Temporal convolutional autoencoder for unsupervised anomaly detection in time series

Thill, M.; Konen, W.; Wang, H.; Bäck, T.H.W.

Citation
Thill, M., Konen, W., Wang, H., & Bäck, T. H. W. (2021). Temporal convolutional autoencoder for unsupervised anomaly detection in time series. *Applied Soft Computing*, *112*. doi:10.1016/j.asoc.2021.107751

Version:	Publisher's Version
License:	Licensed under Article 25fa Copyright Act/Law (Amendment Taverne)
Downloaded from:	https://hdl.handle.net/1887/3280042

Note: To cite this publication please use the final published version (if applicable).

Applied Soft Computing 112 (2021) 107751

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Temporal convolutional autoencoder for unsupervised anomaly detection in time series

Markus Thill [a,*], Wolfgang Konen [a], Hao Wang [b], Thomas Bäck [b]

[a] TH Köln – University of Applied Sciences, 51643 Gummersbach, Germany
[b] Leiden University, LIACS, 2333 CA Leiden, The Netherlands

ARTICLE INFO

Article history:
Received 2 February 2021
Received in revised form 17 July 2021
Accepted 22 July 2021
Available online 2 August 2021

Keywords:
Anomaly detection
Deep learning
TCN
Autoencoder
Mahalanobis distance

ABSTRACT

Learning temporal patterns in time series remains a challenging task up until today. Particularly for anomaly detection in time series, it is essential to learn the underlying structure of a system's normal behavior. Periodic or quasiperiodic signals with complex temporal patterns make the problem even more challenging: Anomalies may be a hard-to-detect deviation from the normal recurring pattern. In this paper, we present TCN-AE, a **t**emporal **c**onvolutional **n**etwork **a**uto**e**ncoder based on dilated convolutions. Contrary to many other anomaly detection algorithms, TCN-AE is trained in an unsupervised manner. The algorithm demonstrates its efficacy on a comprehensive real-world anomaly benchmark comprising electrocardiogram (ECG) recordings of patients with cardiac arrhythmia. TCN-AE significantly outperforms several other unsupervised state-of-the-art anomaly detection algorithms. Moreover, we investigate the contribution of the individual enhancements and show that each new ingredient improves the overall performance on the investigated benchmark.

© 2021 Elsevier B.V. All rights reserved.

1. Introduction

The digitization of companies and their processes, households, and many other public institutions is progressing at an increasing pace. Due to the extensive cross-linking and networking of systems, large amounts of complex data are continuously generated. Today, companies are equipping even the smallest devices with numerous sensors that consistently supply various measurements and other indicators. New technologies for the Internet of things, cyber–physical systems (CPS), and other related domains enable manufacturers to equip such devices with notable computing power, networking capabilities, and numerous sensors. These devices can consistently record and distribute various measurements and other information. In the past, data was often analyzed and interpreted manually by human experts. This has become a considerable challenge in recent years due to the sheer endless amounts of data and the associated manifold complexity of the data. In this context, the necessity for automated, intelligent, and adaptable analytical approaches arose, and researchers invested much effort into developing new methods and algorithms.

Especially in the field of machine learning (ML) and particularly deep learning (DL), the research community has made notable progress in recent years. ML and DL models are capable of learning solely from data and can be trained for various tasks. However, still many challenges and unresolved questions remain up until today. For example, many ML/DL algorithms have to be supervised and commonly require large amounts of labeled data for the training process. To generate labeled datasets, often a tedious and time-consuming labeling process is needed, which has to be carried out by human domain experts. In some application areas, the available labeled data is not sufficient to train a model successfully, and also, the accessible data that could be labeled is somewhat limited. In particular, in anomaly detection applications, one has to deal with datasets with little labeled data. Anomaly detection is concerned with the task of discovering events that deviate from what is considered normal (nominal)ehavior, occurring in unusual or unexpected situations. Typically, anomalies indicate a malfunction, an error, or another issue in the monitored system and mainly require immediate action to prevent (further) damage or harm. Identifying anomalous behavior is becoming increasingly important in many fields, such as predictive maintenance, fraud detection, networking, and health monitoring systems (HMS). In this work, discussed below in greater detail, we will study a particular anomaly detection problem taken from the field of HMS.

Systematically, anomalous events can be categorized [1] as point (a single data instance being an statistical outlier), contextual (only anomalous under a certain condition on the data), and collective anomalies (a set of data points differs from the others while each constituting point is very likely normal when considered alone). For example, in the electrocardiogram (ECG)

* Corresponding author.
E-mail addresses: markus.thill@th-koeln.de (M. Thill), wolfgang.konen@th-koeln.de (W. Konen), h.wang@liacs.leidenuniv.nl (H. Wang), t.h.w.baeck@liacs.leidenuniv.nl (T. Bäck).

https://doi.org/10.1016/j.asoc.2021.107751
1568-4946/© 2021 Elsevier B.V. All rights reserved.

signals, arrhythmias is typically seen as a collective anomaly, as illustrated in Fig. 1.

As already noted, labeled data for anomaly detection tasks are usually relatively sparse or not available at all. Therefore, many machine learning algorithms in this field rely on unsupervised learning techniques instead of supervised approaches. Unsupervised learning algorithms attempt to learn patterns or structure in the data without relying on expert knowledge. For anomaly detection, a common assumption is that a large proportion of the available data instances represent nominal (normal) behavior (however, not necessarily exclusively nominal), which can be learned by a model. Based on its understanding of what is usual or expected, the model can later be used in deployment to distinguish between normal and abnormal patterns.

To understand the complexity of anomaly detection in quasiperiodic temporal data, we show in Fig. 1 an anomaly example. The normal signal is quasiperiodic (peak height and other small details may differ from period to period to some extent). The anomalous region has a very similar peak, yet it comes earlier than expected. The model that predicts anomalies thus has to learn the shape of the peak and when to expect it. This requires the processing of long-range information since the local neighborhood of the anomalous peak looks absolutely normal.

In this paper, we propose a novel temporal autoencoder architecture based on convolutional neural networks, in the following referred to as TCN-AE, capable of processing long-range information in time series. 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. TCN-AE 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). After training TCN-AE, we use the reconstruction error as an indicator for abnormal behavior. We analyze, discuss, and compare the capabilities of TCN-AE on a challenging real-world HMS application, namely the detection of arrhythmias in electrocardiogram (ECG) signals of heart patients.

We formulate the following key questions that drive our research:

- How well 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 paper 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 important to process the data on different time scales (like TCN and wavelets do) and to utilize the information from different time scales in the anomaly detection process.

The rest of this paper is organized as follows: Sections 1.1 and 1.2 present work being related to our approach while Section 2 summarizes the TCN approach and introduces the autoencoder enhancement and other enhancements added to TCN by us. In Section 3, we present our experimental setup, and in Section 4, we show and discuss the results obtained in this work. Finally, we conclude this work in Section 5.

1.1. Related work

Anomaly detection is a broad research field, where especially machine learning (ML) approaches became increasingly popular over the past years. An important subdomain is (unsupervised) anomaly detection in time series. A recent review on unsupervised learning in general is given in [2]. Clustering, undoubtedly the most well-known sub-field of unsupervised learning, is covered in twelve chapters of [2]; a recent review of clustering methods in general is found in [3]. A very comprehensive review of anomaly detection methods was given by Chandola et al. [1]. Goldstein & Uchida [4] give a recent review and comparative evaluation of *unsupervised* anomaly detection methods. A recent survey by Thudumu et al. [5] covers the state of the art in anomaly detection methods for high dimensional data and/or big data. Basora et al. [6] provide a comprehensive taxonomy for all major anomaly detection methods: These methods can be divided into distance-based (including clustering-based), ensemble-based (including isolation forests [7] and LOF [8]), statistical and reconstruction-based (including PCA and neural networks) anomaly detection methods.

Reconstruction-based methods can be described in a nutshell as follows: Similar to forecasting, which is a well-studied problem in the literature, one common approach is to build an (auto-) regressive model for the time series and to use the prediction errors as indicators for anomalous behavior. Commonly, the models used range from simple linear functions [9] to LSTM [10,11] networks and convolutional neural networks [12]. Other popular approaches are based on autoencoders (AE), which learn to compress and reconstruct time series (or segments thereof) and detect anomalies based on the reconstruction error [13–15]. Several works use variational autoencoders to learn the time series behavior and compute a reconstruction probability, which serves as an anomaly score [16–19]. A relatively new area of time series anomaly detection is concerned with the application of generative adversarial networks (GANs) [20–22]. Very recently, the attention mechanism has been applied to capture long-term temporal correlations [23] and the graph neural networks (GNNs) are employed to learn the cross-correlations between different series explicitly [24]. Other popular approaches are based on finding discords in time series using symbolic representations [25–27].

Today, anomaly detection is used in many application domains. Some recent examples of such works are anomaly detection in avionics [6], in network security and intrusion detection [28,29], in the energy sector [30] and in the financial domain [31].

1.2. Related work in ECG and TCN

Many articles are concerned with the detection of arrhythmias in ECG signals. However, work on unsupervised approaches appears to be less common in this field. The presented methods are mostly supervised algorithms that are trained to classify different arrhythmia types. Luz et al. give a comprehensive overview of various classification approaches for ECGs in [32]. In [33], Hannun et al. designed a 34-layer convolutional deep neural network to classify 12 different heart arrhythmia types. However, due to its nature, the architecture is supervised and requires annotated data for training. The authors use a massive labeled dataset containing 91,232 single-lead ECGs from 53,549 patients (not publicly available). The trained model achieves very high accuracy on a cardiologist level. Several researchers base their approaches on the discrete wavelet transform (DWT) [34–37]. Thomas et al. [36] extract, next to other features (some of which might partially require expert knowledge, such as the RR-intervals), dual-tree

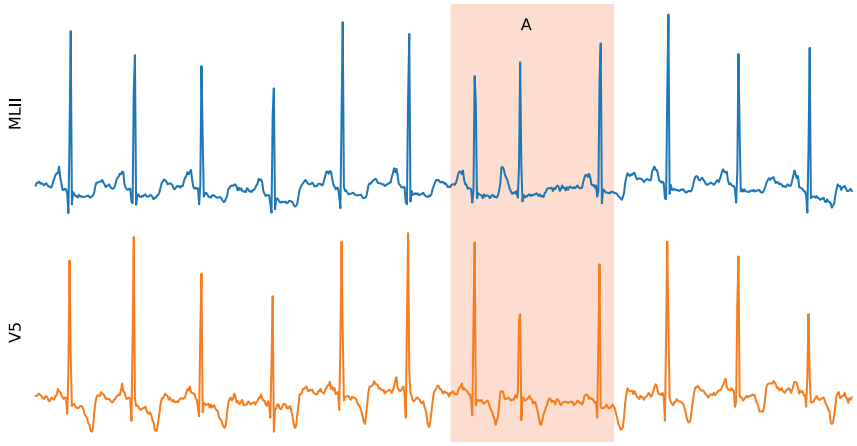

Fig. 1. An anomaly example from the ECG data. This is just one type of anomaly out of the set of 9 different anomaly types (to be discussed later in more detail, see Table 1).

complex wavelet-based features from the ECG signal and train a neural network for four arrhythmia classes. We found several works that introduce anomaly detection methods in ECG readings [10,11,38–41]: Chauhan & Vig [38] train an LSTM network and analyze 1-minute segments of 48 ECG recordings (each about 30 min long) from the MIT-BIH ECG benchmark [42–44]. However, [10,38] have the drawback that they have to divide the data into training, validation, and test sets, which have to be divided further into normal and anomalous subsets. In practice, this leads to difficulties since expert knowledge will be required to classify and split the data before the training process. In [11], the ideas of [10,38] are extended, and also the requirement of different data splits is eliminated. Sivaraks et al. [41] use motif discovery and propose an approach for robust anomaly detection in ECG data.

Publicly, there are only relatively few benchmarks for ECG arrhythmia detection available: The MIT-BIH Arrhythmia database [42–44], the CU Ventricular Tachyarrhythmia Database [45], and the St. Petersburg INCART 12-lead Arrhythmia Database [42]. We will use the MIT-BIH benchmark in this work since it contains the most patients (47 patients) and sufficiently long ECG recordings (30 min), and it is the most commonly used benchmark in the literature.

Several earlier works inspired the TCN-AE architecture that is presented in this paper: While Holschneider et al. applied dilated convolutions in their "algorithme à trous" algorithm in the field of wavelet decomposition already in 1990 [46], 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. [47] introduced the WaveNet architecture, which uses dilated convolutions for the generation of raw audio. Yu & Koltun [48] successfully employed dilated convolutions to the task of semantic image segmentation. Later, Bai et al. [49] 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, no earlier work employs TCNs in an autoencoder-like architecture. We only found one approach for time series anomaly detection that is based on TCNs [50]. However, it does not use autoencoders. Its general idea is more similar to [10], and [11], which use forecasting errors as an indicator for abnormal behavior.

2. The TCN Autoencoder

This section introduces the Temporal Convolutional Network Autoencoder (TCN-AE), describes its main components, and discusses a few of its properties and application areas for time series analysis. We will start with a baseline architecture and successively add several enhancements to this architecture. As the name suggests, TCN-AE is a convolutional architecture. Convolutional neural networks are broadly and with great success used in computer vision applications, where other fully connected/dense architectures commonly suffer from the curse of dimensionality [51,52]. Convolutional nets have several beneficial properties, such as translation invariance, weight sharing, and computational efficiency. These properties make them especially suitable 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.

2.1. Intuition

Conceptually, TCN-AE, as proposed here, is similar to other autoencoder architectures. However, it differs from regular autoencoders because it replaces the fully connected/dense layers with dilated 1D-convolutional layers. Due to this, the network can take into account temporal relationships in the data more naturally and is flexible regarding inputs of variable size. Furthermore, the 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 usually has fewer weights than dense AE architectures.

The key idea of TCN-AE is to learn how to efficiently encode a d-dimensional input sequence of length T_{train} by compressing it along the time axis and possibly also along the feature axis. Subsequently, TCN-AE attempts to decode the compressed representation and reconstruct the original input sequence again.

The intuition behind this approach is that due to the bottleneck in the architecture, the network is forced to identify useful temporal patterns in the data, which in turn allows finding efficient encodings of the input. 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 [53].

Our proposed TCN-AE architecture consists of several building blocks, which will be described in the following.

2.2. Dilated convolutions

Convolutional layers in neural networks are comprised of digital filters, which remove or amplify specific 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 nth 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:

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

where $*$ denotes the convolution operator, $y[n] \in \mathbb{R}$ is the output of the filter, $h[i] \in \mathbb{R}$ is the ith 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 sliding along the time axis, the operation is usually referred to as one-dimensional convolution. The filter's behavior is determined by $h[n]$ (e.g., low-pass or high-pass characteristics). The central idea of convolutional neural networks is not to predetermine $h[n]$ 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:

$$y[n] = (\mathbf{x} * \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \mathbf{x}[n-i]. \tag{2}$$

In contrast to the regular convolution operation (as specified above), the dilated convolution [48] 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 taps $\mathbf{h}[i]$ and $\mathbf{h}[i+1]$. The dilated convolution is written as:

$$y[n] = (\mathbf{x} *_q \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \mathbf{x}[n-qi]. \tag{3}$$

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

In many applications also acausal convolutions are used (e.g. [54]). In this case, future values of a sequence $x[n]$ will be processed to generate output $y[n]$:

$$y[n] = (\mathbf{x} *_q \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \mathbf{x}\left[n - q\lfloor i - k/2 \rfloor\right] \tag{4}$$

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 ECG anomaly detection task. Note that this comes at the cost of slight delays in online settings.

2.2.1. 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 composed of, or is comprising 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., [47,49]) utilize dilated convolutions to create a hierarchical temporal model with a large receptive field that is 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 without reducing the resolution, contrary to pooling or strided convolutions. In general, the receptive field r for the causal and acausal case is given by:

$$r_{\text{causal}} = k2^{L-1}, \tag{5}$$

$$r_{\text{acausal}} = \lfloor k/2 \rfloor (2^{L+1} - 2) + 1, \tag{6}$$

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 size of the receptive field will be $3 \times 2^4 = 48$ for the causal case and $2^6 - 1 = 63$ for the acausal setting.

In summary, a convolutional layer can be mainly described by 3 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.

2.2.2. 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. [46] proposed an efficient algorithm for computing the non-decimating DWT using dilated convolutions. Current deep learning architectures [47–49] 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 filter weights for the DWT 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).

2.3. Temporal convolutional networks

The temporal convolutional network (TCN) [49] is inspired by several convolutional architectures [47,55–57], but differs from these approaches insofar as it combines simplicity, autoregressive prediction, residual blocks, and very long memory. Essentially, a TCN is a stack of n residual blocks which chain two sub-blocks sequentially. Each sub-block comprises a sequence of a dilated convolutional layer, a weight normalization layer [58], a ReLu activation function [59], and a spatial dropout layer [60]. Furthermore, a skip (residual) connection [61] bypasses the residual block and is added to the its 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 $\mathbf{y} : \mathbb{T} \to \mathbb{R}^{n_{\text{filters}}}$.

2.4. Baseline TCN-AE

We use TCN as a building block for a baseline temporal autoencoder, in the following referred to as baseline TCN-AE. In later sections, we will modify the baseline TCN-AE and add further enhancements to the architecture.

The baseline TCN-AE consists of an encoder network $enc(\cdot)$ and a decoder network $dec(\cdot)$.

The encoder $enc(\cdot)$, shown in Fig. 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] [62,63] and finally down-samples the series along the time axis by a specified factor using an average-pooling layer. 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 $\mathbf{g}[n] = enc(\mathbf{x}[n])$, where $\mathbf{g} : \{0, 1, \ldots, T/s - 1\} \to \mathbb{R}^c$.

The decoder $dec(\cdot)$, shown in Fig. 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 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) $\hat{\mathbf{x}}[n] = dec(\mathbf{g}[n])$, $\hat{\mathbf{x}} : \mathbb{T} \to \mathbb{R}^d$.

For the sake of simplicity, the TCN architecture of the encoder and the decoder is the same. However, this identical structure is by no means necessary. In principle, the TCN can be parameterized differently in the encoder and decoder.

2.5. Unsupervised anomaly detection with TCN-AE

Due to the bottleneck in the architecture, the training procedure forces TCN-AE to learn compressed encodings of the input sequences, which 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, which 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 $\mathbf{E} \in \mathbb{R}^{\ell \times d}$. 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 abnormal/anomalous points in the ℓ-dimensional space. The outlier detection algorithm outputs an anomaly score, which decides the occurrence of an anomaly based on some threshold. After experimenting with most of well-known outlier detection algorithms, e.g., local outlier factor (LOF) [8], we discovered that a simple approach based on the Mahalanobis distance (line 16 in Algorithm 1) 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.[2] We summarize the anomaly detection algorithm for TCN-AE in Algorithm 1.

Algorithm 1 General anomaly detection algorithm using the TCN-AE architecture.

1: **Adjustable parameters**:
2: $\mathcal{M}_\tau$: anomaly threshold (see Section 3.3), ℓ: error window length
3: T_{train}: length of training sub-sequences, B: batch size
4:
5: **function** ANOMALYDETECT($\mathbf{x}[n]$) ▷ $\mathbf{x} : \mathbb{T} \to \mathbb{R}^d$, $\mathbb{T} = \{0, 1, \ldots, T-1\}$
6: **Construct** model TCNAE() and **Initialize** the trainable parameters
7: $\mathbf{X}_{\text{train}} \leftarrow$ { Sub-sequences of length T_{train} taken from $\mathbf{x}[n]$ }
8: **for** $\{1 \ldots n_{\text{epochs}}\}$ **do**
9: TRAIN(TCNAE, $\mathbf{X}_{\text{train}}$) ▷ Train with random mini-batches of size B
10: **end for**
11: $\hat{\mathbf{x}}[n] \leftarrow$ TCNAE($\mathbf{x}[n]$) ▷ Encode and reconstruct $\mathbf{x}[n]$
12: $\mathbf{e}[n] \leftarrow \mathbf{x}[n] - \hat{\mathbf{x}}[n]$ ▷ reconstr. error $\mathbf{e} : \mathbb{T} \to \mathbb{R}^d$
13: $\mathbf{E}[n] \leftarrow$ SLIDINGWINDOW($\mathbf{e}[n]$, ℓ) ▷ $\mathbf{E} : \mathbb{T} \to \mathbb{R}^{\ell \times d}$
14: $\mathbf{E}'[n] \leftarrow$ RESHAPE($\mathbf{E}[n]$) ▷ $\mathbf{E}' : \mathbb{T} \to \mathbb{R}^{\ell \cdot d}$
15: $\boldsymbol{\mu}, \boldsymbol{\Sigma} \leftarrow$ ESTIMATE($\mathbf{E}'[n]$) ▷ Mean $\boldsymbol{\mu} \in \mathbb{R}^{\ell \cdot d}$, Cov. Mat. $\boldsymbol{\Sigma} \in \mathbb{R}^{\ell \cdot d \times \ell \cdot d}$
16: $M[n] \leftarrow (\mathbf{E}'[n] - \boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1} (\mathbf{E}'[n] - \boldsymbol{\mu})$ ▷ Mahalanobis distance
17: $a[n] \leftarrow \begin{cases} 0 & \text{if } M[n] < \mathcal{M}_\tau \\ 1 & \text{else} \end{cases}$ ▷ Binary anomaly flags
18: **return** $a[n]$ ▷ Return anomaly flag for each time series point
19: **end function**

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 Section 3.4.

2.6. Enhancements of the baseline TCN-AE

2.6.1. 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.

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

[2] In the extreme case where the number of signals in the series is larger than the window size l, the maximum likelihood estimate covariance will be singular. To resolve this issue, one can always compute the pseudo-inverse of the singular matrix or even better, resort to the so-called regularization technique to learn a non-singular one.

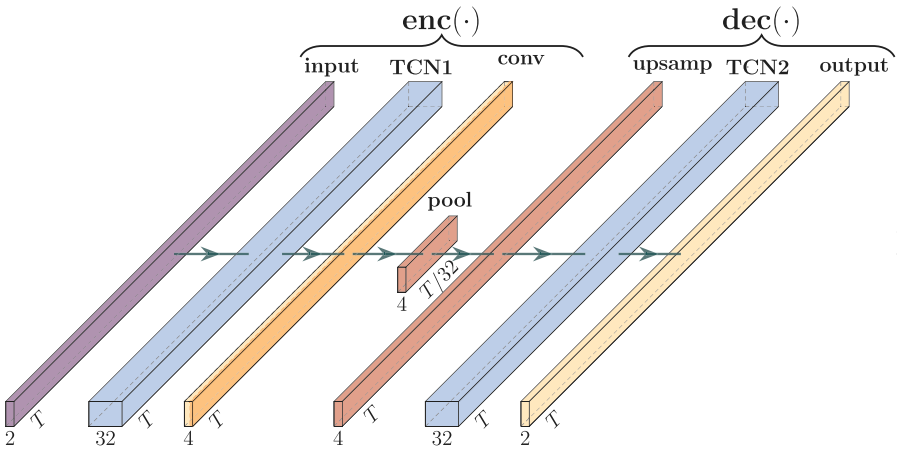

Fig. 2. Architecture of the baseline TCN-AE as described in Section 2.4. The input of TCN-AE is a sequence $\mathbf{x}[n]$ with length T and dimensionality $d = 2$ for the ECG data. The layers "conv" and "output" are 1×1 convolutions with $c = 4$ and $d = 2$ filters, respectively. Afterwards, the output of the encoder is downsampled by an average-pooling layer ("pool") with a pool size $s = 32$.

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 network layer. In our setup, we use a concatenation layer at the end of the encoder and decoder, which collects the outputs of all previous dilated convolutional layers.

In the encoder shown in Fig. 3, we add skip connections from every dilated convolutional layer to the encoder's bottleneck (after reducing the number of channels to 16 by a 1×1-convolution), 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. 4, we also place skip connections from each dilated convolutional layer to the output. 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 [61], 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 [61] (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 [64], 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 by decreasing the number of filters in all layers. 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).

2.6.2. 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 Figs. 3 and 4).

2.6.3. Utilizing hidden representations for the anomaly detection task

While studying the relation of dilated convolutions to the DWT (Section 2.2.2), we noticed some similarities to our prior work [65]: 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 2.6.4) 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. 3 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 ($q = 1, 2, \ldots, 64$) 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 1. With this approach, we cannot 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. 3 and 4 to keep the complexity of the figures manageable.

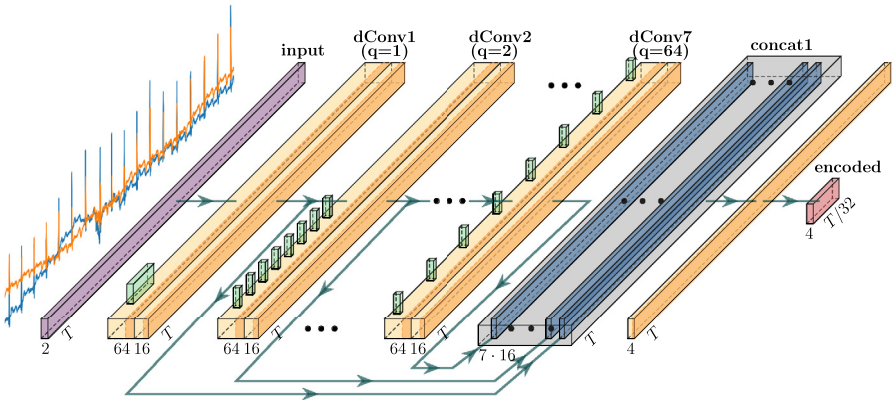

Fig. 3. The architecture of TCN-AE's encoder. The two-dimensional input ECG signal (purple) of length T is passed through a stack of dilated convolutional layers (light orange, dConv1–dConv7). The light green boxes (infront of dConv1–dConv7) represent the filters of the dilated convolutions. Each dilated 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 (inside concat1) 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 (Fig. 4).

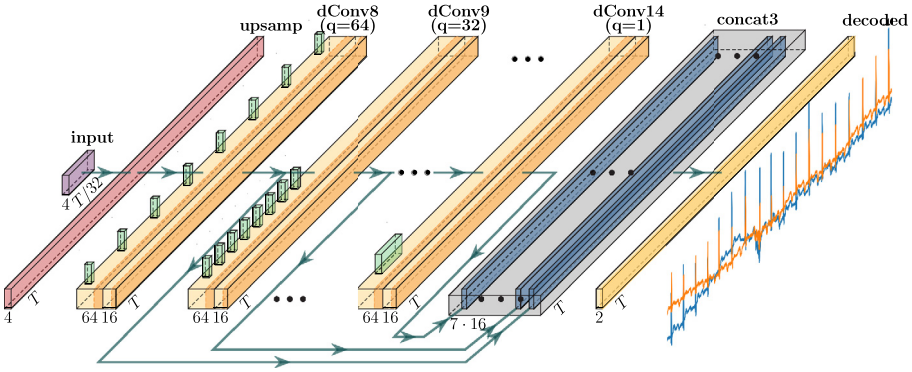

Fig. 4. The architecture of TCN-AE's decoder. A compressed representation is given as input (purple) and then upsampled (upsamp, red layer) to the original length T. Similar to Fig. 3, a stack of dilated convolutional layers operates on the upsampled signal, and the outputs of the 1×1 convolutions are concatenated in the concat3 block. Finally, the output layer (convolution with linear activation) reconstructs the original two-dimensional ECG sequence (last yellow block, decoded).

2.6.4. 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) [62,63], 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. Consequently, 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.

2.6.5. 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.

3. Experimental setup

3.1. The MIT-BIH Arrhythmia database

The MIT-BIH Arrhythmia database [42–44] contains two-channel electrocardiogram (ECG) signals of 48 patients of the Beth Israel Hospital (BIH) Arrhythmia Laboratory. Each signal was recorded with a sampling frequency of 360 Hz and has a length of approximately half an hour, which corresponds to 650 000 data points each. The two channels recorded are the modified limb lead II (MLII) and a modified lower lead V1, in a few cases V5. Each recording contains on average 2160 ± 365 heartbeats, and in total, there are 54 087 heartbeats.

There are many different events in all ECG time series, which human experts labeled. The whole list of heartbeat annotations is found in [43]. Our unsupervised approach investigated in this paper rests on the assumption that most time-series data is normal (nominal) and that anomalous events are relatively seldom. Therefore we select from the 48 time series (patients) only those 25 with 250 or fewer anomalous events. The selected time series contain 721 events from nine anomaly classes listed in Table 1. A more detailed database description can be found in [44].

Table 1
Anomaly types in the 25 ECG signals considered for the experiments. The descriptions are taken from [43]. The second column shows the overall number of the various anomaly types for the 25 considered ECG signals. All anomalies listed in this table are collective anomalies, as described in Section 1.

Code	#	Description
a	11	Aberrated atrial premature beat
A	235	Atrial premature beat
e	10	Atrial escape beat
f	22	Fusion of paced and normal beat
F	25	Fusion of ventricular and normal beat
J	3	Nodal (junctional) premature beat
V	374	Premature ventricular contraction
x	19	Non-conducted P-wave (blocked APC)
|	22	Isolated QRS-like artifact
Σ	721	

3.2. Preprocessing and data preparation

Since the raw 2-dimensional ECG signals contain a lot of noise and exhibit non-stationary behavior, we perform a few preprocessing steps before training the models. Initially, each signal is filtered with a bandpass filter parameterized with the cutoff frequencies of 2 and 20 Hz. These are commonly used values for the processing of ECG signals in practice (cf. Pan–Tompkins algorithm [66]). This bandpass filter removes most of the high-frequency noise in the signal and drifts in the baseline. To reduce the training time of each model and the model complexity, we down-sample each ECG signal by a factor of $n_{samp} = 5$. This reduces the length of the ECG signal from originally 650 000 time-steps to just $T = 130\,000$ time-steps without losing too much information from the signal.

We normalize each input time series to zero mean and unit variance before generating the training samples. Finally, we extract training samples of length T_{train} using a sliding window.

3.3. Algorithmic setup

We compare our unsupervised TCN-AE algorithm to ten other unsupervised anomaly detection algorithms, of which four are DL-based and 6 are not. The DL-based models are: DNN-AE [67], LSTM-ED [14], LSTM-AD [11], and NuPIC [68]. 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 DL algorithms, except NuPIC (which does not support GPU capabilities), on a Tesla P100 GPU. The remaining algorithms are, if not mentioned otherwise, parallelized and run on 40 Intel(R) Xeon(R) E5-2699 CPU cores with 2.20 GHz each. All algorithms require a set of hyperparameters, which we will describe in the following. Parameters common to all algorithms are summarized in Table 2. We tuned the parameters (except for TCN-AE and NuPIC) using the HYPEROPT library [69]. For TCN-AE, we manually investigated different parameter settings, and for NuPIC, we use the recommended parameter settings [70].

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

The non-DL algorithms are parameterized as described below. If not mentioned otherwise, we use the default parameter settings specified in the corresponding papers or packages. We combine the non-DL approaches with sliding windows with a window size w to generate meaningful time series embeddings. These embeddings are passed to the outlier detection algorithms.

SORAD [9]: SORAD is an online time series anomaly detection algorithm based on recursive least squares (RLS) [71, Chapter 13] estimation. It predicts the target horizons $H = (1, 3, \ldots, 49)$ and builds an error model based on these predictions. It uses a sliding window of size $w = 128$ and a forgetting factor of $\lambda = 0.98$.

LOF [8]: We use the scikit-learn [72] (v0.23.2) implementation of the Local Outlier Factor (LOF) algorithm. The sliding window size is set to $w = 64$, the number of neighbors to 20, and the leaf size to 30.

IF [7]: The Isolation Forest (IF) model (scikit-learn [72], v0.23.2) uses a number of 1000 base estimators in the ensemble and a sliding window size of $w = 50$.

GMM & *BGMM*: For the Gaussian Mixture Model (GMM) and the variational Bayesian Gaussian Mixture Model (BGMM), we again use scikit-learn (v0.23.2) [72] and set $w = 64$, the number of mixture components to 5, and the number of initializations to 3,

OCC-SVM [73]: A fast GPU-based implementation of the One-Class Classification Support Vector Machine (OCC-SVM) is available in the ThunderSVM [74] Python package (v0.3.12). We use OCC-SVM with a polynomial kernel $K(\mathbf{x}, \mathbf{y}) = (\gamma \mathbf{x}^\intercal \mathbf{y} + c)^d$, where $d = 3$, $c = 1$, and $\gamma = 1/128$ (default for the selected window size of $w = 64$).

The DL algorithms are configured as follows:

DNN-AE [67]: We use a PyTorch [75] implementation for the anomaly detection algorithm based on a deep autoencoder [13]. The algorithm requires several parameters, which we choose as follows: a hidden size of $h = 6$ for the bottleneck (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 [14] 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 LSTM layer having 50 units.

NuPIC [68]: Numenta's anomaly detection algorithm has a broad range of hyper-parameters that have to be set. We use the parameters recommended by the authors in [70]. It is possible to tune the parameters with an internal swarming tool [76]. However, this is a time-expensive process that is not feasible for the large benchmark.

LSTM-AD [11]: 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 $H = (1, 3, \ldots, 49)$.

TCN-AE (baseline): The settings of the baseline TCN-AE model (Fig. 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.

TCN-AE (final): We implemented TCN-AE using the Keras [77] & TensorFlow framework [78]. An overview of the architecture with its parameters is given in Figs. 3 and 4. 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 1 is $\ell = 128$. To verify the

Table 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 [79]
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 [79]

training process and to ensure that the model does not overfit, we track the training and validation loss for the individual time series. In Fig. A.13, we plot the logcosh loss exemplarily for several time series.

3.4. Performance measures

In the MIT-BIH benchmark, only the R-peaks of the QRS complex are labeled as either normal or with the corresponding arrhythmia type. However, it is usually impossible to locate an anomaly at exactly one point of the time series. In most cases, an anomaly is a longer temporal pattern that is unusual given its temporal context. Therefore it is not meaningful to label individual points of the time series as anomalous. Instead, we specify an anomaly window for each anomaly. The anomaly window is centered around the given label. It contains 400 (80, after downsampling the time series) data points before and after the label, which corresponds to approximately 2 s (or roughly one heartbeat before and after the label). The task of the anomaly detection algorithms is to detect the anomalies within the specified anomaly window. One or several correct detections within an anomaly window will be counted as one true positive (TP). On the other hand, if an algorithm fails to identify an anomaly within the window, a false negative (FN) will be attributed. Any detection outside of an anomaly window will be counted as false positives (FP). 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:

$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}}\ ,\ \text{Rec} = \frac{\text{TP}}{\text{TP} + \text{FN}}\ ,\ F_1 = 2 \cdot \frac{\text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}. \tag{7}$$

Each algorithm outputs an anomaly score for each point in the time series. Low values indicate nominal behavior, and high values indicate that an unusual situation has been observed. In order to classify each point as either nominal or anomalous, a so-called anomaly threshold is required. However, the threshold trades off false negatives (recall) and false positives (precision). The algorithms are typically compared based on the so-called equal accuracy (EAC), where precision and recall are approximately equal to compare algorithms based on these two objectives. We will use EAC as one performance indicator. Another possibility is to 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 ten 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 [80] and report the p-values.

Table 3
Summary of all TCN-AE variants.

Variant	Section	Comment
Baseline	2.4	Baseline algorithm based on TCNs without any enhancements
noSkip	2.6.1	Skip-Connections removed from the architecture
noInvDil	2.6.2	Use same dil. rate ordering for encoder & decoder
noLatent	2.6.3	Do not use hidden representations for anomaly detection
noRecon	2.6.3	Only use hidden representations of encoder for anomaly detection
noMapReduc	2.6.4	Do not use the Map reduction layers
noAnomScoreCorr	2.6.5	Do not correct the baseline of the anomaly score
Final	2.6	Final TCN-AE with all enhancements

4. Experiments, results & discussion

We started our experiments with the baseline TCN-AE model (Section 2.4). The initial results on the ECG benchmark were already promising, but the algorithm still performed similar to LSTM-AD and DNN-AE concerning the $F1$-score (Table 5), leaving room for improvements. While analyzing the baseline TCN-AE, we developed several ideas for improvements, which we introduced in Section 2.6. The resulting (final) TCN-AE showed significantly improved performance, achieving 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.

Fig. 5 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 Fig. 6. 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 abnormal pattern is incorrectly reconstructed, which results in a significant reconstruction error that can easily be detected.

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 Section 2.6 (see Table 3).

Although it is challenging to accurately measure each element's effect on the final result (since there might be some interaction effects between elements[3]), 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

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

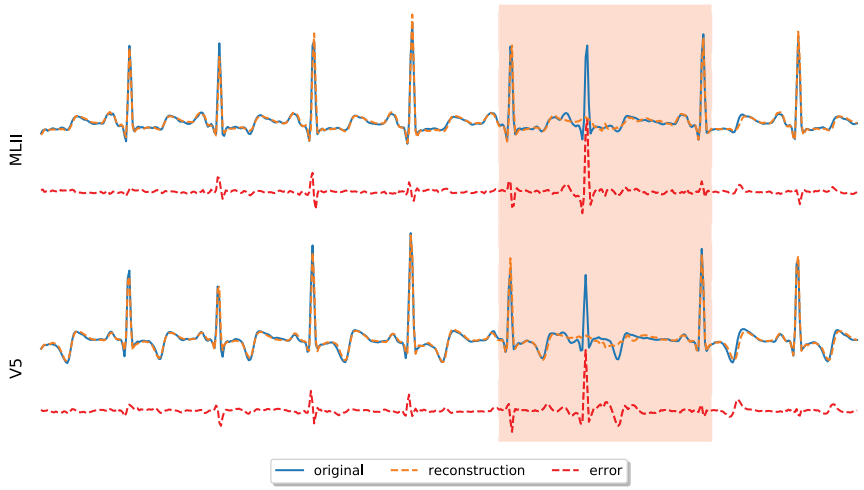

Fig. 5. Excerpt showing how the final TCN-AE model reconstructs the modified limb lead II (MLII) and the modified lead V1 of ECG signal #1. TCN-AE has difficulties in reconstructing actual anomalous behavior (highlighted with the red shaded area). Due to the resulting large error, the algorithm later correctly detects an anomaly (true positive).

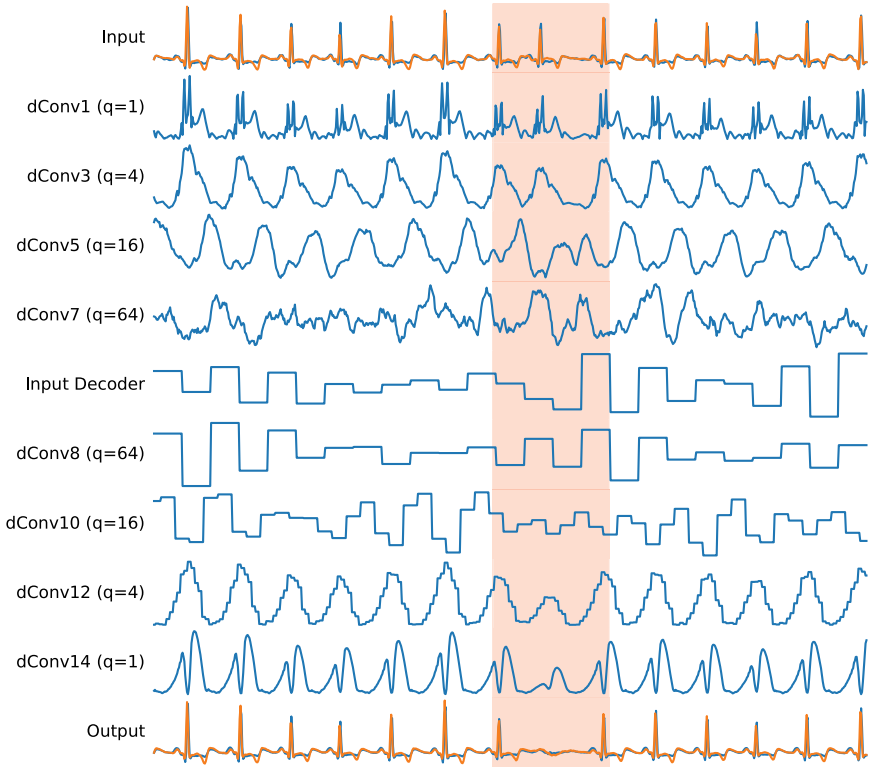

Fig. 6. Activations inside the trained final TCN-AE model for several layers of the network. For each dilated convolutional layer, we plot the channel (signal) with the largest mean absolute activation. If we compute and plot the mean over all channels, we get structurally relatively similar results. 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.

Wilcoxon test signalizes the significance of the result. In Table 4, we summarize the different variants of the TCN-AE algorithm.

Overall, all the individual enhancements significantly improve the performance of TCN-AE on the ECG benchmark. As summarized in Table 4, 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 Fig. 7 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.3. 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 in Table 4 and Fig. 7, 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

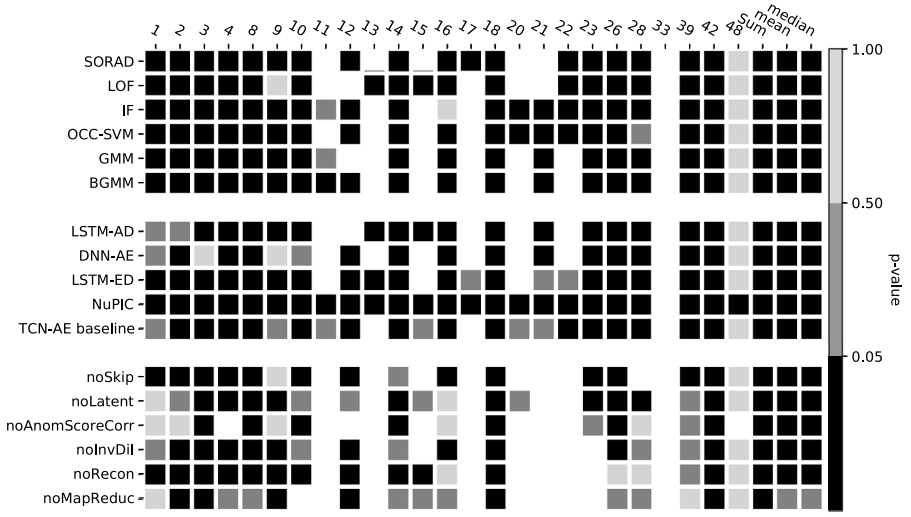

Fig. 7. Heatmap, showing the p-values for the comparison of our final TCN-AE model to various algorithms and other variants of TCN-AE for all 25 ECG signals of our benchmark. We use a one-sided Wilcoxon signed-rank test for the F_1-scores of ten runs each. The test has the null hypothesis that the median 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 ten rows represent anomaly detection algorithms from the literature, while the remaining rows are for different variants of the TCN-AE algorithm. The final TCN-AE model is the model with all extensions (that are switched off one at at time in the last six rows) switched on. 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 F_1-scores of TCN-AE and the compared algorithm are exactly the same. The exact F_1-scores are given in Tables 8, A.2, and A.3.

Table 4
Impact of the individual TCN-AE components for the ECG 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	–

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.

Note that in our setup, it is also principally possible to add all feature maps which are passed through the skip connections (since all feature maps have the same shape) instead of concatenating them. However, we have found that adding the feature maps ($F_1 = 0.908 \pm 0.005$) leads to slightly worse results than concatenation ($F_1 = 0.926 \pm 0.003$). One advantage of using concatenation might be that the network can learn to ignore certain channels in the following layer if they are irrelevant to the learning task.

We also experimented with different variants of a dense TCN-AE similar to a DenseNet [64] (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 [61] or DenseNets [64]. Some of the observed improvements might be due to a smoother curvature of the loss landscape [81], alleviation of 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[4]) 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 $F_1 \approx 0.90$. 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 Fig. 7, 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.

[4] 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.)

Detecting anomalies in hidden representations of time series. As discussed in Section 2.6.3, 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 [65], 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. 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 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 12 in Algorithm 1). Although we observe a drop in the F_1-score for 3 ECG signals (#1, #16, #48) in Fig. 7 (Table A.3), the overall results suggest that this approach generally improves the results (Table 4). 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 (Fig. 7 & Table A.3).

Similarly, in our next experiment, we tried also to utilize the decoder's hidden representations. However, this did not further affect 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. Table 4 shows 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 2.6.4) 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 Fig. 7 (Table A.3).

Anomaly score baseline correction. Also, the correction of the anomaly score baseline using a Butterworth filter, as described in Section 2.6.5, has a notable impact on the final results. Although there is only a significant improvement for 7 out of 25 time series (Fig. 7), the overall F_1 drops by around 4% if we turn off the baseline correction of the anomaly score, as reported in Table 4. Instead of using a filter, we also tested the more advanced baseline correction algorithm by Zhang et al. [82], and obtained results which did not significantly differ (F_1-score of 0.920 ± 0.003).

4.2. Comparison to other algorithms

In Table 5, 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 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. From the non-DL algorithms, LOF obtains the best results, outperforming the DL algorithms NuPIC and LSTM-ED. But overall, the DL approaches appear to have a slight advantage on the ECG benchmark.

4.2.1. 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 Fig. 8, 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 ten 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.

4.2.2. Performance for individual anomaly types

The ECG benchmark contains nine different anomaly types, as summarized in Table 1. 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 6 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 four out of nine anomaly types, it is among the top five algorithms in each case. Furthermore, it produces significantly fewer FPs in the EAC setting (if we permitted TCN-AE also to have more FPs, similar to the other algorithms, it would detect even more anomalies). Interestingly, TCN-AE performs slightly worse for the fusion beats "f" and "F" than DNN-AE. Fusion beats usually differ from normal heartbeats not by their timing but by their shape. Therefore, temporal dependencies play only a minor role in the detection of fusion beats. It seems that DNN-AE (which is similar to TCN-AE in some aspects but has a significantly smaller temporal receptive field) can detect these shape variations slightly better than TCN-AE.

Overall, 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.

4.2.3. Computing anomaly score with little labeled data

While we used EAC to determine all anomaly thresholds for the results presented in Table 5, 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 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.1. We observe that the F_1-score for all algorithms deteriorates compared to Table 5, 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 the runtimes of all algorithms and the number of trainable parameters. All TCN-AE variants are

Table 5
Summary for the ECG data (mean $\pm$ σ_{mean} of 10 runs, except for the deterministic SORAD, LOF, and 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. The first six rows represent the non-DL algorithms.

Algorithm	TP	FN	FP	Prec	Rec	F1	p
OCC-SVM	431.0 ± 0.0	290.0 ± 0.0	318.0 ± 0.0	0.575 ± 0.000	0.598 ± 0.000	0.586 ± 0.000	2.531e−3
SORAD	519.0 ± 0.0	202.0 ± 0.0	206.0 ± 0.0	0.716 ± 0.000	0.72 ± 0.00	0.718 ± 0.000	2.531e−3
IF	529.8 ± 1.6	191.2 ± 1.6	194.5 ± 1.9	0.731 ± 0.003	0.735 ± 0.002	0.733 ± 0.002	2.531e−3
GMM	534.6 ± 3.7	186.4 ± 3.7	196.0 ± 3.7	0.732 ± 0.005	0.741 ± 0.005	0.737 ± 0.005	2.531e−3
BGMM	554.3 ± 2.0	166.7 ± 2.0	175.6 ± 2.2	0.759 ± 0.003	0.769 ± 0.003	0.764 ± 0.003	2.531e−3
LOF	566.0 ± 0.0	155.0 ± 0.0	161.0 ± 0.0	0.779 ± 0.000	0.785 ± 0.000	0.782 ± 0.000	2.531e−3
NuPIC	224.0 ± 0.0	497.0 ± 0.0	497.0 ± 0.0	0.311 ± 0.000	0.311 ± 0.000	0.311 ± 0.000	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	–

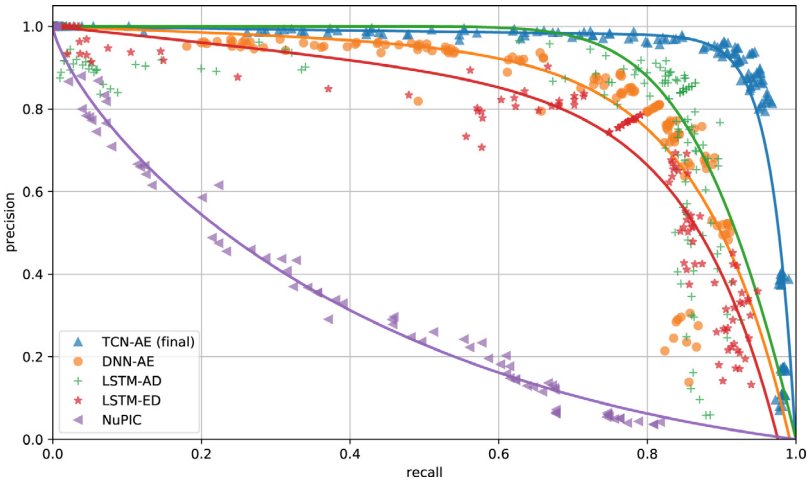

Fig. 8. Precision–recall curves for the individual DL algorithms on the ECG data. Shown are the fits through around 100 points which were generated by evaluating precision & recall for different thresholds. For each algorithm, except NuPIC, 10 runs were performed.

Table 6
Number of detected anomalies for the individual algorithms, broken down by anomaly type (in total nine types, see Table 1). 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. The DL approaches (last six rows) are separated from the non-DL approaches with a line.

Algorithm	a	A	e	f	F	J	V	x	|	FN	FP
OCC-SVM	9.0	74.0	5.0	4.0	16.0	3.0	305.0	5.0	10.0	290.0	318.0
SORAD	9.0	108.0	5.0	17.0	17.0	**4.0**	337.0	6.0	11.0	207.0	210.0
GMM	9.1	178.0	6.3	8.3	14.7	2.4	301.0	5.4	9.4	186.4	196.0
IF	9.8	173.5	**7.9**	1.4	15.6	3.0	302.5	5.0	11.1	191.2	194.5
BGMM	9.4	189.7	6.1	10.5	16.4	2.3	303.4	5.7	10.8	166.7	175.6
LOF	8.0	211.0	7.0	18.0	16.0	3.0	277.0	12.0	**14.0**	155.0	161.0
NuPIC	2.0	32.0	1.0	1.0	5.0	2.0	172.0	3.0	6.0	497.0	497.0
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

considerably faster than the remaining 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 750 s per time series) since no GPU acceleration is available.

4.3. Discussion

Only for the last ECG recording (#48), our final version of TCN-AE performs significantly worse than most 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 for this time series. Exemplarily, we illustrate the cause of this FP in Fig. 9: 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. 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. 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. Nevertheless, some further

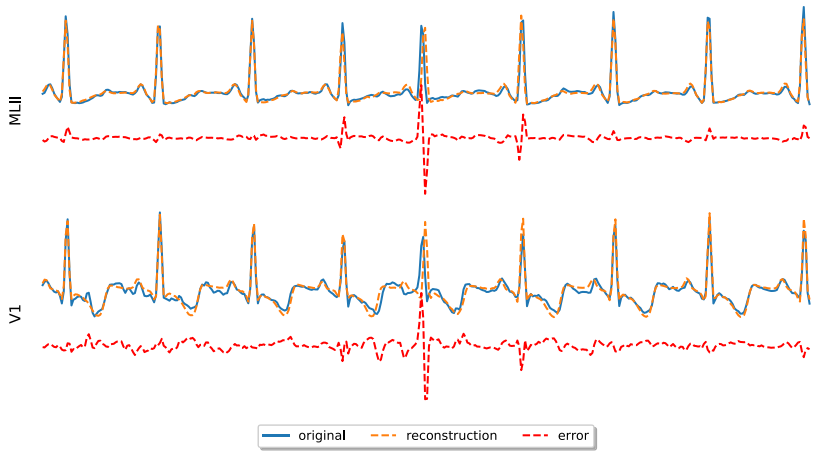

Fig. 9. Similar to Fig. 5, but now showing an excerpt of ECG signal #48. For both signals, one can observe a slight shift to the right in the reconstruction of the 5th R-peak, which results in an unusually large error and, ultimately, the TCN-AE algorithm falsely detects an anomaly (false positive) at this position. Taking the (ECG) signal's variability into account, in order to prevent situations like these, is addressed in our ongoing work.

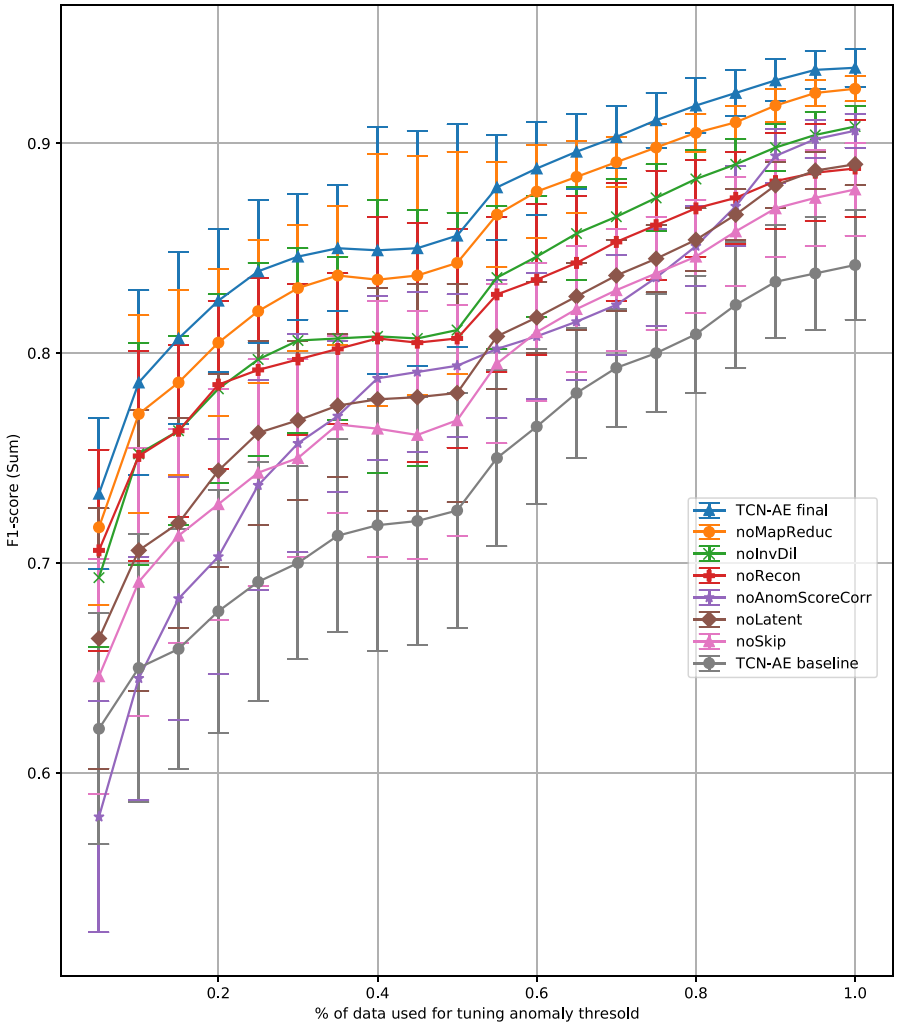

Fig. A.10. Results when tuning the algorithmic variants of TCN-AE on different subsets of the ECG data (mean and standard deviation of 10 runs).

Table 7
Computation times: average (per time series) and total (10 runs, all time series) for all algorithms evaluated in this paper. *TCN-AE final* is faster than *baseline* since it effectively has less layers. On the left side, the times for the non-DL models are shown.

Algorithm	Mean (s)	Total (h)	#Par/10^3	Algorithm	Mean (s)	Total (h)	#Par/10^3
SORAD	**12.5 ± 0.1**	**0.9**	**0.257**	**final**	**47.6 ± 0.4**	**3.3**	**123.8**
OCC-SVM	26.2 ± 1.8	1.8	–	baseline	97.0 ± 1.1	6.7	208.4
BGMM	127.1 ± 3.7	8.8	–	DNN-AE	237.5 ± 10.3	16.5	242.1
GMM	150.4 ± 9.4	10.4	–	LSTM-ED	626.5 ± 30.8	43.5	62.5
IF	161.8 ± 9.7	11.2	–	NuPIC	752.0 ± 23.4	52.4	–
LOF	173.7 ± 43.1	12.1	–	LSTM-AD	1615.0 ± 127.6	112.2	465.6

Table 8
F_1-scores (mean ± σ_{mean}) of TCN-AE and the other DL algorithms on all 25 time series of the ECG 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 highlight the corresponding field. The results for the non-DL algorithms are shown in Table A.2.

	NuPIC		LSTM-ED		DNN-AE		LSTM-AD		TCN-AE
	F1	p	F1	p	F1	p	F1	p	F1
1	0.03 ± 0.00	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 ± 0.0	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 ± 0.000	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 ± 0.0	0.001	0.5 ± 0.0	0.001	0.0 ± 0.0	0.001	0.250 ± 0.083	0.002	**1.0 ± 0.0**
8	0.088 ± 0.000	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 ± 0.000	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 ± 0.000	0.001	0.811 ± 0.016	0.002	0.975 ± 0.005	0.096	0.907 ± 0.016	0.002	**0.987 ± 0.000**
11	0.0 ± 0.0	0.001	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
12	0.0 ± 0.0	0.001	0.5 ± 0.0	0.001	0.650 ± 0.077	0.004	**1.0 ± 0.0**	–	**1.0 ± 0.0**
13	0.0 ± 0.0	0.001	0.950 ± 0.026	0.042	**1.0 ± 0.0**	–	0.833 ± 0.000	0.001	**1.0 ± 0.0**
14	0.524 ± 0.000	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 ± 0.0	0.001	**0.909 ± 0.000**	–	**0.909 ± 0.000**	–	0.727 ± 0.000	0.001	**0.909 ± 0.000**
16	0.33 ± 0.00	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 ± 0.0	0.001	0.9 ± 0.1	0.159	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
18	0.24 ± 0.00	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 ± 0.0	0.001	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
21	0.0 ± 0.0	0.001	0.9 ± 0.1	0.159	0.1 ± 0.1	0.001	0.500 ± 0.167	0.013	**1.0 ± 0.0**
22	0.0 ± 0.0	0.001	0.900 ± 0.071	0.09	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
23	0.19 ± 0.00	0.001	0.814 ± 0.025	0.004	0.910 ± 0.009	0.003	0.919 ± 0.010	0.01	**0.952 ± 0.000**
26	0.169 ± 0.000	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 ± 0.000	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 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**
39	0.25 ± 0.00	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 ± 0.000	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 ± 0.0	0.002	**1.0 ± 0.0**	0.987	**1.0 ± 0.0**	0.987	**1.0 ± 0.0**	0.987	0.875 ± 0.042
Σ	0.311 ± 0.000	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 ± 0.000	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 ± 0.000	0.002	0.837 ± 0.009	0.003	0.883 ± 0.003	0.003	0.882 ± 0.006	0.003	**0.953 ± 0.002**

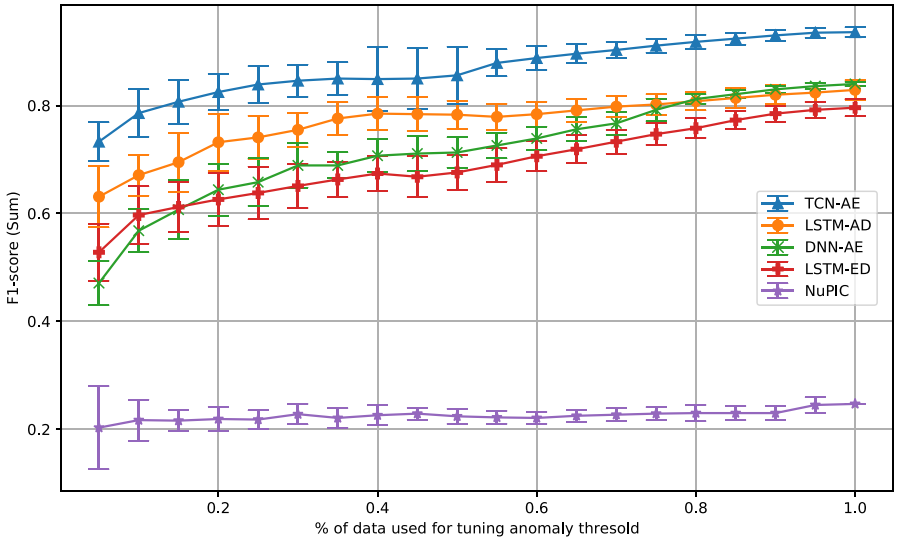

Fig. A.11. Results when tuning various algorithms on different subsets of the ECG data (mean and standard deviation of 10 runs).

Table A.1
F_1-scores (mean $\pm$ σ_{mean}) of TCN-AE and the other algorithms. Same as Table 8, except that we permit the algorithms to determine an anomaly threshold from only 10% of the anomaly labels, as described in Section 3.4.

	NuPIC		LSTM-ED		DNN-AE		LSTM-AD		TCN-AE
	F1	p	F1	p	F1	p	F1	p	F1
1	0.077 ± 0.012	1.94e−18	0.475 ± 0.012	2.07e−18	0.809 ± 0.016	0.00108	0.801 ± 0.017	0.000857	**0.831 ± 0.014**
2	0.164 ± 0.038	8.63e−15	0.254 ± 0.014	2.37e−10	0.276 ± 0.012	4.85e−10	**0.561 ± 0.014**	1.0	0.467 ± 0.024
3	0.082 ± 0.016	1.95e−18	0.77 ± 0.01	4.6e−09	0.729 ± 0.017	9.57e−08	0.345 ± 0.014	1.95e−18	**0.845 ± 0.011**
4	0.106 ± 0.035	6.3e−06	0.112 ± 0.018	1.94e−18	0.144 ± 0.018	1.29e−17	**0.420 ± 0.015**	1.0	0.278 ± 0.031
8	0.294 ± 0.041	1.95e−18	0.861 ± 0.009	8.61e−06	0.538 ± 0.013	7.07e−18	0.789 ± 0.009	1.22e−11	**0.903 ± 0.010**
9	0.108 ± 0.027	2.01e−18	0.471 ± 0.009	6.23e−13	0.544 ± 0.013	0.0103	0.397 ± 0.021	6.39e−11	**0.573 ± 0.011**
10	0.509 ± 0.064	3.79e−15	0.691 ± 0.012	5.3e−06	0.644 ± 0.024	9.3e−13	**0.796 ± 0.011**	0.955	0.785 ± 0.014
11	0.043 ± 0.018	1.17e−18	0.515 ± 0.034	0.0807	0.285 ± 0.028	7.4e−15	**0.640 ± 0.039**	0.967	0.556 ± 0.023
12	0.09 ± 0.06	1.8e−18	0.351 ± 0.019	1.27e−15	0.343 ± 0.027	1.25e−14	0.578 ± 0.028	0.381	**0.593 ± 0.022**
13	0.010 ± 0.007	1.33e−18	0.718 ± 0.022	4.99e−08	0.588 ± 0.028	2.25e−15	0.624 ± 0.029	9.26e−14	**0.817 ± 0.022**
14	0.372 ± 0.080	2e−18	0.793 ± 0.015	4.31e−08	0.614 ± 0.011	9.52e−18	0.684 ± 0.019	1.42e−15	**0.869 ± 0.010**
15	0.006 ± 0.003	1.78e−18	0.574 ± 0.018	4.46e−10	0.702 ± 0.023	0.791	**0.781 ± 0.008**	1.0	0.692 ± 0.015
16	0.264 ± 0.019	1.93e−18	0.816 ± 0.010	1.1e−06	0.612 ± 0.008	3.78e−17	0.873 ± 0.010	0.00805	**0.883 ± 0.013**
17	0.001 ± 0.001	1.97e−19	0.097 ± 0.029	4.66e−16	0.10 ± 0.03	9.66e−16	0.10 ± 0.03	9.66e−16	**0.791 ± 0.029**
18	0.251 ± 0.004	1.95e−18	0.482 ± 0.007	1.95e−18	0.660 ± 0.017	3.06e−18	0.837 ± 0.006	4.07e−14	**0.917 ± 0.005**
20	0.007 ± 0.007	3.74e−18	0.368 ± 0.032	0.000163	0.288 ± 0.032	1.1e−07	0.329 ± 0.036	0.00217	**0.436 ± 0.032**
21	0.004 ± 0.004	1.47e−18	0.361 ± 0.033	2.1e−11	0.230 ± 0.021	2.88e−12	0.391 ± 0.021	0.000737	**0.558 ± 0.038**
22	0.337 ± 0.067	1.75e−17	0.500 ± 0.024	1.1e−07	0.538 ± 0.026	1.81e−05	0.621 ± 0.022	0.212	**0.629 ± 0.018**
23	0.281 ± 0.032	1.94e−18	0.504 ± 0.022	2.74e−17	0.769 ± 0.017	0.000898	0.740 ± 0.016	2.52e−05	**0.830 ± 0.012**
26	0.135 ± 0.023	1.95e−18	0.561 ± 0.024	7.71e−11	0.460 ± 0.019	5.74e−18	0.381 ± 0.013	5.01e−14	**0.665 ± 0.015**
28	0.426 ± 0.038	1.94e−18	0.704 ± 0.019	2.45e−06	0.723 ± 0.020	0.00418	0.775 ± 0.012	0.00466	**0.801 ± 0.010**
33	0.0 ± 0.0	1.34e−05	**0.092 ± 0.014**	1.0	0.062 ± 0.007	0.999	0.002 ± 0.001	1.34e−05	0.033 ± 0.007
39	0.150 ± 0.019	1.95e−18	0.719 ± 0.010	1.77e−17	0.781 ± 0.019	2.93e−06	0.823 ± 0.011	6.18e−08	**0.898 ± 0.007**
42	0.468 ± 0.021	1.95e−18	0.738 ± 0.014	3.59e−13	0.736 ± 0.023	6.34e−07	0.714 ± 0.012	6.9e−16	**0.844 ± 0.008**
48	0.008 ± 0.004	1.8e−18	0.544 ± 0.024	1.39e−08	0.498 ± 0.032	2.97e−05	**0.703 ± 0.024**	0.925	0.667 ± 0.022
Σ	0.217 ± 0.012	1.95e−18	0.597 ± 0.005	1.95e−18	0.568 ± 0.004	1.95e−18	0.671 ± 0.004	9.52e−18	**0.786 ± 0.004**
Mean	0.168 ± 0.009	1.95e−18	0.523 ± 0.004	1.95e−18	0.507 ± 0.004	1.95e−18	0.588 ± 0.003	3.78e−18	**0.686 ± 0.005**
Median	0.120 ± 0.015	1.95e−18	0.544 ± 0.006	1.95e−18	0.531 ± 0.008	1.95e−18	0.650 ± 0.007	1.22e−17	**0.747 ± 0.007**

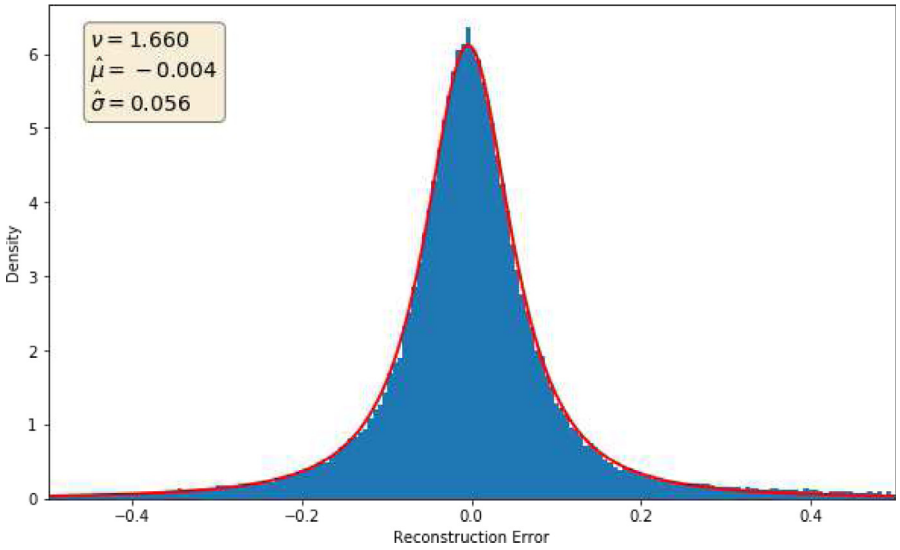

Fig. A.12. Histogram of the reconstruction error (in blue) for one dimension of the error matrix $\mathbf{E}'$. We found that the reconstruction errors are bell-shaped (elliptic in higher dimensions). Although the reconstruction errors are not Gaussian, they closely follow a t-distribution with a mean $\hat{\mu}$, a standard deviation $\hat{\sigma}$, and ν degrees of freedom, as indicated by the estimated distribution (red line).

investigation is required to understand why all algorithms have difficulties with time series #33.

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 [83], a F_1-score of $F_1 = 0.93$ is obtained, if we consider only the anomaly classes also used in this work. In [84], 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 4).

A particular limitation of our algorithm is that it cannot be used for time series that contain many anomalies. We have exemplarily demonstrated this issue for an ECG time series (#30), which contains many anomalous events (990 V-events, 370 F-events, with only 1500 normal heartbeats). As expected, TCN-AE breaks down in such a setting: The obtained F_1-score is $F_1 =$

Table A.2
Similar to Table 8. F_1-scores (mean $\pm \sigma_{\text{mean}}$) of TCN-AE and the other non-DL algorithms on all 25 time series of the ECG benchmark (highest values in boldface).

Patient No	OCC-SVM		SORAD		IF		GMM		BGMM		LOF		TCN-AE
	F1	p	F1	p	F1	p	F1	p	F1	p	F1	p	F1
1	0.058 ± 0.000	0.00224	0.348 ± 0.000	0.00224	0.480 ± 0.013	0.0025	0.902 ± 0.008	0.0328	0.875 ± 0.012	0.00609	0.882 ± 0.000	0.00224	**0.919 ± 0.006**
2	0.333 ± 0.000	0.00248	0.333 ± 0.000	0.00248	0.267 ± 0.045	0.0025	0.217 ± 0.061	0.00348	0.350 ± 0.046	0.00303	0.5 ± 0.0	0.0294	**0.733 ± 0.083**
3	0.314 ± 0.000	0.0022	0.769 ± 0.000	0.0022	0.154 ± 0.015	0.00246	0.475 ± 0.016	0.00252	0.564 ± 0.016	0.00246	0.846 ± 0.000	0.0022	**0.933 ± 0.008**
4	0.0 ± 0.0	0.000783	0.5 ± 0.0	0.000783	0.300 ± 0.082	0.00193	0.5 ± 0.0	0.000783	0.5 ± 0.0	0.000783	0.0 ± 0.0	0.000783	**1.0 ± 0.0**
8	0.574 ± 0.000	0.00193	0.765 ± 0.000	0.00193	0.70 ± 0.06	0.00225	0.902 ± 0.005	0.00245	0.885 ± 0.004	0.00201	0.922 ± 0.000	0.00193	**0.981 ± 0.003**
9	0.436 ± 0.000	0.00245	0.393 ± 0.000	0.00245	0.389 ± 0.006	0.00247	0.312 ± 0.006	0.0025	0.359 ± 0.013	0.0025	**0.75 ± 0.00**	0.998	0.674 ± 0.011
10	0.658 ± 0.000	0.000783	0.9 ± 0.0	0.000783	0.750 ± 0.017	0.0022	0.710 ± 0.026	0.0024	0.827 ± 0.011	0.00245	0.9 ± 0.0	0.000783	**0.987 ± 0.000**
11	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	0.9 ± 0.1	0.159	0.9 ± 0.1	0.159	0.200 ± 0.133	0.00234	**1.0 ± 0.0**	–	**1.0 ± 0.0**
12	0.4 ± 0.0	0.000783	0.5 ± 0.0	0.000783	0.05 ± 0.05	0.00114	**1.0 ± 0.0**	–	0.500 ± 0.105	0.00415	**1.0 ± 0.0**	–	**1.0 ± 0.0**
13	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	0.5 ± 0.0	0.000783	**1.0 ± 0.0**
14	0.88 ± 0.00	0.00232	0.92 ± 0.00	0.00232	0.878 ± 0.003	0.00246	0.824 ± 0.012	0.00253	0.831 ± 0.010	0.00252	0.931 ± 0.000	0.00516	**0.945 ± 0.004**
15	**0.909 ± 0.000**	–	**0.909 ± 0.000**	–	**0.909 ± 0.000**	–	**0.909 ± 0.000**	–	**0.909 ± 0.000**	–	0.769 ± 0.000	0.000783	**0.909 ± 0.000**
16	0.886 ± 0.000	0.00114	0.933 ± 0.000	0.00114	**0.990 ± 0.001**	0.999	0.832 ± 0.011	0.00252	0.833 ± 0.012	0.0025	0.402 ± 0.000	0.00114	0.953 ± 0.001
17	**1.0 ± 0.0**	–	0.0 ± 0.0	0.000783	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
18	0.433 ± 0.000	0.00234	0.507 ± 0.000	0.00234	0.760 ± 0.009	0.00252	0.571 ± 0.027	0.00253	0.686 ± 0.011	0.00253	0.872 ± 0.000	0.00234	**0.962 ± 0.003**
20	0.0 ± 0.0	0.000783	**1.0 ± 0.0**	–	0.0 ± 0.0	0.000783	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
21	0.0 ± 0.0	0.000783	**1.0 ± 0.0**	–	0.500 ± 0.167	0.0127	0.400 ± 0.163	0.00715	0.700 ± 0.153	0.0416	**1.0 ± 0.0**	–	**1.0 ± 0.0**
22	0.667 ± 0.000	0.000783	0.0 ± 0.0	0.000783	0.633 ± 0.145	0.0192	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	0.667 ± 0.000	0.000783	**1.0 ± 0.0**
23	0.829 ± 0.000	0.000783	0.762 ± 0.000	0.000783	0.834 ± 0.009	0.00225	0.762 ± 0.014	0.00224	0.814 ± 0.015	0.00232	0.905 ± 0.000	0.000783	**0.952 ± 0.000**
26	0.554 ± 0.000	0.00225	0.634 ± 0.000	0.00225	0.258 ± 0.015	0.00252	0.612 ± 0.023	0.00252	0.631 ± 0.008	0.00247	0.593 ± 0.000	0.00225	**0.806 ± 0.011**
28	**0.912 ± 0.000**	0.5	0.765 ± 0.000	0.00149	0.884 ± 0.002	0.00285	0.598 ± 0.006	0.00243	0.607 ± 0.019	0.00246	0.794 ± 0.000	0.00149	**0.912 ± 0.004**
33	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**
39	0.198 ± 0.000	0.00193	0.76 ± 0.00	0.00193	0.80 ± 0.08	0.00249	0.911 ± 0.006	0.00242	0.910 ± 0.006	0.00249	0.909 ± 0.000	0.00193	**0.952 ± 0.003**
42	0.72 ± 0.00	0.0024	0.72 ± 0.00	0.0024	0.830 ± 0.003	0.00245	0.801 ± 0.010	0.0025	0.790 ± 0.013	0.00252	0.857 ± 0.000	0.0024	**0.909 ± 0.006**
48	**1.0 ± 0.0**	0.987	**1.0 ± 0.0**	0.987	**1.0 ± 0.0**	0.987	**1.0 ± 0.0**	0.987	**1.0 ± 0.0**	0.987	**1.0 ± 0.0**	0.987	0.875 ± 0.042
Σ	0.586 ± 0.000	0.00253	0.718 ± 0.000	0.00253	0.733 ± 0.002	0.00253	0.737 ± 0.005	0.00253	0.764 ± 0.003	0.00253	0.782 ± 0.000	0.00253	**0.926 ± 0.003**
Mean	0.55 ± 0.00	0.00253	0.657 ± 0.000	0.00253	0.611 ± 0.007	0.00253	0.725 ± 0.009	0.00253	0.711 ± 0.008	0.00253	0.76 ± 0.00	0.00253	**0.896 ± 0.006**
Median	0.564 ± 0.000	0.00246	0.761 ± 0.000	0.00246	0.752 ± 0.008	0.00253	0.82 ± 0.01	0.00253	0.804 ± 0.006	0.00253	0.865 ± 0.000	0.00246	**0.953 ± 0.002**

Table A.3
F_1-scores (mean $\pm$ σ_{mean}) of the individual TCN-AE variants on all 25 time series of the ECG benchmark. 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 highlight the corresponding field.

Patient No	Baseline		noSkip		noLatent		noInvDil		noRecon		noMapReduc		noAnomScoreCorr		Final
	F1	p	F1	p	F1	p	F1	p	F1	p	F1	p	F1	p	F1
1	0.768 ± 0.079	0.069	0.844 ± 0.039	0.018	0.940 ± 0.001	0.995	0.913 ± 0.005	0.051	0.846 ± 0.014	0.002	0.937 ± 0.002	0.978	**0.941 ± 0.000**	0.995	0.919 ± 0.006
2	0.617 ± 0.043	0.043	0.667 ± 0.000	0.042	0.667 ± 0.078	0.168	0.600 ± 0.067	0.035	0.662 ± 0.005	0.046	0.600 ± 0.067	0.035	**0.800 ± 0.022**	0.841	0.733 ± 0.083
3	0.877 ± 0.008	0.003	0.883 ± 0.009	0.002	0.831 ± 0.014	0.002	0.862 ± 0.006	0.002	0.882 ± 0.013	0.002	0.909 ± 0.011	0.041	0.846 ± 0.006	0.002	**0.933 ± 0.008**
4	0.300 ± 0.082	0.002	0.750 ± 0.083	0.013	0.5 ± 0.0	0.001	0.5 ± 0.0	0.001	0.55 ± 0.05	0.001	0.900 ± 0.067	0.079	**1.0 ± 0.0**	–	**1.0 ± 0.0**
8	0.811 ± 0.016	0.003	0.837 ± 0.015	0.002	0.811 ± 0.012	0.003	0.958 ± 0.004	0.003	0.970 ± 0.004	0.006	0.977 ± 0.002	0.111	0.839 ± 0.015	0.003	**0.981 ± 0.003**
9	0.613 ± 0.045	0.077	**0.675 ± 0.009**	0.556	0.643 ± 0.007	0.007	0.643 ± 0.011	0.025	0.602 ± 0.011	0.004	0.618 ± 0.012	0.007	0.673 ± 0.007	0.556	0.674 ± 0.011
10	0.948 ± 0.016	0.014	0.962 ± 0.007	0.008	0.980 ± 0.005	0.09	0.982 ± 0.003	0.079	0.979 ± 0.004	0.029	**0.987 ± 0.000**	–	0.975 ± 0.004	0.023	**0.987 ± 0.000**
11	0.9 ± 0.1	0.159	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
12	0.750 ± 0.083	0.013	0.600 ± 0.067	0.002	0.95 ± 0.05	0.159	0.650 ± 0.077	0.004	0.600 ± 0.067	0.002	0.750 ± 0.112	0.029	**1.0 ± 0.0**	–	**1.0 ± 0.0**
13	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
14	0.903 ± 0.009	0.009	0.938 ± 0.005	0.088	0.938 ± 0.001	0.039	0.938 ± 0.006	0.124	0.922 ± 0.003	0.003	0.939 ± 0.005	0.2	0.924 ± 0.005	0.005	**0.945 ± 0.004**
15	0.894 ± 0.010	0.079	**0.909 ± 0.000**	–	0.894 ± 0.010	0.079	**0.909 ± 0.000**	–	0.841 ± 0.008	0.001	0.894 ± 0.010	0.079	**0.909 ± 0.000**	–	**0.909 ± 0.000**
16	0.905 ± 0.009	0.002	0.932 ± 0.002	0.002	0.956 ± 0.005	0.764	0.949 ± 0.002	0.029	0.962 ± 0.001	0.994	0.950 ± 0.001	0.051	**0.982 ± 0.002**	0.998	0.953 ± 0.001
17	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
18	0.777 ± 0.031	0.003	0.809 ± 0.011	0.003	0.865 ± 0.017	0.002	0.942 ± 0.004	0.002	0.946 ± 0.003	0.002	0.943 ± 0.005	0.006	0.899 ± 0.003	0.003	**0.962 ± 0.003**
20	0.800 ± 0.133	0.079	**1.0 ± 0.0**	–	0.967 ± 0.033	0.159	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
21	0.9 ± 0.1	0.159	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
22	0.867 ± 0.054	0.023	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**	–	**1.0 ± 0.0**
23	0.857 ± 0.014	0.002	0.931 ± 0.008	0.017	0.938 ± 0.006	0.032	**0.952 ± 0.000**	–	**0.952 ± 0.000**	–	**0.952 ± 0.000**	–	0.946 ± 0.007	0.159	**0.952 ± 0.000**
26	0.638 ± 0.011	0.002	0.671 ± 0.012	0.002	0.646 ± 0.015	0.002	0.694 ± 0.006	0.002	**0.817 ± 0.013**	0.764	0.768 ± 0.015	0.061	0.635 ± 0.016	0.003	0.806 ± 0.011
28	0.874 ± 0.008	0.008	0.912 ± 0.004	–	0.874 ± 0.006	0.005	0.905 ± 0.004	0.179	**0.926 ± 0.005**	0.987	0.906 ± 0.006	0.207	0.924 ± 0.005	0.977	0.912 ± 0.004
33	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**	–	**0.0 ± 0.0**
39	0.859 ± 0.012	0.003	0.897 ± 0.010	0.002	0.945 ± 0.003	0.029	0.952 ± 0.003	0.5	0.950 ± 0.003	0.159	**0.955 ± 0.002**	0.921	0.95 ± 0.01	0.405	0.952 ± 0.003
42	0.825 ± 0.009	0.002	0.853 ± 0.006	0.002	0.831 ± 0.009	0.002	0.882 ± 0.002	0.002	0.868 ± 0.005	0.002	0.885 ± 0.003	0.004	0.849 ± 0.011	0.003	**0.909 ± 0.006**
48	0.950 ± 0.033	0.958	**1.0 ± 0.0**	0.987	0.889 ± 0.039	0.841	**1.0 ± 0.0**	0.987	**1.0 ± 0.0**	0.987	0.975 ± 0.025	0.949	0.875 ± 0.042	–	0.875 ± 0.042
Σ	0.826 ± 0.007	0.003	0.861 ± 0.008	0.003	0.871 ± 0.007	0.003	0.904 ± 0.003	0.003	0.907 ± 0.004	0.003	0.913 ± 0.002	0.008	0.890 ± 0.005	0.003	**0.926 ± 0.003**
Mean	0.785 ± 0.009	0.003	0.843 ± 0.009	0.003	0.843 ± 0.007	0.003	0.849 ± 0.005	0.003	0.851 ± 0.006	0.003	0.874 ± 0.008	0.07	0.879 ± 0.004	0.003	**0.896 ± 0.006**
Median	0.868 ± 0.006	0.003	0.911 ± 0.007	0.003	0.919 ± 0.007	0.003	0.939 ± 0.004	0.003	0.939 ± 0.003	0.003	0.950 ± 0.002	0.069	0.934 ± 0.005	0.003	**0.953 ± 0.002**

Table B.4
Sensitivity of TCN-AE (final) towards the max. dilation rate q_{max}. 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. The default value is used in the main part of the paper. For $q_{max} = 1024$, only a single run was performed, due to server issues.

q_{max}	TP	FN	FP	Prec	Rec	F1
2	493.7 ± 4.0	227.3 ± 4.0	226.9 ± 3.7	0.685 ± 0.005	0.685 ± 0.006	0.685 ± 0.005
4	644.5 ± 1.6	76.5 ± 1.6	82.5 ± 1.6	0.887 ± 0.002	0.894 ± 0.002	0.890 ± 0.002
8	647.3 ± 1.5	73.7 ± 1.5	78.8 ± 1.5	0.891 ± 0.002	0.898 ± 0.002	0.895 ± 0.002
16	651.1 ± 1.1	69.9 ± 1.1	74.8 ± 1.2	0.897 ± 0.002	0.903 ± 0.002	0.90 ± 0.02
32	656.7 ± 1.7	64.3 ± 1.7	68.1 ± 1.6	0.906 ± 0.002	0.911 ± 0.002	0.908 ± 0.002
64 (default)	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003
128	673.8 ± 0.7	47.2 ± 0.7	52.4 ± 0.6	0.928 ± 0.001	0.935 ± 0.001	0.931 ± 0.001
256	668.8 ± 1.0	52.2 ± 1.0	56.4 ± 1.0	0.922 ± 0.001	0.928 ± 0.001	0.925 ± 0.001
512	666.5 ± 1.0	54.5 ± 1.0	59.4 ± 1.0	0.918 ± 0.001	0.924 ± 0.002	0.921 ± 0.001
1024	652.0 ± 0.0	69.0 ± 0.0	74.0 ± 0.0	0.898 ± 0.000	0.904 ± 0.000	0.901 ± 0.000

Table B.5
Similar to Table B.4. Sensitivity of TCN-AE (final) towards the number of filters n_{filters} used in the dilated convolutional layers.

n_{filters}	TP	FN	FP	Prec	Rec	F1
2	507.8 ± 10.5	213.2 ± 10.5	202.7 ± 6.1	0.714 ± 0.009	0.704 ± 0.015	0.709 ± 0.012
4	527.3 ± 9.5	193.7 ± 9.5	187.7 ± 4.5	0.737 ± 0.008	0.731 ± 0.013	0.734 ± 0.011
8	655.4 ± 1.8	65.6 ± 1.8	70.3 ± 1.7	0.903 ± 0.003	0.909 ± 0.003	0.906 ± 0.003
16	661.6 ± 1.2	59.4 ± 1.2	65.5 ± 1.2	0.910 ± 0.002	0.918 ± 0.002	0.914 ± 0.002
32	663.3 ± 1.5	57.7 ± 1.5	63.0 ± 1.5	0.913 ± 0.002	0.920 ± 0.002	0.917 ± 0.002
64 (default)	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003
128	659.8 ± 1.4	61.2 ± 1.4	66.4 ± 2.1	0.909 ± 0.003	0.915 ± 0.002	0.912 ± 0.003
256	653.6 ± 2.8	67.4 ± 2.8	68.9 ± 2.1	0.905 ± 0.003	0.907 ± 0.004	0.906 ± 0.002

Table B.6
Similar to Table B.4. Sensitivity of TCN-AE (final) towards the kernel size k used in the dilated convolutional layers.

k	TP	FN	FP	Prec	Rec	F1
2	640.2 ± 1.8	80.8 ± 1.8	85.8 ± 1.7	0.882 ± 0.003	0.888 ± 0.003	0.885 ± 0.003
4	651.8 ± 1.6	69.2 ± 1.6	73.8 ± 1.3	0.898 ± 0.002	0.904 ± 0.002	0.901 ± 0.002
8 (default)	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003
16	665.6 ± 1.2	55.4 ± 1.2	60.8 ± 1.2	0.916 ± 0.002	0.923 ± 0.002	0.920 ± 0.002
32	635.7 ± 1.6	85.3 ± 1.6	93.4 ± 1.5	0.872 ± 0.002	0.882 ± 0.002	0.877 ± 0.002

Table B.7
Similar to Table B.4. Sensitivity of TCN-AE (final) towards the sample rate s used in the bottle neck of the architecture.

s	TP	FN	FP	Prec	Rec	F1
1	617.5 ± 1.9	103.5 ± 1.9	107.0 ± 1.8	0.852 ± 0.003	0.856 ± 0.003	0.854 ± 0.003
2	618.9 ± 2.0	102.1 ± 2.0	105.4 ± 1.9	0.854 ± 0.003	0.858 ± 0.003	0.856 ± 0.003
4	622.6 ± 1.7	98.4 ± 1.7	101.3 ± 1.5	0.860 ± 0.002	0.864 ± 0.002	0.862 ± 0.002
8	631.0 ± 1.7	90.0 ± 1.7	92.7 ± 1.6	0.872 ± 0.002	0.875 ± 0.002	0.874 ± 0.002
16	649.2 ± 1.3	71.8 ± 1.3	75.9 ± 0.9	0.895 ± 0.001	0.90 ± 0.02	0.898 ± 0.002
32 (default)	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003
64	646.1 ± 1.2	74.9 ± 1.2	79.4 ± 1.4	0.891 ± 0.002	0.896 ± 0.002	0.893 ± 0.002
128	640.9 ± 1.5	80.1 ± 1.5	85.1 ± 1.4	0.883 ± 0.002	0.889 ± 0.002	0.886 ± 0.002
256	483.8 ± 9.7	237.2 ± 9.7	230.7 ± 6.0	0.677 ± 0.008	0.671 ± 0.014	0.674 ± 0.010

Table B.8
Similar to Table B.4. Sensitivity of TCN-AE (final) towards the dimension of the encoded feature map used in the bottle neck of the encoder (last conv. layer in Fig. 3).

dim_{encoded}	TP	FN	FP	Prec	Rec	F1
1	632.0 ± 5.7	89.0 ± 5.7	93.8 ± 5.7	0.871 ± 0.008	0.877 ± 0.008	0.874 ± 0.008
2	652.9 ± 4.8	68.1 ± 4.8	73.4 ± 4.8	0.899 ± 0.007	0.906 ± 0.007	0.902 ± 0.007
4 (default)	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003
8	662.8 ± 1.2	58.2 ± 1.2	63.4 ± 1.2	0.913 ± 0.002	0.919 ± 0.002	0.916 ± 0.002
16	661.9 ± 0.9	59.1 ± 0.9	64.2 ± 0.9	0.912 ± 0.001	0.918 ± 0.001	0.915 ± 0.001
32	657.9 ± 1.2	63.1 ± 1.2	67.5 ± 1.1	0.907 ± 0.002	0.912 ± 0.002	0.910 ± 0.002

0.175 ± 0.018, with only $TP = 148.7 \pm 17.4$, $FN = 1225.3 \pm 17.4$, and $FP = 175.6 \pm 22.6$.

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 $1 \ldots 64$ or $1 \ldots 256$ and obtain similar results in both cases).

Table B.9
Similar to Table B.4. Sensitivity of TCN-AE (final) towards the number of channels used in the skip connections, as described in Section 2.6.1 (the connections ending in the "concat"-layers in Figs. 3 & 4). For # channels = None, no skip connections are employed.

# channels	TP	FN	FP	Prec	Rec	F1
None	622.4 ± 5.7	98.6 ± 5.7	102.9 ± 5.7	0.858 ± 0.008	0.863 ± 0.008	0.861 ± 0.008
1	645.6 ± 3.3	75.4 ± 3.3	80.2 ± 3.5	0.890 ± 0.005	0.895 ± 0.004	0.892 ± 0.005
2	661.0 ± 1.7	60.0 ± 1.7	64.4 ± 1.7	0.911 ± 0.002	0.917 ± 0.002	0.914 ± 0.002
4	662.6 ± 1.5	58.4 ± 1.5	63.3 ± 1.4	0.913 ± 0.002	0.919 ± 0.002	0.916 ± 0.002
8	669.8 ± 0.7	51.2 ± 0.7	56.1 ± 0.7	0.923 ± 0.001	0.929 ± 0.001	0.926 ± 0.001
16 (default)	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003
32	668.5 ± 0.6	52.5 ± 0.6	57.5 ± 0.8	0.921 ± 0.001	0.927 ± 0.001	0.924 ± 0.001
64	664.9 ± 1.0	56.1 ± 1.0	61.5 ± 1.0	0.915 ± 0.001	0.922 ± 0.001	0.919 ± 0.001

Table B.10
Similar to Table B.4. Sensitivity of TCN-AE (final) towards the number of filters used in the feature map reduction layers, as described in Section 2.6.4. The last line indicates the results when no feature map reduction layers are used at all.

# channels	TP	FN	FP	Prec	Rec	F1
2	658.3 ± 1.0	62.7 ± 1.0	67.2 ± 1.0	0.907 ± 0.001	0.913 ± 0.002	0.910 ± 0.001
4	659.9 ± 1.3	61.1 ± 1.3	66.0 ± 1.5	0.909 ± 0.002	0.915 ± 0.002	0.912 ± 0.002
8	665.5 ± 1.3	55.5 ± 1.3	60.8 ± 1.5	0.916 ± 0.002	0.923 ± 0.002	0.920 ± 0.002
16 (default)	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003
32	670.1 ± 0.8	50.9 ± 0.8	55.8 ± 0.8	0.923 ± 0.001	0.929 ± 0.001	0.926 ± 0.001
None	660.7 ± 1.3	60.3 ± 1.3	65.1 ± 1.4	0.910 ± 0.002	0.916 ± 0.002	0.913 ± 0.002

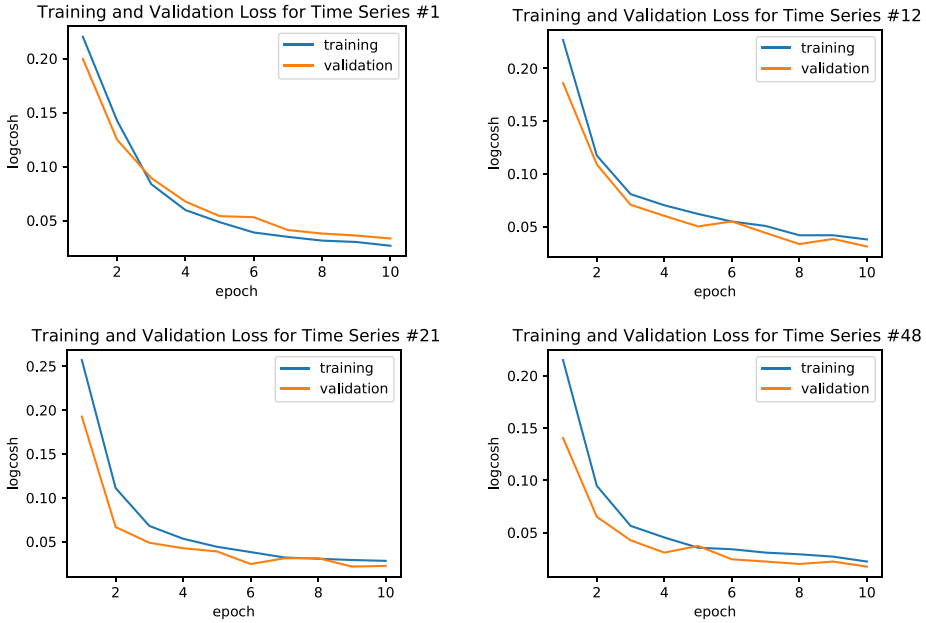

Fig. A.13. Training and validation loss, exemplarily shown for the ECG time series #1, #12, #21 and #48. During the training process, around 10% of the data is used for validation purposes.

- 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: The parallelizable convolution operation allows processing the time series very fast using GPUs (in this study, less than 50 s per time series, as shown in Table 7).
- 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.

5. Conclusion and future work

In this paper, 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.

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. The algorithm 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 five times faster in our experiments than the second-fastest DL algorithm (DNN-AE, see Table 7).

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 quasiperiodic 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) Researching the capabilities of TCN-AE in the field of time series compression. (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.

CRediT authorship contribution statement

Markus Thill: Writing – original draft, Writing – review & editing, Conceptualization, Methodology, Software. **Wolfgang Konen:** Writing – Original Draft, Writing - review & editing, Conceptualization, Supervision, Funding Acquisition. **Hao Wang:** Writing – review & editing, Conceptualization. **Thomas Bäck:** Writing – review & editing, Conceptualization, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Extended results

See Figs. A.10–A.13 and Tables A.1–A.3.

Appendix B. Basic sensitivity analysis

See Tables B.4–B.10.

References

[1] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Comput. Surv. 41 (3) (2009) 15:1–15:58, http://dx.doi.org/10.1145/1541880.1541882.

[2] M.E. Celebi, K. Aydin, Unsupervised Learning Algorithms, Springer, 2016.

[3] D. Xu, Y. Tian, A comprehensive survey of clustering algorithms, Ann. Data Sci. 2 (2) (2015) 165–193.

[4] M. Goldstein, S. Uchida, A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data, PLoS One 11 (4) (2016) e0152173.

[5] S. Thudumu, P. Branch, J. Jin, J.J. Singh, A comprehensive survey of anomaly detection techniques for high dimensional big data, J. Big Data 7 (1) (2020) 1–30.

[6] L. Basora, X. Olive, T. Dubot, Recent advances in anomaly detection methods applied to aviation, Aerospace 6 (11) (2019) 117.

[7] F.T. Liu, K.M. Ting, Z.-H. Zhou, Isolation forest, in: Eighth IEEE International Conference on Data Mining, IEEE, 2008, pp. 413–422.

[8] M.M. Breunig, H.-P. Kriegel, R.T. Ng, J. Sander, LOF: identifying density-based local outliers, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 93–104.

[9] M. Thill, W. Konen, T. Bäck, Online anomaly detection on the webscope S5 dataset: A comparative study, in: I. Skrjanc, S. Blazic (Eds.), Evolving and Adaptive Intelligent Systems, EAIS, Ljubljana, Slovenia, IEEE, 2017, pp. 1–8, http://dx.doi.org/10.1109/EAIS.2017.7954844.

[10] P. Malhotra, L. Vig, G.M. Shroff, P. Agarwal, Long short term memory networks for anomaly detection in time series, in: 23rd European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium, April 22-24, 2015, 2015, pp. 89–94.

[11] M. Thill, S. Däubener, W. Konen, T. Bäck, Anomaly detection in electrocardiogram readings with stacked LSTM networks, in: P. Barancíková, M. Holena (Eds.), Proc. of the 19th Conf. Information Technologies - Applications and Theory (ITAT), in: CEUR Workshop Proceedings, vol. 2473, CEUR-WS.org, 2019, pp. 17–25.

[12] M. Munir, S.A. Siddiqui, A. Dengel, S. Ahmed, DeepAnT: A deep learning approach for unsupervised anomaly detection in time series, IEEE Access 7 (2019) 1991–2005, http://dx.doi.org/10.1109/ACCESS.2018.2886457.

[13] S. Hawkins, H. He, G. Williams, R. Baxter, Outlier detection using replicator neural networks, in: International Conf. on Data Warehousing and Knowledge Discovery, Springer, 2002, pp. 170–180.

[14] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, G. Shroff, LSTM-based encoder-decoder for multi-sensor Anomaly Detection, 2016, CoRR abs/1607.00148. arXiv:1607.00148.

[15] M. Roy, et al., A stacked autoencoder neural network based automated feature extraction method for anomaly detection in on-line condition monitoring, 2018, CoRR abs/1810.08609. arXiv:1810.08609.

[16] J. An, S. Cho, Variational autoencoder based anomaly detection using reconstruction probability, Spec. Lect. IE 2 (1) (2015) 1–18.

[17] J. Pereira, M. Silveira, Unsupervised anomaly detection in energy time series data using variational recurrent autoencoders with attention, in: M.A. Wani, et al. (Eds.), 17th IEEE International Conf. on Machine Learning and Applications, ICMLA, Orlando, FL, IEEE, 2018, pp. 1275–1282, http://dx.doi.org/10.1109/ICMLA.2018.00207.

[18] M. Sölch, J. Bayer, M. Ludersdorfer, P. van der Smagt, Variational inference for on-line anomaly detection in high-dimensional time series, 2016, CoRR abs/1602.07109. arXiv:1602.07109.

[19] H. Xu, et al., Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in Web applications, in: Proc. of the 2018 World Wide Web Conf., 2018, pp. 187–196.

[20] W. Jiang, Y. Hong, B. Zhou, X. He, C. Cheng, A GAN-based anomaly detection approach for imbalanced industrial time series, IEEE Access 7 (2019) 143608–143619, http://dx.doi.org/10.1109/ACCESS.2019.2944689.

[21] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, S.-K. Ng, MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks, in: International Conf. on Artificial Neural Networks, Springer, 2019, pp. 703–716.

[22] D. Li, D. Chen, J. Goh, S. Ng, Anomaly detection with generative adversarial networks for multivariate time series, 2018, CoRR abs/1809.04758. arXiv:1809.04758.

[23] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, N.V. Chawla, A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 1409–1416.

[24] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, Q. Zhang, Multivariate time-series anomaly detection via graph attention network, in: C. Plant, et al. (Eds.), 20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020, 2020, pp. 841–850, http://dx.doi.org/10.1109/ICDM50108.2020.00093.

[25] E. Keogh, J. Lin, A. Fu, HOT SAX: Efficiently finding the most unusual time series subsequence, in: Fifth IEEE International Conference on Data Mining (ICDM'05), Ieee, 2005, pp. 8–pp.

[26] E. Keogh, J. Lin, S.-H. Lee, H. Van Herle, Finding the most unusual time series subsequence: algorithms and applications, Knowl. Inf. Syst. 11 (1) (2007) 1–27.

[27] G. Li, O. Bräysy, L. Jiang, Z. Wu, Y. Wang, Finding time series discord based on bit representation clustering, Knowl.-Based Syst. 54 (2013) 243–254.

[28] M. Ahmed, A.N. Mahmood, J. Hu, A survey of network anomaly detection techniques, J. Netw. Comput. Appl. 60 (2016) 19–31.

[29] A. Aleroud, G. Karabatis, Contextual information fusion for intrusion detection: a survey and taxonomy, Knowl. Inf. Syst. 52 (3) (2017) 563–619.

[30] Y. Himeur, K. Ghanem, A. Alsalemi, F. Bensaali, A. Amira, Artificial intelligence based anomaly detection of energy consumption in buildings: A review, current trends and new perspectives, Appl. Energy 287 (2021) 116601.

[31] M. Ahmed, A.N. Mahmood, M.R. Islam, A survey of anomaly detection techniques in financial domain, Future Gener. Comput. Syst. 55 (2016) 278–288.

[32] E.J.d.S. Luz, W.R. Schwartz, G. Cámara-Chávez, D. Menotti, ECG-based heartbeat classification for arrhythmia detection: A survey, Comput. Methods Programs Biomed. 127 (2016) 144–164.
[33] A.Y. Hannun, P. Rajpurkar, M. Haghpanahi, G.H. Tison, C. Bourn, M.P. Turakhia, A.Y. Ng, Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network, Nat. Med. 25 (1) (2019) 65.
[34] T. Tuncer, S. Dogan, P. Pławiak, U.R. Acharya, Automated arrhythmia detection using novel hexadecimal local pattern and multilevel wavelet transform with ECG signals, Knowl.-Based Syst. 186 (2019) 104923.
[35] S. Sahoo, B. Kanungo, S. Behera, S. Sabut, Multiresolution wavelet transform based feature extraction and ECG classification to detect cardiac abnormalities, Measurement 108 (2017) 55–66.
[36] M. Thomas, M.K. Das, S. Ari, Automatic ECG arrhythmia classification using dual tree complex wavelet based features, AEU - Int. J. Electron. Commun. 69 (4) (2015) 715–721.
[37] E. Alickovic, A. Subasi, Medical decision support system for diagnosis of heart arrhythmia using DWT and random forests classifier, J. Med. Syst. 40 (4) (2016) 108.
[38] S. Chauhan, L. Vig, Anomaly detection in ECG time signals via deep long short-term memory networks, in: 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), IEEE, 2015, pp. 1–7.
[39] A. Adler, M. Elad, Y. Hel-Or, E. Rivlin, Sparse coding with anomaly detection, J. Signal Process. Syst. 79 (2) (2015) 179–188, http://dx.doi.org/10.1007/s11265-014-0913-0.
[40] G. Chakraborty, T. Kamiyama, H. Takahashi, T. Kinoshita, An efficient anomaly detection in quasi-periodic time series data—A case study with ECG, in: I. Rojas, H. Pomares, O. Valenzuela (Eds.), Time Series Analysis and Forecasting, Springer International Publishing, 2018, pp. 147–157, URL http://link.springer.com/10.1007/978-3-319-96944-2_10.
[41] H. Sivaraks, C.A. Ratanamahatana, Robust and accurate anomaly detection in ECG artifacts using time series motif discovery, Comput. Math. Methods Med. (2015).
[42] A.L. Goldberger, L.A. Amaral, L. Glass, J.M. Hausdorff, P.C. Ivanov, R.G. Mark, J.E. Mietus, G.B. Moody, C.-K. Peng, H.E. Stanley, Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals, Circulation 101 (23) (2000) e215–e220.
[43] G.B. Moody, R.G. Mark, PhysioNet: The MIT-BIH Arrhythmia Database, physionet.org, 1992, https://www.physionet.org/physiobank/database/mitdb/.
[44] G.B. Moody, R.G. Mark, The impact of the MIT-BIH Arrhythmia Database, IEEE Eng. Med. Biol. Mag. 20 (3) (2001) 45–50.
[45] F. Nolle, F. Badura, J. Catlett, R. Bowser, M. Sketch, CREI-GARD, a new concept in computerized arrhythmia monitoring systems, Comput. Cardiol. 13 (1986) 515–518.
[46] M. Holschneider, R. Kronland-Martinet, J. Morlet, P. Tchamitchian, A real-time algorithm for signal analysis with the help of the wavelet transform, in: J.-M. Combes, A. Grossmann, P. Tchamitchian (Eds.), Wavelets, Springer Berlin Heidelberg, 1990, pp. 286–297, URL http://link.springer.com/10.1007/978-3-642-75988-8_28.
[47] A. van den Oord, et al., WaveNet: A generative model for raw audio, 2016, CoRR abs/1609.03499. arXiv:1609.03499.
[48] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, in: 4th International Conf. on Learning Representations, ICLR, 2016, pp. 1–13.
[49] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018, CoRR abs/1803.01271. arXiv:1803.01271.
[50] Y. He, J. Zhao, Temporal convolutional networks for anomaly detection in time series, in: Journal of Physics: Conference Series, Vol. 1213, IOP Publishing, 2019, 042050.
[51] A. Choromanska, M. Henaff, M. Mathieu, G.B. Arous, Y. LeCun, The loss surfaces of multilayer networks, in: G. Lebanon, S.V.N. Vishwanathan (Eds.), Proc. of the 18th Int. Conf. on Artificial Intelligence and Statistics, AISTATS,, in: JMLR Workshop and Conference Proceedings, vol. 38, 2015, pp. 192–204, URL http://proceedings.mlr.press/v38/choromanska15.html.
[52] T.A. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, Q. Liao, Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review, Int. J. Autom. Comput. 14 (5) (2017) 503–519.
[53] M. Thill, W. Konen, T. Bäck, Time series encodings with temporal convolutional networks, in: Bioinspired Optimization Methods and their Applications, Springer International Publishing, 2020, p. n.a., (accepted Aug. 2020).
[54] Y.A. Farha, J. Gall, MS-TCN: multi-stage temporal convolutional network for action segmentation, in: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Computer Vision Foundation / IEEE, 2019, pp. 3575–3584.
[55] Y.N. Dauphin, A. Fan, M. Auli, D. Grangier, Language modeling with gated convolutional networks, in: Proc. of the 34th International Conf. on Machine Learning - Volume 70, in: ICML'17, JMLR.org, 2017, pp. 933–941.
[56] J. Gehring, M. Auli, D. Grangier, D. Yarats, Y.N. Dauphin, Convolutional sequence to sequence learning, 2017, CoRR abs/1705.03122. arXiv:1705.03122.
[57] N. Kalchbrenner, E. Grefenstette, P. Blunsom, A convolutional neural network for modelling sentences, in: Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Baltimore, Maryland, 2014, pp. 655–665, http://dx.doi.org/10.3115/v1/P14-1062.
[58] T. Salimans, D.P. Kingma, Weight normalization: A simple reparameterization to accelerate training of deep neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 901–909.
[59] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proc. of the 27th Int. Conf. on Machine Learning (ICML), 2010, pp. 807–814.
[60] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[61] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[62] M. Lin, Q. Chen, S. Yan, Network in network, 2013, arXiv preprint arXiv:1312.4400.
[63] C. Szegedy, et al., Going Deeper with convolutions, 2014, CoRR abs/1409.4842. arXiv:1409.4842.
[64] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.
[65] M. Thill, W. Konen, T. Bäck, Time series anomaly detection with discrete wavelet transforms and maximum likelihood estimation, in: O. Valenzuela, I. Rojas, et al. (Eds.), Intern. Conference on Time Series (ITISE), Vol. 2, 2017, pp. 11–23.
[66] J. Pan, W.J. Tompkins, A real-time QRS detection algorithm, IEEE Trans. Biomed. Eng 32 (3) (1985) 230–236.
[67] M. Fischer, W. Gierke, T. Kellermeier, A. Kesar, A. Stebner, D. Thevessen, Anomaly Detection on Time Series: An Evaluation of Deep Learning Methods, GitHub, 2019, https://github.com/KDD-OpenSource/DeepADoTS.
[68] M. Taylor, et al., numenta/nupic: 1.0.5, Zenodo, 2018, https://doi.org/10.5281/zenodo.1257382.
[69] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, D.D. Cox, Hyperopt: a Python library for model selection and hyperparameter optimization, Comput. Sci. Discov. 8 (1) (2015) 014008.
[70] A. Lavin, S. S. Ahmad, Evaluating real-time anomaly detection algorithms – the Numenta Anomaly Benchmark, in: IEEE Conf. on Machine Learning and Applications (ICMLA), 2015, pp. 38–44, http://dx.doi.org/10.1109/ICMLA.2015.141.
[71] S. Haykin, Adaptive Filtering Theory, fifth ed., Pearson, 2013.
[72] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Weiss, V. Dubourg, J. Vanderplas, A. Passos, M. Brucher, M. Perrot, E. Duchesnay, et al., Scikit-learn: Machine learning in python, J. Mach. Learn. Res. (JMLR) 12 (2011) 2825–2830.
[73] B. Schölkopf, R.C. Williamson, A.J. Smola, J. Shawe-Taylor, J.C. Platt, Support vector method for novelty detection, in: Advances in Neural Information Processing Systems, 2000, pp. 582–588.
[74] Z. Wen, J. Shi, Q. Li, B. He, J. Chen, ThunderSVM: A fast SVM library on GPUs and CPUs, J. Mach. Learn. Res. (JMLR) 19 (2018) 797–801.
[75] A. Paszke, et al., PyTorch: An imperative style, high-performance deep learning library, in: H. Wallach, et al. (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.
[76] S. Ahmad, Running swarms, 2017, http://nupic.docs.numenta.org/0.6.0/guide-swarming.html.
[77] F. Chollet, et al., Keras, 2015, https://keras.io.
[78] M. Abadi, A. Agarwal, P. Barham, et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, https://www.tensorflow.org/.
[79] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proc. of the 13th Int. Conf. on Artificial Intelligence and Statistics, 2010, pp. 249–256.
[80] F. Wilcoxon, Individual comparisons by ranking methods, in: Breakthroughs in Statistics, Springer, 1992, pp. 196–202.
[81] H. Li, Z. Xu, G. Taylor, C. Studer, T. Goldstein, Visualizing the loss landscape of neural nets, in: Advances in Neural Information Processing Systems, 2018, pp. 6389–6399.
[82] Z.-M. Zhang, S. Chen, Y.-Z. Liang, Baseline correction using adaptive iteratively reweighted penalized least squares, Analyst 135 (2010) 1138–1146, http://dx.doi.org/10.1039/B922045C.
[83] M. Lagerholm, C. Peterson, G. Braccini, L. Edenbrandt, L. Sornmo, Clustering ECG complexes using Hermite functions and self-organizing maps, IEEE Trans. Biomed. Eng. 47 (7) (2000) 838–848.
[84] M.G. Tsipouras, D.I. Fotiadis, D. Sideris, Arrhythmia classification using the RR-interval duration signal, in: Computers in Cardiology, IEEE, 2002, pp. 485–488.