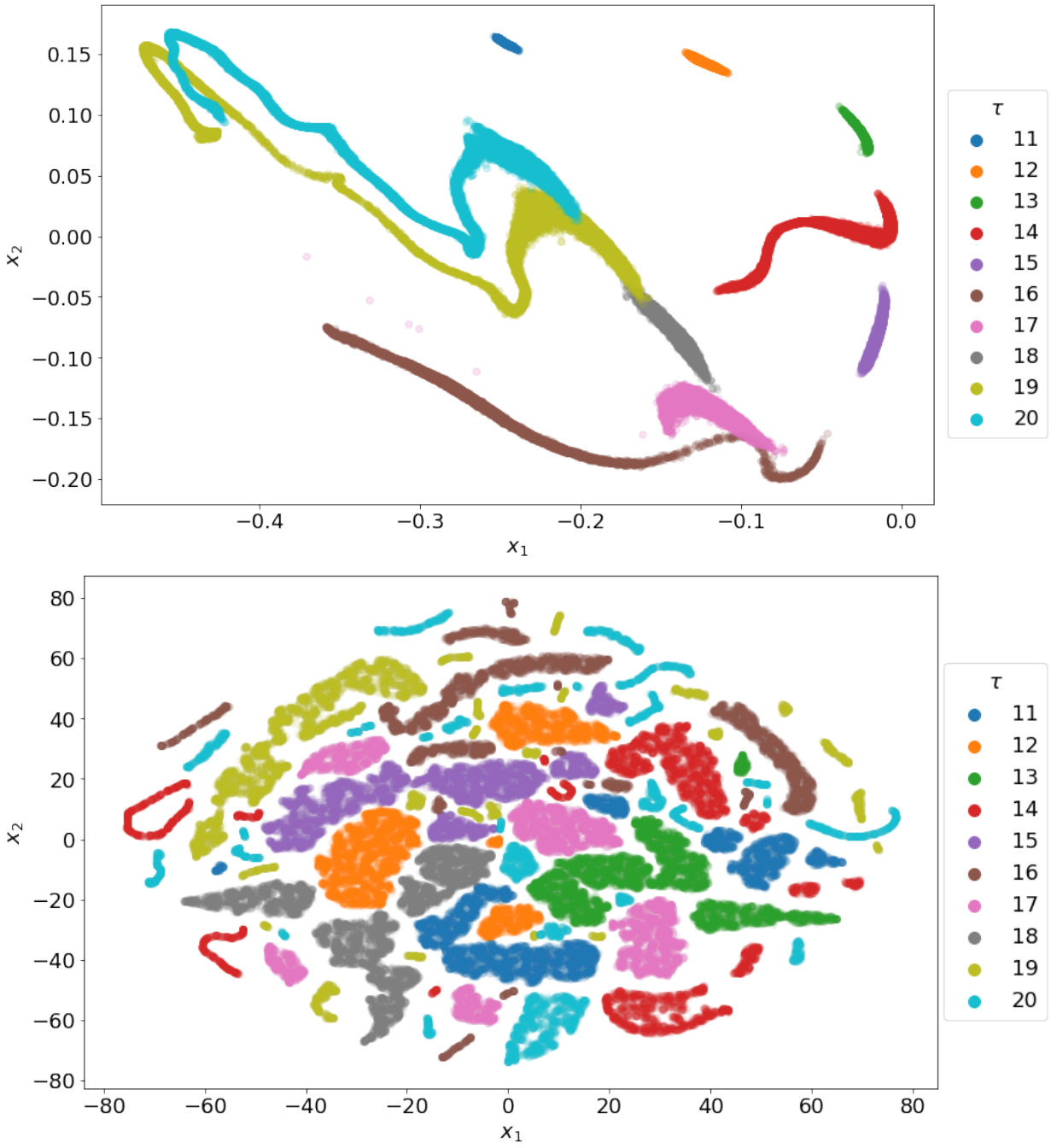

Figure 7.3: Top: 2d-representation of 10^5 (10^4 for each τ) different Mackey-Glass time series using TCN-AE (baseline). The (unsupervised) algorithm is capable of learning an encoding which separates the MG time series fairly well according to their τ value.
Bottom: 2d-representation of the same MG time series, but now using t-SNE [105] to find suitable encodings.

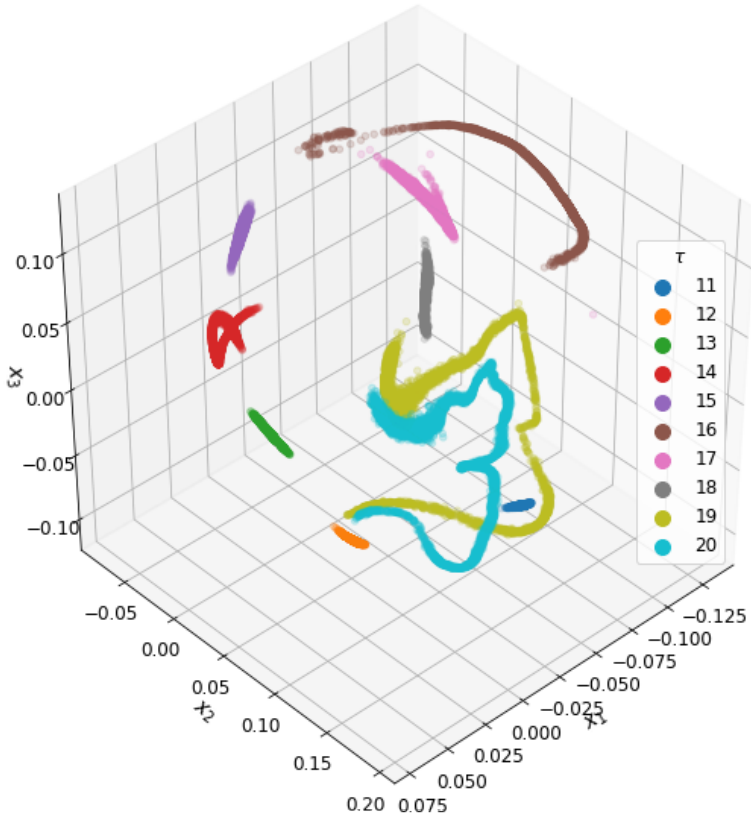

Figure 7.4: Similar to Fig. 7.3 (top). But now we encode each MG time series into a 3d-vector.

for MG time series. t-SNE has more difficulties in comparison to TCN-AE to cluster all sequences with a particular time delay parameter τ in only *one* connected region.

7.3.2.3 Anomaly Detection on the Mackey-Glass Anomaly Benchmark

In a second experiment, we compare TCN-AE to several state-of-the-art anomaly detection algorithms on the Mackey-Glass Anomaly Benchmark. For each algorithm, except NuPIC, ten runs were performed. Hence, for each algorithm and time series, ten different models are trained, and each model is evaluated on the other nine time series. NuPIC is entirely deterministic and does not require several runs. Additionally, as described in Section 7.3.2.1, the anomaly threshold for each algorithm and time series is tuned on ten different subsequences. We add up the TP, FN, and FP over all ten time series and summarize the results in Tab. 7.1. Up to 100 anomalies can be detected in total. We can see that the (deep) DNN-AE detects most of the anomalies (approx. 92), missing only about eight on average. However, this result is achieved at the expense of producing many false-positives. Overall, DNN-AE produces more than 60 false positives on average, while TCN-AE produces

less than one. Hence, DNN-AE achieves the highest recall among all algorithms but ranks only 3$^{\text{rd}}$ in F_1-score, due to its low precision. TCN-AE scores best in F_1-score and precision. NuPIC has the poorest performance in all measures.

Table 7.1: Results for MGAB. The results shown here (mean and standard deviation of 10 runs and ten sub-sequences, are for the sum of TP, FN, and FP over all ten time series. For each algorithm and time series, the anomaly threshold was tuned on 10% of the data using a cross-validation approach: the threshold is tuned on ten different 10%-sequences of the data.

Algorithm	TP	FN	FP	Precision	Recall	F_1-score
NuPIC [160]	3.00	97.00	132.00	0.02	0.03	0.03
LSTM-ED [108]	14.60 ± 5.86	85.40 ± 5.86	57.00 ± 20.43	0.21 ± 0.08	0.15 ± 0.06	0.17 ± 0.06
DNN-AE [58]	91.79 ± 1.22	8.21 ± 1.22	62.58 ± 13.65	0.60 ± 0.06	0.92 ± 0.01	0.72 ± 0.04
LSTM-AD [161]	88.80 ± 2.59	11.20 ± 2.59	0.62 ± 0.61	0.99 ± 0.01	0.89 ± 0.03	0.94 ± 0.01
TCN-AE [this work]	90.54 ± 1.72	9.46 ± 1.72	0.20 ± 0.47	1.00 ± 0.01	0.91 ± 0.02	0.95 ± 0.01

7.3.3 Discussion

The initial results that we obtained with our new TCN-AE architecture are promising. The learned representations (Fig. 7.3) on different MG time series appear to be useful and may reveal interesting insights. For anomaly detection, we achieve with TCN-AE and LSTM-AD the highest F_1-score on the non-trivial MG benchmark. Remarkably, all algorithms except NuPIC require many trainable weights. TCN-AE had 164 451 parameters, DNN-AE 241 526, LSTM-ED 244 101 and LSTM-AD 464 537. That is, the other high-performing algorithms require 50%–300% more trainable weights than TCN-AE.

Generally, we would expect TCN-AE to perform better than, for example, DNN-AE on tasks where a larger receptive field (memory) is required in order to detect anomalies since its hierarchical architecture allows to exponentially increase the receptive field while the number of parameters scales linearly.

Although the initial results of TCN-AE on MGAB look promising and although we could observe that the algorithm is capable of learning representations of MG time series, there are several limitations of the algorithm, which leave room for improvement and which we are planning to address in the future work: (1) Many parameters (approximately 160 000) are required for satisfactory MGAB results. TCN-AE's performance significantly drops if the number of filters n_{filters} and/or the kernel size k is reduced. (2) Baseline TCN-AE is somewhat sensitive towards the maximum dilation rate $q_{\max}$. For example, if we add a dilated convolutional layer with $q_{\max} = 32$ to both TCNs in the architecture, the performance significantly drops. (3) The net requires relatively many epochs until it learns the subtle differences between nominal and abnormal MG time-series patterns. TCN-AE requires ≥ 40 training epochs to learn to detect anomalies on the MG time series. It has to

be investigated if this holds for other (real-world) applications as well and if optimizations of the training-configuration might reduce the required epochs.

7.3.4 Summary

In this section, we proposed with TCN-AE an autoencoder architecture for multivariate time series. We evaluated it on various Mackey-Glass (MG) time series for two relevant tasks: representation learning and anomaly detection. The initial results on various Mackey-Glass (MG) time series are promising. TCN-AE could learn a very interesting representation in only two dimensions, which accurately distinguishes MG time series differing in their time delay values τ (Section 7.3.2.2). On the Mackey-Glass Anomaly Benchmark (MGAB), TCN-AE achieved better anomaly detection results than other state-of-the-art anomaly detectors (Section 7.3.2.3).

In the following section, we will address the limitations of baseline TCN-AE mentioned before and propose several extensions to improve the overall performance of TCN-AE.

7.4 An Improved TCN-AE Architecture

The goal of this section is to improve the baseline TCN-AE architecture based on the limitations discussed in the previous section. Overall, we suggest six modifications. We analyze, discuss, and compare the capabilities of the improved TCN-AE architecture on a challenging real-world HMS application, namely the detection of arrhythmias in electrocardiogram (ECG) signals of heart patients.

7.4.1 Enhancements of the Baseline TCN-AE

7.4.1.1 Skip Connections

While experimenting with the encoder and decoder's dilation rates, we noticed that the performance of the baseline TCN-AE is somewhat sensitive to the choice of the maximum dilation rate q_{max}. We believe that this problem occurs because only the TCN's final dilated convolutional layer is passed on to the following layer, i.e., the original TCN does not provide any mechanisms for feature reuse. However, especially for TCNs, which process a time series signal at different time scales, it might be detrimental to solely use the last dilated convolutional layer's output, since other time scales might also carry essential information. Instead, it should be possible to access the features at all time scales.

To provide for the possibility of feature reuse in TCN-AE, we add so-called skip connections to our architecture. A skip connection copies the output of a particular layer and concatenates it to the input of a subsequent layer of the network. In our setup, we use a concatenation layer in the end of encoder and decoder, which collects the outputs of all previous dilated convolutional layers.

In the encoder shown in Fig. 7.5, we add skip connections from every dilated convolutional layer to the encoder's bottleneck (after reducing the number of channels to 16), where the outputs of the individual layers are concatenated along the channel axis. The bottleneck reduces the number of channels of the concatenated outputs with a 1×1-convolution and downsamples the resulting signal to obtain a compressed encoding.

In the decoder shown in Fig. 7.6 we also place skip connections from each dilated convolutional layer to the output, where lastly, a 1×1 convolution reconstructs the time series with the original dimension d.

Relation to other Architectures Many modern DL architectures adopt skip connections. In ResNets [60], for example, shortcut connections perform an identity mapping, skipping one or more layers. Their outputs are then added to the skipped layers' outputs (not concatenated as in our approach). ResNets were among the first architectures that address the so-called degradation problem [60] (a problem observed in practice, where very deep neural networks surprisingly produce higher training errors than shallow nets) and have shown to improve the results on many problems.

In a DenseNet [66], each layer uses the output of all preceding layers as input and passes on its output to all subsequent layers. Due to this structure, many direct connections are necessary (in a network with L layers, there are $L(L+1)/2$ direct connections). Nonetheless, the authors could significantly reduce the number of required parameters in the overall network, since the number of filters in all layers could be decreased. DenseNets address similar problems as ResNets and are insofar more similar to our TCN-AE in that they also concatenate the feature maps of previous layers and do not add them (as in ResNets).

7.4.1.2 Dilation Rate Ordering

In the setup of the baseline TCN-AE, we use the identical TCN architecture for the encoder and decoder, with the same number of filters n_{filters}, filter lengths k and dilation rates q_i. In the decoder of the baseline TCN-AE, right after the upsampling layer, the first dilated convolutional layer has a dilation rate of $q = 1$. However, if we keep in mind that the upsampling layer uses sample-and-hold interpolation, which repeats each sample $s = 32$ times, a dilation rate of $q = 1$ might be ineffective. Due to the upsampled signal's coarse structure, the filters are mostly moved over ranges of identical values. A straightforward yet beneficial enhancement is to reverse the dilation rates in the decoder. Hence, now the last dilated convolutional layer before the output layer will have a dilation rate $q = 1$, the penultimate layer $q = 2$, and the first layer (after the upsampling layer) a dilation rate of 2^{L-1}. With this approach, larger dilation rates are used on coarser levels. In our architecture with $L = 7$ dilated convolutional layers, we use the dilation rates $(1, 2, 4, \ldots, 64)$ for the encoder, and the dilation rates $(64, 32, 16, \ldots, 1)$ for the decoder (see the green sticks in Fig. 7.5 and 7.6).

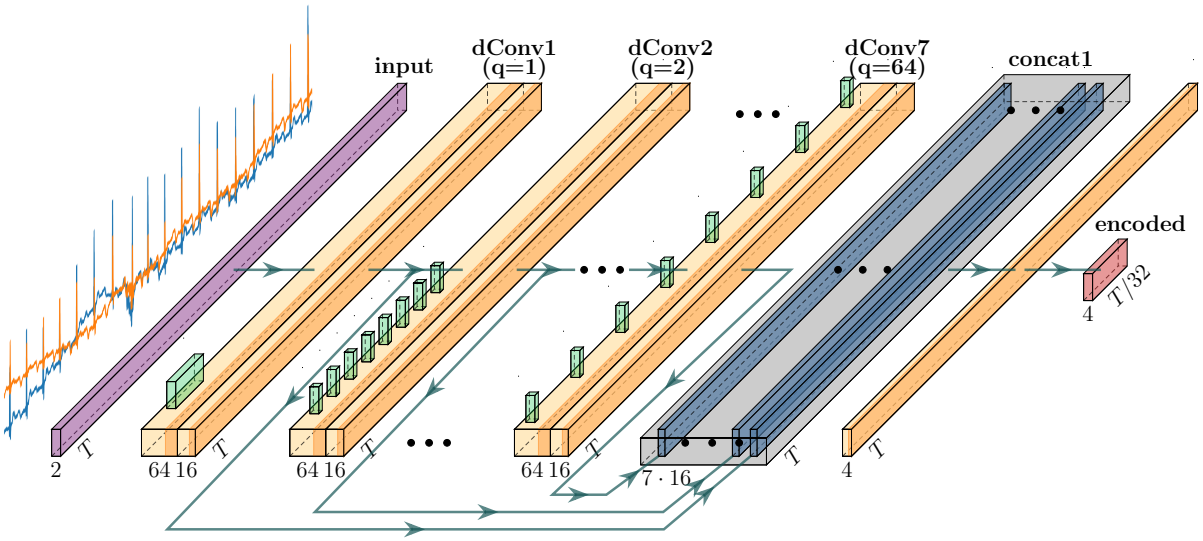

Figure 7.5: The architecture of TCN-AE's encoder. The two-dimensional input ECG-signal (purple) of length T, is passed through a stack of dilated convolutional layers (light orange, dConv1 – dConv7). The light green boxes represent the filters of the dilated convolutions. Each dilated convolutional layer is followed by a 1×1 convolution, which reduces the number of channels to 16. The outputs of the 1×1 convolutions are also concatenated in the block concat1 (blue). The dark blue blocks are identity mappings, not altering the tensors. Overall, seven tensors are concatenated, resulting in $7 \cdot 16 = 112$ channels. Finally, the concatenated tensor is compressed into the final encoded representation (red). The compressed representation of the original input is then passed to the decoder (Figure 7.6).

7.4.1.3 Utilizing Hidden Representations for the Anomaly Detection Task

While studying the relation of dilated convolutions to the DWT (Section 7.2.3), we noticed some similarities to our prior work [164]: In that work, we used the DWT to analyze a time series signal on different frequency scales to detect anomalous behavior. Each frequency scale was analyzed independently, and the aggregated results then led to an anomaly score for each data point of the time series. Similarly, transferred to the TCN-AE architecture, one could imagine that each dilated convolutional layer's output corresponds to an individual frequency/time scale, which already might carry useful information for the anomaly detection task. Hence, it could be sensible to look at the reconstruction error signal of TCN-AE and also individual hidden representations of the network to identify anomalies.

We take the output of each map-reduction layer (see section 7.4.1.4) in the encoder and reduce the feature map channels with a 1×1-convolution to size one. This is like taking each blue bar from Fig. 7.5 and reducing it to one output channel. We then stack each of the reduced outputs onto the reconstruction error signal. If there are seven dilated convolutional layers in the encoder ($q = 1 \ldots 64$) and the reconstruction error signal is two-dimensional, seven additional hidden representations will be stacked onto the reconstruction error signal. We end up with a 9-dimensional signal to which we apply Algorithm 8. With this approach,

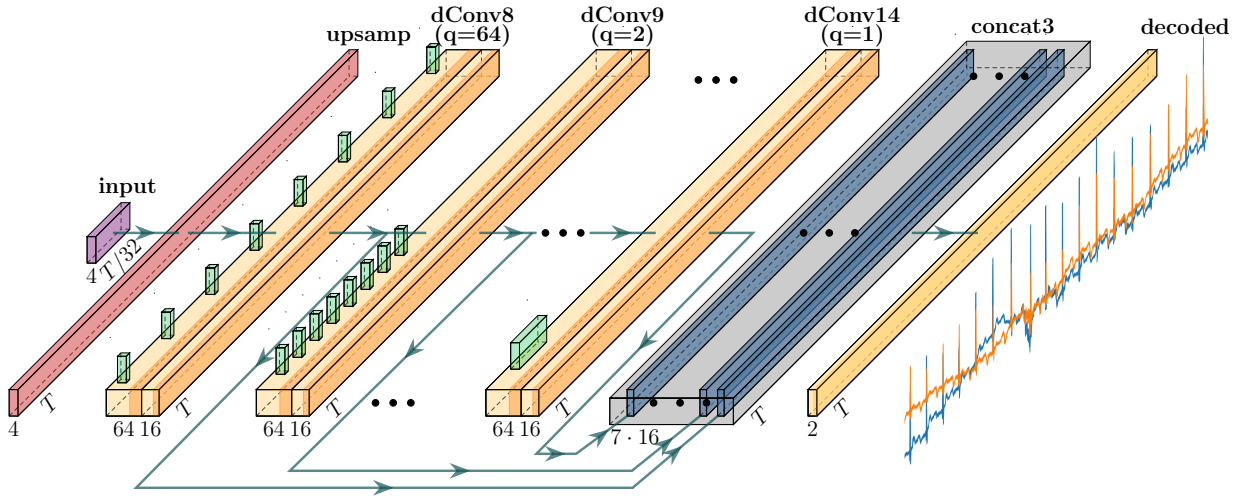

Figure 7.6: The architecture of TCN-AE's decoder. A compressed representation is given as input (purple) and then upsampled (red layer) to the original length T. Similar to Figure 7.5, a stack of dilated convolutional layers operates on the upsampled signal and the outputs of the 1×1 convolutions are concatenated in the concat3 block. Finally, the output layer (convolution with linear activation), reconstructs the original two-dimensional ECG sequence.

we can not only search for anomalies in the reconstruction error but also find irregularities in various hidden feature representations of the input time series.

Note that this enhancement is not shown in Figs. 7.5 and 7.6 to keep the complexity of the figures manageable.

7.4.1.4 Feature Map Reduction

A more technical enhancement of TCN-AE is the introduction of convolutional map reduction layers (commonly referred to as 1×1 convolutional layers) [96, 156], which are regularly used in practice to reduce the dimensionality (the number of channels) of feature maps and effectively reduce the number of trainable parameters in the overall architecture. We experimented with 1×1 convolutional layers and found that they allow reducing the overall number of parameters in the network, without sacrificing performance. Additionally, we could observe a slight improvement in the training time. We place 1×1 convolutions after each dilated convolutional layer, which reduces the number of channels from 64 to 16.

7.4.1.5 Anomaly Score Baseline Correction

While visualizing the anomaly score of the TCN-AE model for a few time series, we noticed that the anomaly score did not always have a constant baseline, as one would expect. We observed slight drifts in the baseline, which made it hard in some cases to find a suitable

threshold value. One reason for this phenomenon could be that certain statistical properties of the signal (such as the random noise) change over time. Since these drifts correspond to low-frequency components in the anomaly score, a simple way to remove them is to filter the anomaly score. We decided to use a second-order Butterworth filter with a cutoff frequency of 1Hz to remove the baseline wandering.

7.4.2 Experimental Setup

In this chapter, we compare all considered algorithms on the ECG-25 benchmark, described in Section 2.3. If not stated otherwise, we sum TP, FN, and FP over all 25 ECG time series. From these three quantities, the well-known metrics precision (Prec), recall (Rec), and F_1-score are derived.

7.4.2.1 Algorithmic Setup

We compare our unsupervised TCN-AE algorithm to four other unsupervised anomaly detection algorithms: DNN-AE [46], LSTM-ED [108], LSTM-AD [161], and NuPIC [160]. They are based on deep autoencoders (DNN-AE), LSTM networks (LSTM-ED and LSTM-AD), and hierarchical temporal memory, HTM (NuPIC).

All anomaly detection algorithms are trained in an unsupervised fashion. The actual anomaly labels are only used at test time. The training process passes the complete time series to the anomaly detection algorithm, and the algorithm learns a model for the provided data and returns an anomaly score for each data point of the time series. We trained all algorithms, except NuPIC (which does not support GPU capabilities), on a Tesla P100 GPU. All algorithms require a set of hyperparameters, which we will describe in the following. Parameters common to all algorithms are summarized in Tab. 7.2. We tuned the parameters (except for TCN-AE and NuPIC) using the HYPEROPT library [14]. For TCN-AE, we manually investigated different parameter settings, and for NuPIC, we use the recommended parameter settings [89].

To obtain statistically sound results, we run each anomaly detection algorithm ten times on all 25 ECG time series.

DNN-AE [46]: We use a PyTorch [130] implementation for the anomaly detection algorithm based on a deep autoencoder [58]. The algorithm requires several parameters, which we choose as follows: hidden size of $h = 6$ for the bottle neck (which results in a compression factor of $T_{\text{train}}/h = 25$ for each sequence). Finally, we set $\%_{Gaussian} = 1\%$, which specifies that 99% of the data is used to estimate a Gaussian distribution for the anomaly detection task.

LSTM-ED [108] is also implemented using PyTorch and has the following parameter setting: $\%_{Gaussian} = 3\%$. Both, encoder and decoder use a stacked LSTM network with two layers, each having LSTM layer having 50 units.

NuPIC [160]: Numenta's anomaly detection algorithm has a broad range of hyperparameters that have to be set. We use the parameters recommended by the authors

in [89]. It is possible to tune the parameters with an internal swarming tool [3]. However, this is a time-expensive process which is not feasible for the large benchmark.

LSTM-AD (Chapter 6, [161]): A 2-layer LSTM network with 256 units in the first layer and 128 units in the second layer is used. The target horizons are chosen to be $H = (1, 3, \ldots, 49)$.

SORAD (Chapter 4, [163]) We use SORAD with mini-batch RLS – using small batches to update the model instead of single examples (see Appendix B.1) – as a simple baseline method. The batch size is set to $\mu = 256$. The target horizons $H = (1, 3, \ldots, 49)$ are the same as for LSTM-AD. The forgetting factor is set to $\lambda = 0.98$. The window length for the sliding window is $w = 128$. Hence, the number of trainable weights of the model is 129 (including one bias weight). The regularization parameter was set to $\beta = 10^{-5}$.

TCN-AE (baseline): The settings of the baseline TCN-AE model (Figure 7.2) mostly correspond to the settings of the final variant. Only the maximum dilation rate is chosen smaller so that $q = (1, 2, \ldots, 32)$ and the number of filters for each dilated convolutional layer is reduced to $n_{\text{filters}} = 32$. Nonetheless, the number of trainable parameters of the baseline TCN-AE is larger due to the two consecutive layers which are created for each individual dilation rate. For baseline TCN-AE, we use an existing TCN implementation in Keras [140].

TCN-AE (final): We implemented TCN-AE using the Keras [30] & TensorFlow framework [1]. An overview of the architecture with its parameters is given in Figures 7.5 and 7.6. In both encoder and decoder we use 7 dilated convolutional layers each, with the dilation rates $q = (1, 2, \ldots, 64)$ (encoder) and $q = (64, 32, \ldots, 1)$ (decoder), $n_{\text{filters}} = 64$ filters with a kernel size of $k = 8$, and a ReLU activation. Each dilated convolutional layer is followed by a 1×1 convolutional layer with $n_{\text{filters}} = 16$ filters, which reduces the feature maps from 64 to 16. The sample rate of the average pooling layer is $s = 32$ and the error window length for the anomaly detection in Algorithm 8 is $\ell = 128$. For MGAB, we use $q = (1, 2, \ldots, 16)$, $n_{\text{filters}} = 32$, $k = 25$, $B = 64$, $n_{\text{epochs}} = 10$, $T_{\text{train}} = 1050$, $n_{\text{filters}} = 16$ filters for the skip connections, $s = 6$, and $\ell = 128$.

Table 7.2: Summary of the common parameters of the neural-network-based anomaly detection algorithms used in this work.

Algorithm	B	n_{epochs}	T_{train}	Loss	Optimizer	Initializer
TCN-AE	64	10	1024	logcosh	Adam	Glorot Normal [50]
DNN-AE	100	25	150	MSE	Adam	$\mathcal{U}(-\sqrt{k}, \sqrt{k}), k = \frac{1}{fan_{in}}$
LSTM-ED	100	10	30	MSE	Adam	$\mathcal{U}(-\sqrt{k}, \sqrt{k}), k = \frac{1}{fan_{in}}$
LSTM-AD	512	25	256	MAE	Adam	Glorot Uniform [50]

7.4.2.2 Evaluation

For most results presented in this section, we use the EAC criterion to determine precision, recall and F_1-score. To evaluate the performance of the algorithms over a wide range of anomaly thresholds, we also generate a precision-recall curve. In the other cases, we select an optimal threshold in a supervised manner for a small fraction of the time series data and then apply this threshold to the overall time series. If not stated otherwise, we select a segment containing 10% of a time series and find the threshold which maximizes the F_1-score on this small subset. Since the results may vary, depending on which 10%-segment is used, we repeat the whole evaluation procedure 10 times and average the results: adjust the threshold on 10% of the data, evaluate on the remaining 90% of the data. We assess the significance of the results with the non-parametric Wilcoxon signed-rank test [178] and report the p-values.

7.4.3 Evaluation of TCN-AE on MGAB

Before performing the experiments on the ECG-25 benchmark, we first run our enhanced TCN-AE algorithm on MGAB and compare the results with the results reported in the previous section. The anomaly threshold is determined by using 10% of the anomaly labels supervisedly, as described in the experimental setup (Sec. 7.3.2.1) of the previous section. The results for the final TCN-AE model are summarized in Table 7.3 and compared to the three other best models. The results for TCN-AE (baseline), LSTM-AD and DNN-AE are copied from Table 7.1. We can see that the performance of TCN-AE (final) is similar to TCN-AE (baseline) and LSTM-AD. However, instead of originally 164 451 trainable weights, TCN-AE (final) now only has 38 423 weights. Also the computation time could be drastically reduced from an aaverage time of 65 seconds (baseline) to about 17 seconds / time series.

Table 7.3: Similar to Table 7.1. Here, we also list the results for TCN-AE (final). Additionally, we also list the p-values, indicating the significance of the results.

Algorithm	TP	FN	FP	Precision	Recall	F_1
DNN-AE [58]	91.8 ± 1.2	8.2 ± 1.2	62.6 ± 13.6	0.600 ± 0.058	0.918 ± 0.012	0.724 ± 0.043
LSTM-AD (Ch. 6,[161])	88.8 ± 2.6	11.2 ± 2.6	0.6 ± 0.6	0.993 ± 0.007	0.888 ± 0.026	0.937 ± 0.014
TCN-AE (final)	90.6 ± 1.9	9.4 ± 1.9	0.4 ± 0.7	0.995 ± 0.008	0.906 ± 0.019	0.948 ± 0.010
TCN-AE (baseline)	90.5 ± 1.7	9.5 ± 1.7	0.2 ± 0.5	0.997 ± 0.011	0.905 ± 0.021	0.949 ± 0.010

7.4.4 Experiments, Results & Discussion for the ECG-25 Benchmark

We started our experiments with the baseline TCN-AE model (Section 7.3.1). The initial results on the ECG-25 benchmark were already promising, but the algorithm still performed

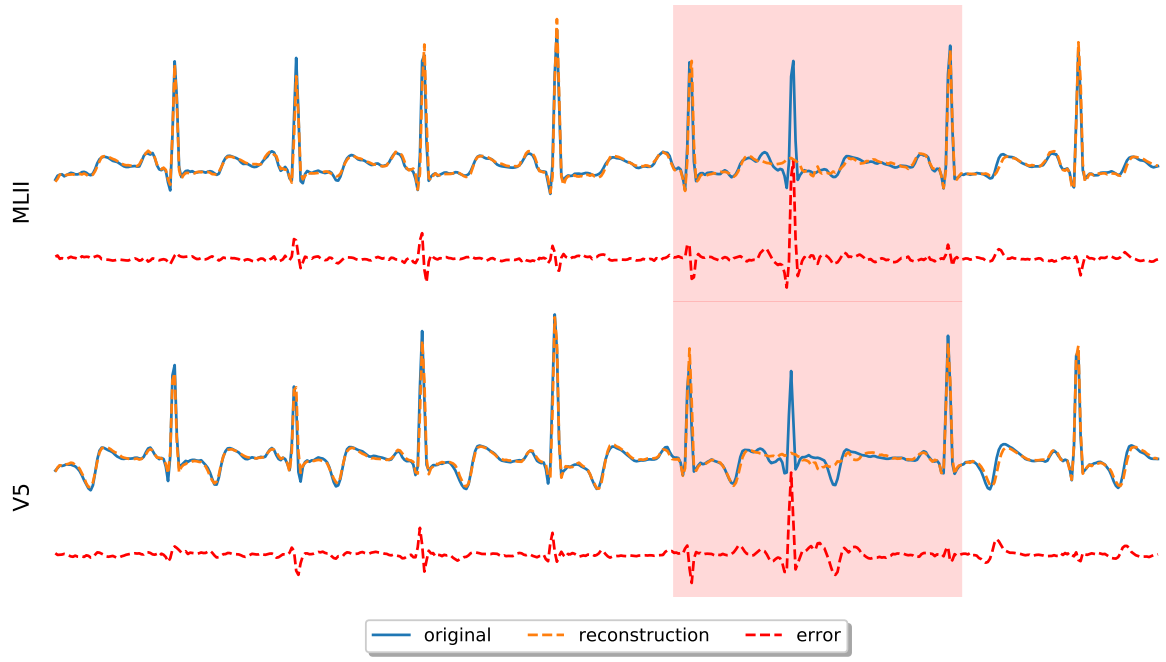

Figure 7.7: Excerpt showing how the final TCN-AE model reconstructs the modified limb lead II (MLII) and the modified lead V1 of ECG signal #1. TCN-AE has difficulties in reconstructing actual anomalous behavior (highlighted with the red shaded area). Due to the resulting large error, the algorithm later correctly detects an anomaly (true positive).

similar to LSTM-AD and DNN-AE concerning the $F1$-score (Table 7.6), leaving room for improvements. While analyzing the baseline TCN-AE, we developed several ideas for improvements, which we introduced in Section 7.4.1. The resulting (final) TCN-AE showed significantly improved performance, achieving a higher precision and recall on 15 out of 25 time series of the benchmark. We perform a more detailed analysis of the contribution of the individual enhancements in the following Section.

Figure 7.7 depicts an example for ECG signal #1, where the TCN-AE algorithm has difficulties reconstructing the original time series due to an actual anomaly present. In this case, the large construction error is correctly interpreted as anomalous behavior. For the same example, we visualize selected activations of several layers inside the trained TCN-AE in Figure 7.8. While the ECG's general patterns are still visible in the initial layers of the encoder, these disappear in later layers, and the activations do not seem to carry any information appearing useful to the human eye. After being passed through the bottleneck and upsampled again, only a 4-channel (from which one is depicted in the graph) step-shaped signal remains. However, remarkably, the decoder can almost accurately restore the original input sequence solely from this coarse representation (sixth row in the plot). Only

the anomalous pattern is incorrectly reconstructed, which results in a large reconstruction error that can easily be detected.

7.4.4.1 Contribution of the Individual TCN-AE Components

In the following, we describe the impact of individual enhancements on the final TCN-AE, which we introduced in Sec. 7.4.1.

Variant	Section	Comment
baseline	7.3.1	Baseline algorithm based on TCNs without any enhancements
noSkip	7.4.1.1	Skip-Connections removed from the architecture
noInvDil	7.4.1.2	Use same dil. rate ordering for encoder & decoder
noLatent	7.4.1.3	Do not use hidden represenations for anomaly detection
noRecon	7.4.1.3	Only use hidden representations of encoder for anomaly detection
noMapReduc	7.4.1.4	Do not use the Map reduction layers
noAnomScoreCorr	7.4.1.5	Do not correct the baseline of the anomaly score
final	7.4.1	Final TCN-AE with all enhancements

Table 7.4: Summary of all TCN-AE variants

Although it is challenging to accurately measure each element's effect on the final result (since there might be some interaction effects between elements[2]), we can approximately quantify the improvements with the following approach: In order to measure the contribution of component C on the final result, we run TCN-AE on the benchmark with this specific component turned off. If the component has a positive impact on the model, we expect a poorer result, and the differences in precision, recall, and F_1-score serve as a rough indicator for the contribution of the component. Additionally, the p-value of the one-sided Wilcoxon test signalizes the significance of the result. In Table 7.5, we summarize the different variants of the TCN-AE algorithm. Further results are listed in Appendix A.

Overall, all the individual enhancements significantly improve the performance of TCN-AE on the ECG-25 benchmark. As summarized in Table 7.5, the precision, recall and F_1-score all improve by around 10%. All enhancements have a significant impact on the increase in performance, as indicated by the corresponding p-values. The p-values are also illustrated graphically in a heat map in Figure 7.9 for all 25 time series of the benchmark. We can see that the algorithm achieved an improvement for most time series. The exact F_1-scores for each TCN-AE variant and time series can be found in Table A.4. This table also highlights the time series for which the p-values are above the significance level of 0.05.

Skip Connections The skip connections in TCN-AE allow the last layers of the encoder and decoder to access all prior layers (having different dilation rates) directly. As shown

[2]We assume that the overall contribution of the individual components is larger than our estimations due to the interaction effects which we cannot measure.

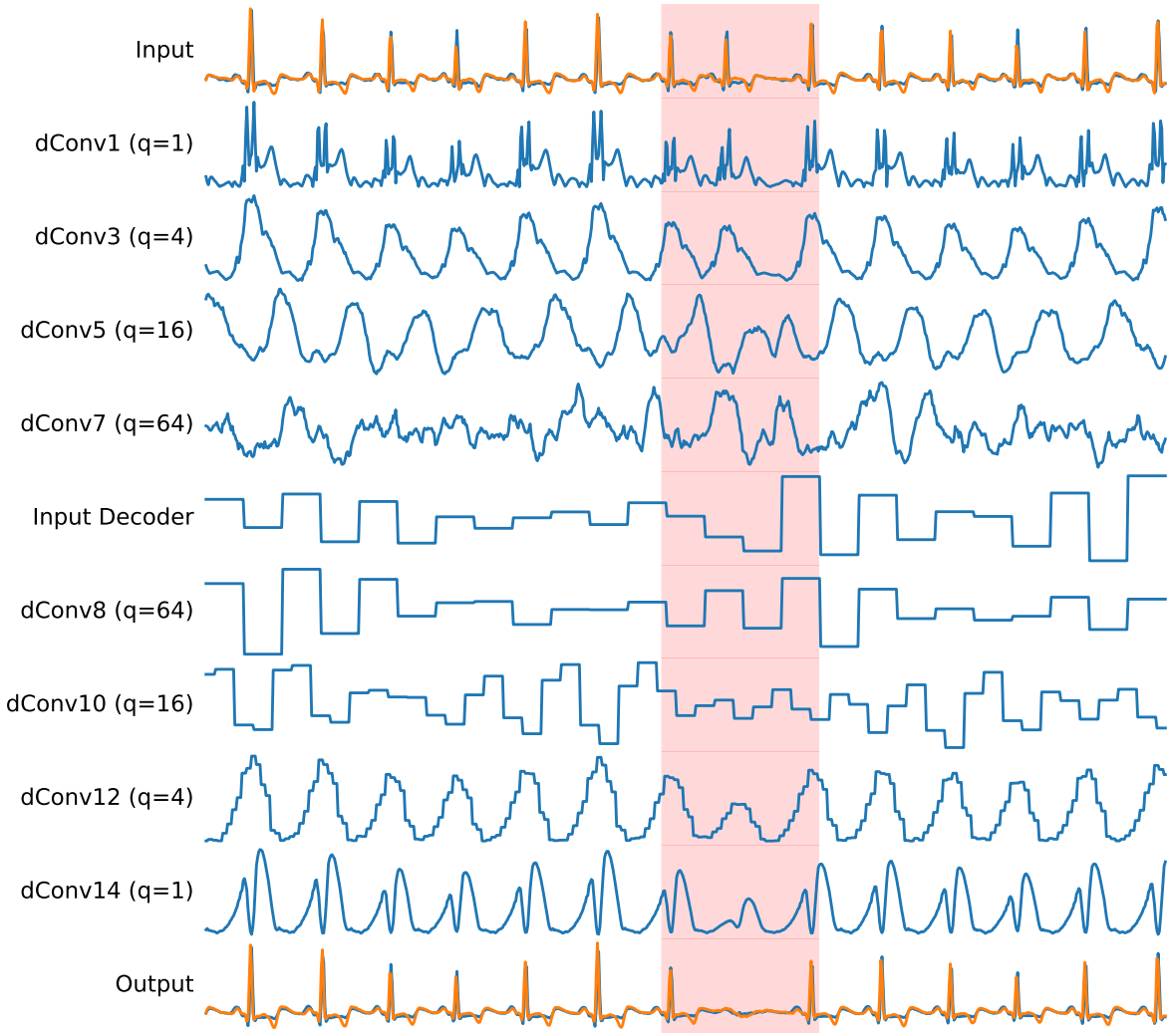

Figure 7.8: Activations inside the trained final TCN-AE model for several layers of the network. For each dilated convolutional layer, we plot the channel (signal) with the largest mean absolute activation. If we compute and plot the mean over all channels, we get structurally relatively similar results. The rows dConv1 – dConv7 refer to the activations of the dilated convolutions inside the encoder, while dConv8 – dConv14 are dilated convolutional layers inside the decoder. The input signal contains an anomaly (atrial premature beat), highlighted with the red-shaded vertical bar. The decoder fails in reconstructing this segment of the time series, which results in a significant deviation between the original and reconstructed signal.

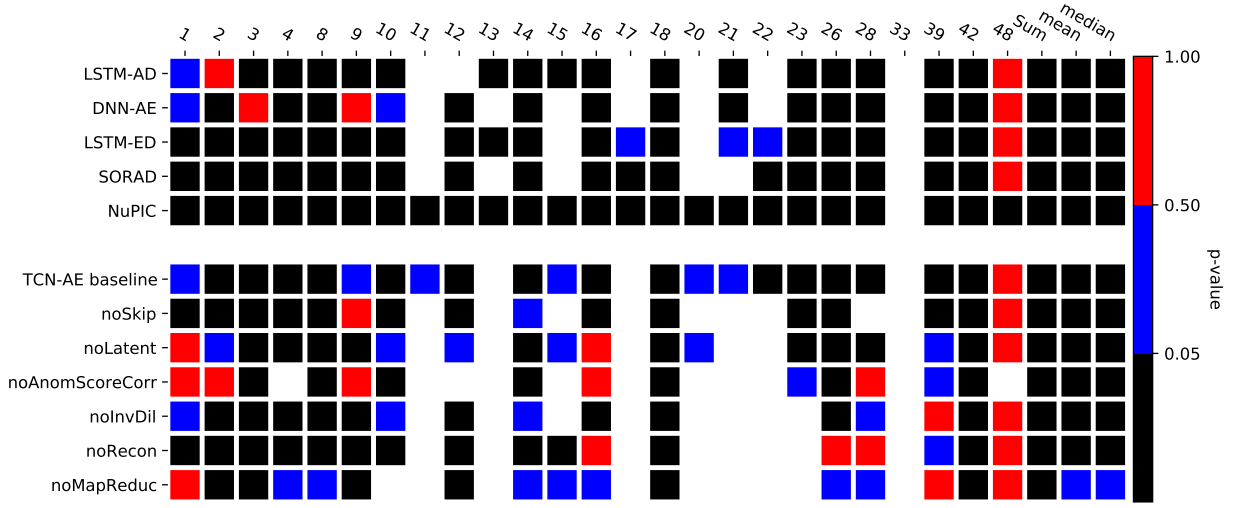

Figure 7.9: Heatmap, showing the p-values for the comparison of our final TCN-AE model to various algorithms and other variants of TCN-AE for all 25 ECG signals of our benchmark. We use a one-sided Wilcoxon signed-rank test for the F_1-scores of ten runs each. The test has the null hypothesis that the median F_1-score of our final TCN-AE is smaller (than the compared algorithm) against the alternative that the median F_1-score is larger. The first four rows represent anomaly detection algorithms from the literature, while the remaining rows are for different variants of the TCN-AE algorithm. In the cases where the p-value is below the significance level of $\alpha = 0.05$, the tiles are colored black (indicating a significantly higher performance of TCN-AE). White tiles indicate that the F1-scores of TCN-AE and the compared algorithm are exactly the same. The exact F_1-scores are given in Table A.4 and Table 7.9.

Table 7.5: Impact of the individual TCN-AE components for the ECG-25 data (mean and standard deviation of 10 runs). The results shown here are for the sum of TP, FN and FP over all 25 time series and were obtained such that an approximate (difference of less than 1% in precision and recall) equal accuracy (EAC) is achieved for each algorithm and time series.

Algorithm	TP	FN	FP	Prec	Rec	F1	p
baseline	597.5 ± 5.1	123.5 ± 5.1	129.0 ± 5.2	0.822 ± 0.007	0.829 ± 0.007	0.826 ± 0.007	2.531e-3
noSkip	622.4 ± 5.7	98.6 ± 5.7	102.9 ± 5.7	0.858 ± 0.008	0.863 ± 0.008	0.861 ± 0.008	2.531e-3
noLatent	629.5 ± 5.2	91.5 ± 5.2	95.3 ± 5.4	0.869 ± 0.007	0.873 ± 0.007	0.871 ± 0.007	2.531e-3
noAnomScoreCorr	644.2 ± 3.6	76.8 ± 3.6	83.2 ± 3.7	0.886 ± 0.005	0.893 ± 0.005	0.890 ± 0.005	2.531e-3
noInvDil	653.6 ± 1.9	67.4 ± 1.9	72.2 ± 1.9	0.901 ± 0.003	0.907 ± 0.003	0.904 ± 0.003	2.531e-3
noRecon	656.9 ± 2.9	64.1 ± 2.9	69.9 ± 2.9	0.904 ± 0.004	0.911 ± 0.004	0.907 ± 0.004	2.531e-3
noMapReduc	660.7 ± 1.3	60.3 ± 1.3	65.1 ± 1.4	0.910 ± 0.002	0.916 ± 0.002	0.913 ± 0.002	8.302e-3
final	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003	–

in Table 7.5 and Figure 7.9, this improvement has the highest impact on the performance of TCN-AE: Without skip connections, the F_1-score drops from $F_1 \approx 0.93$ to $F_1 \approx 0.86$. However, for the model without the skip-connections, we had to decrease the number of filters from $n_{\text{filters}} = 64$ to $n_{\text{filters}} = 32$; otherwise the results would be even worse.

We also experimented with different variants of a dense TCN-AE similar to a DenseNet [66] (connecting all dilated convolutional layers with the preceding ones). However, we decided to no longer pursue this approach, since the dense connections increased the number of parameters and the computation time significantly without considerably improving the results.

Although our primary purpose for the introduction of skip connections is to reduce the sensitivity towards the maximum dilation rate and to enable the encoder/decoder to reuse the features at different time scales, as a side effect, our TCN-AE architecture might also benefit from other advantages associated with ResNets [60] or DenseNets [66]. Some of the observed improvements might be due to a smoother curvature of the loss landscape [93], alleviation the vanishing/exploding gradient problem & degradation problem due to gradient shortcuts through the identity mappings of the skip connections, reduction of parameters, and others.

We also tested higher dilation rates up to $q_{max} = 1024$ and found (apart from the higher computation time and memory requirements[3]) that the results remained almost the same. Only for $q_{max} = 1024$ we could observe a slight drop in the overall F_1-score, from $F_1 \approx 0.93$ to $F_1 \approx 0.91$. Since the results do not change significantly with a larger stack of dilated convolutional layers, this might imply that the TCN-AE model is capable of learning to select the suitable features from the important time scales and to ignore the remaining time scales which do not carry directly relevant features. In the baseline version, where no skip connections were employed, the results significantly deteriorated for inappropriate (too large/small) choices of q_{max}.

Dilation Rate Ordering In Figure 7.9, we can see that this reversed dilation rate scheme improves the results on most of the considered 25 time series. While the reason for the improvement is not entirely apparent yet, we assume that a larger dilation rate is beneficial for the coarse step-shaped signal we have in the first layers of the decoder, and a lower dilation rate is essential when we attempt to reconstruct the details of the original signal.

Detecting Anomalies in Hidden Representations of Time Series As discussed in Section 7.4.1.3, we noticed that there are some similarities between the (stationary) discrete wavelet transform (DWT) and DL architectures, which use stacks of dilated convolutional layers. In [164], we used the DWT to decompose a time series and look for anomalous behavior on different frequency scales. Similarly, we can also utilize the outputs of the individual dilated convolutional layers, which process the time series on different time scales. The general idea is that anomalies might become more apparent on some time scales than on others and that one can already detect anomalous behavior on these hidden representations rather than relying solely on the reconstruction error. In our first experiment, we used the

[3]Since the receptive field of the model increases with larger dilation rates, also longer training sequences are required. (We assume that this is also partially due to the artifacts induced when the filters move within the zero-padded areas.)

encoder's outputs of the dilated convolutional layers, reduced their number of channels to one (with a 1×1 convolution), and stacked them on top of the reconstruction error $\mathbf{e}[n]$ (line 11 in Algorithm 8). Although we observe a drop in the F_1-score for 3 ECG signals (#1, #16, #48) in Figure 7.9 (Table A.4), the overall results suggest that this approach generally improves the results (Table 7.5). The overall F_1-score increases from $F_1 = 0.89$ to $F_1 = 0.93$ and the mean (median) F_1-score is significantly higher (Figure 7.9 & Table A.4).

Similarly, in our next experiment, we tried also to utilize the decoder's hidden representations. However, this did not have any further effect on the algorithm's performance, and we discarded this approach again.

We made another interesting observation: If we entirely remove the decoder after training TCN-AE and solely use the outputs of the encoder's dilated convolutional layers to detect anomalies in the time series, still decent results can be obtained. In Table 7.5, we can see that the F_1-score only drops by about 0.02, although the size of the model (and the computational cost for inference) is effectively halved. This observation might be interesting for practical applications, where memory and computational resources are constrained.

Map Reduction Layers Although the primary purpose for the map reduction layers (Section 7.4.1.4) was to reduce the number of parameters in the overall model, as a side effect, we could observe a slight improvement in the overall performance, when considering the sum over all TP, FP, and FN. However, the improvement is smaller than for the other previously discussed enhancements. The mean (median) F_1-score does not increase, as shown in Figure 7.9 (Table A.4).

Anomaly Score Baseline Correction Also, the correction of the anomaly score baseline using a Butterworth filter, as described in Section 7.4.1.5, has a notable impact on the final results. Although there is only a significant improvement for 7 out of 25 time series (Figure 7.9), the overall F_1 drops by around 4% if we turn off the baseline correction of the anomaly score, as reported in Table 7.5. Instead of using a filter, we also tested the more advanced baseline correction algorithm by Zhang et al. [185], and obtained results which did not significantly differ (F_1-score of 0.920 ± 0.003).

7.4.4.2 Comparison to other Algorithms

In Table 7.6, we summarize the results for all algorithms. The table is sorted according to the F_1-score and shows the p-values for comparing the F_1-scores of the individual algorithms with TCN-AE. The first observation we can make is that TCN-AE (baseline and final variant) outperforms the other five algorithms significantly (p-value < 0.05). On average, TCN-AE detects 81 anomalies more than the second-ranked algorithm, LSTM-AD, while at the same time also producing 80 fewer FPs. The overall F_1-score of TCN-AE is around 15% higher than of DNN-AE and even 20% higher than of LSTM-ED.

Table 7.6: Summary for the ECG-25 data (mean $\pm \sigma_{\text{mean}}$ of 10 runs, except for the deterministic NuPIC algorithm). The results shown here are for the sum of TP, FN and FP over all 25 time series and were obtained such that an approximate (difference of less than 1% in precision and recall) equal accuracy (EAC) is achieved for each algorithm.

Algorithm	TP	FN	FP	Prec	Rec	F1	p
NuPIC	224.0	497.0	497.0	0.311	0.311	0.311	2.531e-3
SORAD	519.0	202.0	206.0	0.716	0.72	0.718	2.531e-3
LSTM-ED	557.3 ± 2.9	163.7 ± 2.9	168.9 ± 2.9	0.767 ± 0.004	0.773 ± 0.004	0.770 ± 0.004	2.531e-3
DNN-AE	583.7 ± 1.1	137.3 ± 1.1	142.9 ± 1.3	0.803 ± 0.002	0.810 ± 0.002	0.806 ± 0.002	2.531e-3
LSTM-AD	589.3 ± 0.8	131.7 ± 0.8	136.4 ± 0.5	0.812 ± 0.001	0.817 ± 0.001	0.815 ± 0.001	2.531e-3
TCN-AE (baseline)	597.5 ± 5.1	123.5 ± 5.1	129.0 ± 5.2	0.822 ± 0.007	0.829 ± 0.007	0.826 ± 0.007	2.531e-3
TCN-AE (final)	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003	–

Precision-Recall Curves Since the anomaly threshold trades off FNs (recall) and FPs (precision), another way of showing the performance of an anomaly detection algorithm is to vary the threshold over a wide range of values and plot the precision and recall for different settings in a graph. In Figure 7.10, we generated such a precision-recall plot for all the compared algorithms. The precision-recall plot can be seen as a bi-objective optimization problem where one attempts to maximize both precision and recall. Each point in the graph is obtained for a specific threshold value. We fit one curve as a rough approximation to the points of the 10 runs. It can be seen that TCN-AE outperforms the other algorithms over a wide range of anomaly thresholds. Especially along the identity line (precision=recall), the difference of TCN-AE to the other algorithms becomes apparent.

Performance for individual Anomaly Types The ECG-25 benchmark contains nine different anomaly types, as summarized in Table 2.2. Since the anomaly types take very different shapes, we were interested in knowing how well TCN-AE can detect the individual types and how it compares to the other anomaly detection algorithms. Table 7.7 shows how many of the individual anomaly types were detected by the respective algorithms for an EAC setting. Although TCN-AE has the most detections for only five out of nine anomaly types, it is among the top three algorithms in each case. Furthermore, it produces significantly less FPs in the EAC setting (if we would permit TCN-AE to also have more FPs, similar to the other algorithms, it would detect even more anomalies). However, we still see room for improvement. A more thorough investigation of the anomaly types that appear to be hard for TCN-AE could lead to new insights and possible new enhancements.

Computing Anomaly Score with little Labled Data While we used EAC to determine all anomaly thresholds for the results presented in Table 7.6, we also investigated how the results change when all algorithms may use only a small fraction of each time series' labels to find a suitable threshold. This approach is more realistic for practical applications

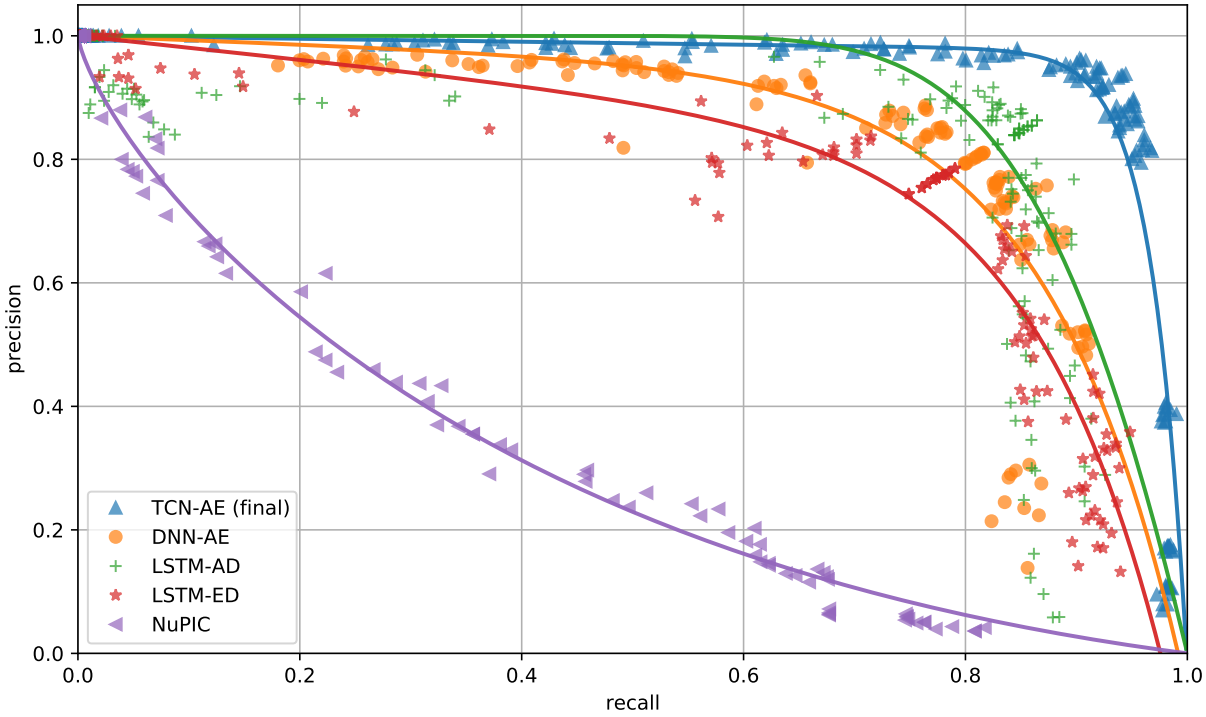

Figure 7.10: Precision-recall curves for the individual algorithms on the ECG-25 data. Shown are the fits through around 100 points which were generated by evaluating precision & recall for different thresholds. For each algorithm, except NuPIC, 10 runs were performed.

since, usually, only little labeled data is available. In our specific experiment the algorithms were only allowed to use 10% of the anomaly labels The detailed results are listed in the appendix in Table A.2. We observe that the F_1-score for all algorithms deteriorates compared to Table 7.6, where the EAC was used. Nevertheless, TCN-AE ($F_1 = 0.79$) still has the highest performance (having the highest F_1-score on 18 out of 25 time series), followed by LSTM-AD ($F_1 = 0.67$), LSTM-ED ($F_1 = 0.6$), and DNN-AE ($F_1 = 0.57$), while NuPIC performs the worst ($F_1 = 0.22$).

One interesting observation is that the recall of all algorithms is significantly higher than the precision. A possible explanation for this is that for many 10% intervals, on which the algorithms optimize their threshold, a threshold value can be found that results in a high recall and high precision. However, this threshold is then too low for the remaining time series, and many FPs are created as a consequence. This problem demonstrates that in practice, more sophisticated methods are necessary to determine a suitable threshold. We are planning to work on this issue in future work.

Finally, we compare in Table 7.8 the runtimes of all algorithms and the number of trainable parameters. All TCN-AE variants are considerably faster than the remaining

Table 7.7: Number of detected anomalies for the individual algorithms, broken down by anomaly type (in total 9 types, see Table 2.2). We count the true detections when an EAC is used. The last two columns (FN & FP) are the false negatives and false positives summed up over all anomaly types.The last row depicts optimal results for each type.

Algorithm	a	A	e	f	F	J	V	x	|	FN	FP
NuPIC	2.0	32.0	1.0	1.0	5.0	2.0	172.0	3.0	6.0	497.0	497.0
SORAD	9.0	108.0	5.0	17.0	17.0	2.0	337.0	6.0	11.0	207.0	210.0
LSTM-ED	10.1	150.5	7.2	18.6	20.0	2.0	328.1	9.1	11.7	163.7	168.9
DNN-AE	9.9	212.4	**7.4**	**21.5**	**21.8**	1.0	290.9	8.9	9.9	137.3	142.9
LSTM-AD	6.4	213.4	4.2	5.8	18.4	2.0	316.9	12.4	9.8	131.7	136.4
TCN-AE (baseline)	9.6	176.9	5.5	18.8	18.9	1.8	345.0	9.0	**12.0**	123.5	129.0
TCN-AE (final)	**10.4**	**213.7**	6.3	20.2	19.5	**2.4**	**369.8**	**16.5**	11.4	**50.8**	**55.8**
Ideal	11	235	10	22	25	3	374	19	22	0	0

algorithms. TCN-AE final, the fastest algorithm is 5× faster than the fastest non-TCN-AE algorithm (DNN-AE). LSTM-AD, ranked 2nd according to the F_1-score, is by far the slowest algorithm and required more than four days to complete ten runs. Furthermore, LSTM-AD has the most trainable parameters. Also, NuPIC is relatively slow (around $750s$ per time series) since no GPU acceleration is available.

Table 7.8: Computation times: average (per time series) and total (10 runs, all time series) for all algorithms evaluated in this chapter. *TCN-AE final* is faster than *baseline* since it effectively has less layers. SORAD and NuPIC are not directly compareable to the other approaches since they are not neural-network-based and run only on one CPU core.

Algorithm	mean (s)	total (h)	#Params/10^3
SORAD (Ch. 4,[163])	12.5±0.1	0.1	0.129
NuPIC [160]	752.0±23.4	52.4	–
TCN-AE final	**47.6±0.4**	**3.3**	**123.8**
TCN-AE baseline	97.0±1.1	6.7	208.4
DNN-AE [58]	237.5±10.3	16.5	242.1
LSTM-ED [108]	626.5±30.8	43.5	134.4
LSTM-AD [161]	1615.0±127.6	112.2	465.6

7.4.4.3 Additional Investigations

Outlier Detection Algorithms We experimented with different outlier detection algorithms using different values of ℓ: The isolation forest [98] and extended isolation forest [57], the one-class support vector machine (OCC-SVM [147]), local outlier factor (LOF) [19], an

Table 7.9: F_1-scores (mean $\pm$ σ_{mean}) of TCN-AE and the other algorithms on all 25 time series of the ECG-25 benchmark (highest values in boldface). The p-values are computed with the one-sided Wilcoxon signed-rank test, in which we compare the final TCN-AE algorithm with the other variants. We compare the F_1-scores of ten runs, which are obtained for an EAC. The Wilcoxon test has the null hypothesis that the median F_1-score of TCN-AE (final) is smaller than the compared algorithm against the alternative that the median F_1-score is larger. In all cases in which we fail to reject the null hypothesis at a confidence level of 5%, we add a grey background the corresponding field.

	NuPIC		LSTM-ED		DNN-AE		LSTM-AD		TCN-AE
	F1	p	F1	p	F1	p	F1	p	F1
1	0.03	0.002	0.572±0.022	0.002	0.907±0.008	0.118	0.904±0.007	0.06	**0.919±0.006**
2	0.5	0.029	0.350±0.017	0.003	0.267±0.045	0.003	**0.783±0.026**	0.5	0.733±0.083
3	0.196	0.002	0.831±0.010	0.002	**0.981±0.006**	0.997	0.376±0.009	0.002	0.933±0.008
4	0.0	0.001	0.5	0.001	0.0	0.001	0.250±0.083	0.002	**1.0**
8	0.088	0.002	0.923±0.012	0.003	0.724±0.010	0.002	0.913±0.007	0.002	**0.981±0.003**
9	0.218	0.002	0.498±0.014	0.002	**0.708±0.005**	0.983	0.582±0.005	0.002	0.674±0.011
10	0.725	0.001	0.811±0.016	0.002	0.975±0.005	0.096	0.907±0.016	0.002	**0.987**
11	0.0	0.001	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
12	0.0	0.001	0.5	0.001	0.650±0.077	0.004	**1.0**	–	**1.0**
13	0.0	0.001	0.950±0.026	0.042	**1.0**	–	0.833	0.001	**1.0**
14	0.524	0.002	0.925±0.004	0.004	0.693±0.005	0.002	0.918±0.002	0.002	**0.945±0.004**
15	0.0	0.001	**0.909**	–	**0.909**	–	0.727	0.001	**0.909**
16	0.33	0.001	0.877±0.013	0.003	0.675±0.010	0.003	0.945±0.003	0.03	**0.953±0.001**
17	0.0	0.001	0.9 ±0.1	0.159	**1.0**	–	**1.0**	–	**1.0**
18	0.24	0.002	0.519±0.010	0.003	0.843±0.003	0.003	0.902±0.005	0.003	**0.962±0.003**
20	0.0	0.001	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
21	0.0	0.001	0.9 ±0.1	0.159	0.1 ±0.1	0.001	0.500±0.167	0.013	**1.0**
22	0.0	0.001	0.900±0.071	0.09	**1.0**	–	**1.0**	–	**1.0**
23	0.19	0.001	0.814±0.025	0.004	0.910±0.009	0.003	0.919±0.010	0.01	**0.952**
26	0.169	0.002	0.724±0.008	0.002	0.646±0.005	0.002	0.240±0.005	0.003	**0.806±0.011**
28	0.507	0.001	0.893±0.009	0.049	0.874±0.006	0.003	0.882±0.010	0.007	**0.912±0.004**
33	**0.0**	–	**0.0**	–	**0.0**	–	**0.0**	–	**0.0**
39	0.25	0.002	0.778±0.018	0.002	0.899±0.006	0.002	0.793±0.012	0.003	**0.952±0.003**
42	0.514	0.002	0.845±0.007	0.003	0.898±0.003	0.045	0.798±0.005	0.003	**0.909±0.006**
48	0.0	0.002	**1.0**	0.987	**1.0**	0.987	**1.0**	0.987	0.875±0.042
Σ	0.311	0.003	0.770±0.004	0.003	0.806±0.002	0.003	0.815±0.001	0.003	**0.926±0.003**
mean	0.179	0.003	0.757±0.008	0.003	0.746±0.005	0.003	0.767±0.006	0.003	**0.896±0.006**
median	0.129	0.002	0.837±0.009	0.003	0.883±0.003	0.003	0.882±0.006	0.003	**0.953±0.002**

elliptic envelope based on the minimum covariance determinant estimator (MCD) [142], and finally, a simple method using only the Mahalanobis distance. Overall, we found the simple Mahalanobis-distance-based approach to deliver the best results. The other algorithms could partially produce similar results but required significantly more computation time and mostly required additional hyper-parameters, which had to be tuned first. Using the Mahalanobis distance as the anomaly score has the advantage that the algorithm only has to compute a mean vector and a covariance matrix, which is computationally less expensive and does not require any additional hyper-parameters. One reason for the higher accuracy

of this method over the other outlier detection algorithms could be that the reconstruction errors are bell-shaped in every dimension (elliptic in higher dimensions), as we observed in visualizations of the error distributions.

Downsampling and Upsampling Approaches We experimented with different approaches to create a bottleneck along the time axis and to restore the original time resolution again. There are several possibilities to decimate/downsample a sequence by a factor s:

1. Resampling: Keep every s-th sample of the sequence. This is the simplest method but could lead to artefacts in the resulting sequence, since there might be aliasing effects if higher frequencies are present.
2. Average pooling: Use an average pooling layer, where groups of size s are averaged. Effectively, the operation is a moving average whose output is resampled. In some sense, average pooling acts as a crude low-pass filter, which removes higher frequencies and reduces aliasing effects.
3. Max-Pooling: Similar to average pooling, with the difference that the maximum value of groups of size s is selected.
4. Strided convolutions: One could use a convolutional layer where the filters are moved with a stride of s. At the same time, the number of filters n_{filters} in this layer can be used to reduce the dimension of the feature map to a desired size.
5. Stepwise downsampling with convolutional layers and (average) pooling: Smoothen the downsampling process by using a sequence of pooling and convolutional layers. For example, to achieve a downsampling rate of $s = 32$, one could use a series of 5 pooling and convolutional layers.
6. Applying a regularizer to the activations: Instead of downsampling the sequence along the time axis, sparsity can also be enforced by applying a regularizer to the activations of the last layer in the encoder. We experimented with L1- and L2-regularization [12] and the Kullback-Leibler [118] divergence as penalty terms.

Accordingly, we also tested several different upsampling techniques:

1. Sample-and-hold interpolation: This is the simplest upsampling approach, where each sample is copied s times in order to recover the original length
2. Linear interpolation: Linearly interpolate between adjacent samples.
3. Tranposed convolutional layer: Use transposed convolutions [99, 40] to obtain the original length of the time series. Transposed convolutions are still used often in practice, but can suffer from so-called checkerboard effects [122].
4. Stepwise upsampling: Analogous to method 5 described in the previous list.
5. Max-Unpooling in combination with Max-Pooling [120].

Surprisingly, we observed that the results for the simple methods average pooling (downsampling step) and sample-and-hold interpolation (upsampling step) are similar and, in some cases, even superior to the supposedly more advanced approaches.

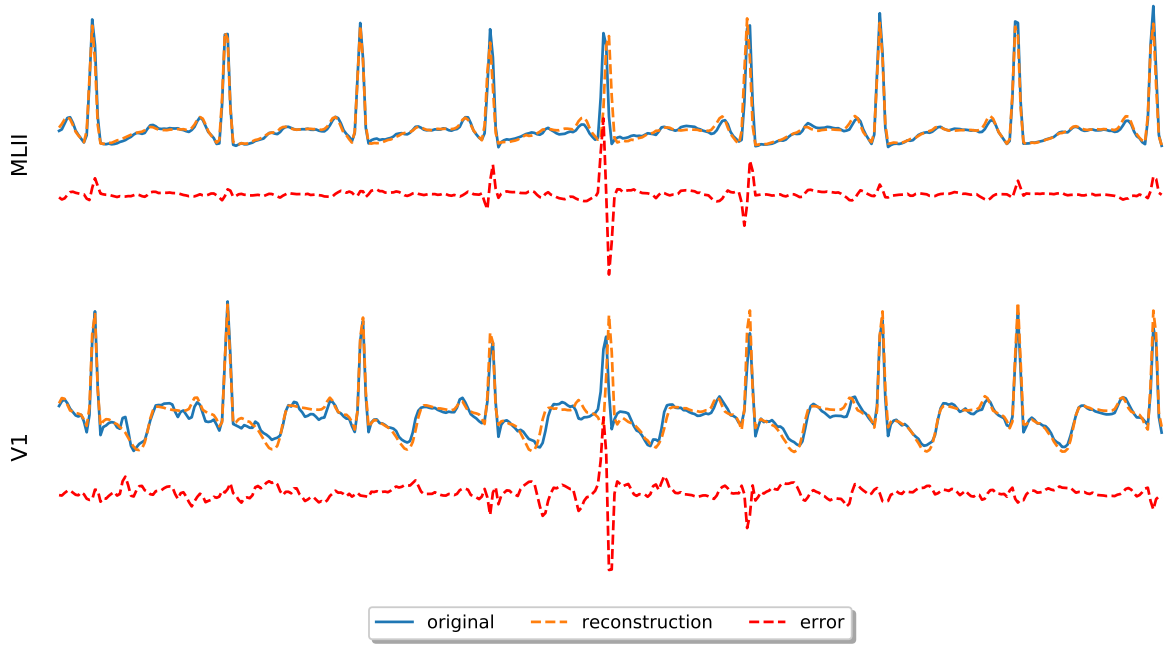

Figure 7.11: Similar to Figure 7.7, but now showing an excerpt of ECG signal #48. For both signals, one can observe a slight shift to the right in the reconstruction of the 5th R-peak, which results in an unusually large error and, ultimately, the TCN-AE algorithm falsely detects an anomaly (false positive) at this position. Taking the (ECG) signal's variability into account, in order to prevent situations like these, is addressed in our ongoing work.

However, it might be possible that more sophisticated down- and upsampling methods are beneficial in other situations, where the time-series data and algorithmic setup is different. Especially for the downsampling, one has to consider that, when using naive decimation approaches, aliasing effects might occur, which introduce artifacts into the compressed signal. We are planning to investigate this topic more thoroughly in the future.

7.4.4.4 Discussion

Only for the last ECG recording (#48), our final version of TCN-AE performs significantly worse most of the other algorithms. Surprisingly, for this time series, the final TCN-AE is also worse than most other variants without one of the enhancements, i.e., it is the combination of all additional modules that leads to the deterioration of the result for ECG signal #48. We found that the final TCN-AE algorithm produced an additional false-positive event, which reduces the overall precision on this time series. Exemplarily, we illustrate the cause of this FP in Fig. 7.11: one can see that a reconstructed R-peak appears slightly shifted, resulting in a large reconstruction error. The reason for this problem could be the quasi-periodic nature of the ECG signal, which is a major challenge for many algorithms.

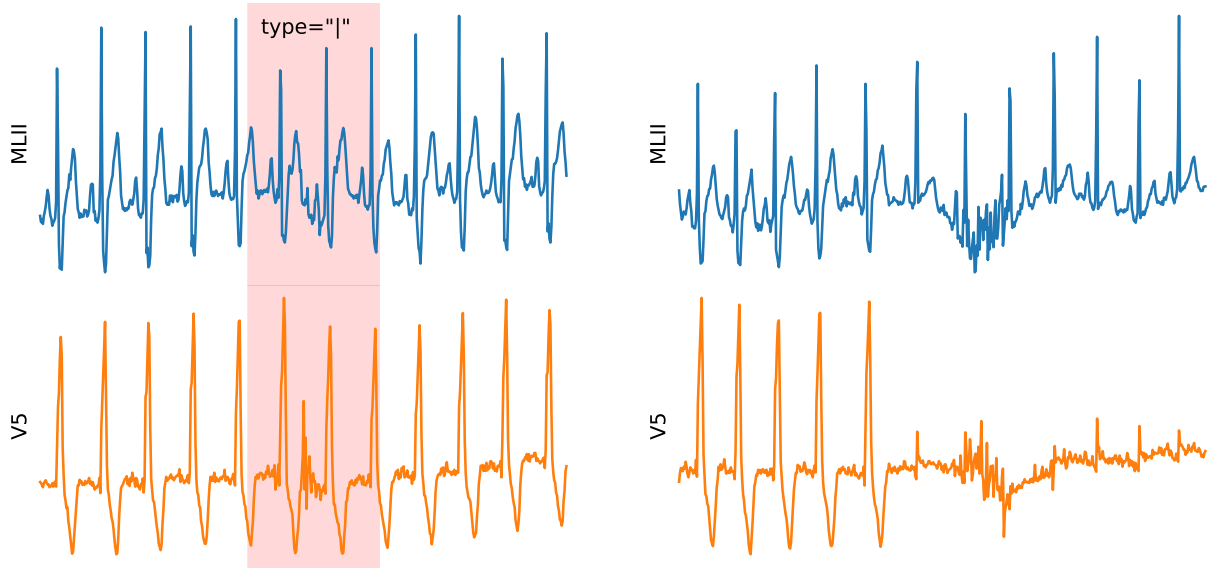

Figure 7.12: Two segments taken from ECG time series #33. This time series only contains one isolated QRS-like artifact (shown on the left side). The segment on the right side does not contain any anomalies, although there is a significant change in the signal. These signal quality changes occasionally occurring in this time series make it rather challenging for the unsupervised anomaly detection algorithms to detect the single anomaly.

We could observe similar events of this kind in a few other time series as well. Without the extra FP event in time series #48, TCN-AE would also achieve an F_1-score of $F_1 = 1.0$.

For time series #33 the results are unusual: we observe that all algorithms have $F_1 = 0$. This is because the time series contains only a single anomaly, which is rather hard to spot, even for the human eye. In Figure 7.12, the problem is illustrated: While the segment on the left contains the anomaly (an isolated QRS-like artifact), the segment shown on the right does not contain any anomaly, although there are significant changes in the signal quality. This makes it rather challenging for unsupervised algorithms to detect the single anomaly. We could observe many such signal quality changes in time series #33. If we discarded time series #33 from the benchmark, the average F_1-score would improve for all algorithms; for example, we would observe an increase from $F_1 = 0.896$ to $F_1 = 0.933$ for TCN-AE.

Also, there are a few other time series (#11, #17, #20, #22, #48) for which the majority of algorithms obtained a perfect F_1-score of $F_1 = 1.0$. These time series (except #48, which has four anomalies) only contain a single anomaly.

Although there are differences to supervised heartbeat classification algorithms, we found that our results are also roughly comparable to a few works in the literature: In [86], a F_1-score of $F_1 = 0.93$ is obtained, if we consider only the anomaly classes also used in this

work. In [169], the F_1-score on the test set for the overlapping anomaly classes is $F_1 = 0.84$, which is slightly lower than the score reported by us ($F_1 = 0.93$, Table 7.5).

Overall, we have shown that the TCN-AE architecture can produce competitive results on a challenging anomaly detection benchmark. We found that the TCN-AE architecture has several appealing properties which can be advantageous in time series anomaly detection, some of which we list in the following:

- Receptive field: Due to the hierarchical dilated convolutional structure, the size of the receptive field of the network can easily be scaled to the requirements of the problem.
- Skip Connections: Due to the introduced skip connections, TCN-AE is less sensitive towards the choice of the dilation rates (for example, we could choose dilation rates $1 \dots 64$ or $1 \dots 256$ and obtain similar results in both cases).
- Utilization of hidden Representations: Outputs of intermediate dilated convolutional layers are utilized, which allows using information processed at different time scales. This information is useful for obtaining an accurate reconstruction of the input and scanning for anomalies at different time scales.
- Fast Training: Due to the parallelizable convolution operation, time series can be processed very fast using GPUs (in this study, less than 50 seconds per time series, as shown in Table 7.8).
- Number of Weights: TCN-AE appears to potentially require less trainable weights than other architectures (e.g., recurrent neural networks) to obtain a good model accuracy. However, this claim has to be verified in more thorough studies in the future.

7.5 Conclusion and Possible Future Work

In this chapter, we introduced a novel temporal convolutional autoencoder (TCN-AE) architecture, which is designed to learn compressed representations of time series data in an unsupervised fashion. It is, to the best of our knowledge, the first work showing the combination of TCN and AE.

Starting with a baseline model and evaluating it on the Mackey-Glass anomaly benchmark (MGAB), we could already obtain promising results: The baseline TCN-AE performed best among 3 other state-of-the-art algorithms and could learn interesting representations of Mackey-Glass time series. However, we also noticed a few limitations which we addressed and led to a new TCN-AE algorithm with several modifications. We demonstrated the new algorithm's efficacy on a challenging real-world anomaly detection task, consisting of 30-minute electrocardiogram (ECG) readings of 25 patients, and could outperform several other unsupervised state-of-the-art algorithms on the investigated problem. Starting with a baseline model, we showed that several extensions are crucial to increase the overall performance. Particularly, we found that skip connections from the encoder's dilated convolutional layers to the bottleneck and skip connections from the decoder's dilated convolutional layers to the final reconstruction (output) layer improve the overall learning significantly.

Furthermore, the utilization of hidden representations (the outputs of the individual dilated convolutional layers) inside TCN-AE appears to be of considerable importance. The results suggest that temporal anomalies become more apparent on some time scales than on others. Another important finding was that TCN-AE was 5 times faster in our experiments than the second fastest algorithm (DNN-AE, see Table 7.8).

In summary, we demonstrated that it is possible to train a deep learning model without supervision, which can be used after training to detect anomalies in multivariate time series data. The novel TCN-AE model proposed in this work appears to be particularly well suited to learn long-range temporal patterns in complex quasi-periodic time series.

In our future research, we are planning to address several aspects of TCN-AE, which have not been thoroughly understood or investigated yet: (a) Application of the architecture to other challenging real-world anomaly detection problems. (b) Gaining more insights from the representations that TCN-AE learns unsupervisedly (Fig. 7.3). (c) Approaches for the determination of suitable anomaly thresholds with severely limited labeled data. (d) Analyzing time series with a higher ratio anomalous/normal data: In this work, we analyzed time series with not more than 250 anomalous events per patient. It is possible that TCN-AE also works for data with more anomalies. We plan to investigate our algorithm for settings with significantly larger anomaly ratios.