POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# Anomaly Detection in Multivariate Time Series: Comparison of Selected Inference Models and Threshold Definition Methods

Tesi di Laurea Magistrale in
Computer Science and Engineering -
Ingegneria Informatica

Author: **Gioele Verze**

Student ID: 966400
Advisor: Prof. Piero Fraternali
Co-advisors: Nicolò Oreste Pinciroli Vago
Academic Year: 2021-22

# Abstract

Anomaly detection is an essential analysis that regards various fields ranging from medical detection to industrial damage detection, from intrusion detection to fraud. It is focused on automatically monitoring different types of data, such as images or time series, to predict and detect possible malfunctions or unexpected and unpredicted behaviors. An accurate, quick, precise, and efficient anomaly detection makes it possible to achieve significant benefits, mainly in time and economics. This work presents anomaly detection methods based on several neural networks method analyzing two different datasets: SKAB and Exathlon. They both contain a multivariate time series that, respectively, record tests on a test bench and some Spark application execution. Different inference models, combined with several thresholding techniques, have been tested to evaluate whether the most relevant contribution to detecting anomalies regards the model or the thresholding method.

**Keywords:** Anomaly Detection, Multivariate Time Series, Threshold

# Abstract in lingua italiana

Il rilevamento delle anomalie è un'analisi essenziale che copre svariati campi, dal rilevamento di anomalie in campo medico a quello dei danni industriali, dal rilevamento delle intrusioni a quello delle frodi. Si concentra sul monitoraggio automatico di diversi tipi di dati, come immagini o serie temporali, per prevedere e rilevare malfunzionamenti o comportamenti inaspettati e non previsti. Un rilevamento accurato, rapido, preciso ed efficiente delle anomalie produce vantaggi significativi, soprattutto in termini di tempo e di costi. Il lavoro presenta metodi di rilevamento delle anomalie basati su diverse reti neurali, analizzando due diversi set di dati: SKAB ed Exathlon. Entrambi contengono una serie temporale multivariata che registrano rispettivamente i test su un banco di prova e l'esecuzione di dieci applicazioni Spark. Sono stati testati diversi modelli di inferenza, combinati con varie tecniche di threshold, per valutare se il contributo più rilevante nell'induviduare le anomalie riguarda il modello o il metodo di threshold.

**Parole chiave:** Anomaly Detection, Serie Temporale Multivariata, Threshold

# Contents

# 1 | Introduction

Anomaly detection is a technique used to identify behaviors that deviate from normality. It allows monitoring and detecting possible anomalous events in the data retrieved to prevent and detect failures, malfunctions, fraud, intrusion, medical diseases, and, more in general, unexpected behavior in a timely fashion to minimize their impact on the overall system [14].

Anomaly detection techniques have been proposed for diverse types of data, including images [19, 36, 36, 87] and time series [12, 63, 86]. This thesis focuses on anomaly detection applied to multivariate time series collected through sensors. In particular, two different multivariate datasets are considered. The first is SKAB [52], a dataset containing experiments collected from sensors installed on a water pump. The other, Exathlon [46], refers to the recording of different repeated executions of ten Spark streaming applications on the same cluster.

In the context of time series, several approaches address anomaly detection tasks [14], including statistical [71], clustering-based [83], nearest neighbors distance-based [117], and ensemble [15] approaches. This thesis focuses on neural networks and a classification-based approach in the context of unsupervised anomaly detection.

Generally, the output of the anomaly detection model could be of two types. It can label each test instance directly, classifying if it is normal or not, or it can return an anomaly score for each test instance representing the degree of anomaly. The higher the score, the more the test instance could be anomalous. A threshold is necessary for models that return a score to distinguish anomalies from normal samples. Typically, higher thresholds minimize false positives (i.e., normal points predicted as anomalous), while lower thresholds minimize false negatives (i.e., anomalous points predicted as normal).

Related works present several techniques to calculate the threshold. An approach proposed by [2, 8, 99], is to use as a threshold a value that maximizes one of the evaluation metrics such as precision, recall, or F1-Score. However, since it needs all the tests set to compute the threshold is not applicable to online anomaly detection. Other approaches, such as MAD, STD, and IQR proposed by [46], consist in calculating the threshold on a small portion of the dataset, called validation, and then applying it to all scores. Another similar approach, computed on a small validation set and proposed by [101], is to consider the maximum score value as a threshold considering all the data in the validation set normal.

Several neural architectures have been proposed and analyzed in detail to compute the anomaly score. Each architecture, according to how it works, generates different results, which are compared together and to state of the art. The simplest architecture used is a Recurrent Neural Network called LSTM, which predicts a future sequence starting from what it has learned in training and based on previous sequences [70]. Then a particular neural network configuration, called autoencoder, is analyzed. Autoencoders are composed of the encoder, the latent space, and the decoder. First, the encoder compresses the input data into a lower-dimensional space (the latent space). Then, the decoder, starting from the latent space data, tries to reconstruct the input sequence. The more the data to analyze in the input are normal and describe the normal behavior learned in training, the more precise and accurate the reconstruction. On the other hand, if the sequence in input describes an anomalous behavior, the reconstruction is not precise and deviates consistently from the ground truth. So, reconstruction and original data are compared and generate an anomaly score to be evaluated to distinguish anomalies. In detail, according to the type of layer used in the implementation of the neural network, are analyzed DENSE-AE, CONV-AE, LSTM-AE, and VAE. The latter is quite different from others, and using it is possible to detect anomalies by analyzing the latent space distribution using KNN [22], Isolation Forest [66], and the method proposed in [114]. Moreover, since the state of the art of examined dataset, they produce good results, ELM-MI [81], and USAD [7] models are adapted and used.

The scope of the work is to analyze different inference models and thresholding techniques to understand which components contribute more to the final metrics result. In other words, according to the different datasets analyzed, the work shows if the most significant contribution to the anomaly detection results belongs to the choice of the model or to the threshold method. What emerges is that the choice of model has a relevant impact on the result. Some models generate a score distribution where normal and anomalous data are clearly separated, and the final results are better than models that separate normal and anomalous data worst. The impact of the threshold technique on the final results is highly dependent on how it is constituted and what values the validation set contains.

The thesis is organized as follows:

**Chapter 2** introduces time series and anomaly detection in general and provides an overview of anomaly detection approaches for time series. It also contains several thresholding techniques with their related work;

**Chapter 3** presents the inference models analyzed and implemented in this thesis;

**Chapter 4** present the datasets, SKAB [52] and Exathlon [46], and provides an overview of relevant hyper-parameters of the machine learning models;

**Chapter 5** presents qualitative and quantitative results for both datasets with the implemented models and compares them to the available state of art results;

**Chapter 6** draws conclusions and proposes future directions for research.

# 2 | Background

## 2.1. Time Series

### 2.1.1. Definition

A time series is a discrete sequence of values indexed by time. The time order is the principal feature of a time series where each value is assigned a timestamp according to when it was observed. The analysis of a time series shows how a variable changes over time and can show the dependencies between different variables.

Time series are used in different fields [30], ranging from statistics to signal processing, finance to pattern recognition, astronomy, and weather forecasting. More generally, time series are used in applied science and engineering domains involving temporal measurements.

### 2.1.2. Components

Time series can be represented as the combination of simpler components that can be added or multiplied according to the formula [42]:

$$y_t = T_t \times C_t \times S_t \times R_t$$

$$y_t = T_t + C_t + S_t + R_t$$

where $y_t$ is a value at the instant $t$ of the time series. The components $S$, $C$, $T$ and $R$ in the formula are respectively [49]:

- **Seasonal component (S)**: the seasonal component refers to the seasonality of a time series. It reflects variations that recur every season to the same extent. For example, in a time series that contains retail sales, there is an evident seasonal component corresponding to December, where, each year, the sales increase due to Christmas shopping. Generally, this component is important when a time series exhibits regular fluctuations based on the season (e.g., every month/quarter/year), which is fixed and known.

- **Cyclical component (C)**: the cyclical component exists when data exhibit rises and falls, not of fixed periods. For example, a time series containing retail sales could have a cyclical component corresponding to boom, slump, recession, and recovery periods.
- **Trend component (T)**: the trend component corresponds to a pattern in data that shows the movement to relatively higher or lower values over a long period. It is observed when there is an increasing or decreasing slope in the time series. The trend usually does not repeat. A trend could be:
  - *Uptrend*: shows a general upward pattern. For example, it is visible in a time series that represents a country's energy consumption; its value is constantly growing over the years.
  - *Downtrend*: shows a general downward pattern. In a time series that reports sales of a particular product, a downtrend may be visible due to a reduction in sales caused by the marketing of another competing good.
  - *Horizontal trend*: no general pattern is observed.
- **Random component (R)**: the random component is unpredictable. Every time series has some unpredictable component that makes data change randomly. These variations are fluctuations in time series that last short and follow no regularity in the occurrence pattern. Random component refers to what is not considered by trend, cyclical, and seasonal variations. For example, [50] shows that the random component may occur due to wars, earthquakes, or floods.

### 2.1.3. Univariate and Multivariate

Time series can be categorized based on the number of values associated with a single timestamp. They can be divided into two categories [12]:

- **Univariate time series**: A univariate time series $X = \{x_t\}_{t \in T}$ is defined as an ordered set of real-valued observation, $x_t$, where each observation is at a specific time $t \in T \subseteq \mathbb{Z}^+$
- **Multivariate time series**: A multivariate time series $X = \{x_t\}_{t \in T}$ is defined as an ordered set of k-dimensional vectors, $x_t = (x_{1t}, ..., x_{kt})$, where each observation is recorded at a specific time $t \in T \subseteq \mathbb{Z}^+$

Compared to a univariate time series, a multivariate time series has a more comprehensive vision of the environment to which it refers, as it relates to multiple quantities.

The analysis of univariate time series considers a single-time dependent variable, whereas an analysis of a multivariate one considers simultaneously more than one variable. Alternatively, univariate analyses can be performed on multivariate time series processing each time-dependent variable without considering the dependencies that may exist between the variables. Considering multivariate time series, dimensionality reduction allows for reducing the number of variables. PCA [5] or autoencoders can perform dimensionality reduction, as shown in [105].

### 2.1.4. Stationarity

Stationarity means that the time series generating process's statistical properties remain constant over time. Not necessarily data is constant. Instead, the way it changes is consistent. A stationary time series' mean, variance, and autocorrelation remain constant. An example of a stationarity time series is visible in Figure 2.1. On the other hand, time series with a changing mean or variance are non-stationary, as shown in Figure 2.2. A time series exhibiting trend or seasonality, for example, would be non-stationary because these components affect both the mean and variance. A further distinction in stationary processes is [23]:

- *Strict stationarity* [84]: The time series $X_t, t \in \mathbb{Z}$ is said to be strict stationary if the joint distribution of $(X_{t_1}, X_{t_2}, ..., X_{t_n})$ is the same as $(X_{t_{1+h}}, X_{t_{2+h}}, ..., X_{t_{n+h}})$ meaning that the joint distribution depends only on $h$ and not on time $(t_1, ...t_n)$.
- *Weak Stationarity*: The time series $X_t, T \in \mathbb{Z}$ is said to be weak stationary if the following three conditions are verified:

$$E[X_t^2] < \infty \quad \forall t \in \mathbb{Z}$$

$$E[X_t] = \mu \quad \forall t \in \mathbb{Z}$$

$$Cov(X_s, X_t) = Cov(X_{s+h}, X_{t+h}) \quad \forall t, s, h \in \mathbb{Z}$$

  where $E[X_t]$ is the expected value at time $t$ and $\mu$ is the mean of the series, $E[X_t^2]$ corresponds to the root-mean-square value and

$$Cov(X_s, X_t) = E[(X_s - X[X_s])(X_t - X[X_t])]$$

  Given the assumption that the mean and the root-mean-squared exist and are finite, strict stationarity implies weak stationarity.

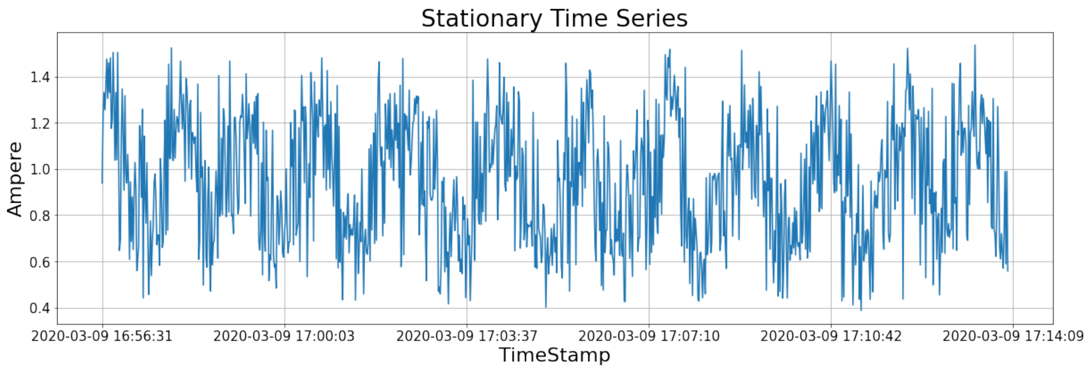


Figure 2.1: Stationary time series of Current value from SKAB dataset[52]

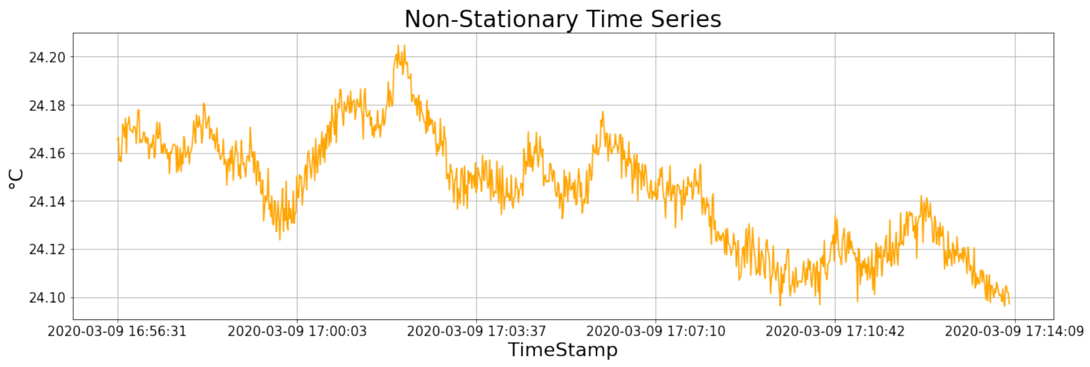


Figure 2.2: Non-stationary time series of Thermocouple value from SKAB dataset[52]

#### 2.1.4.1. Dickey-Fuller test

To determine if a time series is stationary, a method that can be used is the Dickey-Fuller [25] test. It is a unit root test that statistically detects stochastic behaviors in time series using a hypothesis test. Starting from an autoregressive model defined as:

$$Y_t = \rho Y_{t-1} + \epsilon_t$$

Where $Y_t$ is the variable of interest at time $t$, $\epsilon_t$ is an error term, and $\rho$ is a coefficient that defines the unit root, the stationarity classification can be performed by testing the coefficient value according to the following hypothesis:

$$H_0 : \rho = 1$$

$$H_A : \rho \neq 1$$

The model is non-stationary if the null hypothesis is verified. On the other hand, the time series is stationary by rejecting the null hypothesis. This approach generated two problems. Mainly the t-test cannot be applied to an autoregressive model, and, by definition, the null hypothesis has to be $\rho = 0$ and not $\rho = 1$. So, to overcome these problems, the equation can be manipulated by subtracting $Y_{t-1}$ from both sides of the autoregressive model.

$$Y_t - Y_{t-1} = \rho Y_{t-1} - Y_{t-1} + \epsilon_t$$

$$\Delta Y_t = (\rho - 1)Y_{t-1} + \epsilon_t$$

Then, by substituting $\alpha = (\rho - 1)$, the t-test is applied to the null hypothesis $\alpha = 0$.

## 2.2. Anomaly Detection in Time Series

Anomaly detection is a problem that affects a wide variety of domains [113], including cybersecurity, fraud detection, industry [73], and medical analysis [31]. It consists of detecting expectations, deviations, and differences from most of the data [96].

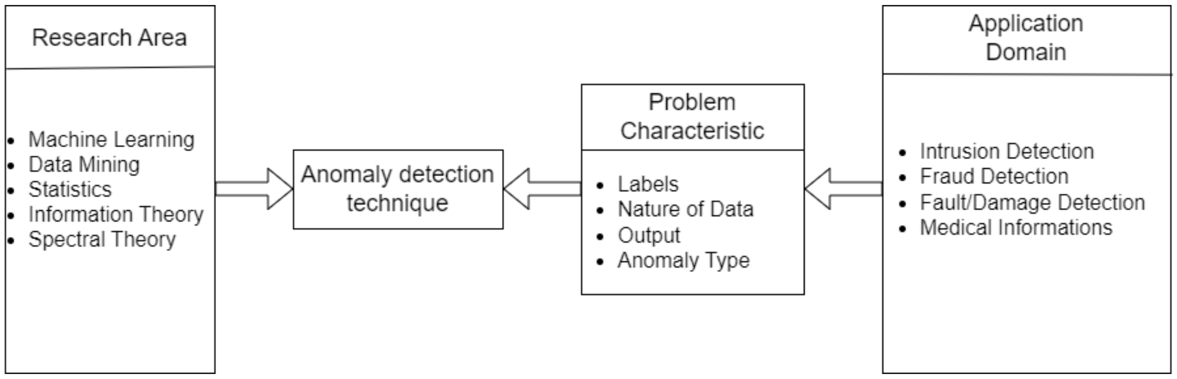


Figure 2.3: Key component of anomaly detection [14]

The basis of anomaly detection is to distinguish normal and anomalous behaviors. This task poses several challenges. First, defining intervals representing every possible normal behavior is challenging, as the boundaries to distinguish normal and anomalous samples are often not precise. This could generate a lot of mispredictions. Moreover, depending on the data, a slight fluctuation may represent anomalous behaviors, while in other data, the same change is irrelevant. For example [14], in the medical field, a small deviation from normal body temperature is an anomaly, while a similar fluctuation in the stock market domain could be normal. The last issue concerns the quality of the data, which can contain noise. The noise is erroneous or random contamination sample recorded incorrectly. The noise is not interesting in anomaly detection, but it can be used to rate the quality of the instrument to collect data [85]. The challenge is distinguishing real anomalies from noise. So, before doing anomaly detection, noise is detected and removed by denoising algorithms [53].

Moreover, in many domains, such as medical and healthy care, normal behavior keeps evolving, so a current definition of normal behavior might not represent the future [14].

### 2.2.1. Defining Anomaly

In the work of [3], outliers are commonly used as synonyms for abnormalities, discordant, deviants, or anomalies. Grubbs [32], in 1969, defines an outlier as follows: "outlier is one that appears to deviate markedly from other members of the sample in which it occurs". Then Hawkins [38], in 1980, redefined it as: "an observation which deviates so significantly from other observations as to arouse suspicion that it was generated by a different mechanism". Finally, a recent definition of outlier is provided by Barnett and Lewis [11], in 1994, defining it as: "observations or a subset of observations which appears to be inconsistent with the remainder of that data set".

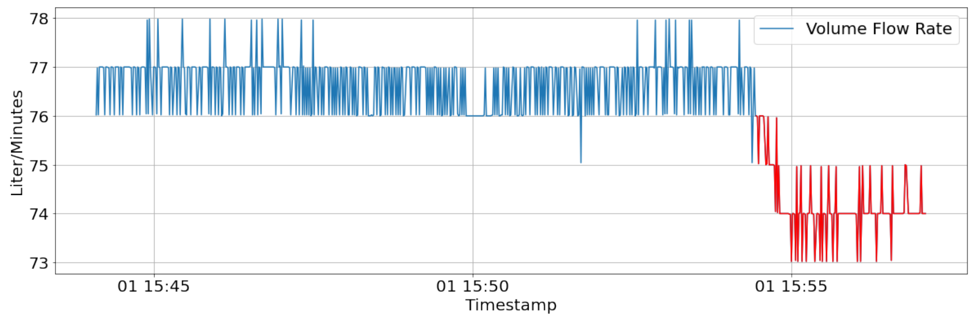
All three definitions have a point in common; when referring to outliers, something anomalous is mentioned, which is very different compared to other data. Despite being very similar, the three definitions have considerable differences. The first two definitions refer only to single and separated anomalies, while the most recent one offers a wider definition by also including sequences of points, which are regarded as anomalous when considered together.

Anomalies can be divided into three main categories: point outliers, collective outliers, and contextual outliers.

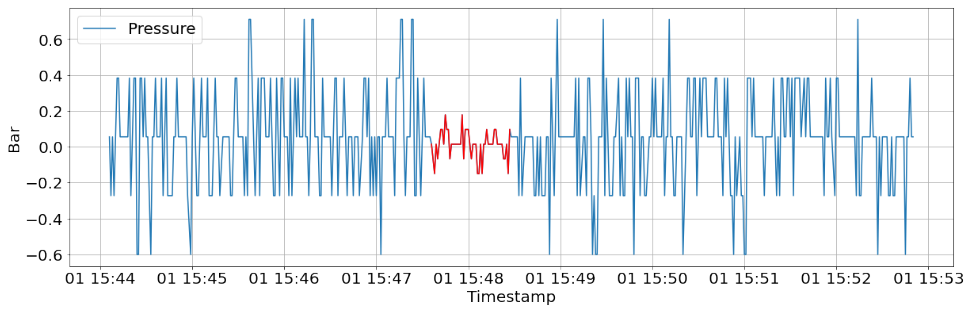
**Point outliers** : point outliers, Figure 2.4, refer to single data instances that deviate significantly from the rest of the data. They represent the simplest type of anomaly. They have a relevant role in time series analysis. For example, these types of anomalies appear in the analysis of credit card transactions where purchases with an unusual transaction value can indicate potential fraud [95].

**Contextual outliers** : contextual outliers, Figure 2.4b, refer to single data instances that, taken individually, appear normal but are anomalous in a specific context. This means that two points with the same value could be classified differently. Contextual anomalies are determined by combining contextual and behavioral attributes [95]. The former is used to determine the context of each data instance. For example, contextual attributes could correspond to latitude and longitude in a spatial dataset or in time series that corresponds to the time. The latter refers to the noncontextual characteristic of an instance corresponding to the value of each sample. An example is temperature measurement, where two equal high temperatures are classified differently, one normal and the other anomalous, according to the period in which they are taken.

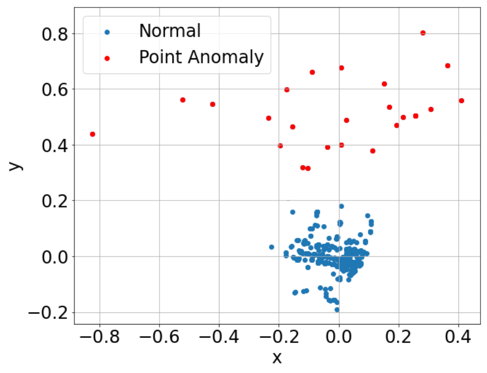
**Collective outliers** : collective outlier, Figure 2.4a, refer to consecutive and related data that are considered abnormal with respect to most of the data in the dataset. A single data instance taken individually from a collective anomaly may not be an anomaly, but the interval in which it is contained is anomalous. However, often the values considered collective outliers are point anomalies, Therefore, these outliers are significant in time-series analysis, indicating anomalies for consecutive timestamps.



(a) **Collective outlier** retrieved from a time series representing the 'Volume Flow Rate' from SKAB [52] dataset



(b) **Contextual outlier** retrieved from a time series representing the 'Pressure' from SKAB [52] dataset



(c) **Point outlier** retrieved by a latent space analysis of VAE, a machine learning model described later in section 3.3 and in section 3.8b

Figure 2.4: Contextual, collective, point outlier representation

### 2.2.2. Anomaly Detection Paradigms

Depending on whether the dataset contains a label representing the anomalies or not, three different learning paradigms [89] can be implemented to detect anomalies.

In **Supervised Learning**, the training set data must be labeled into two different categories: normal and anomalous. In this case, the quality of the training set is essential. Often, conspicuous manual work could be necessary to label data correctly. In supervised learning, the model tries to extract anomalous patterns from training data and detect them in the test set. Despite performing precise and accurate anomaly detection, it has the disadvantage of being an expensive and slow process. Another relevant limit concerned the data. Supervised data represent only existent data which not represents all the possible features and behavior of the system analyzed. So, the supervised model learns in a limited way, not having a complete view of the system [69], and the anomalies detected are only those similar to the ones in the training set.

In **Unsupervised Learning**, the data of the training set are not labeled, so anomalous data are not explicitly marked. This learning paradigm is used to create models able to mark an input value as normal or anomalous based on a model of normal behaviors [26]. Unsupervised learning in anomaly detection of time series consists in learning how the analyzed system normally behaves by learning the pattern and trend of the training data. The anomalies can be detected in different ways. The main consist in [12]:

- isolated samples that deviated consistently from the training data. It is a common approach exploited by Isolation Forest [66], nearest neighbors distance-based model

[33], and clustering-based model [63].

- consider anomalous the samples that, compared to the prediction performed by a neural network model according to what it has learned during the training, differ more than a threshold value. It is a common approach exploited by RNN [70].
- consider anomalous the samples that, after a reconstruction performed by a neural network model according to what it has learned during the training, are reconstructed badly differing from the ground truth more than a threshold value. It is a common approach exploited by autoencoders [54].

Unlike supervised learning, unsupervised learning does not require a labeled dataset, so it can be applied to all datasets. In addition, it is effective for unstructured and huge datasets. Moreover, it can detect all types of anomalies, while supervised anomaly detection can detect only the anomalies contained in training data [13].

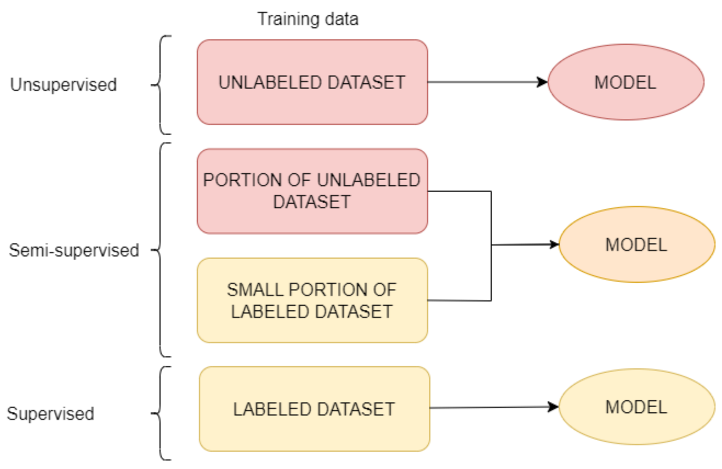


Figure 2.5: Representation of supervised, unsupervised and semisupervised learning

In general **Semi-Supervised Learning**, only a small portion of the training data is labeled. In that case, two different approaches can be used. The first is not to consider the labels and use the unsupervised learning method. Conversely, the second allows using the supervised method thanks to the label propagation [45]. It consists of training the model with the marked data and then using it to label the unlabeled portion of the training set. After that, the training set is ready to be processed by a supervised method. It is a method commonly used in image classification where only a small part of the training set is labeled. In the particular case of semi-supervised learning applied to anomaly detection, the training data has labeled instances only for normal data, and anomalies are not explicitly labeled. Since semi-supervised techniques do not require labels for anomalies, they are more widely applicable than the supervised approach [14].

The approach is to build a model for the normal data and then use it to identify anomalies in the test data. The semi-supervised learning is not commonly used since it is difficult to have a training set that covers all possible anomalous behavior that can occur on the data. It has the advantage of being more accurate than unsupervised learning. Furthermore, compared with supervised learning, it is a faster and less costly process because it does not require all the data in the dataset to be marked.

### 2.2.3. Anomaly Detection Applications

The Anomaly detection field comprises a lot of different usage and application. According to [14] anomaly detection techniques and the field in which anomaly detection is used can be classified as:

- Intrusion Detection
- Fraud Detection
- Medical and Public Health Anomaly Detection
- Industrial Damage Detection
- Image Processing
- Sensor Networks

#### 2.2.3.1. Intrusion Detection

A field in which anomaly detection is involved is intrusion detection [57], which consists of unauthorized access and manipulation of information or making a system unreliable. There are two different ways to classify the intrusion; one can be performed from outside the system, and the other by someone inside who knows the system's vulnerabilities. The approach to detect an intrusion is to create a normal historical profile for each user and then, by comparing them to new activities, detect possible intrusion. Moreover, the available data corresponds to the normal behavior, and usually, anomalous data are few or do not exist. So the most used approach in this domain is the unsupervised and semi-supervised anomaly detection technique which learns the normal behavior and can distinguish anomalies considering the sample that deviate consistently from what the model has learned. The advantage of this approach is the ability to detect any intrusion, including novel attacks on the system. However, a limit of this approach is a high false alarm rate where normal activities could be classified as intrusions being the detection, less accurate than the supervised approach.

#### 2.2.3.2. Fraud Detection

With the expansion of modern technology and global communication, fraud is increasing, and **fraud detection** [57] plays a vital role in the anomaly detection field. Fraudulent activities involve many areas of daily life, such as banking, E-commerce, and mobile communication. They happen when malicious agents consume resources provided by the attacked organization in an unauthorized way. Fraud detection consists in detecting fraud as quickly as possible to prevent economic losses; responsiveness and reactivity are the main features. In addition, fraud detection methods must be continually developed and updated as criminals continue to adapt and create new ways to bypass the existing detection methods. **Credit card fraud** [24] is one of the most relevant fields in which fraud detection is involved. This fraud could happen using a stolen physical card or via the web, where only some card details are needed. To detect fraud can be used different methods that analyze each credit card transaction. The supervised approach consists of comparing each transaction with historical data previously classified. The limitation of this approach is that it detects only fraud of a type that has once occurred. On the other hand, an unsupervised system can detect new types of fraud which do not need prior knowledge of fraudulent and non-fraudulent transactions. In addition, it detects unusual transactions, thus transactions that deviate consistently from normal ones. Fraud detection can also detect **insider trading** [58]. It happens on the market when people make illegal profits by leaking inside information before it is made public. Therefore, to prevent illegal profit fraud has to be detected as soon as possible online. Fraud detection involved also **mobile phone fraud** [9], which consists of phone cloning or subscription fraud using false identification.

#### 2.2.3.3. Medical and Public Health Anomaly Detection

Anomaly detection performs a crucial role also in **medical and public health** information [31]. From physiological data, anomaly detection can be performed to predict the patient's future conditions or diagnose tasks. For diagnosing studies, physiological data are analyzed to recognize pathological signs of medical conditions. So, medical and public health anomaly detection is crucial and needs high precision and accuracy. Typically, this application analyzes data of different types, such as patient age, weight, and blood group or data related to temporal aspects. A relevant use is in analyzing electrical and biomedical signals, such as the electrocardiogram (ECG), electroencephalogram, or magnetoencephalography. For example, studying ECG series anomaly detection methods [17] can screen irregular electrical activities and find stressed regions. Another common use of anomaly detection techniques is the analysis of biomedical images [36] such as xray radiography, computed tomography, or magnetic resonance images. Frequently, the techniques used a semi-supervised approach given an extensive data history with corresponding normal anomaly classification.

#### 2.2.3.4. Industrial Damage Detection

Due to the continuous usage and normal wear and tear, anomaly detection has a relevant role in detecting **industrial damage**. Also, in this field, responsiveness is essential to prevent further breakdowns and economic loss. Therefore, by considering information from different sensors, it is possible to identify malfunctions in industrial machinery to detect anomalies early to prevent damage [86]. Industrial damage detection can be further classified according to its scope. For example, *fault detection in mechanical units* [29] is about monitoring industrial components' performance, such as motor, turbine, and fluid flows. Instead, *structural defect detection* [68] deals with physical structure defects such as beam cracks.

A possible example of industrial damage detection is the one used on the SKAB [52] dataset, used in the thesis's implementation part. In detail, the dataset contains a multivariate dataset that describes a test bench performed on a water pump. By analyzing the multivariate time series, different neural network model tries to detect possible anomalies representing pump malfunction.

#### 2.2.3.5. Image processing

**Image processing** occupies a relevant area in anomaly detection. It consists in analyzing both static images and images that change over time. The static analysis consists of detecting anomaly points or regions that appear abnormal and assume an important role also in medical and public health detection [36] and industrial and damage detection [87]. The video analysis [75] focuses on how images change over time, checking attributes such as color, lightness, and texture to detect anomalies caused by motion, insertion of foreign objects, or intrusions [118]. The principal issue of image processing is to work with a large input size that reduces performance, and for video analysis, online detection is mandatory.

#### 2.2.3.6. Sensor Networks

Anomaly detection has an important role in analyzing data collected from IoT sensors. Sensors, by their findings, can describe the environment in which they are installed. So an accurate analysis of them is helpful to detect any changes in the environmental state and consequently register anomalies such as sensors fault or intrusions. Anomaly detection based on sensor networks is used in very different areas [115]:

- **Environment monitoring**: sensors are used to monitor the natural environment, such as temperature and humidity.
- **Industrial monitoring**: sensors are used to monitor the machinery behavior, and an analysis of their measurement shows and can predict possible malfunctions.
- **Target tracking**: sensors are used to track in a real-time way moving objects.
- **Healt and medical monitoring**: sensor on the human body to monitor electrical and biomedical signals to detect potential diseases.
- **Surveillance monitoring**: sensors are used to monitor a given area to detect unauthorized access and potential attack.

An issue that makes the detection more difficult is that sensor records often contain missing or noisy data due to the communication channel. The presence of noise makes anomaly detection more challenging because detector models must distinguish between the noise and the real anomaly. Also, in this area, anomaly detection techniques are required to operate online to detect anomalies as soon as possible to mitigate the effect of the anomalies.

### 2.2.4. Anomaly Detection Techniques

According to [103] and to [14], there are different anomaly detection techniques based on the problem statement, the input data type, and if they are marked or not, the desired output, and if the problem needs a reactive or proactive response to the anomalies.

#### 2.2.4.1. Classification-based

The classification-based anomaly detection technique [14] is a two-phase algorithm divided into training and testing. It is based on a model, also called a classifier, that, starting from training data, can learn their feature and then apply the classification to the testing data. According to what has been learned, the classified can distinguish between normal and abnormal classes. According to the training data, classification-based anomaly detection techniques can be split into multi-class and one-class anomaly detection techniques. Suppose the training set comprises only one type of data, normal instances. In that case, the one-class classification is used where the model learns a discriminative boundary around the normal sample, and any testing instance that does not fit in it is classified as anomalous. The opposite technique is multi-class classification, where the training data belongs to different classes, and the model learns to distinguish between classes. Approaches that exploit this technique are:

- **Neural networks-based**: This approach uses a neural network model to reconstruct the input data. Then, according to a score function defined by comparing the input data and the reconstructed one, classify each sample as normal or anomalous.
- **Bayesian networks-based** [39]: it is an approach used in the multi-class setting. Bayesian networks are a graphical probabilistic model that represents the dependency between data features in which it is applied throw a direct acyclic graph.
- **Support vector machine based** [97]: Support vector machine is a linear classification method using a kernel function to map training data in a multivariate space. Then, each test instance is checked if it falls into the training region and determines whether it is anomalous.
- **Rule-based** [27]: It learns the rule that captures the system's normal behavior by processing the training data and then considers anomaly the testing data which are not covered by any rule. It works well both for multi-class and one-class settings.

#### 2.2.4.2. Nearest Neighbors Distance-Based

The nearest neighbors distance-based anomaly detection technique [117] checks the number of the neighbors of data, which is defined by a radius. A value is considered an outlier if it does not have enough points in the neighborhood. It can be defined as a region of a size determined by an input parameter around the considered value. This method depends on a Multi-dimensional Index, which controls whether the neighborhood of each data contains enough points to be considered not anomalous. Nearest neighbors distance-based methods also scale well to multidimensional space. A problem can occur if the dataset contains dense and sparse regions because the detection is based on a single value of a custom parameter. The values in dense areas can be classified as normal, and the values in sparse regions can be classified as anomalous increasing the number of false negatives and false positives, respectively.

#### 2.2.4.3. Clustering-based

The clustering-based anomaly detection technique [83] works by grouping similar data into clusters. It is based on the projections of the data into a multidimensional space, and the model, by analyzing the cluster's density, can classify each point as normal or not. There are three different approaches to analyzing the cluster. The first is based on assuming that normal data belongs to a cluster while anomalies do not. The second approach is based on the fact that the more a point far from the cluster centroid, the more is anomalous. Given that, it is assigned to each point an anomaly score. A limit of this approach is that if anomalies are mapped in small clusters, they are not noted. To solve this issue, the third approach is introduced, which analyzes each cluster's density. If data are mapped in small or sparse clusters are considered anomalous instances. On the other hand, normal data are mapped into large and dense clusters. The clustering-based approach works well with unsupervised modality and is fast since, usually, each instance is compared with a small number of clusters. The limit is that it is highly dependent on the clustering algorithms, which are not always optimized for anomaly detection problems. The clustering-based technique may seem similar to the nearest neighbor distance-based one, but they exploit two different aspects. One evaluates each instance concerning the belongingness to a cluster, while the other analyzes each instance concerning its local neighbors.

#### 2.2.4.4. Statistical

The statistical and probabilistic technique [71] is based on modeling data using different distributions and checking how data is probable to belong to the distribution. In other words, it is based on modeling data based on its statistical properties and using this information to estimate whether a test sample comes from the same distribution as the training data or not, being anomalous. According to the technique used, the statistical approach can be classified as parametric or non-parametric. For the parametric technique is assumed that a parametric distribution with known parameters and probability distribution normal data are generated. Then a statistical hypothesis test on the distribution is performed to get the anomaly score of the test instance. Gaussian [90], and Regression [74] model-based are two examples of the parametric technique. Instead, it is called non-parametric, the technique that exploits the nonparametric statistical model. With this technique, the model is not defined a priori but is determined by making some assumptions regarding the data. Examples of non-parametric analysis are histogram-based [55] and kernel function-based [102] models.

#### 2.2.4.5. Ensemble

In addition to being used alone, these techniques presented above can be combined in the technique known as **ensemble** [15]. It is an anomaly detection technique that applies together different algorithms, like predictive or clustering, to classify each data point. Implementing a voting system combines the result of dependent or independent anomaly detection algorithms to obtain a unique result. It can improve the overall success of detection, but according to the methods used, it can significantly increase the complexity and the computational time.

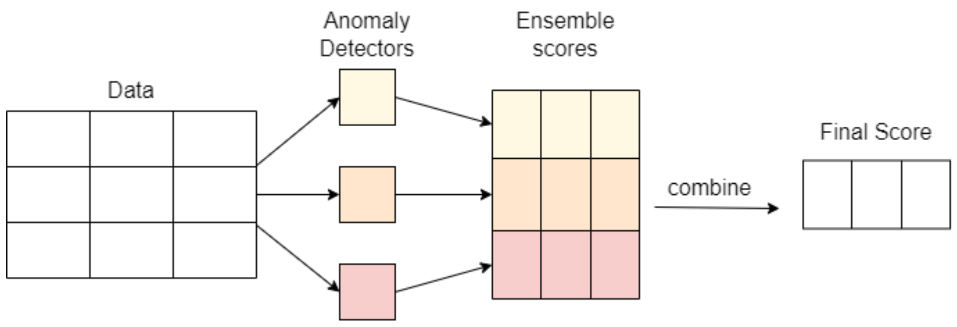


Figure 2.6: Ensembre anomaly detection technique

For example [51], this technique is very powerful in datasets that contain different types of anomalies. It could happen that a technique performs well in finding a certain type of anomaly but cannot find one of another type. Therefore, combining this technique with one that behaves oppositely can improve anomaly detection accuracy.

### 2.2.5. Anomaly Detection Output and Evaluation

The core of anomaly detection algorithms is to detect values that deviate consistently from normal behavior. Even though there are a lot of algorithms and methods for anomaly detection, the output produced could be of two types:

- **Labels**: it consists of assigning a label directly to each test instance, classifying if it is normal or anomalous.
- **Scores**: scoring techniques assign a score to each test instance representing the degree of anomaly. The more the score is high, the more it could be anomalous. Then, by analyzing the score is possible to determine the classification by using a cutoff threshold as a limit value.

Since the score indicates the degree of the anomaly of each sample, it must be evaluated to define if a value has normal or abnormal behavior. So, a good anomaly detection algorithm has to determine an optimal threshold, typically computed on a small portion of the dataset and then applied over the entire dataset. The threshold represents a limit value; the input data is considered anomalous if the score exceeds it. This means that decreasing the threshold value increases the number of positives. On the other hand, by increasing the threshold value, the number of detected anomalies decreases. This value sets the optimal trade-off between false positives and false negatives. There are different techniques to compute a dynamic threshold that works differently according to the distribution and the characteristic of the dataset on which the threshold is computed [111].

### 2.2.5.1. Median Absolute Deviation

The Median Absolute Deviation (MAD) statistic is used in the anomaly detection field to set the threshold. It is a measure of how spread out a set of data is. This method is applied to a validation set of data to set the threshold value before the testing phase. It is essentially based on the **median**, which is the data value in the middle of a list of data ordered in increasing order. The median is less affected by the tail values than the mean. So, it is used instead of the mean when the deviation needs to be less affected by extreme value in the tail.

With the MAD method, the threshold value is calculated by the formula:

$$MAD = 1.4826 \cdot median(|score - median(score)|)$$

$$threshold = median(score) + th_{factor} \cdot MAD$$

Where $MAD$ is the median absolute deviation from the score and the relative median value multiplied by a constant, and $th_{factor}$ is a constant that can assume different values to increase or decrease the threshold. A further clarification is about the constant 1.4826 in the formula. The MAD value can be used similarly to the standard deviation for the mean. To use MAD as a consistent estimator for the estimation of the standard deviation, the correlation is:

$$\sigma = k \cdot MAD$$

Where $k$ is a scale factor depending on the distribution. In this specific case, the scale factor used is the one corresponding to the normal distribution and is defined as:

$$k = \frac{1}{(\Phi^{-1}(\frac{3}{4}))} \approx 1.4826$$

Where $\Phi^{-1}$ corresponds to the inverse of the cumulative distribution function for the standard normal distribution [62].

A possible limitation of the MAD method is caused by the presence of outliers in the dataset on which the method is applied. If there are a lot of outliers, the median is located outside the normal data.

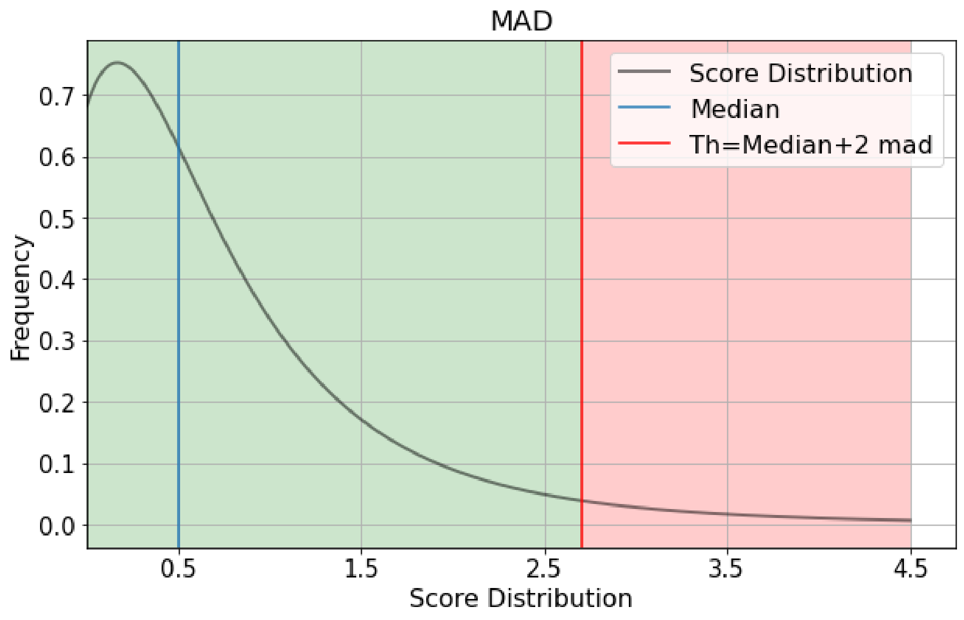


Figure 2.7: MAD threshold method applied on a validation anomaly score distribution

### 2.2.5.2. Inter-Quartile Range

The Inter-Quartile Range (IQR), shown in Figure 2.8, is a common method to find outliers in a data set. Using IQR, the validation dataset is split into four equal parts called quartiles, determined by three different values:

$$Q_1 = quartile(score, 0.25)$$

$$Q_2 = quartile(score, 0.50) = median(score)$$

$$Q_3 = quartile(score, 0.75)$$

The difference between $Q_3$ and $Q_1$ is called Inter-Quartile Range:

$$IQR = Q_3 - Q_1$$

A decision range is defined to detect the outlier using this method, and each point outside this range is considered anomalous. The range is given as follows:

$$LowerBound = Q_1 - th_{factor} \cdot IQR$$

$$UpperBound = Q_3 + th_{factor} \cdot IQR$$

Where $th_{factor}$ is a constant that can assume different values to increase or decrease the threshold. But, given that the score as it is calculated represents how much a value is anomalous and the lower the score is, the lower the probability that the corresponding value is anomalous, it does not make sense to consider the lower bound. So, the threshold is:

$$threshold = UpperBound = Q_3 + th_{factor} \cdot IQR$$

Also, the IQR method is affected by outliers. Problems may occur if Q3 is located within outliers.

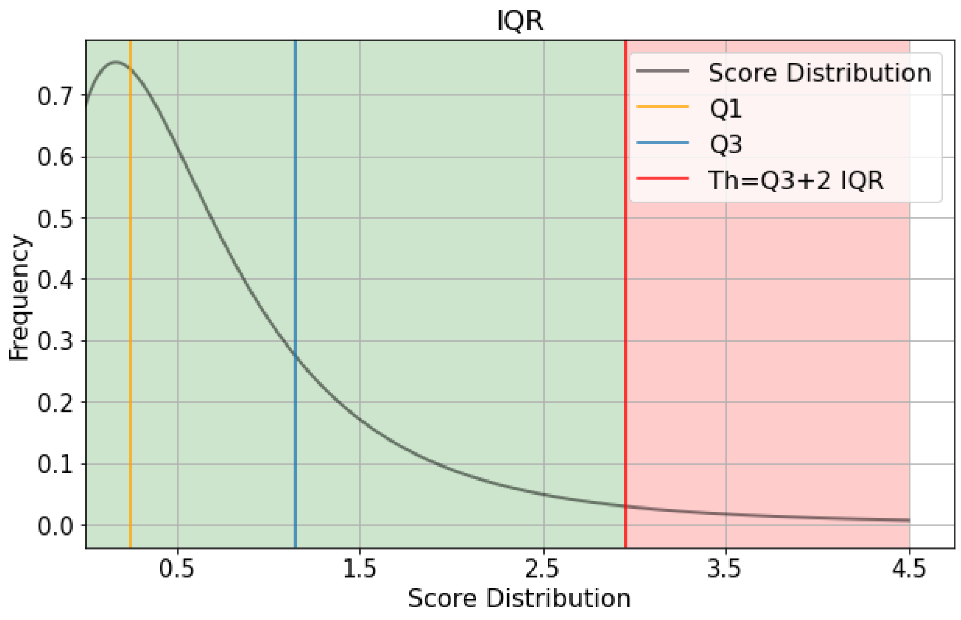


Figure 2.8: IQR threshold method applied on a validation anomaly score distribution

### 2.2.5.3. Standard Deviation

Another method used to find a good threshold in anomaly detection is Mean and Standard Deviation (STD). It consists of calculating the mean and the standard deviation of the score of the validation set. Then the threshold is defined as:

$$threshold = mean(score) + th_{factor} \cdot std(score)$$

Where $th_{factor}$ is a constant that can assume different values to increase or decrease the threshold.

STD method is very sensitive to outliers. Only a few outliers directly affect the mean and the standard deviation value.

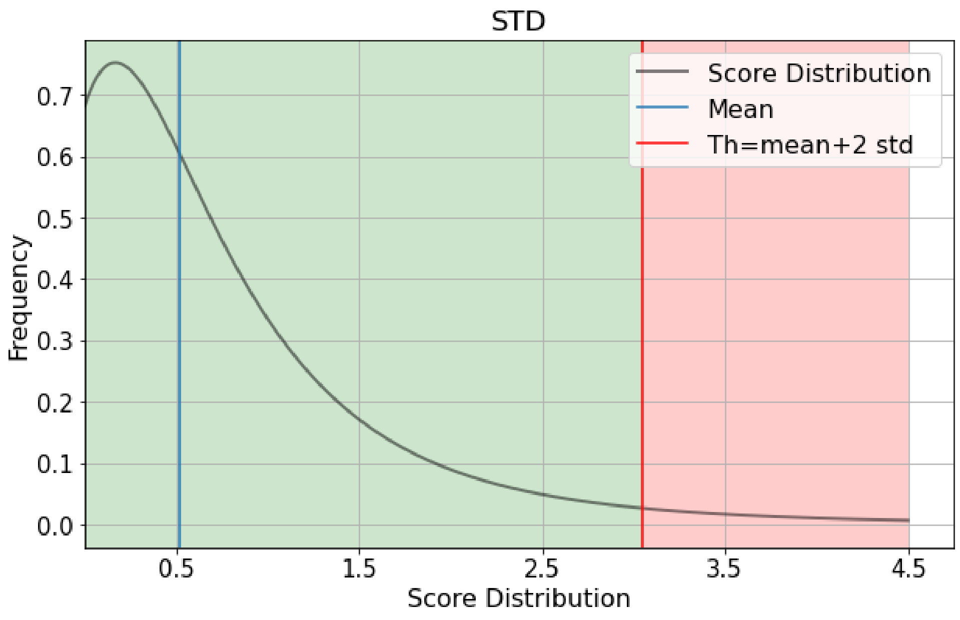


Figure 2.9: STD threshold method applied on a validation anomaly score distribution

#### 2.2.5.4. Max Value

Another intuitive approach that can be used is to set the maximum score value of the validation set as a threshold. The intuition is that since it is composed of only normal data, all the score values of the validation set are acceptable. When this approach is used, it must be checked on the training set. Suppose the validation and training data are very similar, so much so that there is no difference by overlapping them. In that case, it could happen that samples from the training set might be marked as anomalous, which is a contradiction since all the training data are normal. So, before applying the threshold to the test score, there is a check on the training set, and if some anomalous data are detected, the threshold is increased to the max value of the training set.

This technique cannot be used if the dataset on which the threshold is calculated contains outliers. This is because only a single outlier that deviates consistently from other score values generates a threshold too high to evaluate the score effectively.

### 2.2.6. Threshold Related Work

The thresholding techniques just presented are those exploited in the thesis experiments. However, since thresholds are fundamental to evaluating the anomaly score and directly impact the anomaly detection results, several available works propose different techniques.

The work in [112] uses log-likelihood, [10] shows the effectiveness of IQR (Interquartile Range) both in terms of time required for training and ability to detect anomalies when compared with Elliptic Envelope and Isolation Forest. [46] uses IQR (Inter-Quartile Range), MAD (Median Absolute Deviation), and STD (Standard Deviation), as they are among the most used automatic thresholding techniques.

The work in [82], which employs LSTM-based autoencoders for unsupervised anomaly detection, sets the threshold value as the 99.9% quantile of the anomaly scores computed on the test data. The work in [101] uses Graph Neural Networks and LSTMs to consider both the correlation among time series captured from different sensors and the sequential dependency in the temporal dimension. It sets the threshold to the maximum value of the anomaly score computed with an ad hoc validation data set. The study does not specify the thresholding approach used in the compared methods (LSTM-VAE, KNN, and AE). The work in [40] proposes an approach for detecting anomalies in multivariate telemetry time series using LSTMs and a nonparametric dynamic thresholding method that does not assume a specific underlying distribution of the anomaly scores. The method relies on the standard deviation of the smoothed prediction errors and on a single parameter (z), which is set experimentally and shown to have little impact on performances. The work in [64] computes the threshold as the cut-off value leading to the optimal separation between normal and anomalous data in the test set. Usad [7] is another unsupervised anomaly detection approach for multivariate time series data, which selects the threshold as the value that maximizes the F1 score on the test set. The work in [98] exploits the Extreme Value Theory [93] for determining the threshold without resorting to assumptions in the data distribution. Their approach requires tuning two parameters selected empirically, but the effects of their variation are not studied in depth.

## 2.3. Works on Data

Once retrieved, data cannot be used just as they are but have to be checked and transformed to improve the data quality. Therefore, data must be split into three parts to train, validate and test the AI model.

### 2.3.1. Data Preprocessing

After performing different transformations, the data preprocessing phase takes in raw input data and gives output data ready to be split into training, validation, and test set.

It is composed of three different stages:

1. **Data Cleaning**: In this stage, there is a check for missing values, represented by

timestamps with no corresponding values. To solve this issue, there are two different methods[43]:

- *Ignoring missing values*: If the dataset is huge, numerous tuples with missing values can be deleted. Another type of deletion can be done on datasets with multiple features where features characterized by a high percentage of missing data can be deleted. The advantage is to get a more accurate AI model. But, on the other hand, there is a loss of information.
- *Fill in missing values*: Instead of ignoring missing values, another approach is to replace them with other values. To achieve this, many methods exist, such as filling them manually, predicting them using regression methods, or replacing them with the mean, mode, or median. These are commons approach when the dataset size is small and prevent data loss by deleting some timestamps or features. But, it introduces some approximations.

2. **Data Integration**: In this stage, data from multiple sources are merged into a single and larger dataset. Data Integration is not mandatory but is helpful to get a complete dataset.
3. **Data Transformation**: Once the data have been cleaned and integrated, they can be encoded and transformed. First, non-numerical values must be encoded in numerical ones, or columns containing non-numerical data can be eliminated. Then, data could be transformed to facilitate the ML model optimization process and increases the probability of obtaining good results.

   The main methods used for data transformation [4] are:

   - *Normalization*: It is a common approach to re-scale features with a value between two expressed ones, typically between 0 and 1. For each feature, values are transformed according to the formula:

     $$x_{norm} = \frac{x - min(x)}{max(x) - min(x)}(u - l) + l$$

     Where $x_{norm}$ corresponds to the normalized value, $x$ is the value to transform, $min(x)$ and $max(x)$ are, respectively, the minimum and the maximum value of that feature, and $l$ and $u$ corresponds respectively to the lower bound and the upper bound.
   - *Standardization*: It is a common approach to ensure that the mean and the

standard deviation are 0 and 1, respectively. It is calculated as:

$$x_{stand} = \frac{x - \mu(x)}{\sigma(x)}$$

Where $x_{stand}$ corresponds to the standardized value, $x$ is the value to transform, and $\mu$ and $\sigma$ are, respectively, the mean and the standard deviation of the feature values.

The two forms of data transformation have distinct advantages and purposes [94]. Standardization is commonly used when the data to transform fit a Gaussian distribution; if not, normalization is better. Another significant difference concerns the output data. Normalization forces them to be in range, typically between zero and one, while standardization has no range limitation. This is very useful when data with different scales have to be compared end used together. One aspect of preferring standardization over normalization is the presence of outliers in the data; normalization by scaling data to a small range is less resistant to outliers.

### 2.3.2. Data Splitting

To use a dataset in a machine learning model, it has to be split into three parts used to train, validate and test the model.

**Training set** :

It is the data set used to train and make the model learn features and patterns in the data. It is processed in the training phase, where the same dataset is fed to the neural network architecture repeatedly, one time per epoch. The model continues to learn the features of the data and checks how accurately it has learned according to the loss function, a function that computes the distance between the input and the output and indicates how well a model is learning. The training set can be structured in different ways. For Neural networks that have to process images, it is composed of relevant images that best represent the feature that the model needs to learn. On the other hand, for sequential decision trees and random forests, the dataset would be composed of raw data that get classified or processed to extract useful information. In the anomaly detection field, the training dataset comprises only non-anomaly data because the neural network has to learn only how normal data behave.

**Validation set** :

The validation dataset, separated from the training set, is used to validate the neural network model performance during the training. The validation process also gives information on whether the model is learning well during the training or not. In the anomaly detection field, it contains only the non-anomalous data, and usually, on it, the threshold is calculated to evaluate if a value predicted by the model is anomalous or not.

**Test set** :

It is a separate data set used to test the model after completing the training phase. It is composed of data to be processed by the neural network in the way learned during the training phase. In anomaly detection, the test set is composed of both normal and anomalous data.

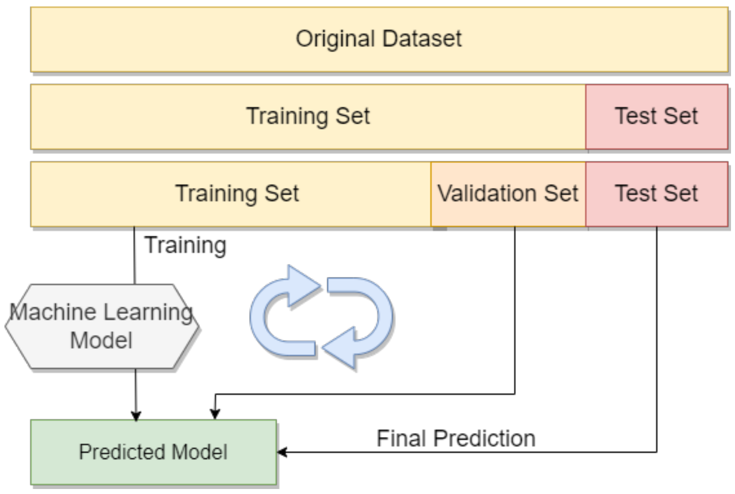


Figure 2.10: How dataset is split

### 2.3.2.1. Problem and Technique

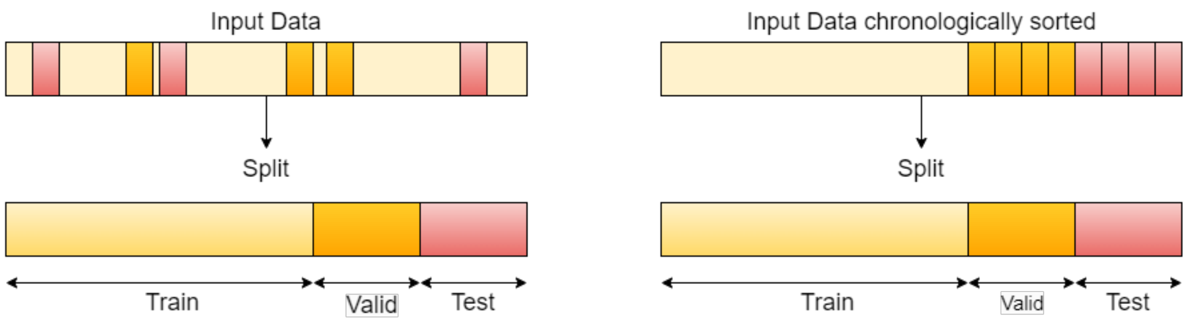
Data splitting is one of the most crucial phases to making a model learn accurately. It is a delicate process, and there is not only one correct way to do it. According to the feature of the data, starting from the entire dataset, there are several techniques [28] to split it into training, validation, and testing sets.

- **Simple random**: Samples from the entire dataset are picked randomly and put in the train, the validation, or the test set according to the ratio set.
- **Stratified random**: Before sampling randomly from the dataset to form the three parts, the dataset is split into small, not overlapping groups based on common

features. Then, data are randomly chosen according to the proportion of the group's size to the entire dataset. Compared with the simple random technique, it has the advantage of representing a more accurate subdivision of the dataset, and it consents to obtain the same results with fewer data in the training set.

- **Temporal**: For each dataset where time is involved, and the scope is to predict something in the future, the technique is to split according to time. Most recent samples are used for the validation and test sets, while the training set comprises the oldest data.

Both simple and stratified random splits are used in image analysis, while on datasets based on time, like time series, the technique used is the Temporal one.



(a) Random splitting (b) Temporal splitting

Depending on the analyzed dataset, you must apply the proper techniques to avoid some problems[56]. The first problem that can be verified is the **overfitting**. It happens when the model does not learn but almost memorizes the training data. The consequence is that the model is very accurate on the training data but inexact on the testing data because it cannot generalize the new data. Furthermore, with overfitting, the model learns random fluctuations or noises in training data instead of the relationship between different features. This noise is a characteristic of the data and, being casual, it does not be learned during the training. The opposite problem is the **underfitting**. It happens when the training set is too poor, and the model does not have enough training data to learn. With underfitting, the model cannot even reconstruct the training data accurately, and consequently, the result is that the model cannot be generalized on test data. In addition to being a data-splitting problem, it could be caused by an overly simple model that cannot learn. Another relevant problem to consider and pay attention to is the **low-quality** of the training set. To improve the model's performance, the quality of the training data is a crucial point. The model cannot perform well if training data is 'garbage'. A slight variation or error in the training set can lead to significant errors in the model performance. This happens because the model learns wrongly, so it cannot generalize the results on the test data.

To solve and attenuate these problems, the choice of the size of the three parts and the data-splitting technique is crucial. There is no unique choice regarding the training, validation, and test size. The percentage of the splitting has to consider the following:

- Computational cost for the training and the evaluation of the model
- Train and test representativeness to mitigate the overfitting and underfitting problems

## 2.4. Result evaluation metrics

This section presents the different techniques used to evaluate how different anomaly detection models detect anomalies. Each technique is based on the confusion matrix which is used to analyze the error made by the machine learning model. The confusion matrix is very useful for evaluating the quality of the classification model. Specifically, it highlights where the model goes wrong, in which instances it responds worse, and which ones are better.

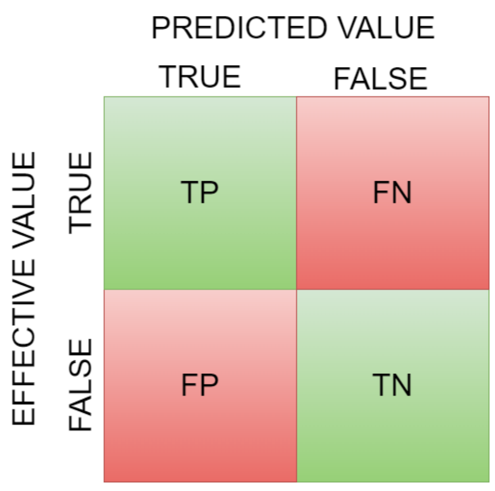


Figure 2.11: Definition of confusion matrix

The confusion matrix is composed essentially of four cells where each cell assumes a value according to the relation between the real value and the predicted one. The four categories are:

- **True Positive (TP)**: refers to a sample belonging to the positive class being classified correctly;
- **True Negative (TN)**: refers to a sample belonging to the negative class being classified correctly;
- **False Positive (FP)**: refers to a sample belonging to the negative class but being classified wrongly because predicted as a positive one;
- **False Negative (FN)**: refers to a sample belonging to the positive class but being classified wrongly because predicted as a negative one;

### 2.4.1. Threshold-independent evaluation

To evaluate how an anomaly detection model is able to detect anomalies independently from the threshold value, the Receiver-Operating-Characteristic (ROC) and the Precision-Recall (PR) curve are used. They both evaluate an anomaly score, obtained by an inference model, to the ground truth value. The ROC curve [78] is a plot that represents the correlation between the false positive rate on the x-axis and the true positive rate on the y-axis for several different thresholds. The true positive rate is defined as:

$$TPR = \frac{TP}{TP + FN}$$

while the false positive rate is calculated as:

$$FPR = \frac{FP}{FP + TN}$$

It is an effective method of evaluating the performance of anomaly detection classification. When analyzing a ROC curve, the interest is placed on the area subtended by the curve itself, called the AUROC. It represents how much a model can distinguish between classes. The higher its value, the better the model can distinguish anomalies from normal values.

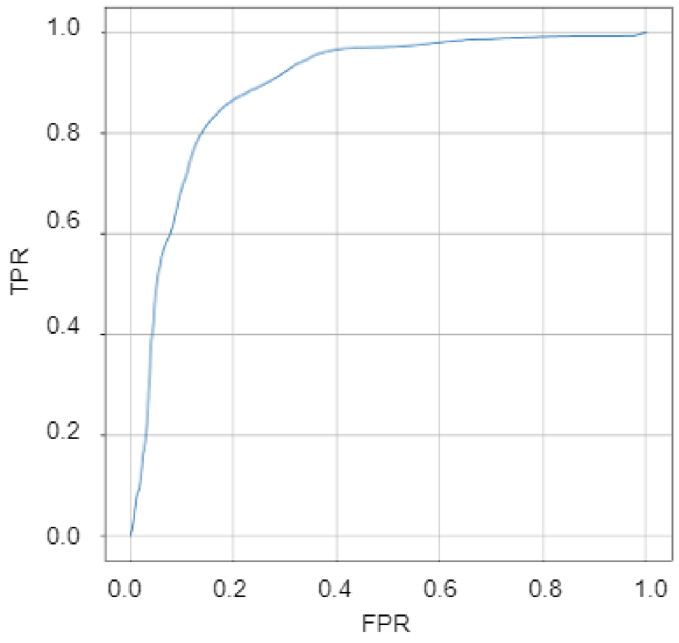


Figure 2.12: Example of ROC curve obtained after processing the test Exathlon dataset [46] with the CONV-AE models. In that case, the AUROC value is equal to 0.89538

Another approach similar to the ROC curve is the PR curve [20]. PR curve is a plot that represents the correlation between the precision on the y-axis and the recall on the x-axis, for different thresholds. The precision is computed as:

$$Precision = \frac{TP}{TP + FP}$$

while the recall is:

$$Recall = \frac{TP}{TP + FN}$$

The precision-recall curve shows the tradeoff between precision and recall for different thresholds. Greater the area under the curve, the better the recall and precision are.

## 2.4.2. Threshold-dependent evaluation

To evaluate how different anomaly detection methods work according to different thresholding techniques, the confusion matrix is computed. It is computed by comparing the ground truth value and the anomaly score after applying the threshold to it. So, it compares the real anomalies to the predicted ones. Starting from the confusion matrix, other different metrics can be calculated to compare the result of the different models.

1. **Precision**: it corresponds to the proportion of true positives to all predicted positive values predicted

$$Precision = \frac{TP}{TP + FP}$$

2. **Recall**: it corresponds to the proportion of the positive value predicted correctly to all the positive presented in the test set

$$Recall = \frac{TP}{TP + FN}$$

3. **Accuracy**: it corresponds to the proportion of samples correctly classified to all samples present in the test set

$$Accurancy = \frac{TP + TN}{TP + FP + TN + FN}$$

4. **F1-score**: it represents the harmonic mean between the precision and the recall

$$F1Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

5. **False Alarm Rate (FAR)**: it is the proportion of false positive samples to all the negative samples in the test set

$$FAR = \frac{FP}{FP + TN}$$

6. **Missing Alarm Rate (MAR)**: it is the proportion of false negative samples to all the positive samples presented in the test set

$$MAR = \frac{FN}{FN + TP}$$

# 3 | Model design

This section presents different types of neural networks, which are the basis of the implementation part of the thesis. Some models are based on autoencoder structures like LTSM-AE, Variational-AE, Convolutional-AE, or USAD, and others, like LSTM or ELM-MI, are based on more straightforward methods.

## 3.1. Long Short-Term Memory

Recurrent neural networks are a particular type of neural network where the output of a step is fed as input in the next one. The problem with standard RNNs is the long-term dependencies that do not allow the RNN to predict the data stored in long-term memory. Long-term memory is used to save into the model the data that happened before the series that is being analyzed. To solve this issue, it is introduced and used the Long Short-Term Memory (LSTM) [65] recurrent neural network. LSTM, thanks to its structure, can retain information about a quite long period and can learn long-term dependencies in sequential data belonging to that period. Due to this feature, they are commonly used for speech recognition, time series forecasting, and language translation.

### 3.1.1. Architecture

To solve the long-term problem of RNN, LSTM has a memory cell that can hold information for an extended period. Three gates control the memory cell: the input, the forget, and the output gate. These gates decide what information to add and remove from the memory cell.

- **Input Gate**: it identifies the essential features and elements that must be added to the memory cell.
- **Forget Gate**: it decides what information contained in the cell memory should be maintained and forgotten.
- **Output Gate**: it is the component in charge of extracting useful information from

the current state to be presented as output.

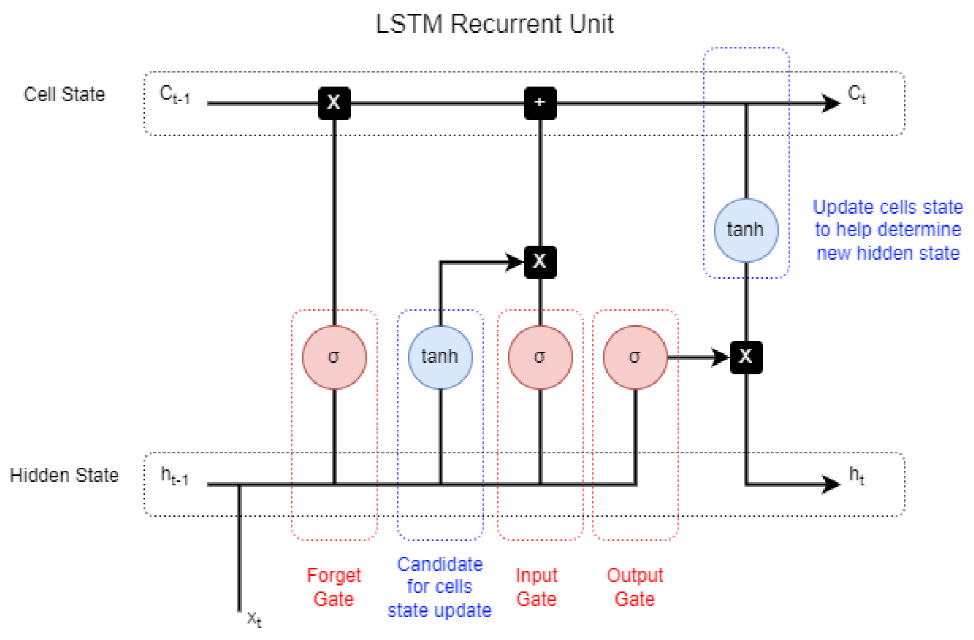


Figure 3.1: LSTM Structure [35]

The LSTM layer introduced before can be used in different configurations as a part of the encoder or the decoder in an autoencoder network or in a simpler structure in a model able to predict a future time sequence starting from a known one. In all the configurations, LSTM layers can be used alone or stacked with others. Stacking LSTM layers makes the model deeper and more accurate. The addition of layers adds levels of abstraction of the input over time. Using a neural network with a single layer can be obtained the same result as a neural network with stacked LSTM layers. To do so, the number of neurons of the single layer has to be increased, but the execution time of the training is increased.

## 3.2. Autoencoder

Autoencoders are artificial neural networks used to learn data in an unsupervised way. Autoencoders aim to learn a lower-dimension representation for higher-dimensional data by training the network to capture the most crucial features of the input data.

### 3.2.1. Architecture

All types of autoencoders have a simple architecture made of three different components:

**Encoder** The encoder is the module that compresses the input data into a representation smaller than the original. Its output is a vector that corresponds to the latent space.

**Bottleneck or Latent Space** The bottleneck is the smallest module composed of the data obtained by the encoding phase. It is the most important part of the network because it contains the compressed knowledge input representation. A compressed representation prevents the network from memorizing the input and overfitting the data. On the other hand, a too-small bottleneck would restrict the amount of information storable, and the input data, encoded, can not be correctly reconstructed by the decoder.

**Decoder** It is the module with the opposite function of the encoder. It takes the compressed data of the bottleneck and reconstructs them into the original input data. Typically, the decoder module comprises the same encoder components in reverse order.

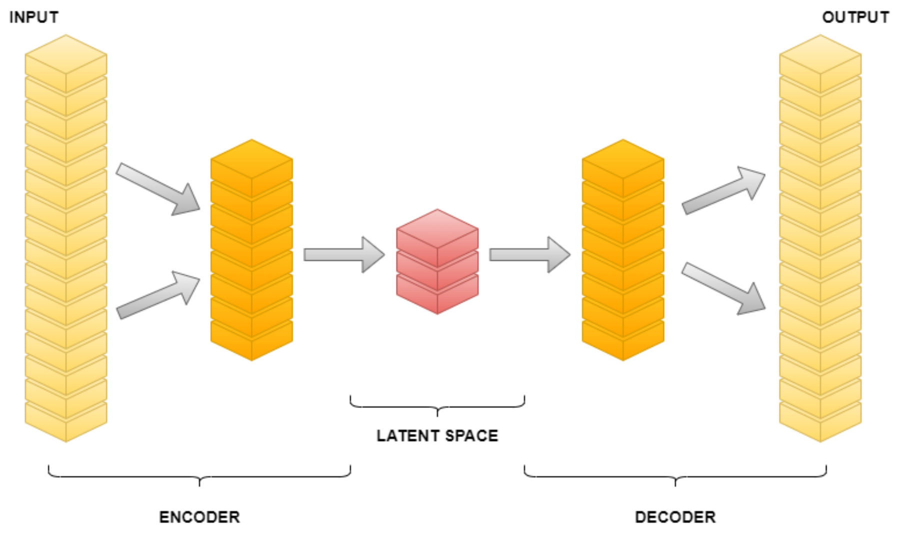


Figure 3.2: Autoencoder architecture representation

### 3.2.2. Autoencoder parameters

Before **training**, the phase in which the neural network learns the model, there are four hyperparameters to define:

1. **Bottleneck size**: The bottleneck size decides how much the data has to be compressed, and it can also act as a regularization term.
2. **Number of layers**: This parameter represents the depth of the encoder and the decoder. A higher depth increases the model's complexity, while a lower depth is

faster to process. A depth model increases the level of abstraction of the data in such a way as to highlight the main features of the input data.

3. **Number of nodes per layer**: According to the number of nodes in the middle, autoencoders can be classified as *Undercomplete Autoencoders* where the dimension of the layer in the middle is less than the input and output one, and as *Overcomplete Autoencoders* where the size of the layers in the middle is greater than the input end output one. Typically in undercomplete autoencoders, starting from the first encoder layer to the latent space, the number of nodes decreases gradually, while the decoder has the opposite behavior.

4. **Loss function**: When data are encoded and then decoded by the autoencoders module, some information is lost. Therefore lossy is a crucial feature of autoencoders, and one of the training scopes is to reduce the loss of each epoch according to the loss function. The loss function corresponds to the reconstruction loss and represents the input and output data differences. Typically, it can be implemented in two different ways according to the application.

    - Binary Classification loss function: Is used in predictive modeling problems where data are assigned one of two categories, and the scope of the function is predicting the probability of the data belonging to one of two categories. The common function used is binary cross-entropy.

    - Regression loss function: Is used in a regression predictive modeling problem that involves data prediction. Typically it consists of comparing the original data with the reconstructed one. There are a lot of different implementations [48] as:

        – Mean Absolute Error (MAE): it is the absolute difference between the original data and the predicted one defined as:

        $$loss = MAE = \frac{1}{N}\sum_{i=1}^{N}|x_i - y_i|$$

        where $x_i$ is the original data, $y_i$ is the prediction, and $N$ is the number of samples. It is used due to its simplicity and because it is less sensitive toward outliers. Conversely, a disadvantage is that all errors are considered equally because of the meaning calculation.

        – Mean Squared Error (MSE): is the squared difference between the input and the reconstructed data. It is defined as:

$$loss = MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$

where $x_i$ is the original data, $y_i$ is the prediction, and $N$ is the number of samples. It is a method used because it is sensible to outliers, but a disadvantage is its quadratic component. If the model makes a single bad prediction, with $x_i - y_i$ greater than 1, the quadratic part of the function amplifies the error.

## 3.3. Variational Autoencoder

Variational Autoencoder (VAE) is a particular type of autoencoder created to improve its effectiveness and add new features as the generative characteristic of the latent space. Autoencoder generally has a non-regularized latent space that can be used only by the decoder to reconstruct the input data. In a VAE, instead of outputting the vector in the latent space, the encoder gives in outputs two different vectors representing the mean and the variance of the distribution of the latent space. VAE, by sampling from mean and variance and with its loss function, forces the latent space to be normally distributed in a regular and continuous way

### 3.3.1. Architecture

The structure of a VAE is the usual architecture of an autoencoder. An encoder, a decoder, and the latent space make it. The decoder works similarly to the one in AE; it takes the content of the latent space and reconstructs it into the input. The relevant differences concern the latent space and the encoder. The latent space is not the output of the encoder; it is obtained by sampling from the mean and the standard deviation obtained by the encoder. The encoder works similarly to the AE, reducing the input size, but it does not return directly in the output the latent space.

### 3.3.2. Loss Function

The loss function of a VAE is different from a standard autoencoder one. It is composed of two components:

$$loss = ReconstructionLoss + SimilarityLoss$$

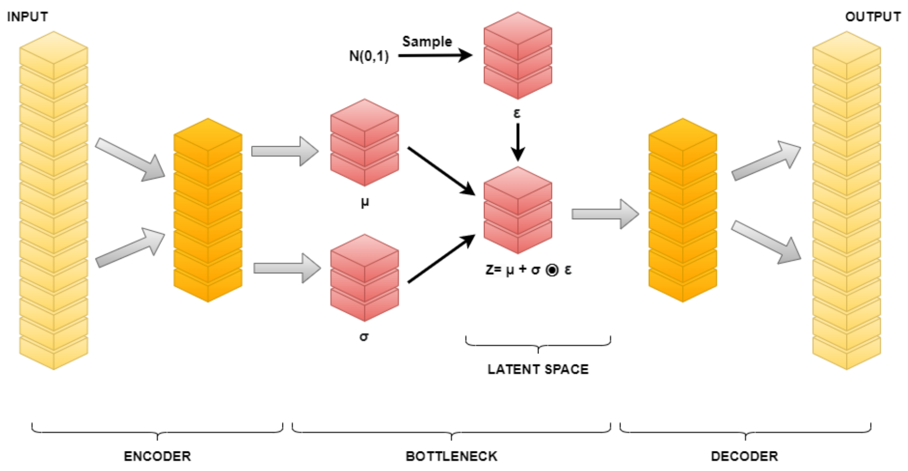


Figure 3.3: VAE structure representation

The reconstruction loss corresponds to the loss of a standard autoencoder and represents the differences between the input and the output data. Less is its value, and the more the network has learned. The similarity loss is a significant parameter introduced in VAE. It corresponds to the Kullback-Leibner divergence between the latent space distribution and a standard Gaussian with zero mean and unit variance. Thanks to this function and another, called the reparameterization trick, the latent space is normally distributed, smooth and continuous. The **Kullback-Leibner divergence** or **KL divergence** is a type of statistical distance. It measures how a probability distribution is different from another.

**KL divergence Definition:** For continuous probability distribution $P$ and $Q$ defined on the same probability space, $X$, KL divergence is defined as [116]:

$$D_{KL}(P||Q) = \int_{x \in X} P(x) log\left(\frac{P(x)}{Q(x)}\right) = -\int_{x \in X} P(x) log\left(\frac{Q(x)}{P(x)}\right)$$

And the probability density function of Normal multivariate distribution is:

$$p(x) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(\mu_1 - \mu_0)^T \Sigma^{-1}(\mu_1 - \mu_0)\right)$$

So, by applying the KL divergence definition to two normal distributions, $N(\mu_p,\Sigma_p)$ and $N(\mu_q,\Sigma_q)$, it is obtained:

$$D_{KL}(p||q) = \mathbb{E}_p[log(p) - log(q)]$$

$$= \frac{1}{2}\mathbb{E}_p(log\left(\frac{|\Sigma_q|}{|\Sigma_p|}\right) - (x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p) + (x-\mu_q)^T\Sigma_q^{-1}(x-\mu_q))$$

$$= \frac{1}{2}\mathbb{E}_p(log\left(\frac{|\Sigma_q|}{|\Sigma_p|}\right) - \frac{1}{2}\mathbb{E}_p((x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p)) + \frac{1}{2}\mathbb{E}_p((x-\mu_q)^T\Sigma_q^{-1}(x-\mu_q))$$

$$= \frac{1}{2}log\left(\frac{|\Sigma_q|}{|\Sigma_p|}\right) - \frac{1}{2}\mathbb{E}_p((x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p)) + \frac{1}{2}\mathbb{E}_p((x-\mu_q)^T\Sigma_q^{-1}(x-\mu_q))$$

Then, since $(x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p)$ in the second term $\in \mathbb{R}$, it can be written as: $tr(x-\mu_p)^T\Sigma_p^{-1}(x-\mu_p)$ where $tr\{\}$ is the trace operator. Now it is obtained that:

$$= \frac{1}{2}tr\{\mathbb{E}_p[(x-\mu_p)^T(x-\mu_p)]\Sigma_p^{-1}\}$$

$$= \frac{1}{2}tr\{\Sigma_p\Sigma_p^{-1}\} = \frac{1}{2}tr\{I_k\} = \frac{k}{2}$$

By combining all this is obtained:

$$D_{KL(p||q)} = \frac{1}{2}\left(log\left(\frac{|\Sigma_q|}{|\Sigma_p|}\right) - k + (\mu_p-\mu_q)^T\Sigma_q^{-1}(\mu_p-\mu_q + tr\{\Sigma_q^{-1}\Sigma_p\}\right)$$

In the VAE case, KL divergence is applied to a standard normal distribution (with zero mean and unit variance) and the normal distribution with mean and variance calculated by the network encoder. So, the KL divergence becomes calculated as follows:

$$D_{KL(p||N(0,I))} = \frac{1}{2}\left[\mu_p^2 + \Sigma_p^2 - k - log|\Sigma_p|\right]$$

## 3.3.3. Reparametrization trick

The VAE calculates parameters by sampling from a distribution with mean and variance calculated by the encoder network. This process is not differentiable; in other words, each network component must be deterministic because it can cause problems during the backpropagation. This issue is easily solved by applying the reparameterization trick to the sampling phase used to generate the latent space. The reparametrization trick treats the random sampling as a noise term sampled from a Gaussian distribution with zero mean and unit variance. So, the noise term is independent and not parameterized by the model. According to the reparameterization trick, the diagram changes in this way 3.4:

When $z$ is sampled stochastically from the parameterized distribution, the propagation flows through a stochastic node. Reparameterization allows a path through deterministic

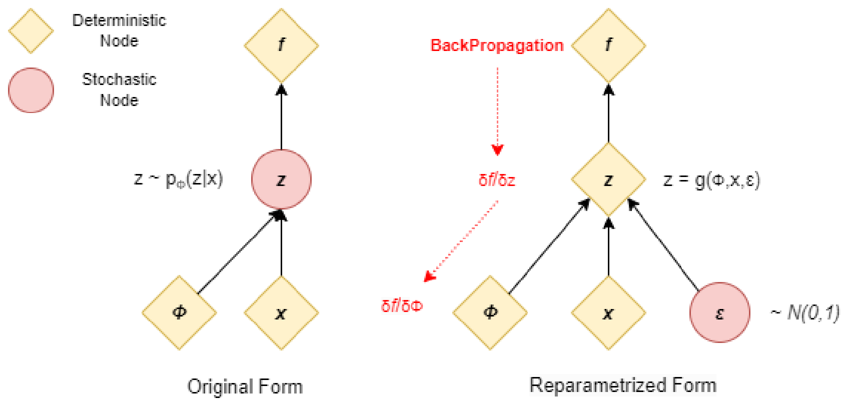


Figure 3.4: Reparameterization trick effect

nodes and relegates the stochastic component to a noise vector separated from the flows. It works thanks to a property of Gaussian distribution that allows scaled sampling from a Gaussian with zero mean and unit variance to a distribution with other mean and variance. $z$ is obtained by:

$$z = \mu + \sigma \odot \epsilon$$

where $\epsilon \sim N(\mathbf{0},\mathbf{1})$ and $z$ is equivalent to $\tilde{z} \sim N(\mu, \sigma)$.

## 3.3.4. Latent Space

The latent space is the core of a VAE. To make generative processes possible, the regularity of the latent space can be expressed through two main properties:

- **Continuity:** It means that two close points in the latent space, when decoded, should not give two completely different contents but something with some features in common.
- **Completeness:** When decoded, z random value sampled from the latent space generates a meaningful output.

To ensure continuity and completeness, it is not sufficient that the decoder input is a distribution instead of being simple points. Furthermore, without a well-defined regularization term, the model behaves almost like a standard autoencoder. To ensure these two properties, the covariance matrix and the mean of the distribution returned by the encoder must be regularized. The regularization is done by enforcing distributions to be closer to a normal one with zero mean and unit variance. In this way, the covariance matrices are closed to the identity, preventing punctual distribution, and the mean is close to 0, preventing points from being too far apart from each other.

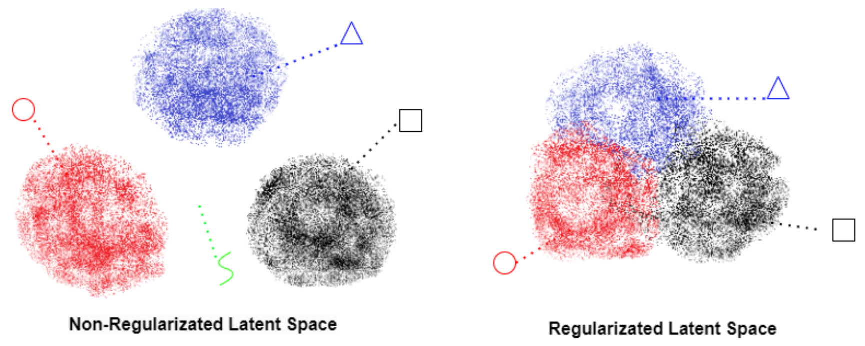


Figure 3.5: Latent space view without and with regularization

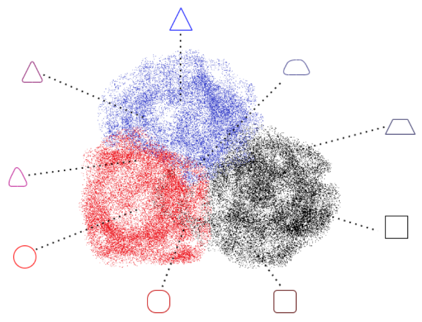


Figure 3.6: Relationship between neighbor points in the latent space

Moreover, continuity and completeness, obtained with regularization, tend to create a relationship over the information contained in the latent space. In other words, a point of the latent space between the mean of two encoded distributions should be decoded in something that is somewhere between the data encoded in the two distributions.

### 3.3.5. Difference between AE and VAE

Simple AE and VAE, being both autoencoders, have the same structure. They both be composites with an encoder, a decoder, and the latent space. The only difference is how the latent space is obtained and its properties.

| **AE** | **VAE** |
|---|---|
| • The latent space is composed of a compressed transformation of the input data encoded. Each input data has a deterministic mapping into the latent space. This means that different training on the same dataset always generates the same latent space. | • The latent space obtained by sampling the mean and the variance is determined in a non-deterministic way. This means that different training on the same dataset can create a different latent space shape |
| • The latent space being not regularized is not complete. This means that it has some discontinuity. | • The latent space does not have any discontinuity, so it is continuous and smooth. This means that data are not encoded in separated and distant clusters corresponding to their feature, but the cluster in which they are encoded are near and intersect. In the intersection of the cluster are encoded the data with mixed features, while in the center of each cluster are encoded data with 'pure' features. |
| • The latent space is not able to generate new output. This means that only data encoded into the latent space can be decoded, and a random sample from the latent space generates a non-significative output. | • Due to the feature of its latent space, the most significant difference is the generative ability. This means that a random sampling decoded from the latent space generates a new output unrelated to an input sequence. |

### 3.3.6. Variational Autoencoder approaches

Due to how they manage the latent space, Variational Autoencoders, compared to other autoencoders, can be used in the anomaly detection field using four approaches.

- Reconstructed value: it is the only approach that can be used for all autoencoders.

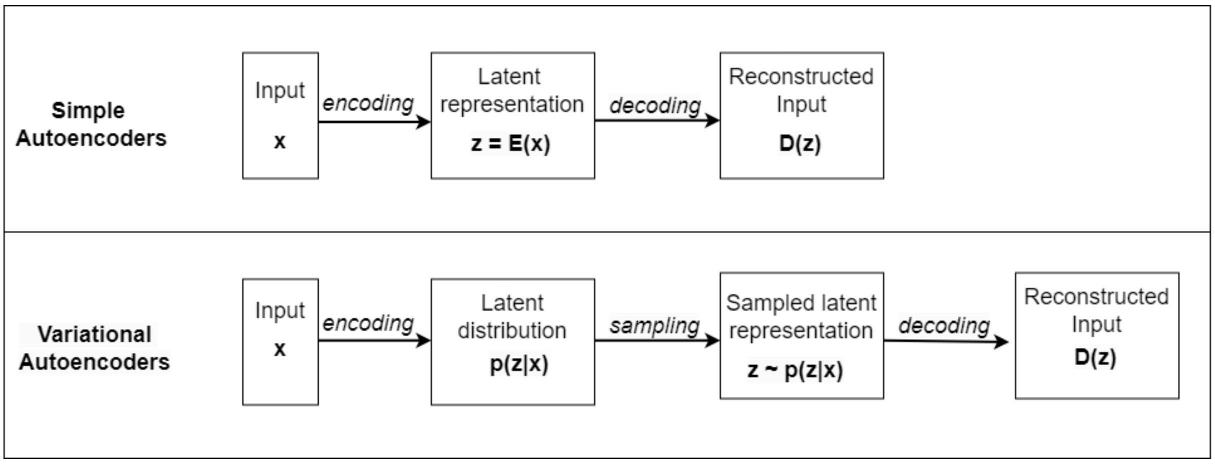


Figure 3.7: Difference between AE (deterministic) and VAE (probabilistic)

It compares the ground truth to the processed and reconstructed data and detects anomalies according to the comparison results and the threshold.

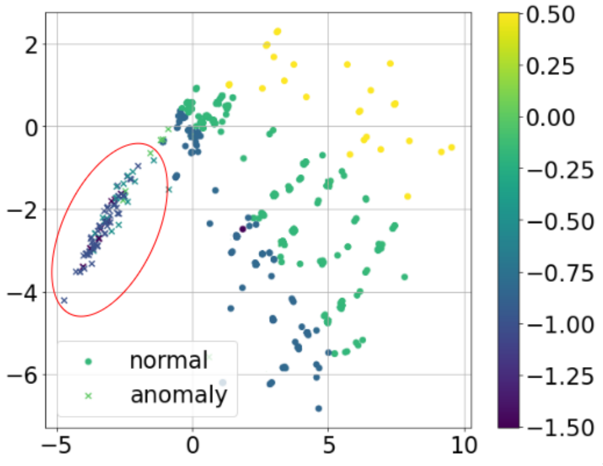
- VAE combined with K-nearest neighbors: it is an approach that analyzes the latent space distribution with the K-nearest neighbor method [34].
- VAE combined with isolation forest: it is an approach that analyzes the latent space distribution with isolation forest [67].
- VAE with re-encoding: it is an approach that compares two different latent space distributions. The first is obtained by a single encoding, while the second is obtained after an encoding-decoding-encoding process.

The following sections present each method concerning the latent space in depth.

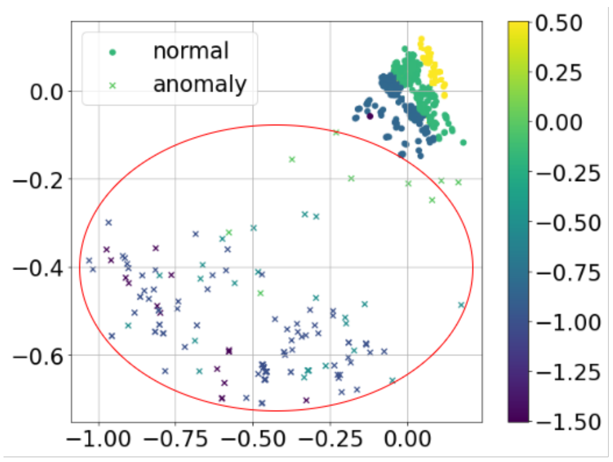
#### 3.3.6.1. Latent space methods

VAE can detect anomalies by analyzing the latent space distribution, while non-variational AE cannot. This is because of the different characteristics of the two models described in Section 3.3.5. VAE latent space, contrary to an AE one, is regularized, smooth, continuous, and has generative properties. To better understand these differences, it is necessary to analyze and compare the latent space distribution obtained by the two algorithms.

As Figure 3.8 shows, the scatter plots of the two latent spaces obtained using VAE and AE have very different shapes. They both represent the same encoded time series from the SKAB dataset, which contains normal and anomalous data. For a better comparison, samples taken from the time series with the same value are represented in the plots with the same color. In addition, the two plots represent anomalous samples using crosses and



(a) AE latent space distribution



(b) VAE latent space distribution

Figure 3.8: The two figures compare the latent space of a simple AE to the latent space of a VAE. The color represents the value of each sample before being processed by the model. The shape of each point corresponds to its classification: the cross represents the anomalies, while the bullets are the normal ones. They are obtained by processing the same time series taken from the SKAB dataset [52]

normal samples using points. The first difference between the two plots is the distribution of the points. For VAE, normal points are mapped compactly, assuming a distribution with zero mean and unit variance, and the more the input data are similar, the more are mapped closer. Instead, anomalous points are mapped sparsely and are easy to isolate and detect. Concerning the AE latent space, it is immediately noticeable that it is non-regularized. As a result, normal points are mapped sparsely and more distant compared to the normal points in the VAE latent space, with the consequences explained in Section 3.3.4. The principal feature that allows anomaly detection by analyzing the latent space is where anomalous and non-anomalous data are mapped. AE latent space, normal data take up all the space, and outliers are mapped together, making it more difficult to efficiently apply algorithms based on density distribution or nearest neighbor distance. On the other hand, in VAE latent space, all normal data are mapped close together, being similar data, and anomalous data are mapped far away from that dense region. For this reason, anomalies can be separated better from normal points.

#### 3.3.6.2. VAE and KNN

The K-nearest neighbors (KNN) [34] is a non-parametric classification method. It can be applied to identify anomalies directly on the time series or on preprocessed data. In this case, it is used to analyze the latent space generated by the encoder to detect outliers.

Using it with VAE is very effective since the encoder is able to amplify the difference between normal and anomalous data. Given a sample to be classified, the algorithm forms its neighborhood by retrieving its $k$ nearest neighbors. Then a majority voting system is used among data in its neighborhood to classify it. To define the concept of the nearest neighbor, it is important to define a distance metric. A metric [22] must respect four properties:

- non-negative: $d(x, y) \geq 0$
- identity: $d(x, y) = 0$ if and only if $x = y$
- symmetry: $d(x, y) = d(y, x)$
- triangle inequality: $d(x, z) \geq d(x, y) + d(y, z)$

where $x$, $y$ and $z$ represent three different points, and $d(x, y)$ refers to the distance between two points. The distance metrics most used in KNN are the Euclidean and the Manhattan distance because they scale well to multiple dimensions [60]. In detail, they are defined as:

- **Manhattan distance** [21]: it is the sum of the absolute difference between points across all dimensions, and it is calculated as:

$$D(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

  Where $x = (x_1, x_2, .., x_n)$ and $y = (y_1, y_2, .., y_n)$ are two points in the $n$-dimensional space.

- **Euclidean distance** [110]: it is the shortest distance between two points calculated as:

$$D(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

  Where $x = (x_1, x_2, .., x_n)$ and $y = (y_1, y_2, .., y_n)$ are two points in the $n$-dimensional space.

The number of neighbors composing the neighborhood is based only on the parameter $k$, so the choice of its value is fundamental for accurate classification. A possible option is to run the algorithm many times with different $k$ values and choose the one corresponding to the best results. In the anomaly detection field, based on historical data, the $k$ value can be stable whether the process is subject to anomalies or whether the anomalies are rare. So, according to the percentage of anomalies in the entire dataset is possible to set the

$k$ parameter. Namely, if the anomalies are rare, the number of neighbors will be greater than the case with several anomalies [100]. KNN categorizes test samples according to

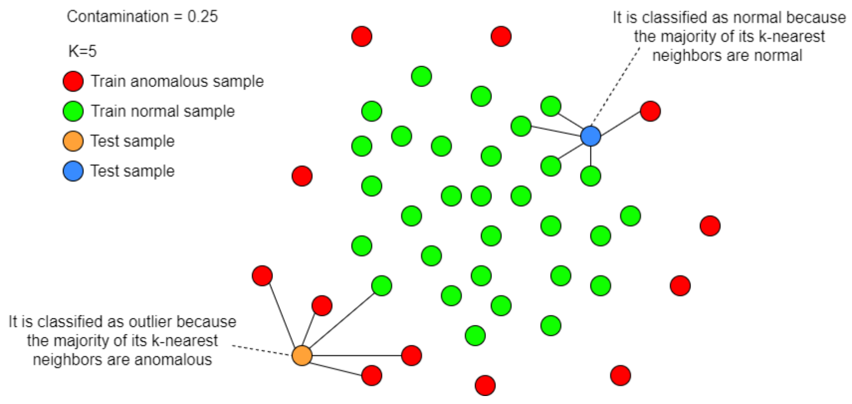

Contamination = 0.25
K=5
Train anomalous sample
Train normal sample
Test sample
Test sample
It is classified as normal because the majority of its k-nearest neighbors are normal
It is classified as outlier because the majority of its k-nearest neighbors are anomalous

Figure 3.9: The figure represents the training data distribution represented by the red and green points and two test samples, colored in blue and orange, which have to be classified. The color of the training points is chosen according to the *contamination* parameter, which determines the percentage of training data to be considered anomalous. Concerning the test sample, their k-nearest points are retrieved to classify them.

the training set. For classification, the training set is composed of data belonging to different categories known in advance. This method works well for supervised anomaly detection, where the training data are labeled anomalous or normal. However, a slightly different approach must be used in unsupervised anomaly detection, where the training data comprises only normal data. Therefore, it is necessary to indicate a *contamination* term showing the percentage of training data to be classified as anomalous, and it is used as a threshold. The more the *contamination* value is small, the higher the threshold is, and fewer anomalies are detected. Analyzing the 3.9 is useful to understand this concept better. The figure contains the training data distribution with points colored in green and red and two test points that have to be classified. According to the *contamination* [34] value, the training data are divided into anomalous, colored in red and mapped at the edges of distribution, and into normal, colored in green. When a test sample has to be classified is compared with its k-nearest neighbor. The test point colored in orange is classified as anomalous since the majority of its k-nearest neighbor are anomalous, while the test point colored in blue is normal, being most of his neighbors normal.

### 3.3.6.3. VAE and Isolation Forest

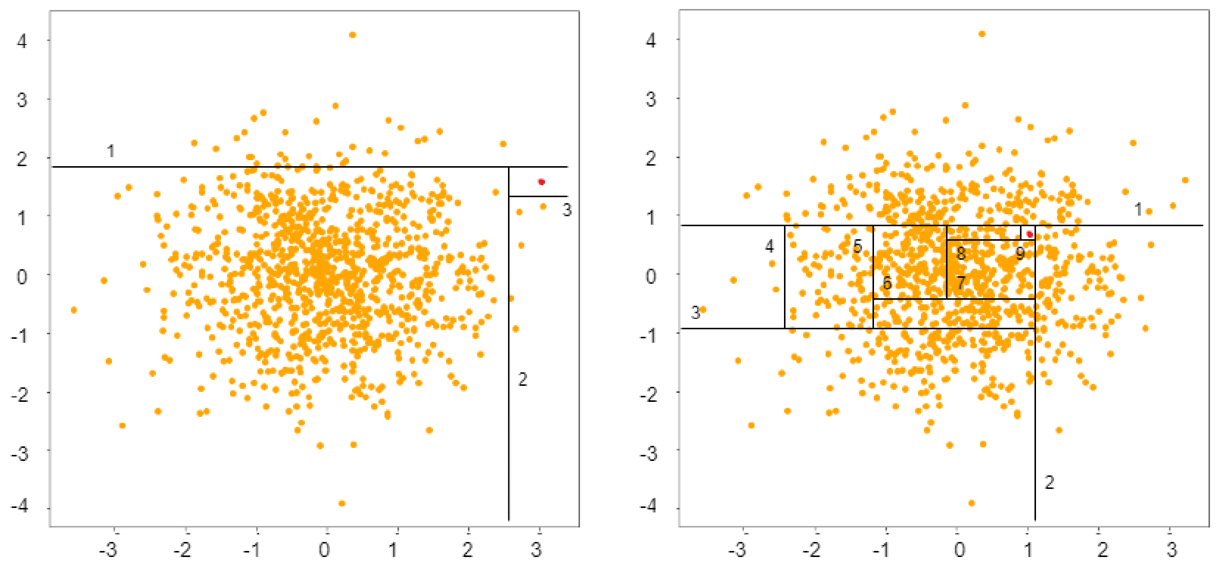
Instead of KNN, a latent space analysis can be performed using Isolation Forest [67]. After the encoding phase, it is generated the latent space contains normal data in dense regions and anomalies in sparse regions. Also, Isolation Forest is an effective method that can be applied directly to the time series. However, since the difference between normal and anomalous data is amplified after being processed by an encoder, Isolation Forest is more effective for analyzing the latent space.

Isolation Forest is an unsupervised decision-tree-based algorithm. The concept behind Isolation Forest is to separate a value from the rest of the distribution. So, it adapts well to anomaly detection because outliers, being few and different, are easier to isolate.

The process used to isolate each point is called random partitioning. First, it divides the distribution, which contains the samples to evaluate, into two parts. Then each part is recursively divided until all points are separated. In that way, a tree is generated, its root corresponds to the initial distribution, its nodes to the partitioning, and the leaves to the isolated point. The partitioning is called random because the value to perform the splitting is casual and different execution generates different trees. By analyzing the tree, the result is that outliers are more likely to be isolated in early partitions, so the probability that points in the tree with shorter paths are anomalous is higher. On the other hand, more random splittings are necessary to isolate normal points because they are located in dense regions. Thus, they are inserted in the tree with a long path from the root.

Being a random splitting process, to obtain consistent results, the tree must be computed several times, and the path length of each point corresponds to the average one over the number of trees. The Isolation Forest process comprises two phases: fitting and prediction. The fitting stage takes as input the training set to create the distribution, which is used to analyze the test instance in the prediction phase. The prediction phase generates the anomaly score for each test instance, corresponding to the average depth level of the tree in which the corresponding point is isolated. To evaluate the score, a threshold is necessary. It is defined by a *contamination* parameter which indicates the proportion of outliers in the dataset and is used to set the depth of the tree over which a point is normal. If the depth in which a test sample is isolated is greater than the limit set by the threshold, the point is classified as normal. On the other hand, it is an anomaly.

Isolation Forest has some advantages compared to KNN. It utilizes no distance or density measures to detect anomalies reducing the computational costs [67]. Isolation forest has a linear time complexity since the execution time increases linearly with the input size [109], and low memory requirements are necessary to obtain the same result of distance and density models. Moreover, Isolation Forest can scale well, handling large datasets and high-dimensional problems.



(a) Isolation of an anomalous point (b) Isolation of a normal point

Figure 3.10: The two plots represent the same scatter plot of samples from a normal distribution. The horizontal and vertical lines are the random split performed by Isolation Forest, and the red points represent the isolated points.

### 3.3.6.4. VAE and Re-encoding

VAE with re-encoding is the last anomaly detection method based on the Variational Autoencoder. It exploits the concepts presented in [114]. Usually, the anomaly detection methods base on autoencoder models exploits the reconstruction error. First, they compare the ground true value with the same value processed and reconstructed by the autoencoder model generating a score. Then they compare the score with a limit value to detect anomalies. The VAE with re-encoding is based on a completely different approach. It compares two latent spaces, one obtained after encoding the input sequence and the other obtained by encoding, decoding, and re-encoding it.

**Intuition:** Analyzing a sequence reconstructed by a VAE is necessary to understand why this model works well. In unsupervised anomaly detection, a neural network model is trained with a training dataset that not contains anomalies. So, the model learns the normal behavior of the time series and can reconstruct the test sequence in two different ways. If the test sequence represents normal behavior, the model accurately reconstructs the sequence. In contrast, if the test sequence represents an unexpected behavior, the result is that the reconstruction deviates significantly from the input sequence. In that way, starting from an input sequence $x$ and processing it with the VAE, a new sequence

$x'$ is obtained in output. $x'$ is similar to $x$ for non-anomalous sequences and deviates consistently for anomalous sequences. Then, applying an encoding to $x'$, a re-encoded latent space is obtained. By comparing it with the latent space, anomalies can be found. Normal data with a similar value in $x$ and $x'$ are encoded closer, while outliers, having a different value in $x$ and $x'$, are mapped in a different position amplifying their diversity.

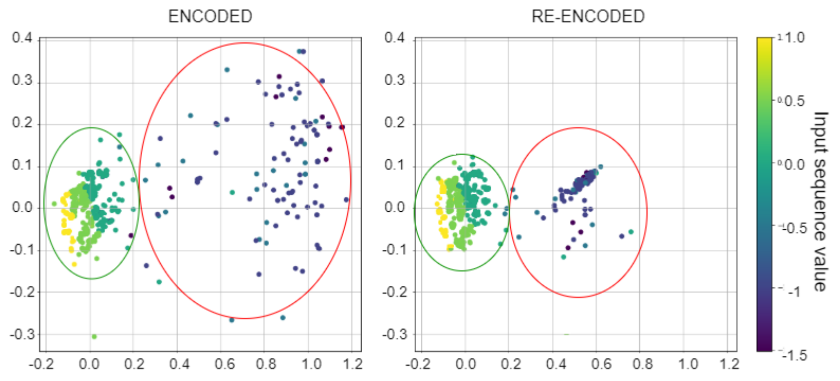


Figure 3.11: The figure compares two latent space distributions according to how they are computed. The left plot represents the latent space obtained after the encoding phase, while the right one is obtained after a re-encoding. Normal values circled in green are encoded similarly in the two latent spaces. Instead, the two latent spaces' anomalous values circled in red are mapped differently. The figure is obtained by analyzing a file from the SKAB dataset [52]

**Model and score definition:** The model used is a VAE. It comprises an encoder that gives the mean and the variance vector as output, a decoder that decodes the latent space data, and the latent space obtained by sampling from the encoder output.

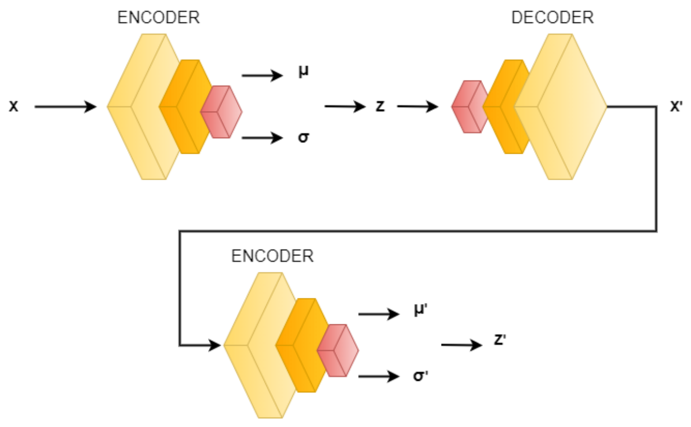


Figure 3.12: The structure of the network and the different steps to which each test sequence is subjected to obtain each variable of the score function.

First, the input sequence $x$ is encoded to obtain the $\mu$, $\sigma$ vector, and the latent space $z$. Then the latent space is decoded to obtain the reconstructed data $x'$. The last step is to process $x'$ throw the same encoder used before getting $\mu'$, $\sigma'$ vector, and the reconstructed latent space $z'$.

In that way, what remains to be done is to define the score function. It can be defined as:

$$score(x) = \alpha||x - x'||_2 + \beta||z - z'||_2$$

Where $\alpha$ and $\beta$ are two positive parameters that summed are equal to one and indicate which part of the score is predominant. In the case of anomaly detection based on reconstruction loss, $\beta$ is equal to zero to minimize the contribution of the re-encoded latent space, and $\alpha$ is one. On the other hand, if $\beta$ is one, by definition, $\alpha$ must be zero, and the score corresponds only to the mean squared error between the latent space and the re-encoded one.

## 3.4. Convolutional-Autoencoder

A Convolutional Neural Network (CNN) is a particular neural network comprising one or more convolutional layers designed for processed data arrays such as images. CNNs are commonly used in computer vision to process and classify images but have succeeded in natural language processing and time series analysis. CNNs are very powerful in image processing since they are very good at picking up patterns in input images. The power of Convolutional neural networks comes from the convolutional layers that, stacked on each other, can recognize the specific feature of the input. A common configuration of CNN is in autoencoder shape using convolutional and deconvolutional layers.

### 3.4.1. Architecture

CNNs are composed of convolutional layers. The convolutional layer is visualized as kernels, which slide over the input images and try to find patterns. Where the part of the images matches the kernel's pattern, it returns a large positive value; conversely, if it does not correspond, the kernel returns zero or a smaller value. The CNN comprises a series of these layers, an activation function, and a downscaling convolutional layer repeating many times. Repeating this combination, the network can detect more complex patterns. In the anomaly detection field, the convolutional layers can be used in Convolutional Autoencoder for Image Noise Reduction. The anomalous data are seen as noise, so the model can detect anomalies by comparing the input and the denoise images in the output.

The convolutional autoencoder implemented comprises two convolutional layers and two deconvolutional layers to reconstruct the denoised output image.

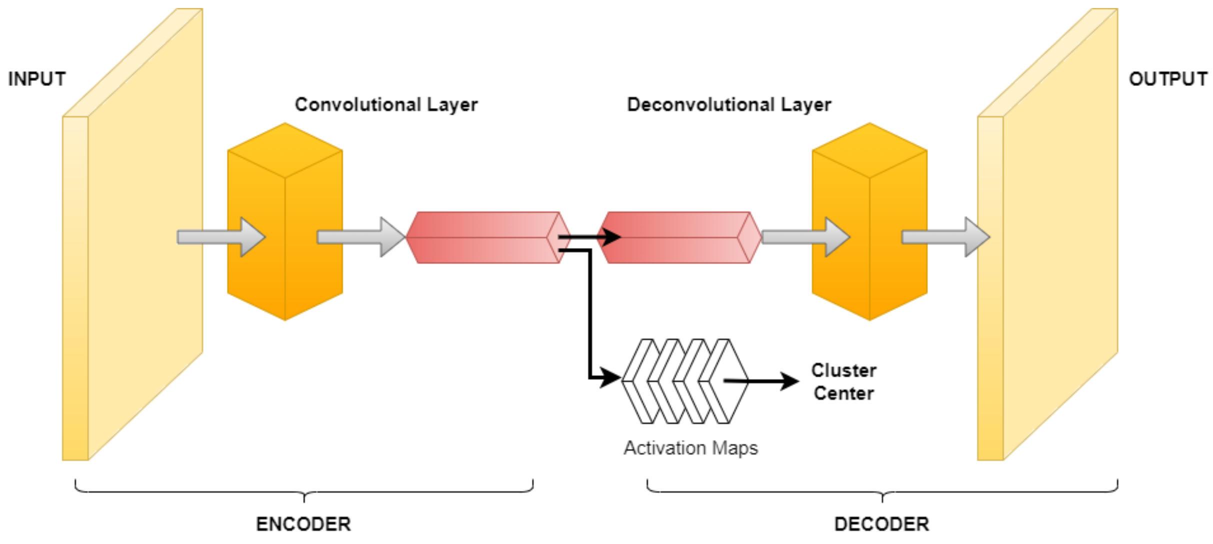


Figure 3.13: Convolutional Autoencoder structure

Usually, the convolutional autoencoder works on images where the shape of the input is $H \times W \times C$ where $H$ is the height, $W$ is the width, and $C$ is the number of channels of the images. To adapt the model to a time series analysis, the input has to be a single time instant of the time series. So, in the input shape, $H$ corresponds to the window size, $W$ is the number of features of the multivariate series, and $C$ is equal to one.

### 3.4.2. CONV-AE vs. LSTM-AE

LSTM and CNN were initially used in different fields; one was created to process data dependent on time, while the other was designed to process static data, such as images, not reliant on time. New implementations point out that CNN can achieve LSTM results, namely predicting sequences, but in a much faster and more computationally efficient manner [107]. An LSTM model is designed to work differently than a CNN and is used to process time-dependent data and make predictions given data sequences. In contrast, CNN is used to exploit spatial correlation in data, so it works well on images. Both models, especially in an autoencoder configuration, can analyze time series focusing on different aspects. The LSTM models are focused on the dependency of data on time. In contrast, due to its characteristic, CNN focuses on analyzing separated and time-independent images exploiting the correlations between channels of the multivariate time series. Despite CONV-AE and LSTM-AE being autoencoders, they have considerable differences [107]. The first concerns the encoder. The encoder of LSTM-AE tries to compress the data into a smaller representation by mapping, in the latent space, similar points near. On the other hand, the encoder of a CONV-AE tries to extract relevant patterns and the correlation between the multivariate time series features by applying several convolutional filters. So, also, the latent space is different. For LSTM-AE, it contains a compressed representation of the input, while for CONV-AE, the latent space corresponds to feature maps. Feature maps contain data that help identify relevant input data attributes. For example, it contains edges, vertical lines, and horizontal lines in the image analysis. Concerning the decoder, the differences are about how they work. CONV-AE decoder is made by deconvolutional layers, which implement the opposite function of the convolutional layer. It is used to upsample input and learn how to fill in details during the model training process. The decoder of LSTM-AE instead is used to decode the compressed latent space bringing it back into the data input size.

## 3.5. Unsupervised Anomaly Detection

Unsupervised Anomaly Detection (USAD) [6] is based on the Autoencoder architecture with two-phase training. The intuition behind it is to combine the anomaly detection technique of autoencoder by, in the first phase of the training, comparing input and reconstructed data and, in the second phase, amplifying the reconstruction error of inputs containing anomalies.

### 3.5.1. Architecture

The USAD model is essentially composed of two autoencoders connected to each other. They have a common encoder, and each autoencoder has its own decoder. Respectively the two autoencoders are:

$$AE_1(W) = Dec_1(Enc(W))$$

$$AE_2(W) = Dec_2(Enc(W))$$

where $W$ corresponds to the input data, $Enc$ is the typical encoder, $Dec_1$ is the decoder of the first autoencoder, and $Dec_2$ is the decoder of the second one.

The model needs two phases to be trained as composed of two autoencoders. One aims to learn and reconstruct the input data. The other is done in an adversarial way where $AE_1$ tries to deceive $AE_2$ and $AE_2$ tries to recognize if data are real, coming from the encoder, or reconstructed, coming from the other autoencoder. The two phases are:

1. **Autoencoder training** During this stage, the two autoencoders are trained with

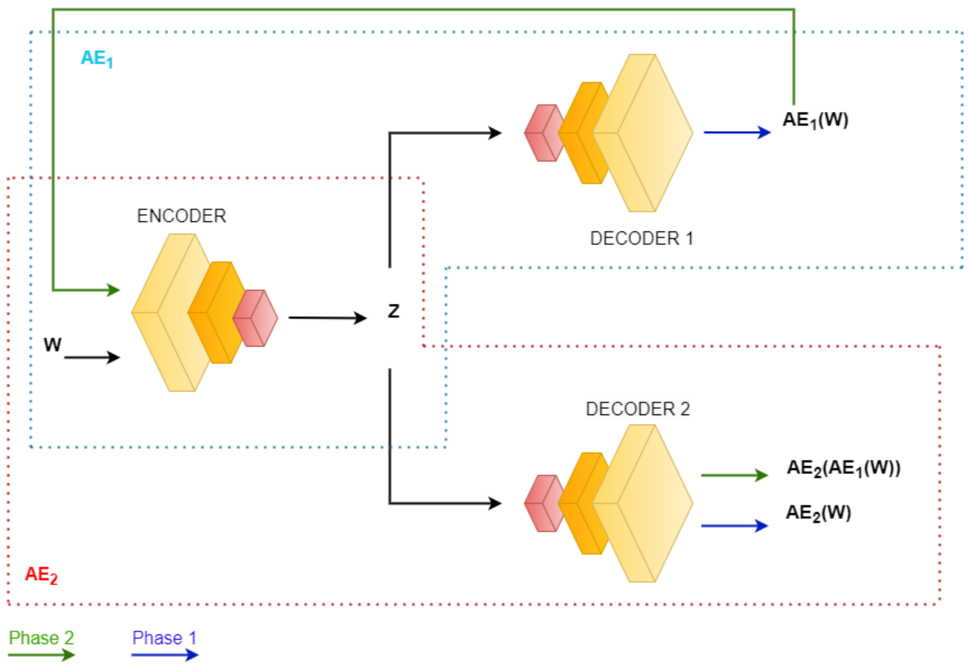


Figure 3.14: USAD training and architecture representation

the usual training of an autoencoder model. The encoder encodes the input data into the latent space, and the decoder decodes it to reconstruct the input data. In this phase, the two autoencoders work in a parallel way, and the learning is regulated by the reconstruction loss defined as:

$$Loss_{AE1} = ||W - AE_1(W)||_2$$

$$Loss_{AE2} = ||W - AE_2(W)||_2$$

And corresponds to the mean squared error between the input and reconstructed data.

2. **Adversarial training** The scope of this phase is to amplify the reconstruction error of inputs that contains anomalies. It exploits the feature of autoencoders that, when they receive in input a sequence with anomalies, tend to reconstruct it in a way that looks like a sequence with normal data which differs significantly from the original one. So, to amplify the reconstruction error, it is sufficient to process the input series twice. More in detail, the input series is first encoded and decoded by $AE_1$, then the result is encoded and decoded by $AE_2$. Moreover, in this phase, the two autoencoder work in different ways according to their loss function:

$$Loss_{AE1} = +||W - AE_2(AE_1(W))||_2$$

$$Loss_{AE2} = -||W - AE_2(AE_1(W))||_2$$

The scope of $AE_1$ is to minimize the difference between the input data and the data obtained by processing them in the two autoencoders. On the other hand, the purpose of $AE_2$ is exactly dual. $AE_2$ tries to maximize the difference between the input data and the data obtained by processing them in the two autoencoders learning to distinguish if the data received in input comes from the original sequence or if it is already processed by $AE_1$

Combining the two-phase, it is obtained a complete loss function for both autoencoders:

$$Loss_{AE1} = \frac{1}{n}||W - AE_1(W)||_2 + (1 - \frac{1}{n})||W - AE_2(AE_1(W))||_2$$
$$Loss_{AE2} = \frac{1}{n}||W - AE_2(W)||_2 - (1 - \frac{1}{n})||W - AE_2(AE_1(W))||_2$$

Where $n$ represents the epoch of the training. For low epochs, the Autoencoder training loss is dominant, while during the training, increasing the epoch value, the adversarial training makes the most significant contribution to the loss.

### 3.5.2. Detection Phase

The scope of this phase (Figure 3.15) is to define a score function that, compared to a threshold, classifies a value as anomalous or not. The score is defined according to the formula:

$$score(W) = \alpha||W - AE_1(W)||_2 + \beta||W - AE_2(AE_1(W))||_2$$

Where $\alpha + \beta = 1$ are used to balance the false and true positives, and according to the value of these two parameters, it is possible to have scores with different sensitivities on a single trained model. This happens because the two terms of the equation assume slightly different values. What is expected is that the reconstructed error is smaller than the error obtained after processing the input first with $AE_1$ and then with $AE_2$. More in detail, if $\alpha$ is greater than $\beta$, the score assumes a lower value, and so the value of positive reduces. On the other hand, if $\alpha$ is lower than $\beta$, the score takes higher values as the significant contribution is made by double encoding decoding, and many positives are detected.

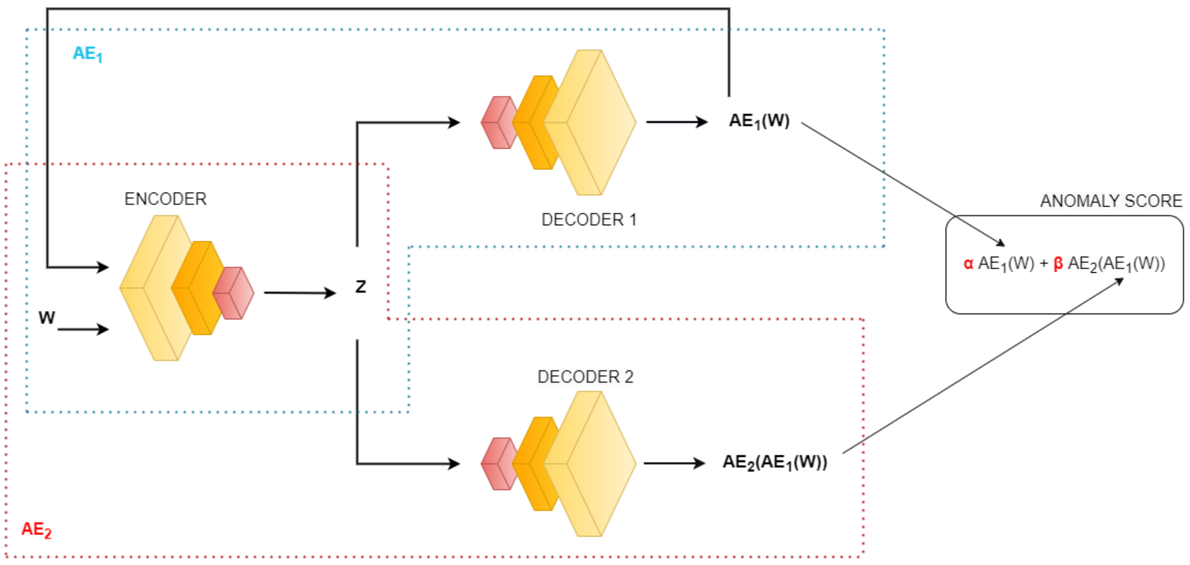


Figure 3.15: USAD score definition

## 3.6. ELM-MI

Extreme Learning Machine (ELM) is a feedforward neural network with a single hidden layer born to feature learning, clustering, regression, and classification. Standard learning approaches learn by using the backpropagation technique. It is the method of fine-tuning the weight of each neural network layer according to the error, calculated by the comparison between the input and the output obtained in the previous epoch. By proper tuning of weights, the reconstruction error is reduced, and the network learns. ELM uses a completely different approach because it does not use backpropagation. The hidden node parameters might be assigned randomly or inherited from the predecessor and never updated. In this way, the model learns the weights of the network nodes in a single step. So, ELM can get good results performing faster than backpropagation networks. Thanks to some changes, Extreme Learning Machines (ELM) can be used in the anomaly detection field. Firstly, it is combined with Mutual Information (MI), a method that measures the relationship between two variables, to obtain an algorithm called ELM-MI [81]. Then an important change has to be made in the structure of ELM. Being randomly initialized on the test data can only be applied to offline inference, and the principal feature of an anomaly detection problem is the response time. To solve this problem, the solution is to use dynamic kernel selection, which, comparing training and test sequences, can determine the networks parameter adaptively for each test sequence.

### 3.6.1. Architecture

ELM-MI method is composed of two different phases. The first is the training phase, where is applied on data a hierarchical clustering procedure while in the second, the test phase, the parameters of radial basis function kernels (RBK) are determined for each test sequence based on the clustered data training, and anomalies are detected.

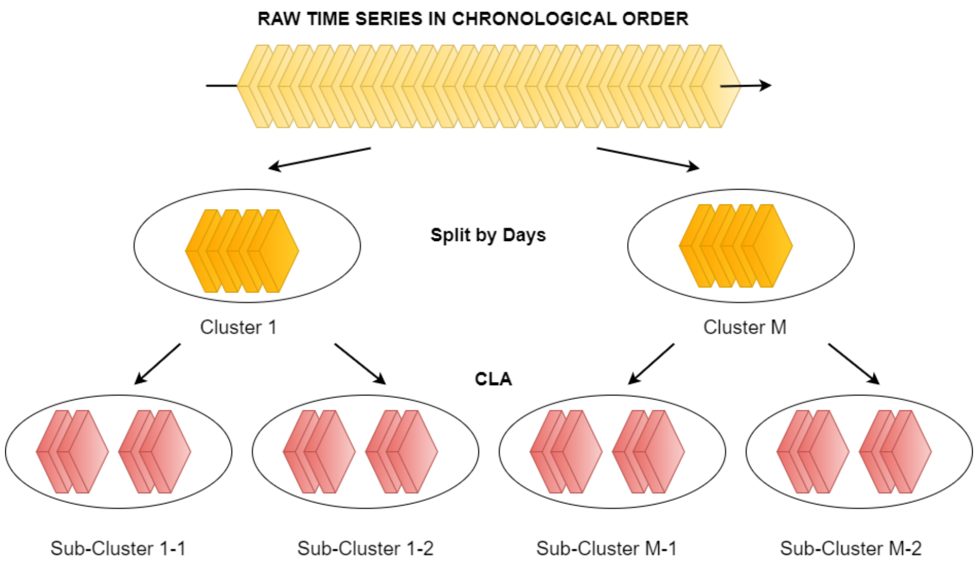


Figure 3.16: ELM-MI division in clusters splitting by days and applying CLA method

During the hierarchical clustering procedure, the first split is done in days if the training data are longer than one day. However, this division is not performed for small datasets that cover a short duration, less than a day. This happens because it is supposed that small datasets have similar data, and no division is necessary. Then, to obtain a second layer division, the daily clusters are split into sub-clusters. In general, any clustering technique can be used. Specifically, as [81], the clustering method used is called Communication with Local Agents (CLA) [106]. Figure 3.16. CLA uses a local gravitation approach reflecting the relationship between a data point and its local neighbors. It considers each data point as an object with mass that impacts its neighbors. More in detail, each point is subject to an attraction force, represented as a vector, generated for each of its neighbors according to their distance, so the closer the neighbor is, the greater the value of the force is. By the vector sum of the forces to which each data point is subject, it is assigned a parameter called local resultant force (LRF) as shown in Figure 3.17.

Moreover, according to the LRF value, each data point is assigned a parameter called centrality (CE), representing the distance from the center of the future cluster. The algorithm needs the number of neighbors in a cluster to obtain these two parameters

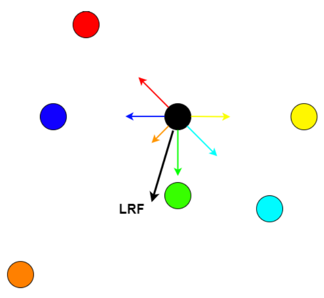


Figure 3.17: The black arrow corresponds to the LRF value of the black data point. It corresponds to the vectorial sum of the force generated by its neighbors represented by the colored arrow

as input. By computing LRF and CE, each data point finds a local agent with a large CE and a small LRF and represents a central candidate of the future cluster. Then, by communicating with each other, local agents set the cluster center and connect as a cluster. Figure 3.18. So, points in the cluster center have a higher density than their neighbors and a larger distance from points in other clusters.

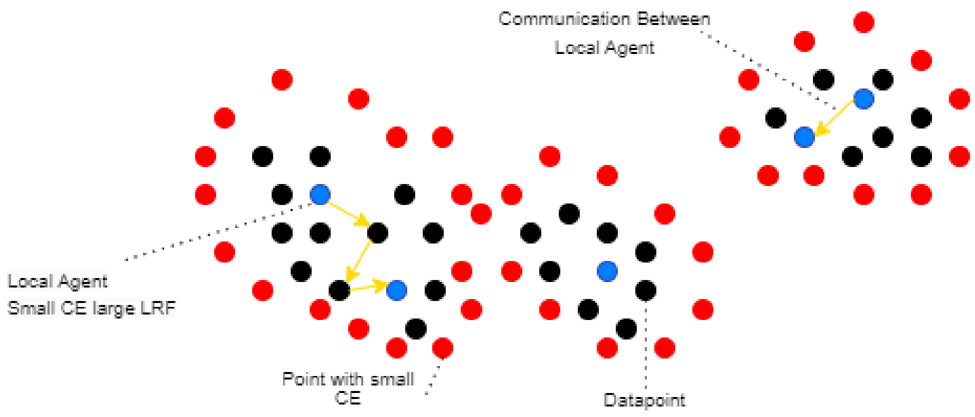


Figure 3.18: The red dots are the point with the smallest CE, the blue dots are ones individuated as local agents, and the yellow row represents the communication between local agents to create the different clusters

The second phase of the ELM-MI method is where the anomaly data are detected. First of all, thanks to the Dynamic Kernel Selection for each test sequence $x \in \mathbb{R}^{W \times N}$ is assigned the corresponding cluster generated in the training phase. To do this, it is calculated the Euclidean distance between $x$ and the center $\mathbf{c}$ of each cluster by the formula:

$$d(\mathbf{x}, \mathbf{c}) = \sqrt{(\mathbf{x} - \mathbf{c})^2}$$

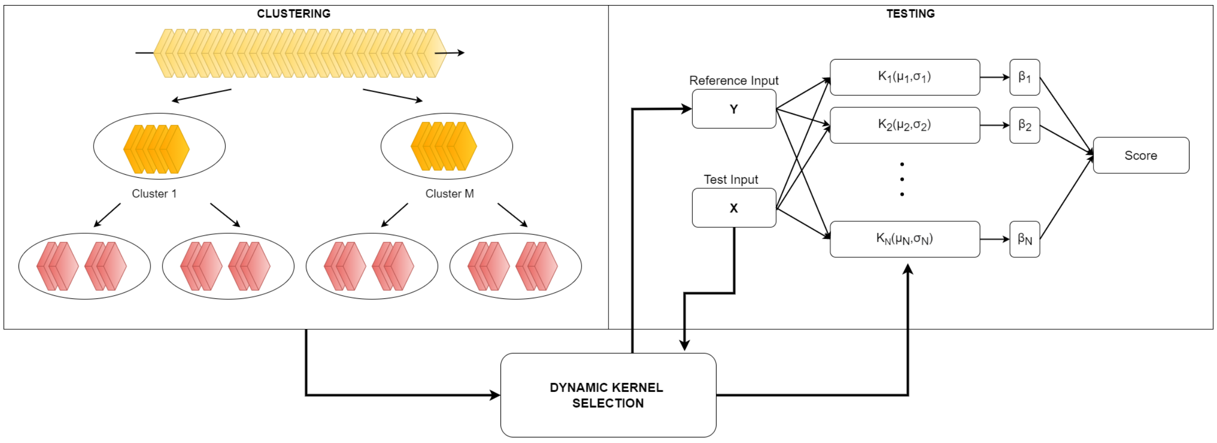


Figure 3.19: ELM-MI framework

So, the test sequence belongs to the cluster where $d(\mathbf{x}, \mathbf{c})$ is less. Having a 2-layers clustering division, this step is applied sequentially, before on the days split, then to the cluster created by CLA. After being assigned to the corresponding cluster, the next step is to apply the Mutual Information (MI) method, which measures the correlation between two random variables. To use this method for anomaly detection, the test sequence is compared with a sequence containing non-anomalous data sampled from the training set and belonging to the same cluster. The lower the MI value is, the more likely the test sequence is anomalous. Being $x \in \mathbb{R}^{W \times N}$ the test sequence and $y \in \mathbb{R}^{W \times N}$ the referencing sequence, where $W$ represents the window size, and $N$ represents the number of the feature, the MI value for $x$ and $y$ can be calculated as:

$$MI(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \int \int \left( \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})p(\mathbf{y})} - 1 \right)^2 p(\mathbf{x})p(\mathbf{y}) d\mathbf{x} d\mathbf{y}$$

where $p(\mathbf{x})$ and $p(\mathbf{y})$ represent the marginal probabilities of $x$ and $y$, and $p(\mathbf{x}, \mathbf{y})$ is the joint probability. Then, $\frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{x})p(\mathbf{y})}$ function can be approximated as a linear combination of multiple kernels defined as:

$$\frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})p(\mathbf{y})} = \sum_{i=1}^{N} \beta_i k_i(\mathbf{x}, \mathbf{y})$$

where $k_i$ is the kernel function, $\beta_i$ is the weight of the corresponding kernel, and $N$ corresponds to the number of kernels for each cluster. The kernel function can be defined as:

$$k_i(\mathbf{x}, \mathbf{y}) = exp\left( -\frac{||\mathbf{x} - \mu_i||^2}{2\sigma_i^2} \right) exp\left( -\frac{||\mathbf{y} - \mu_i||^2}{2\sigma_i^2} \right)$$

where $\mu_i$ and $\sigma_i$ represent the mean and standard deviation of the $i$-th kernel.

Starting from the vector of all the kernel functions, $k = [k_1(\mathbf{x}, \mathbf{y}), .., k_N(\mathbf{x}, \mathbf{y})]^T$, the vector of kernel weight, $\beta = [\beta_i, ..\beta_N]^T$, can be calculated by the formula:

$$\beta = (\mathbf{k}\mathbf{k}^T + \lambda\mathbf{I})^{-1}\mathbf{k}$$

where $\lambda$ is a regularization parameter, and $\mathbf{I}$ is the identity matrix. Usually, in anomaly detection problems, the higher the anomaly score is, the higher the probability of being anomalous. So, to respect this rule, the score is calculated by the negative of the estimated *MI*. The anomaly score is:

$$s(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\beta^T\mathbf{k}\mathbf{k}^T\beta - \mathbf{k}^T\beta + \frac{1}{2}$$

Once obtained, the anomaly score undergoes two further steps. First, on the score is applied a maximum limit equal to 0.5. Each score values greater than 0.5 assumes that value. The second step is to apply a smooth function to the score. It consists in averaging the value of the score by several consecutive values equal to the smooth parameter. This means that a point with a high anomaly score impacts the score of neighboring points. Then the anomaly score is compared with a threshold to determine whether it corresponds to an anomalous value or not.

## 3.7. Anomaly score range

In this section, it is shown the range of values that the score can assume according to the model used to process the data.

| Model | Min Score | Max Score |
|---|---|---|
| AE | 0 | No Limits |
| VAE + ReEnc | 0 | No Limits |
| VAE + Isof | 0 | 1 |
| VAE + KNN | 0 | 1 |
| LSTM | 0 | No Limits |
| USAD | 0 | No Limits |
| ELM-MI | 0 | 0.5 |

Table 3.1: Score range according to the models. Concerning AE includes all the autoencoder implementations (DENSE-AE, LSTM-AE, VAE, CONV-AE). Further clarifications are about VAE+Isof and VAE+KNN which the score obtained by the model is directly the classification in normal and anomalous

# 4 | Implementation and Dataset analysis

## 4.1. Introduction

This chapter presents the implementation of the different neural network models introduced in Chapter 3 by explaining and describing the implementation choices and the datasets used for evaluating the methods. To process and visualize the data and evaluate anomaly detection scores, the programming language used is `Python` [104]. More in detail, the libraries that facilitate the coding are:

- `Pandas` [77]: a Python library that provides fast, flexible, and expressive data structures designed to process data easily and intuitively. It is used to manage the data frames and the time series.
- `Numpy` [37]: a Python extension module providing a convenient and efficient way to handle a significant amount of data. It is used to manage arrays and matrices.
- `Matplotlib` [41]: a Python library for creating static visualization.

`Python` is also used for the presence of machine learning libraries such as:

- `Sklearn` [80] is focused on machine learning tools with mathematical, statistical, and general-purpose algorithms to implement machine learning models. In addition, it is used to scale data and to implement machine learning algorithms like KNN.
- `Keras` [16] is a library used to implement the neural networks models together with the `Tensorflow` [1] framework.
- `PyTorch` [79] is a framework that combines efficient GPU-accelerated backend libraries from `torch` with Python frontend and supports a wide variety of deep learning models.

All the NN models tested in the thesis are implemented in `Python` except ELM-MI [81], which is implemented in `Matlab` [44] consistently with the original implementation.

## 4.2. Dataset

### 4.2.1. SKAB dataset

#### 4.2.1.1. Dataset analysis

The SKAB dataset contains multivariate time series collected from sensors installed on a pump which undergoes several tests on a test bench. The tests simulate different work conditions to record normal behavior and stress the pump to generate anomalies.

In detail, the sensors installed on the pump measure the following quantities:

- **Accelerometer1RMS**: shows a vibration acceleration $[m/s^2]$
- **Accelerometer2RMS**: shows a vibration acceleration $[m/s^2]$
- **Current**: shows the amperage on the electric motor $[Ampere]$
- **Voltage**: shows the voltage on the electric motor $[Volt]$
- **Temperature**: shows the temperature of the engine body $[°C]$
- **Thermocouple**: represents the temperature of the fluid in the circulation loop $[°C]$
- **RateRMS**: Represent the circulation flow rate of fluid inside the loop $[L/min]$
- **Pressure**: it corresponds to the pressure after the water pump $[bar]$

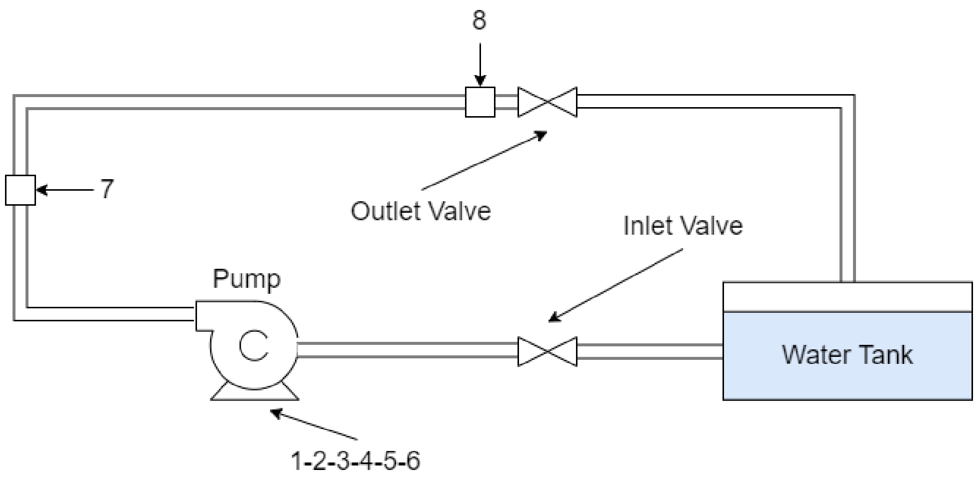


Figure 4.1: The test bench schema. The numbers 1, 2, 3, 4, 5, and 6 correspond to the sensors installed on the pump that measure vibrations, current, voltage, water temperature, and thermocouple. Sensors 7 and 8 are positioned on the water circuit, measuring the pressure and volume flow rate of the fluid.

The dataset is formed by 34 files divided according to the anomalies they contain, divided into three different groups of manually generated anomalies. Table 4.1 presents the anomaly groups. The first type of anomaly is caused by closing the outlet valve of the flow from the pump, and the second by closing the inlet valve to the pump. The last group comprises data obtained from other experiments. For example, some are generated by the simulation of fluid leaks and fluid additions, others by changing the behavior of the rotor.

| **Anomaly Type** | **Number of files** | **Description** |
|---|---|---|
| T1: Outlet Valve Closed | 16 | Anomalies obtained by closing the outlet valve of the pump |
| T2: Inlet Valve Closed | 4 | Anomalies obtained by closing the inlet valve of the pump |
| T3: Others | 14 | Simulation of fluid leaks<br>Simulation of fluid additions<br>Change the behavior of the rotor imbalance<br>A slow increase in the liquid quantity<br>A sudden increase in the liquid quantity<br>An increase in the liquid temperature |

Table 4.1: SKAB dataset anomalies classification

The dataset is labeled, meaning that each sensor measurement is assigned a value indicating whether it is retrieved during an anomaly phase or not. This information is not used during the training of the models since only the test set contains anomalies. Labeled data are used only to evaluate the performances of the model using the different metrics presented in Section 2.4.

#### 4.2.1.2. Data processing

The first phase of data processing is data cleaning. First, the presence of missing values is addressed as proposed in [52], which ignores them and does not replace missing values.

Then, data are scaled. Data collected by different sensors can take values of different magnitudes, so to have comparable values among the features, data are scaled by a min-max scaler defined in Section 2.3.1.

Then, data must be split into training, validation, and testing, as proposed in [52]. The first 400 samples of each file compose the training set, and the successive 50 samples correspond to the validation set, on which the threshold for discerning normal and anomalous values is calculated. The remaining samples contain anomalies and are used for testing.

Finally, if the proposed method requires overlapping windows, the impact of five window lengths ranging from 3 to 48 timestamps is evaluated.

### 4.2.2. Exathlon dataset

#### 4.2.2.1. Dataset analysis

The labeled Exathalon [46] dataset is built from recording the repeated execution of ten different Spark stream processing applications on a 4-node cluster. In this dataset, the user clicks streams from the WorldCup 1998 Website are recorded at a given input rate. To have a more realistic setting, the recorded applications are run concurrently in batches of 5 to 10. The recording is done by Spark Monitoring and Instrumentations Interface, which gives 2,283 features, and the Operating System monitoring of each of the 4 cluster nodes. Each record has a sample rate of one second. The dataset is composed of 93 traces of 7 hours on average. Of these traces, 59 traces record the normal execution of the Spark streaming application, and 34 traces contain disturbed data by events injected during known and labeled time intervals. The anomalies can be classified into different categories, as also shown in Table 4.2:

- **Bursty Input Traces (T1)**: there are six different traces with these types of anomalies. This error happens when the input rate of the data sender is increased significantly for 15-30 minutes
- **Bursty Input Until Crash Traces (T2)**: seven different traces contains this type of anomaly. They are typically shorter than the others and happen when the input rate of the data sender is increased greatly and maintained until the application crashes.
- **Stalled Input Traces (T3)**: these anomalies happen when the data sender stops sending data for about 15 minutes. There are four traces containing this type of anomaly.
- **CPU Contention Traces (T4)**: these six traces are composed of the anomalies where a Python program consumed all the CPU cores available on a given node during a given period
- **Process Failure Traces**: these traces can contain two types of anomalies: **Driver Failure (T5)**: when the driver process of the application failed and **Executor**

| Trace Type | Anomaly Type | # of Traces | Anomaly Instance | Anomaly length: min, avg, max | Data Items |
|---|---|---|---|---|---|
| Undisturbed | N/A | 59 | N/A | N/A | 1.4M |
| Disturbed | T1: Bursty Input | 6 | 29 | 15min - 22min - 33min | 360K |
| Disturbed | T2: Bursty Input until crash | 7 | 7 | 8min - 35min - 1.5hours | 31K |
| Disturbed | T3: Stalled Input | 4 | 16 | 14min - 16min - 16min | 187K |
| Disturbed | T4: CPU Contention | 6 | 26 | 8min - 15min - 27min | 181K |
| Disturbed | T5: Driver Failure | 11 | 9 | 1min - 1min - 1min | 128K |
| Disturbed | T6: Executor Failure | | 10 | 2min - 23min - 2.8hours | |
| Disturbed | T7: Unknown | 11 | 13 | 1min - 11min - 6min | 132K |

Table 4.2: Exathlon anomalies classification [46]

**Failure (T6)**: when an executor process of the application failed.

- **Unknown (T7)**: corresponds to anomalies detected manually because they cannot be classified according to the previous categories

### 4.2.2.2. Data Processing

Data processing is split into five stages: Data Partitioning, Data Transformation, AD Modelling, AD Inferring, and AD Evaluation. All these operations are the same done in the referencing paper and repository to have comparable results because they are applied to the same processed dataset.

**Data Partitioning** The first stage operates directly on raw data. First, it performs data cleaning by replacing missing data with the mean. The next step is to concatenate all the undisturbed traces into the training set to train models implemented with a unique training set. Instead, the disturbed traces are the ones used to test the model.

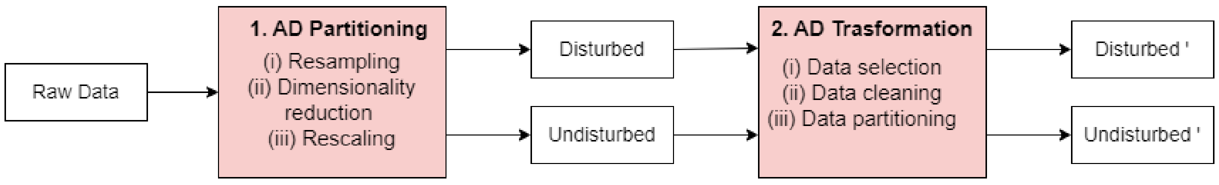


Figure 4.2: Data preparation

**Data Transforming** In this stage, data undergo three different procedures. First, because of the huge number of features, a reduction is performed to reduce the number of features from 2,283 to 19 using Principal Component Analysis (PCA) [59]. PCA is an unsupervised linear transformation technique used mainly for feature extraction and dimension reduction. To reduce the data from a dimension $d$ to the dimension $k$, a $d \times k$-dimensional transformation matrix is constructed. It maps sample vectors onto the new $k$-dimensional feature subspace with fewer dimensions than the original feature space.

Then, resampling reduces the number of timestamps and the cardinality of the time series. The resampling factor used is 15, so a value in the resampled time series corresponds to the mean of 15 consecutive samples. The last operation done in this stage is a min-max rescaling between zero and one. It is done to align features whose raw values may differ by orders of magnitude. For the implemented models that required overlapping windows, such windows have a size of 40 samples.

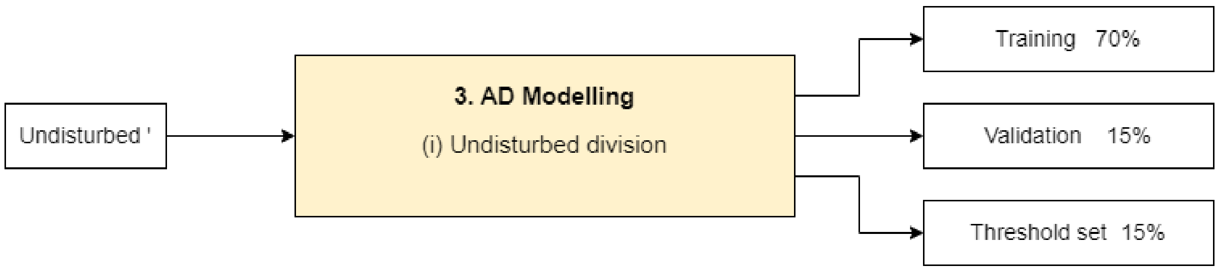


Figure 4.3: Undisturbed trace division

**AD Modelling** All the operations in this stage are applied only to the undisturbed trace. It is split into three parts: training data (70%), validation data (15%), and data on which the threshold is calculated (15%). The training part is used to train each model and the validation part is used to evaluate the performance of the model during the training. For the Exathlon dataset, the threshold is not computed on the validation set but is computed on the remaining 15% of the data. So the threshold is computed on data not used for the training phase of the model. The methods used to determine the threshold are STD, MAD, IQR, and MV.

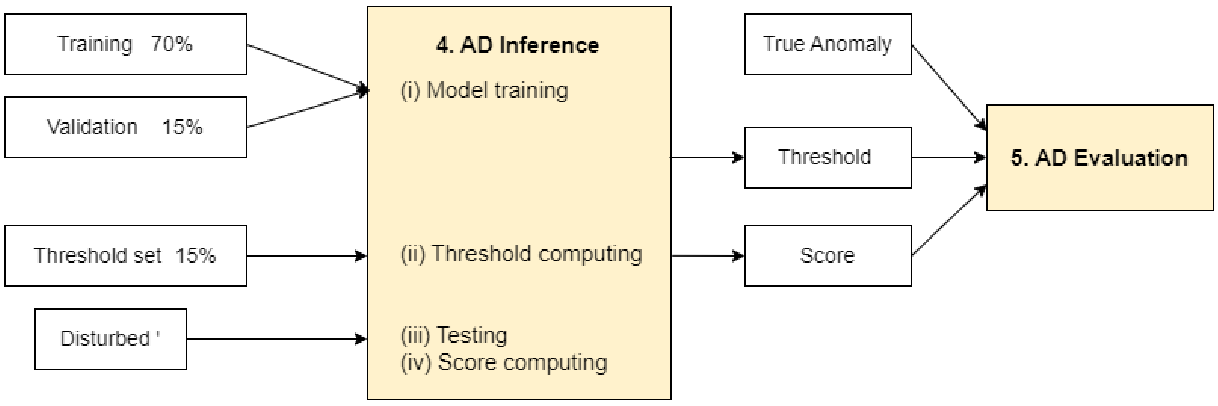


Figure 4.4: Anomaly detection and evaluation

**AD Inference** Once the machine learning model is built and trained, the successive phase focuses on finding anomalies. This stage takes as input the disturbed traces corresponding to the test set, processed in the second stage, and the model processes it. The model gives as the output the predicted data that, compared with the input ones, generate a score for each timestamp. Then the score is compared with the threshold computed at the third stage, and if it is greater, the value is labeled as anomalous.

**AD Evaluation** The final stage inputs the score, the threshold computed in the previous stage, and the real anomalies. First, the score is compared with the threshold, and if it is greater, the value is labeled as anomalous. Then the confusion matrix is computed by comparing the predicted and the real anomalies. Once obtained the confusion matrix, the different metrics like precision, recall, F1-score, false alarm rate (FAR), and missing alarm rate (MAR) are calculated. Thanks to these values, it is possible to compare different models and how they work compared to the state-of-the-art.

## 4.3. Neural Network model

Most of the models tested in the thesis are implemented in `Python` using the `Keras` library. They comprise the LSTM model and all the models based on autoencoders, such as convolutional, LSTM, and Variational. This choice has been made since are very general and commonly used model with several related papers.

Not all of the neural network models analyzed and tested in the development of the thesis are implemented from scratch. In detail, the implementations that follow this choice and are made by readjusting the existing implementation are USAD and ELM-MI.

### 4.3.1. LSTM Autoencoder

The structure of the proposed LSTM Autoencoder is as follows:

| **Layer** | **Output Shape** |
|---|---:|
| InputLayer | (None, WindowSize, NFeature) |
| LSTM | (None, WindowSize, 128) |
| LSTM | (None, 64) |
| z | (None, 64) |
| RepeatVector | (None, WindowSize, 64) |
| LSTM | (None, WindowSize, 64) |
| LSTM | (None, WindowSize, 128) |
| OutputLayer | (None,WindowSize,NFeature) |

Table 4.3: LSTM-AE model detailed implementation. *None* corresponds to the dynamic dimension of a batch, namely the number of samples propagated in the network

More in detail:

- **Input Layer** and **Output Layer** correspond to the layers used to take in input the data to process and to give in the output the predicted value in the correct shape. The input and output layers have the same output shape but have different implementations. The input layer acts as an intermediary to the next layer by checking whether the input has the correct shape. The output layer instead is a *dense* layer that reduces the third dimension from 128 to the number of features. In that way, input and output have the same shape.
- **LSTM** layers are the core of the implementation. The first two consecutive form the encoder, and the last two form the decoder. The principal design choice is to use two consecutive layers with different shapes instead of only one. Stacking LSTM layers makes the model more complex, deeper, and able to capture relevant features in a better way [76]. On the other hand, obtaining the same result using a single LSTM layer is possible, but it requires more neurons and, consequently, is slower [91].
- **z** is the latent space made by a *dense* layer that takes in input data from the second LSTM layer, and its output is processed by the *repeatVector*.
- **RepeatVector** is the layer implemented just before the latent space. It has the role of repeating its input multiple times, transforming the latent space distribution from a two-dimensional vector to a three-dimensional one. The three dimensions correspond to the batch size, the window, and the features.

As an optimizer, it is used *Adam* implementation with its default learning rate.

## 4.3.2. LSTM

LSTM is a simple RNN implemented as follows:

| **Layer** | **Output Shape** |
|---|---:|
| InputLayer | (None, WindowSize, NFeature) |
| LSTM | (None, WindowSize, 144) |
| LSTM | (None,40) |
| OutputLayer | (None, NFeature) |

Table 4.4: LSTM model detailed implementation. *None* corresponds to the dynamic dimension of a batch, namely the number of samples propagated in the network

Relevant considerations have to be made on the shape of the output layer. This method, not being an autoencoder, is not used to reconstruct the input but to predict the next time series value starting from a series long as the window size. So, the output layer is two-dimensional, where the two dimensions correspond to the timestamp and the feature number. Also, in this configuration, two levels of LSTM layers are used, and their size is chosen according to the parameter decided in the Exathlon repository.

### 4.3.3. Variational Autoencoder

All the models in which VAE is involved followed the following implementation:

| Layer | Output Shape |
|---|---:|
| InputLayer | (None, WindowSize, NFeature) |
| LSTM | (None, WindowSize, 128) |
| LSTM | (None, 64) |
| Mean | (None, 64) |
| Variance | (None, 64) |
| z | (None, 64) |
| RepeatVector | (None, WindowSize, 64) |
| LSTM | (None, WindowSize, 64) |
| LSTM | (None, WindowSize, 128) |
| OutputLayer | (None, WindowSize, NFeature) |

Table 4.5: Variational autoencoder detailed implementation. *None* corresponds to the dynamic dimension of a batch, namely the number of samples propagated in the network

More in detail:

- **Input Layer** and **Output Layer** have the same function as in LSTM-AE. The input layer has the function of checking if input data are in the correct configuration. In contrast, the output layer, implemented by a dense layer, reduces the third dimension from 128 to the number of features.
- **LSTM** layers form the encoder and the decoder.
- **RepeatVector** is the layer located immediately after the latent space and has the function to transform 2-dimensional latent space to 3-dimensional. The transformation is done by repeating the latent space vector several times corresponding to the *window size*. The three dimensions correspond to the batch size, the window, and the features.

- **Mean** and **Variance** are two vector obtained from the encoder. They represent the mean and variation of the distribution from which the latent space is sampled.
- **z** is the latent space and is implemented by a `Lambda` layer. This type of layer can execute a function; in this specific implementation, it executes the reparameterization trick described in Section 3.3.3. Taking as input the mean and the variance vector, it executes the sampling function through the reparametrization trick. It takes the *mean* and the *variance* vector as input, and it performs the sampling returning the latent space distribution.

As an optimizer, the *Adam* implementation is used with its default learning rate.

## 4.3.4. Convolutional Autoencoder

The convolutional autoencoder implemented in the thesis is:

| **Layer** | **Output Shape** |
|---|---|
| InputLayer | (None, WindowSize, NFeature) |
| Conv1D | (None, $\frac{WindowSize}{2}$, 128) |
| Dropout(rate=0.2) | (None, $\frac{WindowSize}{2}$, 128) |
| Conv1D | (None, $\frac{WindowSize}{4}$, 64) |
| z | (None, $\frac{WindowSize}{4}$, 64) |
| Conv1D_Transpose | (None, $\frac{WindowSize}{2}$, 64) |
| Dropout(rate=0.2) | (None, $\frac{WindowSize}{2}$, 64) |
| Conv1D_Transpose | (None, WindowSize, 128) |
| OutputLayer | (None, WindowSize, NFeature) |

Table 4.6: Convolutional AE model detailed implementation. *None* corresponds to the dynamic dimension of a batch, namely the number of samples propagated in the network

More in the details:

- **Input Layer** and **Output Layer** correspond exactly to the layers described in the LSTM-AE implementation. The only clarification that needs to be made concerns the window size dimension. Since the encoder comprises two convolutional layers and the dimension corresponding to the window size is halved at each layer, the window size must be divisible by four.
- **Conv1D** layers are used in the encoder. Their convolutional kernels are convolved with their input over a single dimension to extract the most significant features.

- **Conv1DTranspose** layers are used in the decoder and are opposed to Conv1D layers in the encoder having dual behavior. Conv1DTranspose is used for upsampling its input, increasing the output size.
- **Dropout** layer is a regularization term. According to the rate parameter, it randomly ignores some input data, making the training process noisy. The scope is to avoid overfitting.

As an optimizer, it is used *Adam* implementation with its default learning rate.

## 4.3.5. USAD

The USAD model is implemented in `Python` using the library `PyTorch`. USAD is composed of two identical autoencoders with a shared encoder and two separated decoders implemented as:

| Layer | Output Shape |
|---|---|
| | Encoder |
| InputLayer | (None, WindowSize × NFeature) |
| Linear | (None, $\frac{WindowSize \times NFeature}{2}$) |
| Linear | (None, $\frac{WindowSize \times NFeature}{4}$) |
| z : Linear | (None, 64) |
| | Decoder |
| Linear | (None, $\frac{WindowSize \times NFeature}{4}$) |
| Linear | (None, $\frac{WindowSize \times NFeature}{2}$) |
| OutputLayer : Linear | (None, WindowSize × NFeature) |

Table 4.7: USAD detailed implementation. *None* corresponds to the dynamic dimension of a batch, namely the number of samples propagated in the network

More in detail:

- **Input Layer** and **Output Layer** represent the layers used to take in input data and return the processed ones. Differently from the model described before, they do not process a two-dimension array corresponding to the window size and the number of features, but a reshaped one-dimension array with the size equivalent to the window size multiplied by the number of features. Moreover, the output layer is implemented with a *Linear* layer which corresponds to the *dense* one in *Keras*.

- **Linear** layers are used in the encoder and the decoder. The focus is on the output dimension; it is halved at each layer in the encoder and doubled at each decoder layer.

As an optimizer, the *Adam* implementation is used with its default learning rate.

### 4.3.6. ELM-MI

The ELM-MI method is implemented by readjusting the original implementation. The structure is profusely described in Section 3.6.1. The implementation decision regards only the choice of the parameters to pass to the model:

| **Parameter** | **SKAB** | **Exathlon** |
|---|---:|---:|
| Num days | 1 | 1 |
| Num Kernel | 100 | 300 |
| CLA Parameter | 5 | 15 |
| Lamda | 0.01 | 0.01 |
| Smooth Parameter | 50 | 50 |
| Score Limit | 0.5 | 0.5 |

Table 4.8: ELM-MI parameters

The parameter values for the SKAB dataset are the same as written in the reference paper. Therefore, the only estimated parameter value is the *CLA Parameter*, which is decided by executing the original code many times and picking the one that gives the same paper results. The parameters for the Exatlhlon dataset are obtained starting from the value of the SKAB parameter. The parameters for creating the cluster, the number of kernels and the CLA parameter, are incremented according to the ratio of the sizes of the two datasets. The choice of other parameters is to maintain the same value for both datasets.

A parameter that has to be explained is the *smooth parameter*. It is used in the smooth function, which is applied to the score and averages the value of the score by several consecutive values equal to the smooth parameter. This means that a point with a very high anomaly score impacts the score of neighboring points. Furthermore, the score obtained with ELM-MI undergoes the effect of another parameter, the *score limit*. It has the scope to limit the anomaly score to a certain value.

# 5 | Evaluation

## 5.1. Thresholding

Both the SKAB and EXATHLON datasets are processed by the neural network models described in Section 3. Except for VAE+isof and VAE+KNN, all the models return an anomaly score that must be evaluated to determine whether a value is anomalous or not. For each model, excluding ELM-MI and USAD, the score is computed as follows:

$$MSE = \frac{1}{N}\sum_{i=0}^{N}(x_i - y_i)^2$$

Where $N$ is the number of values processed, $x$ is the ground truth, and $y$ is the output of the model. The score corresponds to a reconstruction error for the autoencoder-based models, while for LSTM, the score is a prediction error. For ELM-MI, the score is calculated as thoroughly described in Section 3.6.1 while the USAD score is shown in Section 3.5.2.

In the case of SKAB, a threshold is calculated for each file that composes the dataset, evaluating the score of a small validation set composed of 50 normal values. In the case of EXATHLON, instead, the threshold is computed on the entire validation set. In both cases, four thresholding approaches have been analyzed:

- Maximum Value (MV) computes the threshold as the maximum anomaly score on the validation set:

$$\tau = max(s)$$

- Standard Deviation (STD) relies on the anomaly score mean and standard deviation. In this case, $th_{factor}$ represents the minimum number of standard deviations to consider a score anomalous. The threshold is:

$$\tau = mean(s) + th_{factor} \cdot std(s)$$

- Median Absolute Deviation (MAD) computes the median of the absolute differences between each anomaly score and the median anomaly score, making this approach less sensitive to outliers than the standard deviation. It is calculated as:

$$\tau = median(s) + 1.4826 \cdot th_{factor} \cdot median(|s - median(s)|)$$

- Inter-Quartile Range (IQR) is based on the difference between the 75th percentile ($Q_3$) and the 25th percentile ($Q_1$) of the anomaly scores. It is calculated as follows:

$$\tau = Q_3 + th_{factor} \cdot (Q_3 - Q_1)$$

Where $\tau$ is the threshold and $th_{factor}$ is a constant.

Concerning VAE+isof and VAE+KNN, the output of the methods is not a score, but they directly classify data as normal or anomalous. This is because they are not based directly on a threshold but have a parameter called contamination that affects the classification. Since Isof and KNN analyze the same distribution, it is chosen to use $contamination = 0.001$. To set this value, different tests are executed, and the one corresponding to the best results is selected. Moreover, it is noted that using several contamination values, the results do not vary considerably, specifically by a maximum of 1%. Concerning KNN, another parameter, K, which corresponds to the nearest neighbors, can affect the results. Also, in this case, several tests are performed, and the results vary by a maximum of 1%. So, it is chosen $K = 35$, which has the best results.

## 5.2. SKAB Dataset

### 5.2.1. Analysis of the anomaly score distribution on the validation set

This section presents and analyzes the distributions of the validation set anomaly score used to compute the threshold. Figure 5.1 shows representative validation set unimodal distributions. This distribution is expected, as the validation sets contain only normal values reconstructed by the proposed algorithms. Figure 5.1 also shows that different approaches generate scores in different ranges. For all models, the validation score assumes values which does not deviate much from the score mean being included in a range corresponding to the mean ±3 standard deviations. Consequently, not being values that deviate consistently from the mean, the MV thresholding technique can be used.

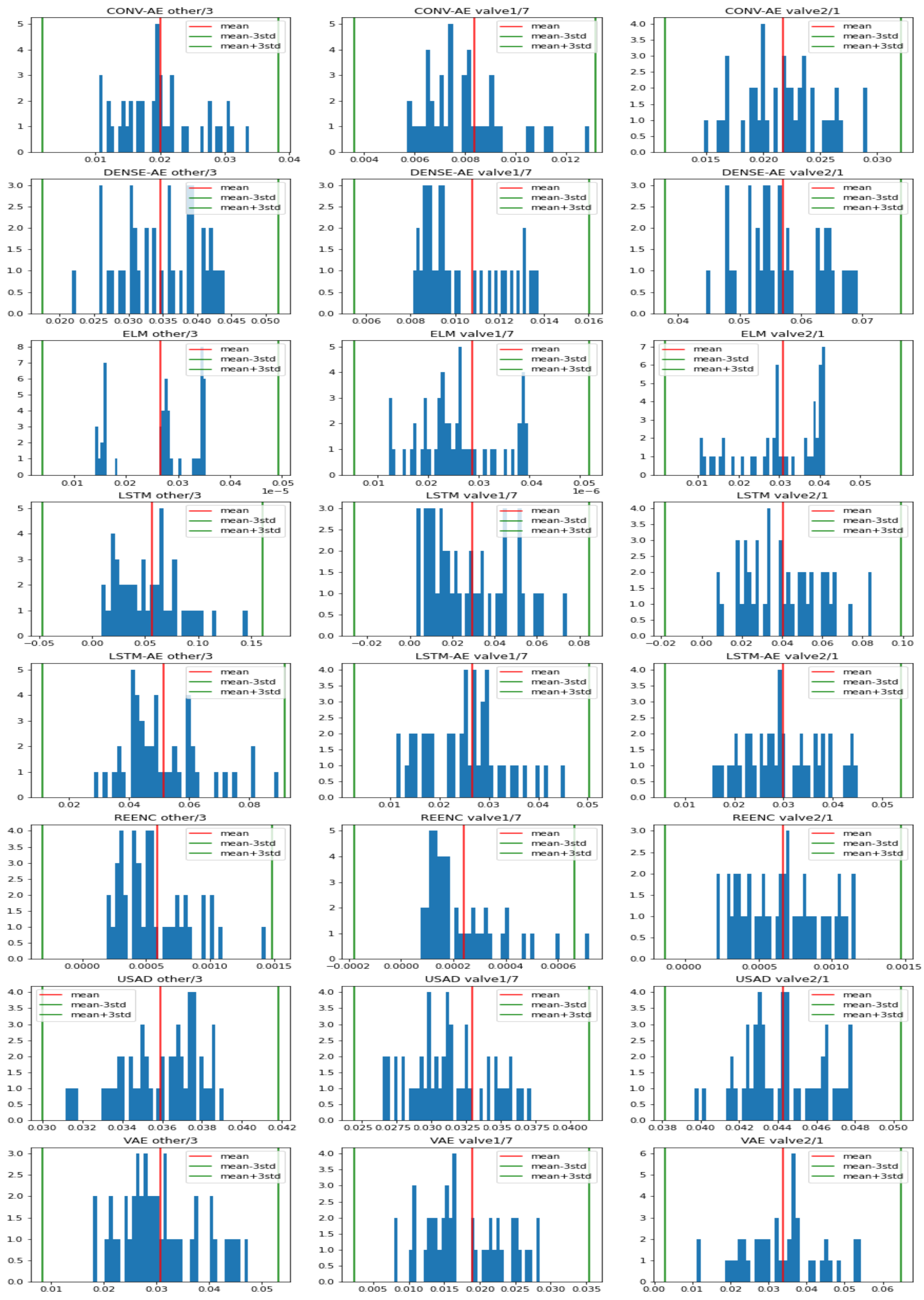


Figure 5.1: Different histograms of the anomaly scores computed on the validation set using all the neural network models. Since the dataset comprises 34 files, too much to be shown in a single graph, in this figure are shown only three different files, one for each type of anomaly

Comparing the plot of different models, it is noticed that, despite assuming the same shape, the scale change according to the models. The autoencoder model generates a score with similar values, such as CONV-AE, DENSE-AE, LSTM-AE, and VAE. This is because they compare the ground truth to the reconstruction; despite the reconstruction being slightly different from each model, the score values do not change significantly. Also, the validation score values are similar for USAD, being an autoencoder too. The LSTM model is the one with the highest validation score value. This happens because working by predicting the following sequence starting from a note one and the prediction is not precis as a reconstruction. The model that gives the lowest score value is VAE+ReEnc. Since the score is obtained by measuring the distance of where points are mapped into two latent spaces, and the data into the latent space assumes shallow values, the result is that the score is of the same order of magnitude.

There are no considerable differences in the different types of anomalies (**T1**, **T2**, **T3** shown in Table 4.1). Therefore, each model has a score comparable for each anomaly type.

### 5.2.2. Analysis of the anomaly score distribution on the test set

This section analyzes the different distributions of the test score upon which the threshold is applied to detect anomalies.

Figure 5.2 presents representative test set anomaly score distributions. Since test sets contain both normal and anomalous data, the score distribution assumes a bimodal distribution. The two peaks correspond respectively to a prevalence of anomalous and normal data. In some cases, the anomaly score distribution is not bimodal. This happens basically for three reasons. The first is that the score of these anomalies assumes values similar to the score of normal data, so the neural network model does not detect some anomalies. Meaning that anomalous and normal scores are mixed, making the separation of anomalous from normal data impossible. The second reason is that anomalous and normal scores do not differ too much to generate two separate visible peaks. This does not mean that the results are bad for sure, but finding a threshold to separate anomalous from normal scores is more challenging. The latter reason concerned the nature of the data on which the score is calculated. The fewer anomalies in the data on which the score is calculated, the less visible the relative peak. The ability to distinguish anomalous and normal values depends on the ability of the model to reconstruct normal values correctly and anomalous values incorrectly. The distributions in Figure 5.2 show that different neural network models generate different anomaly score distributions.

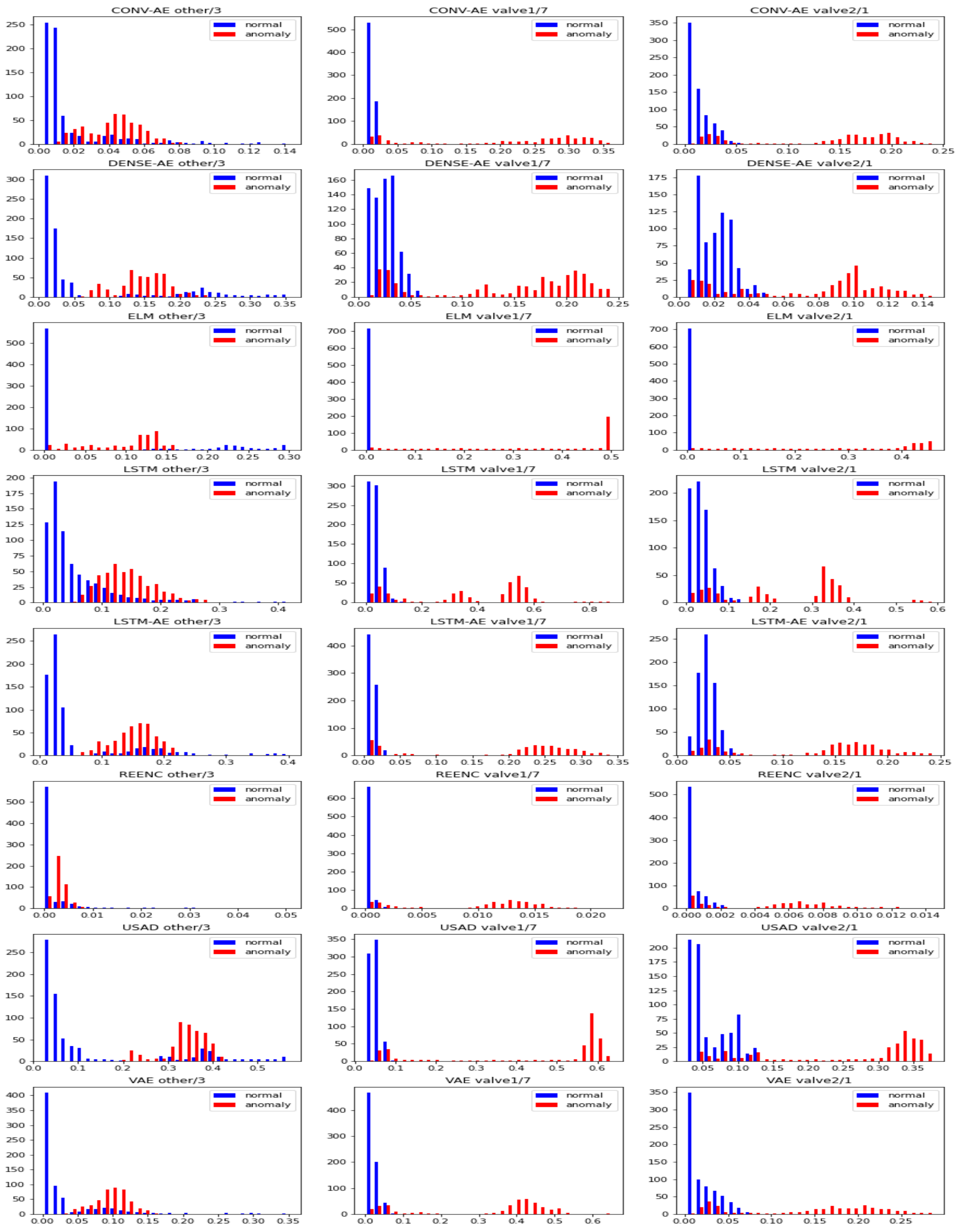


Figure 5.2: Different histograms of the anomaly scores computed on the test set using all the neural network models. Since the dataset comprises 34 files, too much to be shown in a single graph, the figure shows the score of only three files, one for each type of anomaly. Files that correspond to the one represented in Figure 5.1. For a better comparison, the y-axis representing the occurrence of each score value is in logarithmic scale

Autoencoder-based models, excluding DENSE-AE, generate a clearly bimodal score distribution. The reason lies in how they work; they have very inaccurate reconstruction for anomalous data. It is evident for USAD, which has the greatest distance between normal and anomalous distribution. By adversarial training, it tries to amplify the reconstruction error between anomalous values. DENSE-AE have result different from other autoencoders. The reason is that they are the most straightforward implementation of autoencoders, and the reconstruction both for normal and anomalous data is worse, as shown in the following section. Different score distributions are obtained using non-autoencoder models. LSTM, a prediction-based model, cannot separate anomalies from normal data as autoencoders. For this dataset, the prediction is less accurate than the reconstruction. It is evident by comparing the score distribution of normal data for LSTM and VAE. For VAE, scores of normal data are similar, appearing more compactly in the histogram. Concerning ELM-MI, which is clustering based, the score distribution assumes a clearly different shape. Normal data have a similar score, so the score distribution of normal data corresponds to only a column in the histogram. Instead, the anomalous data have distributions similar to other models.

The presence of anomalous and normal scores mixed, indicated by red and blue bars shuffled together in Figure 5.2, corresponds to incorrect prediction value generating a too-low score for anomalous samples and too-high score for normal samples. The consequence is that it is impossible to separate these values, generating many false negatives or false positives.

This section presents a qualitative analysis of the score distribution. The quantitative approach is in Section 5.2.4, where the analysis focuses on the ROC curve.

### 5.2.3. Reconstruction and score analysis

This section presents two different analyses: the reconstruction and the score. The former presents a representative qualitative example of how the autoencoder-based approaches reconstruct the input data. The latter focuses on the anomaly score of all the different models analyzed. The presented plots refer to a file from the SKAB dataset for simplicity. Still, their behavior is analogous and easily extendible to the entire dataset. Below is presented a qualitative analysis of how the eight features of the SKAB multivariate time series are respectively reconstructed by DENSE-AE Figure 5.3, LSTM-AE Figure 5.4, VAE Figure 5.5, and CONV-AE Figure 5.6.

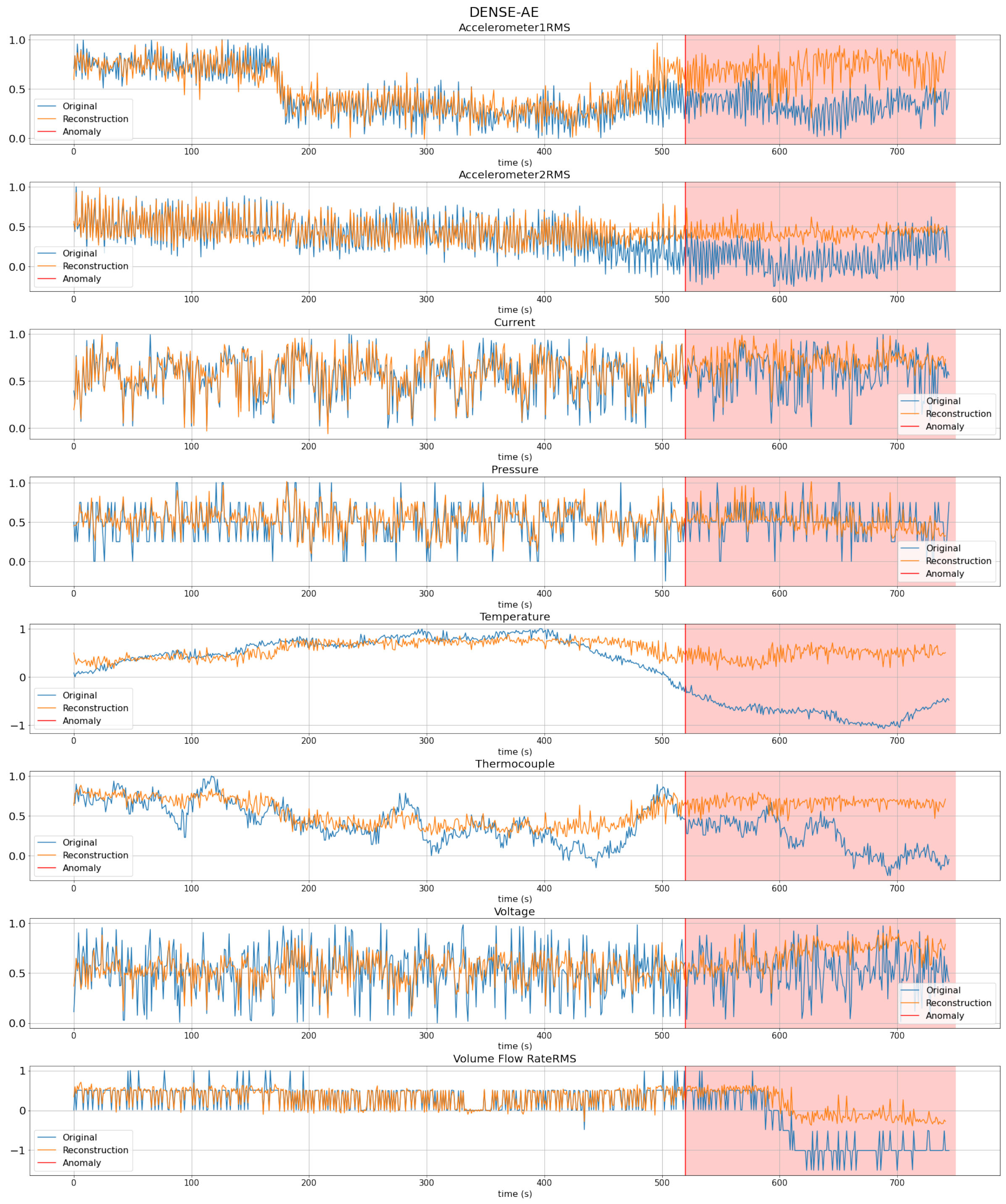


Figure 5.3: Original sequences compared with the reconstruction ones computed by DENSE-AE

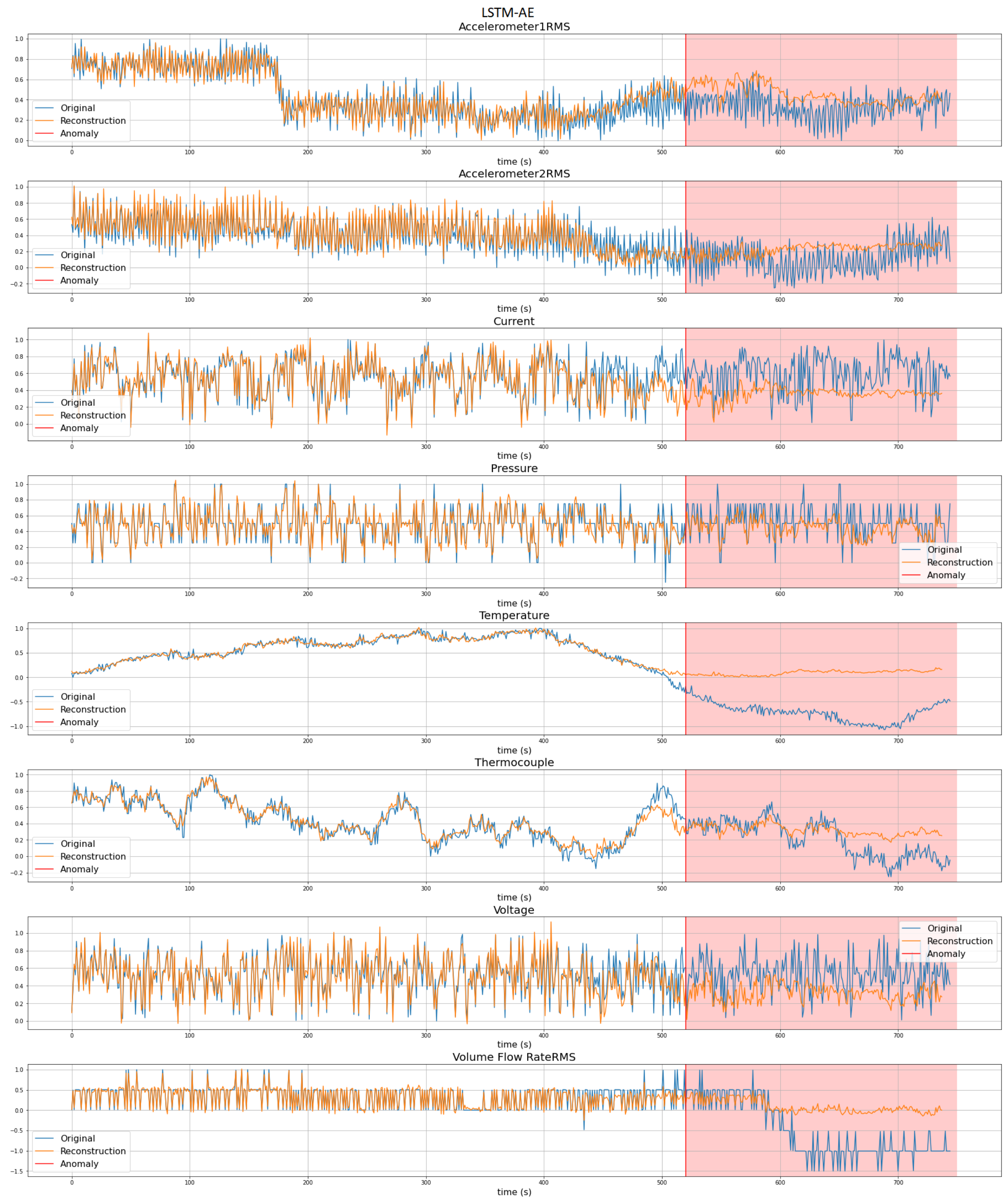


Figure 5.4: Original sequences compared with the reconstruction ones computed by LSTM-AE

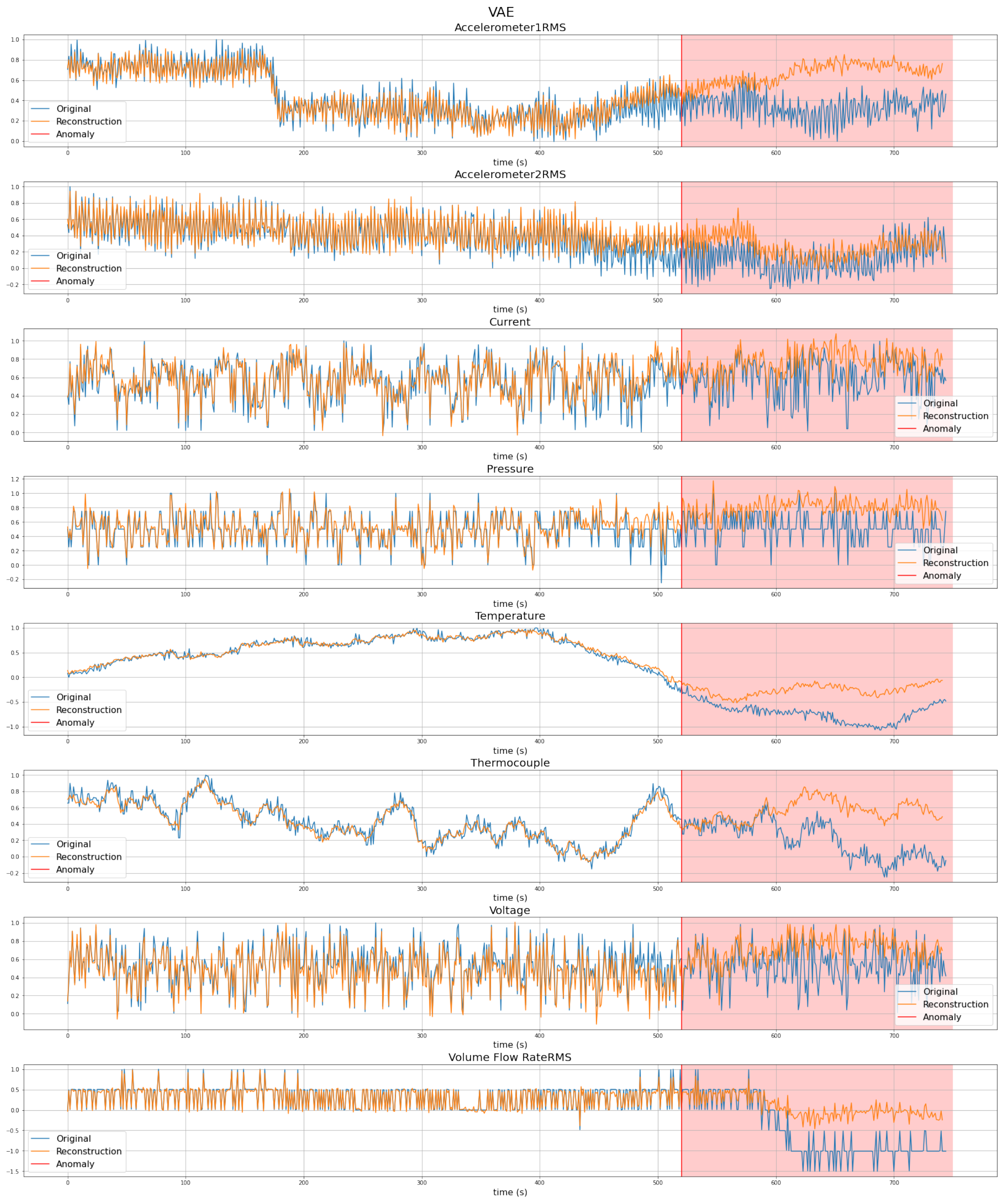


Figure 5.5: Original sequences compared with the reconstruction ones computed by VAE

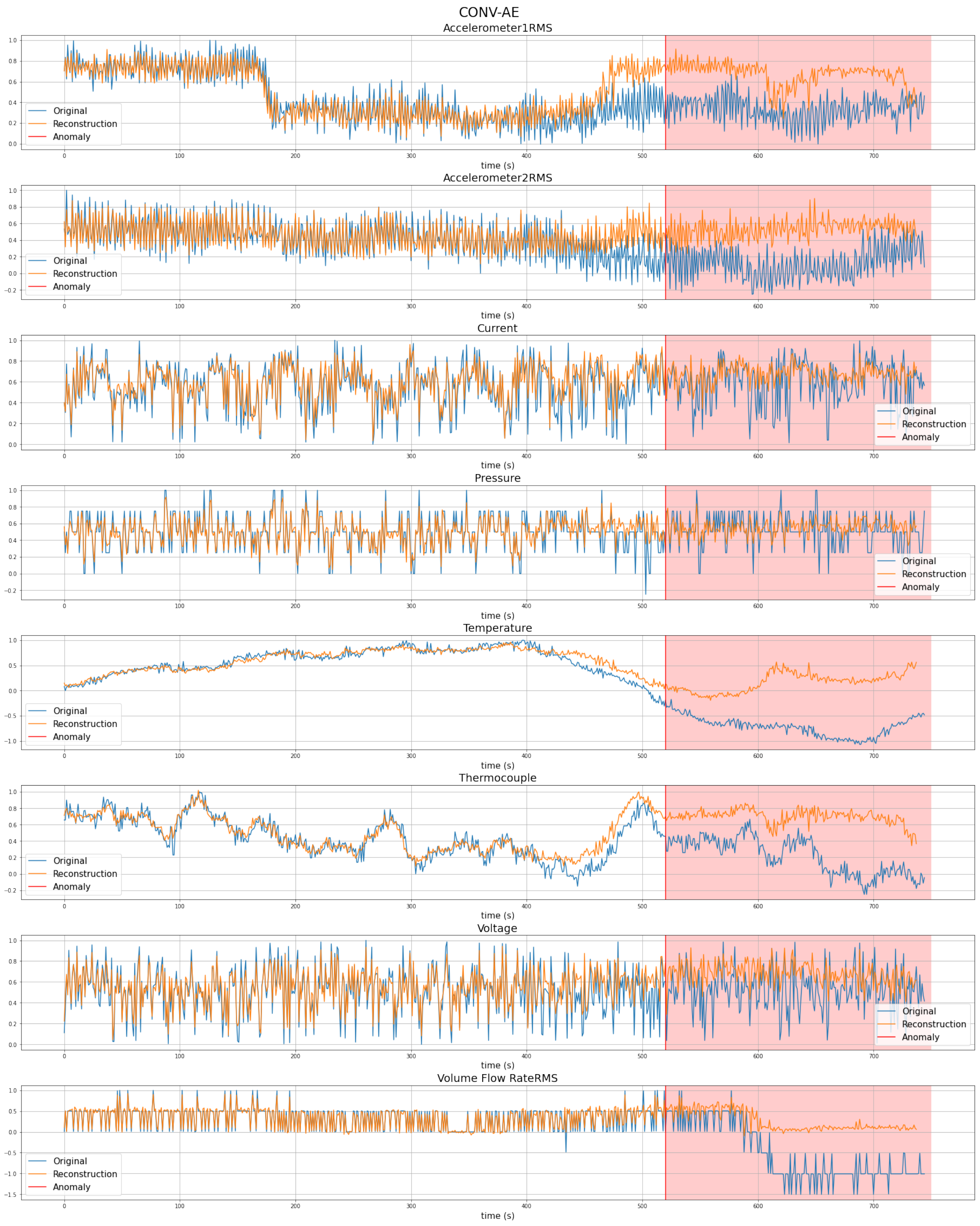


Figure 5.6: Original sequences compared with the reconstruction ones computed by CONV-AE

Looking at the reconstructed figure shown above, it is evident that for each autoencoder, as expected, the reconstruction is closer to the ground truth data for normal data, while it deviates considerably for anomalous data. Moreover, the closer the timestamp is to the anomaly, the more the correspondent sample has a worse reconstruction. It is also shown how the reconstruction change for each feature, so it is evident that the value of some features, such as Temperature, Thermocouple, and Volume Flow rate, change considerably according to the normal and anomalous behavior of the system and have a huge impact on the anomaly score. On the other hand, other feature values, such as Current and Voltage, are less affected by anomalies and have a minimal contribution to the score value.

Comparing the reconstruction of each autoencoder, it is evident that each autoencoder model reconstructs data differently. As Figure 5.3 shows, DENSE-AE is the autoencoder that reconstructs in the worst way. The reason for this behavior is it is the most straightforward implementation of an autoencoder network. It is clear that by using a more complex structure that exploits more advanced layers such as LSTM and convolutional, the reconstructed data are more accurate [18]. Conv-AE, LSTM-AE, and VAE all reconstruct the input accurately, especially, as expected, for normal values, but they have some differences, which this section discusses in more detail. *LSTM-AE* and *VAE* reconstruct quite similarly, being based on the same structure LSTM structure, but due to a better latent space encoding, *VAE* works better, as shown in the following sections. This is evident by observing the reconstruction plots in Figures 5.5 and 5.4, where the original and the reconstructed lines almost coincide.

Concerning *CONV-AE*, analyzing its reconstruction graph (Figure 5.6) and comparing the scale of its score graph with others, it is evident that it reconstructs the input in two ways. The reconstruction is accurate for normal data and diverges a lot for anomalous data.

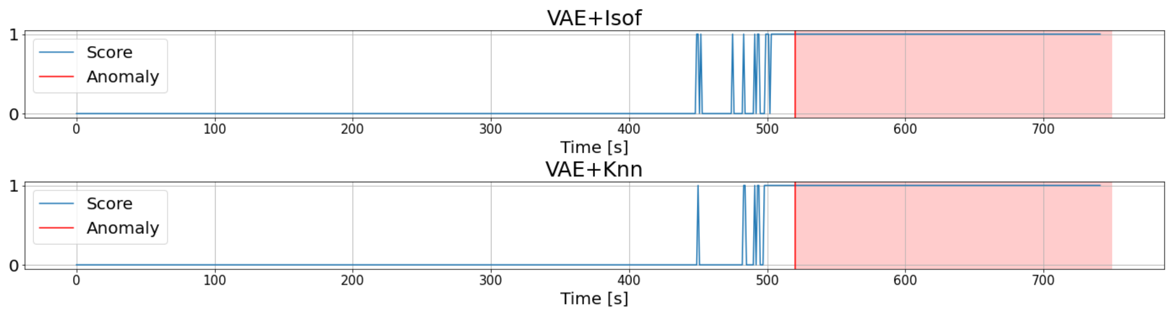


Figure 5.7: Results obtained by processing data with VAE+Isof and VAE+KNN

Regarding the anomaly score, different observations can be made. First, for *VAE+KNN* and *VAE+Isof* shown in Figure 5.7, the anomaly score is not computed. The two models immediately classify each value returning directly if each value is anomalous or not without any further classification.

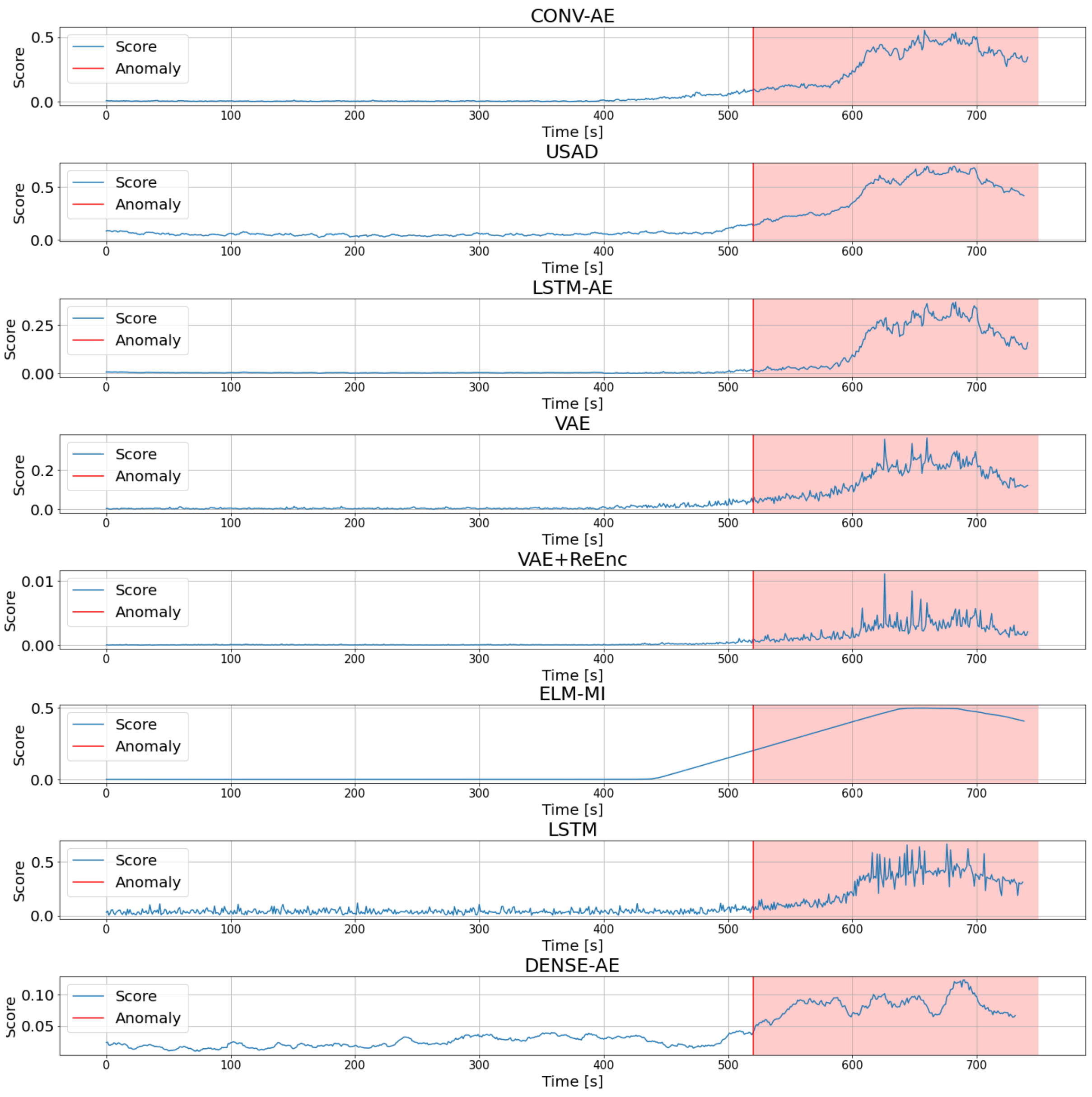


Figure 5.8: Anomaly score of a test of the SKAB dataset processed with the machine learning methods implemented

To better analyze how the anomaly score change according to anomalies, it is necessary to analyze together the quantitative comparison in Table 5.1 and the qualitative comparison shown in Figure 5.8.

By the qualitative comparison emerges that the score of each model assumes high values in correspondence with the anomalies, which means that the reconstruction is bad. On

| | VAE | CONV-AE | DENSE-AE | LSTM-AE | REENC | LSTM | USAD | ELM |
|---|---|---|---|---|---|---|---|---|
| ANOMALY mean | 0.28209 | 0.11170 | 0.08851 | 0.18022 | 0.00651 | 0.26377 | 0.42038 | 0.28743 |
| NORMAL mean | 0.01639 | 0.01158 | 0.03448 | 0.02308 | 0.00017 | 0.03787 | 0.05194 | 0.00724 |
| $\frac{ANOMALY}{NORMAL}$ | 17.21295 | 9.64627 | 2.56739 | 5.22746 | 38.88773 | 6.96490 | 8.09394 | 39.68548 |

Table 5.1: Mean score of normal and anomalous samples relative to the single test shown in Figure 5.8. The last row of the table corresponds to the ratio between normal and anomalous scores. The greater it is. the easier it is to distinguish between normal and anomalous scores

the other hand, the score is very low for normal data, meaning that the neural network model performs well in predicting or reconstructing.

The worst score is the one obtained with DENSE-AE, and it is the score where the means of normal and anomalous scores are more similar, being one twice the other. This happens because, as shown in the early part of this section, it is the autoencoder method that learns worse. This is also confirmed by comparing the normal mean score in the table with another autoencoder. Between autoencoders, CONV-AE performs the best reconstruction of normal data, while the VAE has the higher anomaly score mean. Moreover, VAE is the one where the ratio between the mean of anomalies and normal samples is greater. Under this perspective, also USAD has the same behavior. As designed, it does not focus on reconstructing its input sequence but tries to maximize the score for anomalous data. Concerning LSTM-AE compared with VAE, despite having a similar architecture, worst reconstructs the normal data, and the mean score for anomalies increases less. This happens due to the different latent spaces of the two models, which are presented in Section 3.3.4. For LSTM, the score has more fluctuation than autoencoders. This is related to how the score is calculated. Since LSTM predicts the value of the following timestamp of the input sequence, the score to the mean squared error of only one timestamp. On the other hand, since autoencoders reconstruct the input sequence, the score is computed as the mean square error on several timestamps. The result is that the score is more stable, with fluctuations not so evident. The method that generates the lowest score is *VAE+ReEnc*. This is consistent with what is expected because the score represents a distance, which is very small, between where the same value is mapped in two different latent spaces. Moreover, together with ELM-MI is the method that produces the higher ratio between anomaly and normal score. The last consideration is that the score coming from *ELM-MI* methods differs from the others. It is not characterized by fluctuations typical of other scores but has a less noisy trend. This is exactly the effect of the smooth function used in the algorithm to propagate the anomaly effect of a point to its neighbors. Moreover, the maximum value that the ELM-MI score can assume is 0.5, as specified in [81]. Being obtained by processing the same file, all the scores follow the same trend where the first $\approx 500$ timestamps are associated with low values representing the normal value. Instead, the last $\approx 200$ timestamps are associated with high values representing the anomalies. In addition, in the timestamps preceding the real anomaly, highlighted in red, there is a relevant increment of the anomaly score value, indicating that abnormal behavior is about to occur.

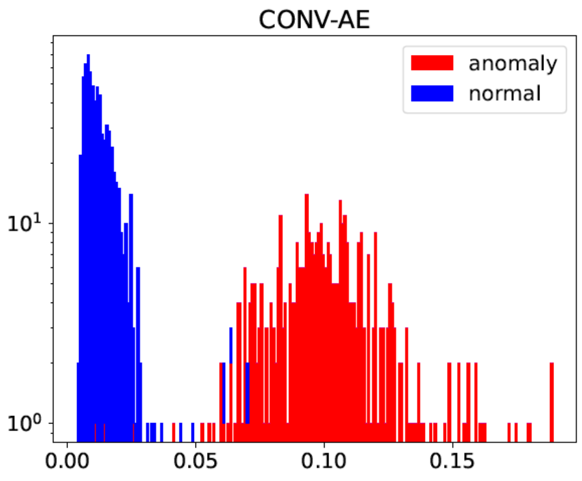
### 5.2.4. AUROC

The Receiver Operating Characteristic (ROC) curve is defined as "a plot of test sensitivity as the y coordinate versus its 1-specificity or false positive rate (FPR) as the x coordinate" [78]. It is an effective method of evaluating the performance of anomaly detection classification. When analyzing a ROC curve, the interest is placed on the area subtended by the curve itself, called the AUROC. It represents how much a model can distinguish between classes. The higher its value, the better the model can distinguish anomalies from normal values. This section analyses the AUROC value obtained from the ground truth value and the scores for each model. Since each file of the SKAB dataset is processed separately and has a different validation set, the AUROC of each model is the average of per-file results and is shown in Table 5.2.

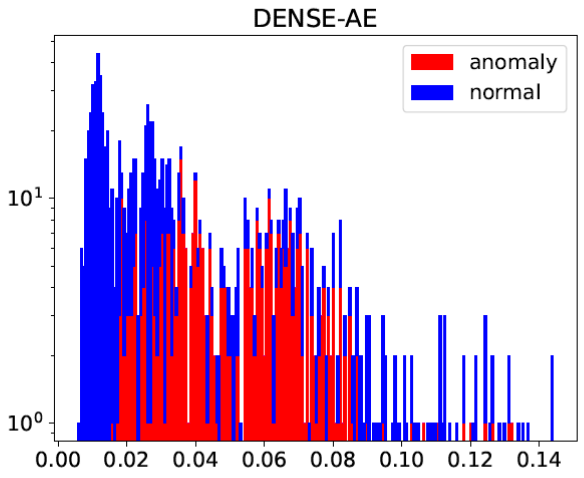
| **METHOD** | Conv-AE | VAE | ReEnc | USAD | ELM-MI | LSTM | LSTM-AE | Dense-AE |
|---|---|---|---|---|---|---|---|---|
| **AUROC** | 0.94221 | 0.92289 | 0.91611 | 0.90503 | 0.89592 | 0.87697 | 0.87045 | 0.85651 |

Table 5.2: AUROC value for each method

Since AUROC measures how well the model separates the positive and negative classes, the distribution of the anomaly score has to be analyzed to understand why different models have different values. In this specific is useful to compare the distribution of the model score with the best and worst AUROC by considering Figure 5.9. It shows the score distribution of CONV-AE, which has the best AUROC, and DENSE-AE, which has the worst AUROC.



(a) Test score obtained from the CONV-AE



(b) Test score obtained from DENSE-AE

Figure 5.9: Anomaly score distribution showing the value assumed by normal and anomalous value

Concerning the CONV-AE, the score distribution is clearly bimodal, with a clear distinction between normal and anomalous points. With a coherent threshold value, it is possible to obtain a precise classification by getting good results in detecting anomalies. On the other hand, the distribution relative to the DENSE-AE score is not bimodal since anomalous and normal value assumes similar value. The effect is that it is more difficult to classify each value correctly. This generates, for every threshold choice, a lot of false positives and negatives, significantly reducing the AUROC value.

AUROC is a score that does not depend on a threshold value, but it can be compared with threshold-dependent evaluations to understand the behavior of each anomaly detection model. To evaluate how accurately a model and a threshold method can distinguish anomalies, the F1-Score is used. It is the harmonic mean between the precision and the recall calculated as:

$$F1Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Where $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$.

By applying the different threshold values, computed by MAD, IQR, STD, and MAX, on the anomaly score, the anomaly prediction is obtained. Then by comparing it to the ground truth, the F1-Score is computed. Finally, having several thresholding techniques and threshold factors, for each machine learning architecture, the maximum F1-Score is compared with the AUROC. As Figure 5.10 shows, AUROC and F1-Score behave similarly. Models such as CONV-AE, VAE, and ReEnc, which have the best AUROC score, are the better for F1-Score. This trend is observed because better separation of anomalies from the normal point allows getting a better F1-Score.

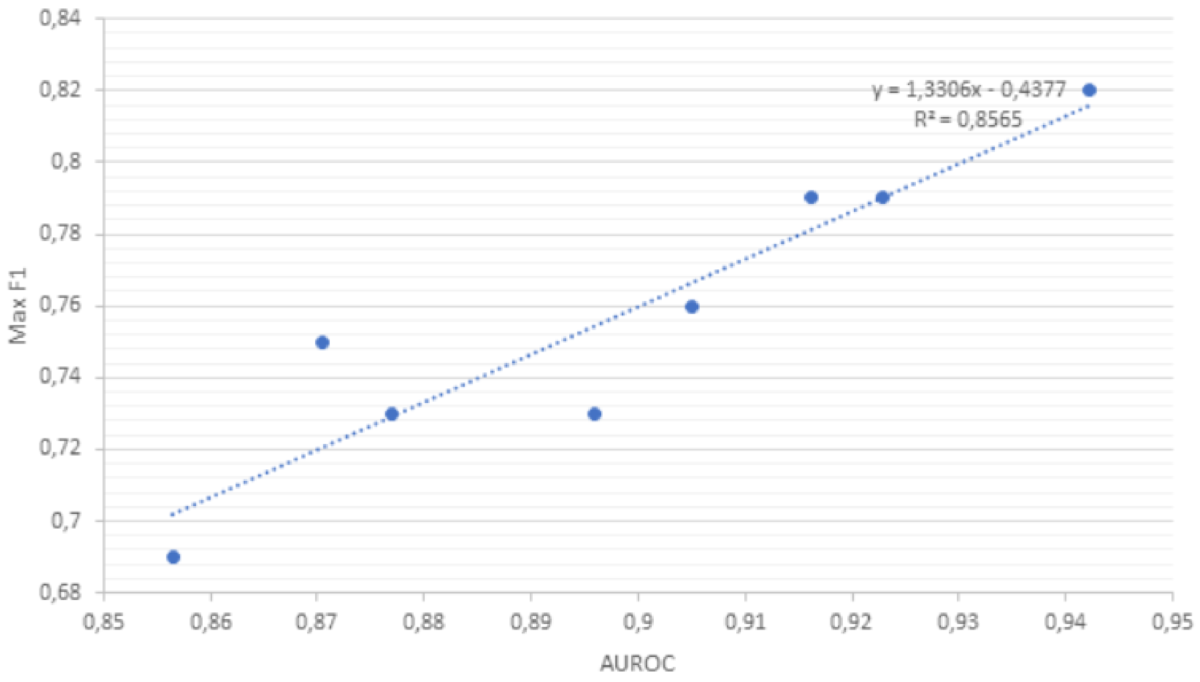


Figure 5.10: Analysis of the relation between the AUROC value and the F1-Score

### 5.2.5. Analysis by overlapping window

After implementing a version of each neural network model and evaluating them by comparing the obtained result to state-of-the-art, the successive step is to evaluate each model by changing the overlapping window size. It consists of changing the temporal view that the single data processed by a model has. Except for ELM-MI and CONV-AE, more in detail, overlapping windows of size 3, 6, 12, 24, and 48 are analyzed, meaning that the small window describes the system behavior for only three seconds while the greater has the longest view of 48 seconds. For ELM-MI, this analysis is not performed since in [81], it is explicitly written that the best results of this model are obtained with a size of 2. Also, for CONV-AE, the windows used are of different sizes than other models. As it is implemented, it requires overlapping windows divisible by four, so the size considered are 8, 24, and 48. Moreover, to have comparable results, independent from the threshold methods and threshold factor, the chosen modality is the MV, assigning the maximum value of the validation score to the threshold. Considering Figure 5.11 and Table 5.3, it is evident that F1-score does not vary much according to the window size. In addition, it has a stable behavior for values like 12 or 24, and the performance slightly decreases for the smallest and maximum sizes.

The only significant changes occur with *VAE+ReEnc* and *VAE+KNN* methods. The two methods behave similarly regarding the *F1-Score* but have opposite behavior concerning the FAR and MAR. Moreover, it can see a significant collapse of the F1-Score with the largest overlapping window size. For *VAE+ReEnc*, it is caused by the small data set in which the threshold is calculated. Having a window of 48 values and a validation set of 50 values, the threshold is calculated on a few values. In detail, the threshold value is very low, detecting a lot of false positives, causing a high FAR and low Precision value. On the other hand, for *VAE+KNN*, since few true positives and a lot of false negatives are detected, the Recall is low and MAR high. What may seem strange is the different behavior of *VAE+KNN* and *VAE+Isof* with a window of size 48, despite the fact they both analyze the same latent space distribution. The different result is due to how *KNN* and *Isolation Forest* work. Since *Isolation Forest* performs recursive division to isolate each value, it is not affected by the number of samples present in the latent space. Contrary, *KNN* is dependent on the parameter $K$, so, having a $K$ similar to the number of validation samples, the model does not have enough points to compute the neighbor distance and classify the test instances correctly.

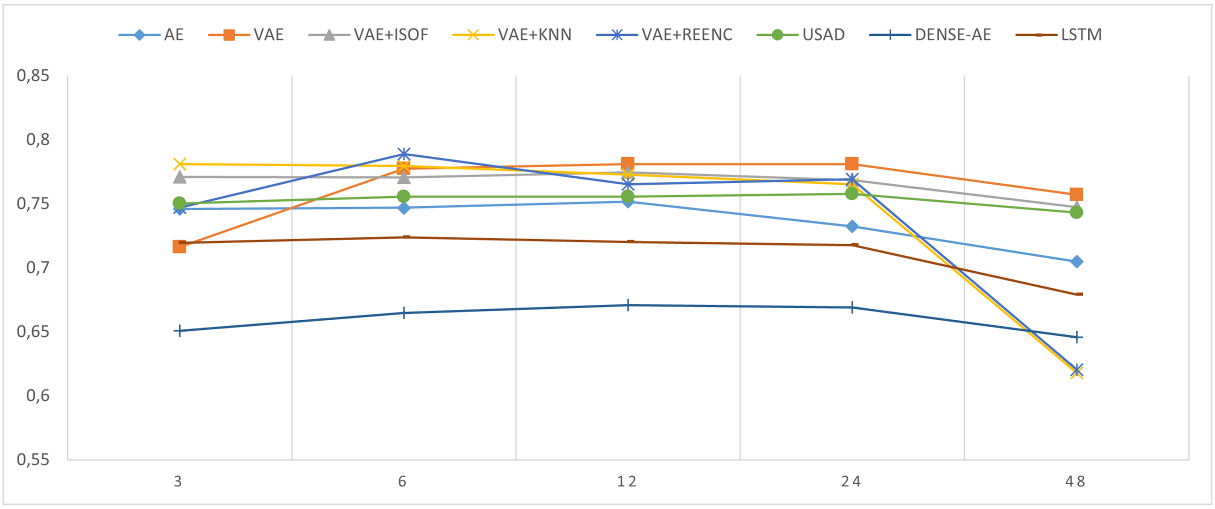


Figure 5.11: Analysis performed on different overlapping window size

To conclude, the window size used in the other sections to obtain all the results is as follows in Table 5.4.

| **Model** | LSTM-AE | VAE | ReEnc | DENSE-AE | CONV-AE | LSTM | USAD | ELM |
|---|---|---|---|---|---|---|---|---|
| **WinSize** | 12 | 12 | 6 | 12 | 8 | 6 | 24 | 2 |

Table 5.4: Window size chosen according to the maximum F1-Score in table 5.3

| MODEL | WINDOWS | F1 | PRECISION | RECALL | FAR | MAR |
|---|---|---|---|---|---|---|
| LSTM-AE | 3 | 0.7459 | 0.7372 | 0.7697 | 11.48% | 27.63% |
| LSTM-AE | 6 | 0.7468 | 0.6879 | 0.8112 | 19.60% | 18.88% |
| LSTM-AE | 12 | **0.7518** | 0.7271 | 0.7784 | 15.48% | 22.16% |
| LSTM-AE | 24 | 0.7323 | 0.6109 | 0.9137 | 30.83% | 8.63% |
| LSTM-AE | 48 | 0.7047 | 0.5692 | 0.9248 | 37.09% | 7.51% |
| VAE | 3 | 0.7163 | 0.7411 | 0.6931 | 12.83% | 30.69% |
| VAE | 6 | 0.7773 | 0.7355 | 0.8243 | 15.71% | 17.57% |
| VAE | 12 | **0.7810** | 0.7360 | 0.8318 | 15.80% | 16.82% |
| VAE | 24 | 0.7809 | 0.6842 | 0.9095 | 21.94% | 9.08% |
| VAE | 48 | 0.7571 | 0.6286 | 0.9519 | 29.80% | 4.81% |
| VAE+ISOF | 3 | 0.7711 | 0.6798 | 0.8907 | 22.22% | 10.93% |
| VAE+ISOF | 6 | 0.7706 | 0.6669 | 0.9124 | 24.15% | 8.76% |
| VAE+ISOF | 12 | **0.7747** | 0.6733 | 0.9133 | 23.45% | 8.67% |
| VAE+ISOF | 24 | 0.7684 | 0.6614 | 0.9168 | 24.86% | 8.32% |
| VAE+ISOF | 48 | 0.7474 | 0.6313 | 0.9157 | 28.33% | 8.43% |
| VAE+KNN | 3 | **0.7811** | 0.7195 | 0.8544 | 17.65% | 14.56% |
| VAE+KNN | 6 | 0.7796 | 0.6952 | 0.8873 | 20.61% | 11.27% |
| VAE+KNN | 12 | 0.7728 | 0.6836 | 0.8914 | 21.68% | 10.68% |
| VAE+KNN | 24 | 0.7651 | 0.6858 | 0.8653 | 21.00% | 13.47% |
| VAE+KNN | 48 | 0.6177 | 0.5369 | 0.7271 | 10.68% | 46.31% |
| VAE+REENC | 3 | 0.7471 | 0.7243 | 0.7714 | 15.56% | 22.86% |
| VAE+REENC | 6 | **0.7888** | 0.7313 | 0.8559 | 16.66% | 14.41% |
| VAE+REENC | 12 | 0.7651 | 0.7043 | 0.8373 | 18.62% | 16.27% |
| VAE+REENC | 24 | 0.7691 | 0.6955 | 0.8601 | 19.96% | 13.99% |
| VAE+REENC | 48 | 0.6202 | 0.8657 | 0.4832 | 49.06% | 13.43% |
| CONV-AE | 8 | **0.8084** | 0.7399 | 0.8908 | 16.59% | 10.91% |
| CONV-AE | 24 | 0.7918 | 0.6934 | 0.9228 | 21.62% | 7.72% |
| CONV-AE | 48 | 0.7646 | 0.6338 | 0.9639 | 29.49% | 3.65% |
| USAD | 3 | 0.7503 | 0.6521 | 0.8834 | 21.82% | 11.66% |
| USAD | 6 | 0.7555 | 0.6612 | 0.8812 | 21.11% | 12.02% |
| USAD | 12 | 0.7556 | 0.6536 | 0.8953 | 21.93% | 11.53% |
| USAD | 24 | **0.7576** | 0.6401 | 0.9278 | 27.64% | 7.21% |
| USAD | 48 | 0.7432 | 0.6123 | 0.9452 | 28.73% | 6.36% |
| DENSE-AE | 3 | 0.6506 | 0.5063 | 0.9098 | 47.20% | 9.02% |
| DENSE-AE | 6 | 0.6647 | 0.5432 | 0.8563 | 36.02% | 14.93% |
| DENSE-AE | 12 | **0.6708** | 0.5576 | 0.8415 | 35.37% | 15.84% |
| DENSE-AE | 24 | 0.6691 | 0.5626 | 0.8254 | 34.25% | 17.02% |
| DENSE-AE | 48 | 0.6457 | 0.5235 | 0.8423 | 37.21% | 15.72% |
| LSTM | 3 | 0.7193 | 0.6551 | 0.7989 | 22.51% | 20.12% |
| LSTM | 6 | **0.7238** | 0.7706 | 0.6824 | 10.76% | 31.76% |
| LSTM | 12 | 0.7201 | 0.6521 | 0.8039 | 22.73% | 19.60% |
| LSTM | 24 | 0.7176 | 0.6166 | 0.8581 | 28.28% | 14.14% |
| LSTM | 48 | 0.6789 | 0.5567 | 0.8873 | 39.45% | 11.25% |

Table 5.3: SKAB analysis according to the size of the windows

### 5.2.6. Threshold

This section analyzes the impact of the four different thresholding techniques on the F1 Score: STD, MAD, IQR, and MV.

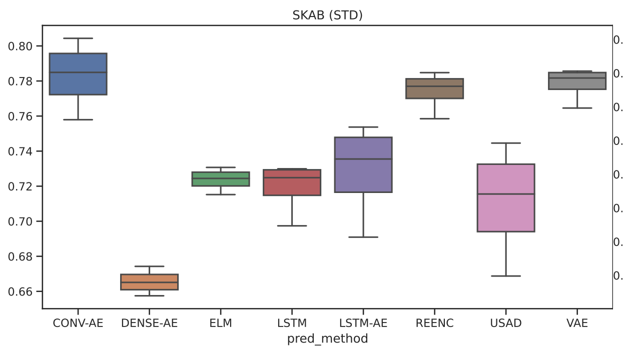
For each threshold technique, the effect that different threshold factors have on the threshold and the overall results are evaluated. IQR, MAD, and STD, since they depend on various thresholding factors, generate several thresholds and results. Further precision regards the F1-Score. Since the SKAB dataset is composed of several test files, the approach chosen is to compute the sum of the confusion matrix value of each file and then compute a single F1-Score. This choice is coherent with the one of [52]. Figure 5.12 compares the F1-Score obtained with these three methods showing similar results. This means that the three ways generate similar threshold values. By comparing the values at line $\Delta$ of Table 5.5, IQR is the approach associated with the lowest difference between the maximum and minimum F1 values for each considered deep learning method. In contrast, the way that, according to the threshold factor, gets the most significant difference between the maximum and the minimum value is STD. This is evident by observing the boxplot where STD shows a broader distribution than IQR. STD and MAD have similar boxplots because they are generated from similar F1-Score. This happens since MAD and STD have correspondent thresholds as reported in Table 5.6

More in detail, in this case [61]:

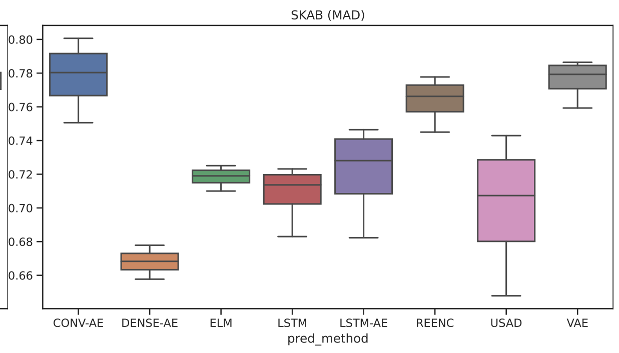
$$mean(s) + std(s) \approx median(s) + 1.4826 \cdot median(|s - median(s)|)$$

Where the $s$ corresponds to the anomaly score. Since the validation score distribution, shown in Section 5.2.1, is almost uniform with no value that deviates consistently from the mean, $mean(s)$ and $median(s)$ coincide. Moreover [88], $std(s)$ and $1.4826 \cdot median(|s - median(s)|)$ almost coincide. So, the result is that the two methods in the SKAB dataset generate a very close threshold, as shown in Table 5.6. Analyzing the table 5.5 and comparing the F1-score corresponding to the threshold method MV to the other, it is noted that for the SKAB dataset, using the maximum value in the validation set is a good choice, obtaining an F1-Score closer to the maximum. It is an applicable thresholding technique on SKAB dataset because, as shown in Section 5.2.1, all the validation scores do not contain values that deviate consistently from the mean. Moreover, as shown in Table 5.6, the thresholds calculated with MV and STD with $th_{factor}$ equal to 2 are similar. Having similar thresholds also, the F1-Score is similar. MV is the method that produces better results for VAE+ReEnc.

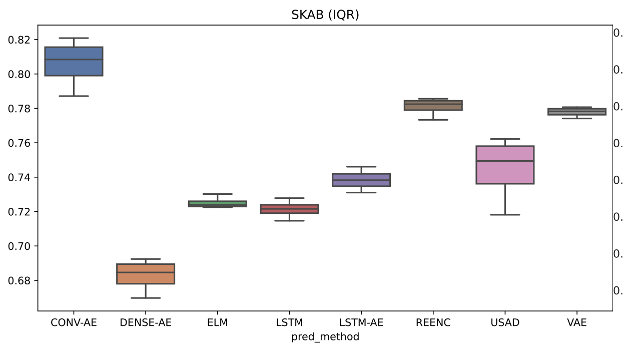
Figure 5.12d, representing the F1-Score behavior according to all the thresholding techniques, shows that MAD, IQR, and STD on the SKAB dataset give similar results.



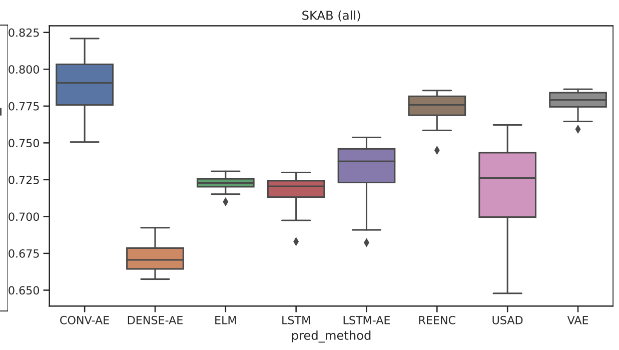
(a) STD threshold technique F1-score results



(b) MAD threshold technique F1-score results



(c) IQR threshold technique F1-score results



(d) Average F1-score results

Figure 5.12: Overall results obtained by computing the threshold in three different ways

| Threshol Method | Th factor | VAE | LSTM-AE | VAE+ReEnc | DENSE-AE | USAD | CONV-AE | LSTM | ELM-MI |
|---|---|---|---|---|---|---|---|---|---|
| IQR | 0.5 | 0.77949 | 0.73597 | 0.77333 | 0.66975 | 0.71814 | 0.78714 | 0.72052 | 0.72460 |
| | 1 | 0.78070 | 0.74608 | 0.78083 | 0.68078 | 0.74219 | 0.80302 | 0.72782 | 0.73019 |
| | 1.5 | 0.77702 | 0.74056 | 0.78405 | 0.68844 | 0.75665 | 0.81383 | 0.72262 | 0.72306 |
| | 2 | 0.77412 | 0.73104 | 0.78557 | 0.69241 | 0.76220 | 0.82086 | 0.71465 | 0.72250 |
| | Δ | 0.00658 | 0.01504 | 0.01224 | 0.02266 | 0.04406 | 0.03373 | 0.01317 | 0.00770 |
| MAD | 0.5 | 0.75930 | 0.68230 | 0.74500 | 0.65771 | 0.64784 | 0.75058 | 0.68300 | 0.71002 |
| | 1 | 0.77460 | 0.71709 | 0.76114 | 0.66520 | 0.69091 | 0.77205 | 0.70880 | 0.71661 |
| | 1.5 | 0.78395 | 0.73905 | 0.77130 | 0.67139 | 0.72375 | 0.78862 | 0.71853 | 0.72142 |
| | 2 | 0.78644 | 0.74644 | 0.77772 | 0.67785 | 0.74292 | 0.80066 | 0.72314 | 0.72509 |
| | Δ | 0.02714 | 0.06414 | 0.03273 | 0.02014 | 0.09508 | 0.05008 | 0.04014 | 0.01507 |
| STD | 0.5 | 0.76457 | 0.69093 | 0.75849 | 0.65747 | 0.66874 | 0.75792 | 0.69738 | 0.71520 |
| | 1 | 0.77885 | 0.72509 | 0.77392 | 0.66211 | 0.70252 | 0.77699 | 0.72058 | 0.72178 |
| | 1.5 | 0.78454 | 0.74590 | 0.78006 | 0.66812 | 0.72854 | 0.79284 | 0.72914 | 0.72708 |
| | 2 | 0.78564 | 0.75371 | 0.78477 | 0.67426 | 0.74458 | 0.80434 | 0.72992 | 0.73071 |
| | Δ | 0.02107 | 0.06279 | 0.02628 | 0.01678 | 0.07584 | 0.04642 | 0.03254 | 0.01550 |
| MAX | | 0.78102 | 0.75184 | 0.78875 | 0.67081 | 0.75759 | 0.80843 | 0.72384 | 0.73056 |
| | minF1 | 0.75930 | 0.68230 | 0.74500 | 0.65747 | 0.64784 | 0.75058 | 0.68300 | 0.71002 |
| | maxF1 | 0.78644 | 0.75371 | 0.78875 | 0.69241 | 0.76220 | 0.82086 | 0.72992 | 0.73071 |
| | Δ Tot | 0.02714 | 0.07142 | 0.04375 | 0.03493 | 0.11435 | 0.07028 | 0.04692 | 0.02069 |

Table 5.5: F1-score comparison between different threshold technique and threshold factor

| Method | Th_factor | VAE | LSTM-AE | VAE+ReEnc | DENSE-AE | USAD | CONV-AE | LSTM | ELM-MI |
|---|---|---|---|---|---|---|---|---|---|
| IQR | 0.5 | 0.04191 | 0.04768 | 0.00086 | 0.04037 | 0.05280 | 0.02205 | 0.06864 | 0.01946 |
| | 1 | 0.04871 | 0.05478 | 0.00105 | 0.04477 | 0.05861 | 0.02472 | 0.08450 | 0.02348 |
| | 1.5 | 0.05552 | 0.06188 | 0.00124 | 0.04917 | 0.06442 | 0.02740 | 0.10036 | 0.02751 |
| | 2 | 0.06232 | 0.06899 | 0.00144 | 0.05356 | 0.07023 | 0.03007 | 0.11622 | 0.03153 |
| MAD | 0.5 | 0.03215 | 0.03766 | 0.00056 | 0.03396 | 0.04315 | 0.01857 | 0.04667 | 0.01452 |
| | 1 | 0.03697 | 0.04263 | 0.00068 | 0.03702 | 0.04672 | 0.02061 | 0.05812 | 0.01754 |
| | 1.5 | 0.04178 | 0.04761 | 0.00080 | 0.04008 | 0.05029 | 0.02265 | 0.06958 | 0.02056 |
| | 2 | 0.04660 | 0.05258 | 0.00091 | 0.04314 | 0.05387 | 0.02468 | 0.08104 | 0.02358 |
| STD | 0.5 | 0.03385 | 0.03907 | 0.00064 | 0.03443 | 0.04620 | 0.01908 | 0.05094 | 0.01378 |
| | 1 | 0.03875 | 0.04414 | 0.00078 | 0.03733 | 0.05064 | 0.02122 | 0.06260 | 0.01617 |
| | 1.5 | 0.04365 | 0.04920 | 0.00093 | 0.04023 | 0.05508 | 0.02336 | 0.07425 | 0.01856 |
| | 2 | 0.04855 | 0.05426 | 0.00107 | 0.04313 | 0.05953 | 0.02549 | 0.08591 | 0.02095 |
| MAX | | 0.05096 | 0.05593 | 0.00128 | 0.04279 | 0.05857 | 0.02675 | 0.09492 | 0.01936 |

Table 5.6: Threshold comparison between different threshold techniques and threshold factor. The threshold value in the table corresponds to the average of all files comprises in the dataset

## 5.2.7. VAE vs ReEncoding

This section analyzes the method of Variational Autoencoder with re-encoding according to the different parameters, $\alpha$ and $\beta$, of its scoring function:

$$score(x) = \alpha||x - x'||_2 + \beta||z - z'||_2$$

The first term corresponds to the score generated by the difference between the input and the reconstructed output, while the difference between the latent space and the latent space by re-encoding gives the second as described in Section 3.3.6.4.

The score with $\alpha = 1$ corresponds to the *VAE* method, while the score with $\beta = 1$ is the *VAE+ReEnc* method. To have comparable results, the same thresholding technique is chosen. In this specific case, the analysis is performed by considering the MV method, assigning to the threshold the maximum value of the validation set.

As Table 5.7 and Figure 5.13 shows, the value of *F1-Score* slightly increases according to the increase of the $\beta$ parameter. The increment is minimal as the extreme cases have little different results.

In other words, the more significant the re-encoding part contributes, the greater the F1-Score. Concerning the *precision* is almost constant while the *recall* follows the *F1-Score* trend reaching the maximum value with the maximum $\beta$.

So, in all the sections where VAE+ReEnc appears, the score considered is the one with

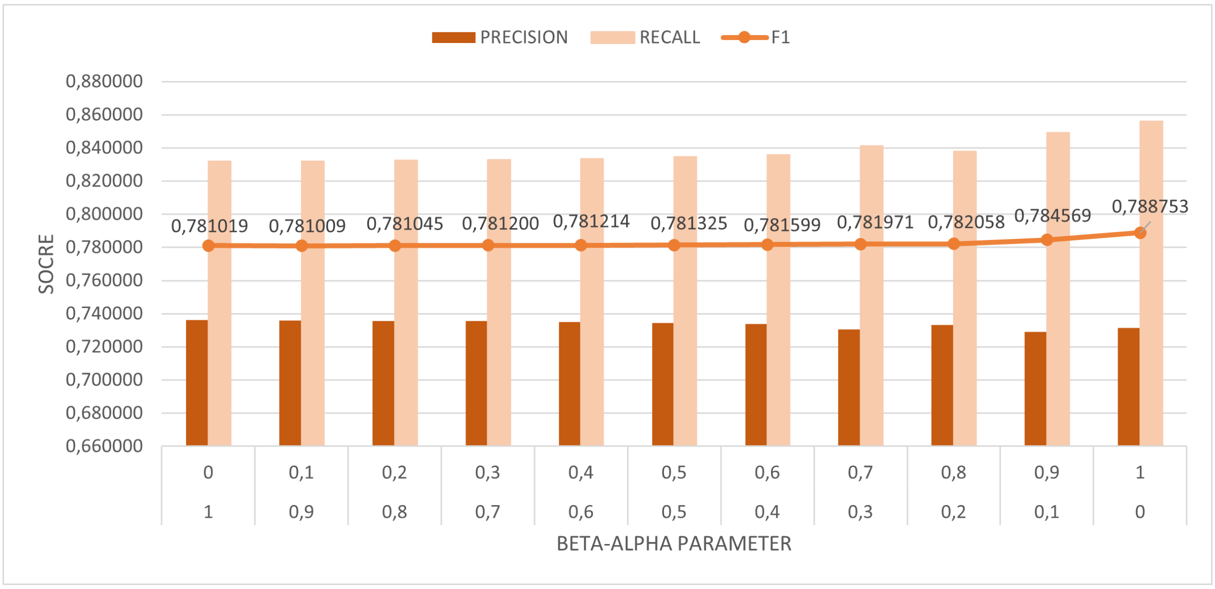


Figure 5.13: Re-Encoding analysis according to $\beta$ and $\alpha$ parameter. It is considered the same input reconstruction $(x')$ and the same latent space reconstruction $(z')$. The only parameters that changes are the ones in the score function.

$\beta = 1$.

| **Alpha** | **Beta** | **F1-score** | **Precision** | **Recall** | **FAR** | **MAR** |
|---|---|---|---|---|---|---|
| 1 | 0 | 0.781019 | 0.736064 | 0.831822 | 15.80% | 16.82% |
| 0.9 | 0.1 | 0.781009 | 0.735862 | 0.832059 | 15.82% | 16.79% |
| 0.8 | 0.2 | 0.781045 | 0.735679 | 0.832374 | 15.85% | 16.76% |
| 0.7 | 0.3 | 0.781200 | 0.735585 | 0.832847 | 15.86% | 16.72% |
| 0.6 | 0.4 | 0.781214 | 0.735118 | 0.833478 | 15.91% | 16.65% |
| 0.5 | 0.5 | 0.781325 | 0.734457 | 0.834582 | 15.99% | 16.54% |
| 0.4 | 0.6 | 0.781599 | 0.733906 | 0.835922 | 16.06% | 16.41% |
| 0.3 | 0.7 | 0.781971 | 0.730472 | 0.841284 | 16.45% | 15.87% |
| 0.2 | 0.8 | 0.782058 | 0.733200 | 0.837893 | 16.15% | 16.21% |
| 0.1 | 0.9 | 0.784569 | 0.729162 | 0.849089 | 16.71% | 15.09% |
| 0 | 1 | 0.788753 | 0.731339 | 0.855949 | 16.66% | 14.41% |

Table 5.7: SKAB ReEncoding analysis according to $\alpha$ and $\beta$ value

### 5.2.8. Thesis vs. State-of-Art

In this section, it is performed an analysis of the results obtained by comparing all the metrics. The analysis can be split into two parts; the first consists of comparing the results obtained in the thesis test to those available in the SKAB repository. The second consists of a comparison of all the available results. The comparison with the SKAB repository comprises both metrics evaluation and training execution time. As Figure 5.14 shows, the model reimplemented performs better than the one implemented in the original repository. However, the values are very different for several reasons:

- **Normalization** vs. **Standardization**: the way to scale data is normalization rather than standardization, which is used in the SKAB repository. This method is chosen as it is better suited for a multivariate time series containing data from different sensors with very different scales. By normalization, it is possible to make sure that all values are scaled in the same range and make the same contribution to the final anomaly score. This improvement is evident by analyzing the *Isolation Forest* results being the only change applied and increases the F1-Score by ten percentual points.
- **Number of Epochs**: it is another relevant difference. In the model implemented in the thesis, the training is performed for 400 epochs, while in the original implementation, the training is performed for a few epochs. This impacts the total execution time of the training but allows better prediction and reconstruction.
- **Different Overlapping Window size**: In the original implementation, the only window analyzed is that long ten timestamps. In the thesis implementation, the analysis is performed on different window sizes described in Section 5.2.5.
- **Different Model Parameters** A relevant change is on model implementation and parameters used as shown in Table 5.8. Several implementations of each model are tested to get better results, and by comparing the results, the one with the best metric value is kept.
- **Different Threshold Technique** The technique used in the original implementation is the IQR technique. First, the quantile at 99 percent on the training and validation set is calculated and then set and assigned as a threshold the three means of the quantile value. The thesis implementation used a different approach. Different thresholding techniques are applied on the same validation set: MAD, IQR, STD, and MV, described at the beginning of Section 5.2. The results in Table 5.9 correspond to the best one obtained by comparing the different threshold techniques.

Concerning the total training time, the model implemented performers worst since the training lasts for many more epochs. Therefore, implementing a model that gives better quality results was preferred at the sacrifice of time. Regarding the training time for each

| | Repository implementation | | My implementation | |
|---|---|---|---|---|
| MODEL | STRUCTURE | W | STRUCTURE | W |
| Conv-AE | conv1D(32)<br>conv1D(16)<br>latentSpace: dense(16)<br>conv1Dtranspose(16)<br>conv1Dtranspose(32) | ✓ | conv1D(64)<br>conv1D(32)<br>latentSpace: dense(32)<br>conv1Dtranspose(32)<br>conv1Dtranspose(64) | ✓ |
| LSTM-AE | lstm(100)<br>latentSpace: dense(100)<br>lstm(100) | ✓ | lstm(128)<br>lstm(64)<br>latentSpace: dense(32)<br>lstm(64)<br>lstm(128) | ✓ |
| VAE | lstm(32)<br>latentSpace(100)<br>lstm(32) | ✓ | lstm(128)<br>lastm(64)<br>latentSpace: dense(32)<br>lstm(64)<br>lstm(128) | ✓ |
| LSTM | lstm(100)<br>lstm(100) | ✓ | lstm(100)<br>lstm(64) | ✓ |
| DENSE-AE | dense(5)<br>dense(4)<br>latentSpace: dense(2)<br>dense(4)<br>dense(5) | ✗ | dense(64)<br>dense(32)<br>latentSpace: dense(16)<br>dense(32)<br>dense(64) | ✓ |

Table 5.8: Comparison between the structure of the implemented model and the SKAB repository ones. The *W* coloumn indicates if the model input has to be in overlapping window configuration or not

epoch, the difference is minimal. First, all the values are measured using the same machine for comparable results. For most models, the training time of the implemented version is slightly greater than the original one as the model complexity increases. They give better metrics results, but there is a relevant impact on the training time. The execution time of *isolation forest* implementations is the same in both versions since the only change is in the method used to scale data. The only improvement in the training time concerns the VAE. The thesis implementation is faster since it uses a more compressed latent space. Having $\mu$ and $\sigma$ vector about half the size, the sampling process to generate the latent space distribution is faster, significantly reducing the total training execution time.

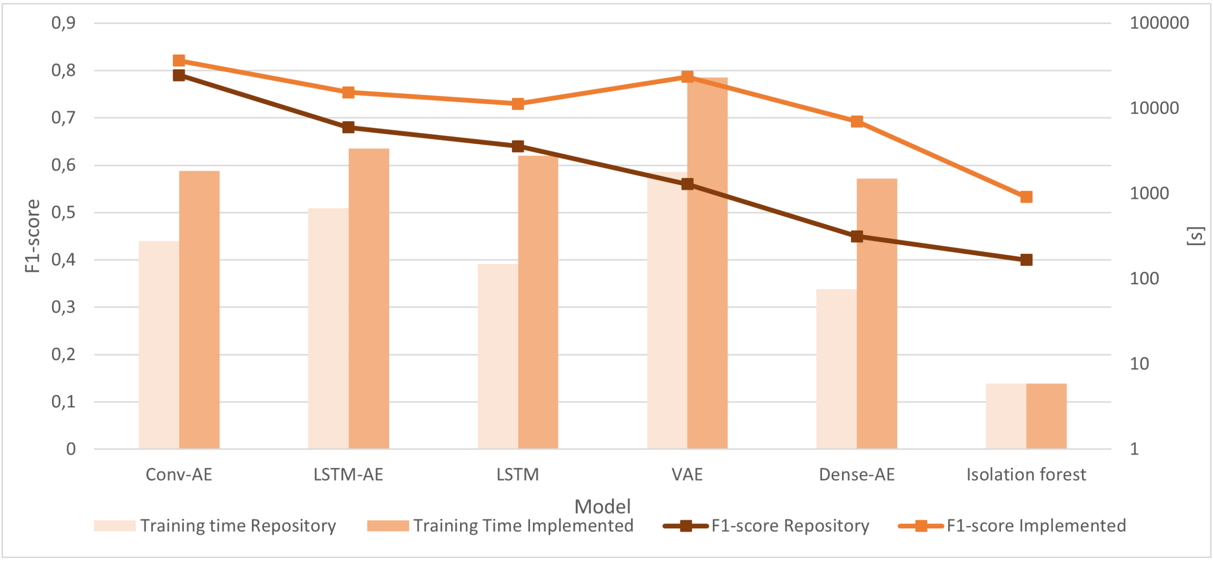


Figure 5.14: It represents the difference between the implemented and available models on the SKAB repository. It contains both F1-Score and the total training time. To show time of different orders of magnitude, a logarithmic scale is used

| | Repository Result | | | My Result | | |
|---|---|---|---|---|---|---|
| Model | F1 | FAR | MAR | F1 | FAR | MAR |
| CONV-AE | 0.79 | 13.69% | 17.77% | 0.8209 | 14.40% | 11.46% |
| LSTM-AE | 0.68 | 14.24% | 35.56% | 0.7537 | 18.15% | 18.80% |
| LSTM | 0.64 | 15.40% | 39.93% | 0.7299 | 14.15% | 27.18% |
| VAE | 0.56 | 9.13% | 55.03% | 0.7864 | 17.82% | 13.40% |
| Dense-AE | 0.45 | 7.56% | 66.57% | 0.6924 | 28.16% | 18.91% |
| Isolation Forest | 0.4 | 15.40% | 40.33% | 0.5327 | 6.19% | 59.04% |

Table 5.9: Comparison of thesis result and SKAB state of art

| | Repository Result | | | My Result | | |
|---|---|---|---|---|---|---|
| **Model** | **Time [s]** | **Epochs** | **[s/Epoch]** | **Time [s]** | **Epochs** | **[s/Epoch]** |
| Conv-AE | 277.37 | 100 | 2.77 | 1849.85 | 400 | 4.62 |
| LSTM-AE | 672.32 | 100 | 6.72 | 3361.6 | 400 | 8.4 |
| LSTM | 149.98 | 25 | 6 | 2773.81 | 400 | 6.93 |
| VAE | 1800.93 | 20 | 90.05 | 22932 | 400 | 57.33 |
| Dense-AE | 75.84 | 40 | 1.9 | 1503.52 | 400 | 3.76 |
| Isolation Forest* | 5.87 | - | - | 5.87 | - | - |

Table 5.10: Comparison of execution time between thesis and SKAB state of art. (*The training is not executed by epochs)

An important consideration must be done about the maximum *F1-Score* value available on state of the art. The best result is calculated in the paper [81] with the ELM-MI method. During the implementation of the thesis, this value is verified, and the conclusion is that the results are computed not considering statistical information on a validation set but directly on the test set. In the paper, to compute the threshold, they find the value that maximizes the *F1-Score* by iterating on test data to find the best threshold.

| **Model** | **F1-Score** | **FAR** | **MAR** |
|---|---|---|---|
| ELM-MI paper implementation | 87.51 | 7.85% | 11.04% |

## 5.2.9. Summary Results

This section compares the overall and summary result of the machine learning methods analyzed and implemented. Concerning the *F1-Score*, as the histogram in Figure 5.15 and Table 5.11 show, the results are similar and within a range of 10 percentage points are contained the results of eight models. This happens since the threshold method is the same and calculated on a small validation set. The model reconstructs or predicts a time series in more or less precise ways, and the threshold values vary according to it. If the model reconstructs the input in an approximative way, the threshold assumes a value greater than a threshold of a more precise model. The model that gives better results is the *CONV-AE* while *VAE* and *LSTM-AE*, having a similar configuration with only a few differences in the latent space, yield the same result. As expected from the analysis carried out in the section 5.2.7, the model *VAE+ReEnc* performs better than simple *VAE*. When dealing with an anomaly detection problem, important considerations must be made on false and missing alarm rate values, where low values are better. All models have similar *FAR* and the better corresponds to the better F1-Score. DENSE-AE has a FAR of about 30% because it is the method that reconstructs the input data in the worst way and a lot of normal data are considered anomalous. By analyzing the *MAR* value, the best one is *ELM-MI*, the model that allows the detection of almost all anomalies. It behaves in this way essentially for a reason. After obtaining an anomaly score where each value corresponds to the score of each timestamp, a smooth function is applied. It consists of propagating the anomaly score of a timestamp to all the scores with a nearby timestamp according to the smooth parameter. This has the effect of increasing the number of anomalies detected, obtaining a high FAR and a low MAR.

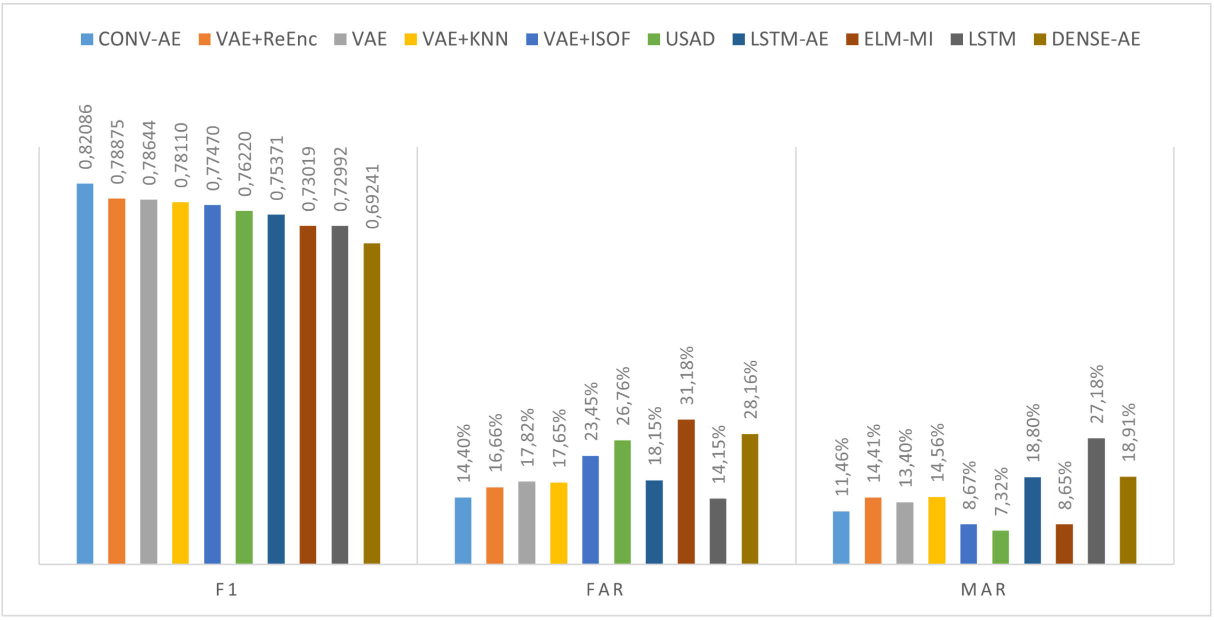


Figure 5.15: Summary results of SKAB dataset

| MODEL | F1 | PRECISION | RECALL | FAR | MAR |
|---|---|---|---|---|---|
| CONV-AE | 0.82086 | 0.76513 | 0.88536 | 14.40% | 11.46% |
| VAE+ReEnc | 0.78875 | 0.73134 | 0.85595 | 16.66% | 14.41% |
| VAE | 0.78644 | 0.72029 | 0.86596 | 17.82% | 13.40% |
| VAE+KNN | 0.78110 | 0.71950 | 0.85440 | 17.65% | 14.56% |
| VAE+ISOF | 0.77470 | 0.67330 | 0.91330 | 23.45% | 8.67% |
| USAD | 0.76220 | 0.64723 | 0.92683 | 26.76% | 7.32% |
| LSTM-AE | 0.75371 | 0.70327 | 0.81195 | 18.15% | 18.80% |
| ELM-MI | 0.73019 | 0.60816 | 0.91351 | 31.18% | 8.65% |
| LSTM | 0.72992 | 0.73162 | 0.72822 | 14.15% | 27.18% |
| DENSE-AE | 0.69241 | 0.60411 | 0.81093 | 28.16% | 18.91% |
| ISOF | 0.50000 | 0.74531 | 0.38212 | 15.40% | 40.33% |

Table 5.11: SKAB summary results

The models with better *F1-Score*, as expected, are the ones that better balance the percentage of MAR and FAR and so Precision and Recall.

## 5.3. Exathlon

### 5.3.1. Analysis of the anomaly score distribution on the validation set

This section discusses the different distributions of the validation score used to compute the threshold. Contrary to the SKAB dataset, which has a validation set for each file, the Exathlon dataset has a unique validation on which a single threshold is computed to evaluate each test score. The validation score distribution, obtained from the different neural network models, is shown in Figure 5.16.

The validation score distribution is very different from the SKAB one. It is not a unimodal distribution. Moreover, it is noticed that the presence of extreme values made the MV value thresholding technique extremely ineffective. For all models, excluding ELM-MI, the validation scores assume similar distribution, having many concentrated values for low scores and few and sparse for high score values assuming a distribution nearly exponential.

The effect is that the USAD thresholds are greater than the ones calculated on the score generated by other models. ELM-MI is very different; the score assumes greater values than the others but less than 0.5, with a completely opposite distribution.

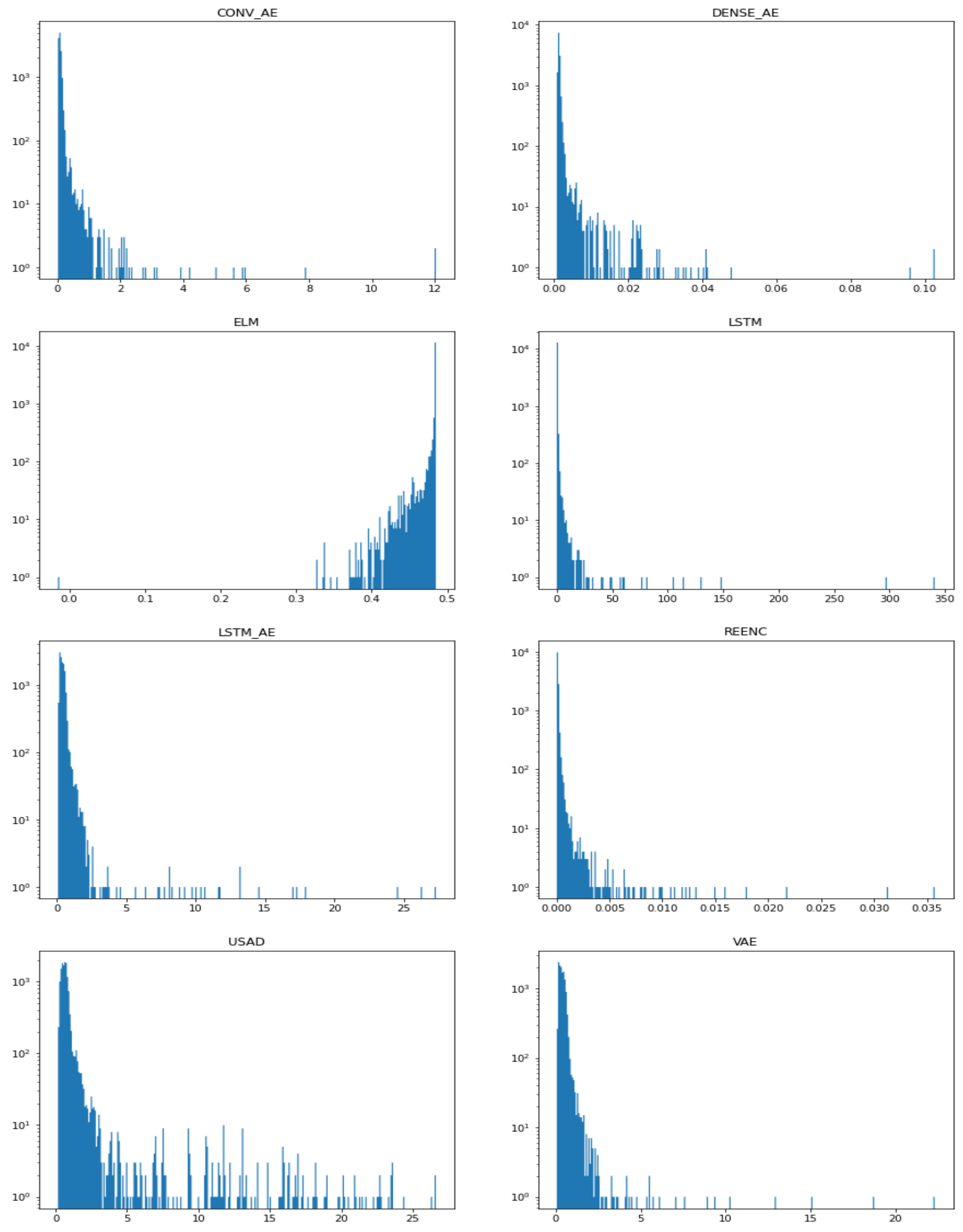


Figure 5.16: Validation scores computed with all the neural network models. To better show all values, the axis y uses a logarithmic scale

## 5.3.2. Analysis of the anomaly score distribution on the test set

This section focuses on analyzing the score distribution of the test set. Having several test sets containing different types of anomalies, the scores assume different distributions according to the anomalies contained.

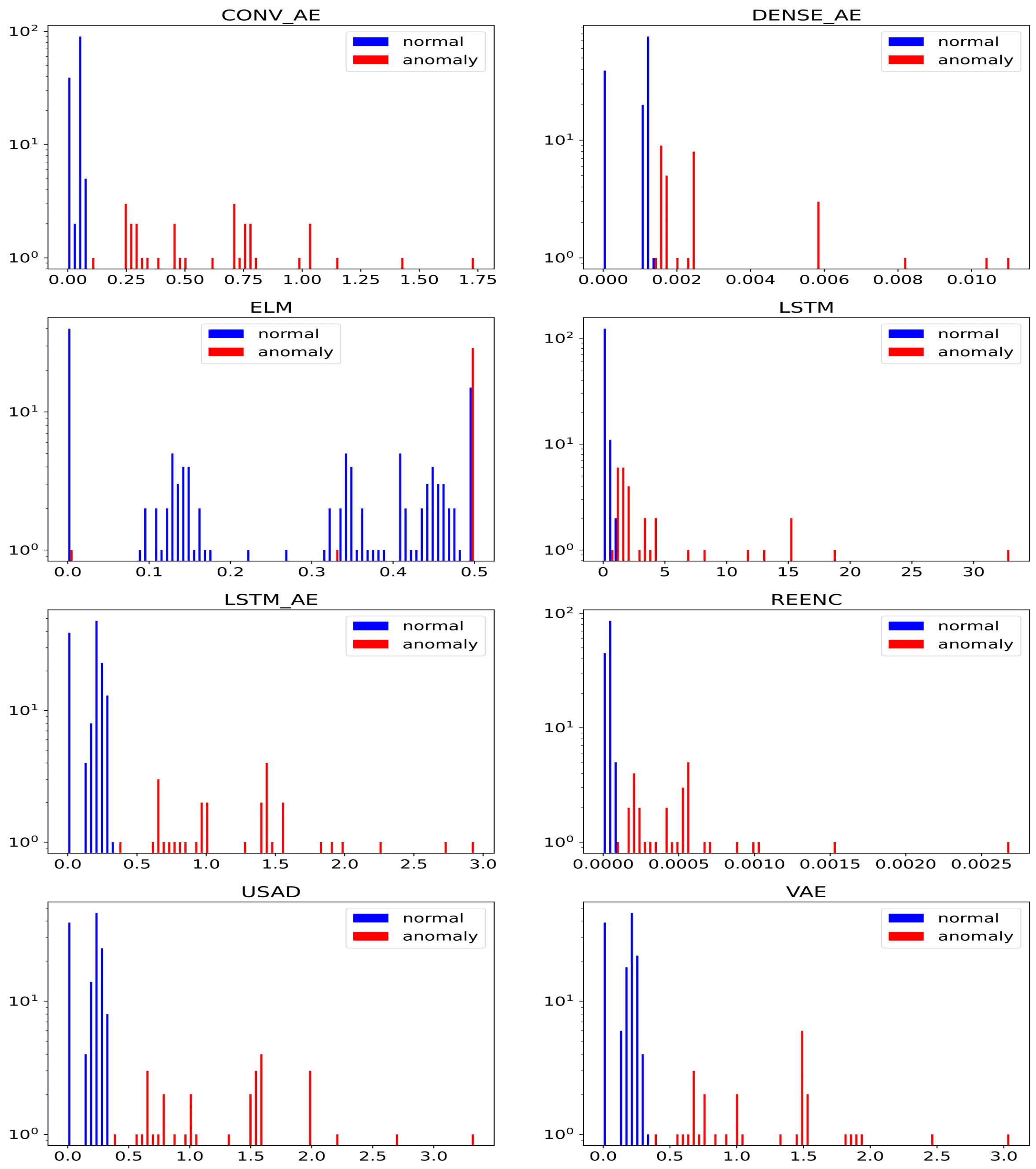


Figure 5.17: Exathlon test score containing anomaly of type **T1** with a visible distinction between normal and anomalous data

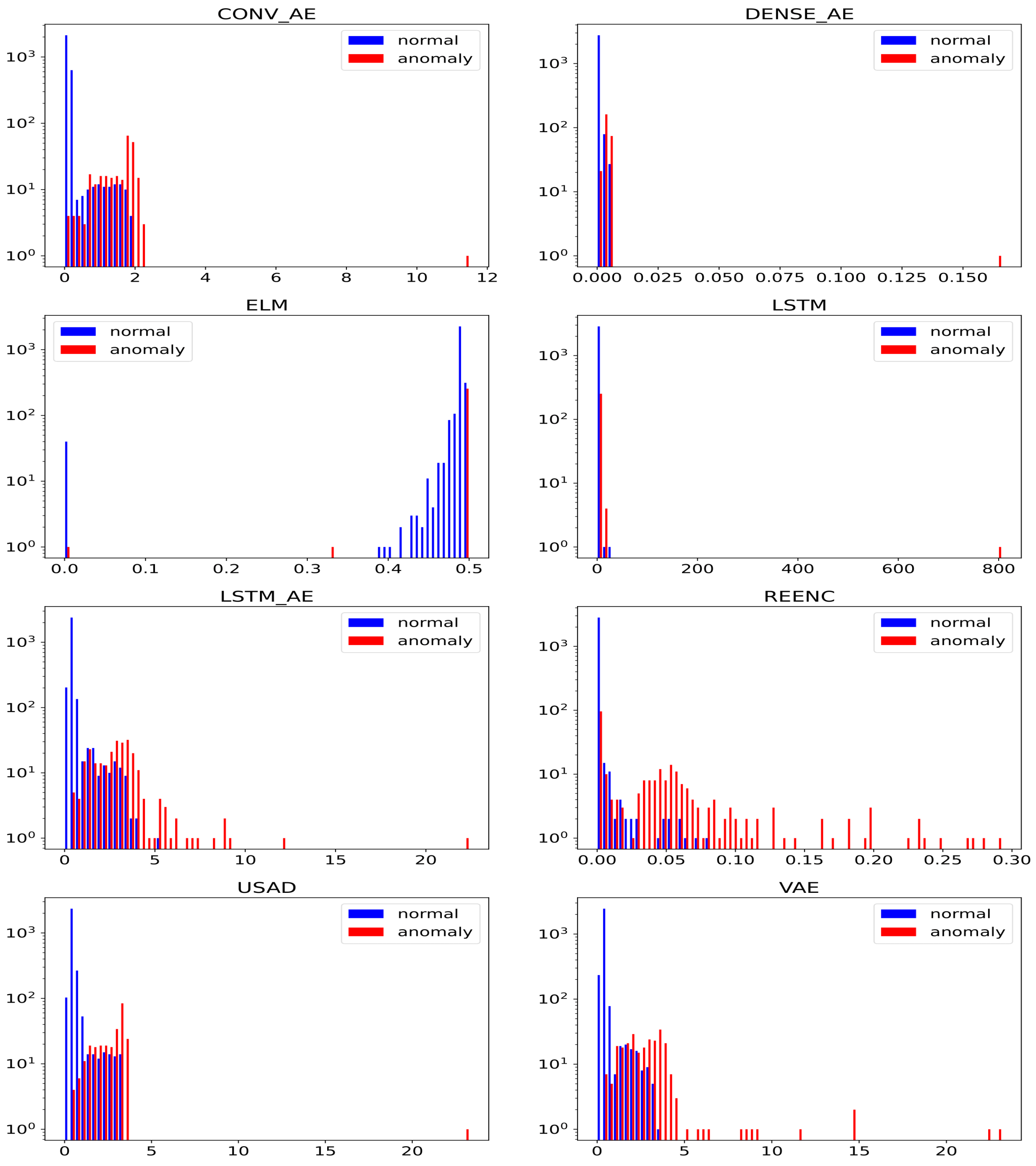


Figure 5.18: Exathlon test score containing anomaly of type **T4** with no distinction between normal and anomalous data

Figure 5.17 represents the score obtained by processing a test file which contains anomaly of type **T1**, Bursty Input. The distributions obtained are not properly bimodal, but normal and anomalous data are well-separated, and it is easy, by applying a threshold, to classify each sample correctly. All the distributions are similar except the ELM-MI one, which is slightly different from the others. All the anomalous values assume closer and not greater than 0.5. This is a consequence of the ELM-MI implementation where the score is limited to 0.5 according to [81]. Not all test distributions follow this behavior. Analyzing Figure 5.20, which contains anomaly of type **T5**, it is noticed the score does not assume a bimodal distribution, and no clear separation between normal and anomalous data is visible. This happens because anomalous and normal data assume similar score values; consequently, the respective distributions overlap. Therefore, finding an optimal threshold is more difficult. In addition, many false positives are detected using a low threshold, and the false negative rate increases using a high threshold value. In that case, the model able to better separate anomalies from normal data is ELM-MI, where anomalous scores are mapped closer and assume a value similar to 0.5. To confirm this, ELM-MI is the method with the best AUROC value, visible in Table 5.12 and analyzed in the following section.

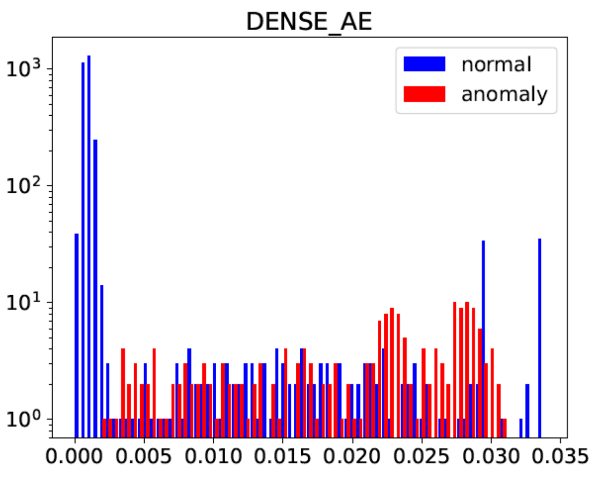
## 5.3.3. AUROC

Models behave in different ways according to the ROC curve. By analyzing the area under the curve, the AUROC value shown in Table 5.12, some models can better separate anomalies from normal values than others.

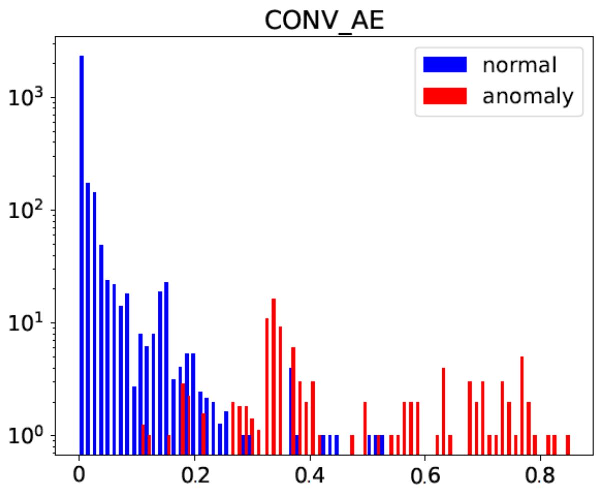
| **METHOD** | ELM-MI | Conv-AE | ReEnc | LSTM | USAD | VAE | LSTM-AE | DENSE-AE |
|---|---|---|---|---|---|---|---|---|
| **AUROC** | 0.91557 | 0.89538 | 0.8880 | 0.87921 | 0.89079 | 0.85385 | 0.84996 | 0.75564 |

Table 5.12: AUROC value for each method

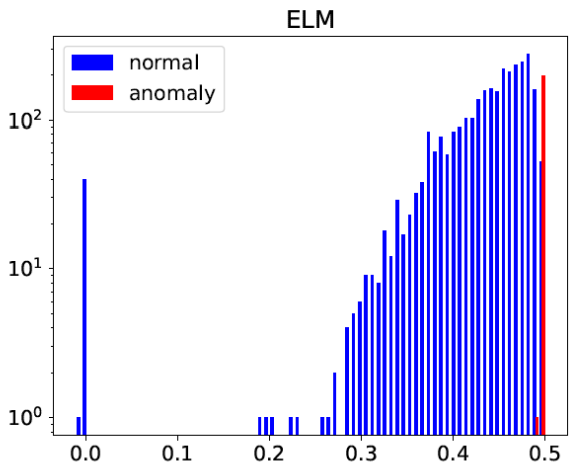
To begin with, it is useful to analyze the best and worst AUROC obtained by autoencoder-based models and then compare them to the one with the best AUROC. To understand why DENSE-AE and CONV-AE have so different AUROC, comparing the score distribution obtained with the two methods is necessary. Figure 5.19 represents the score distribution of the same file processed with the three models. Comparing CONV-AE and DENSE-AE, CONV-AE separates anomalies from normal data cleanly. So it is possible to get a high AUROC score. Concerning the DENSE-AE score, separating anomalous and normal data is tricky, and getting a good AUROC score is impossible. The method with the best AUROC is ELM-MI. Comparing the anomaly score distribution in Figure 5.19, ELM-MI and CONV-AE have a slightly different separation between normal and anomalous data. For ELM-MI, unlike CONV-AE, anomalous and normal scores are not mixed and concentrated in a unique bar in the histogram. As a consequence, ELM-MI has a better AUROC than CONV-AE.



(a) Test score obtained from DENSE-AE



(b) Test score obtained from CONV-AE



(c) Test score obtained from ELM-MI

Figure 5.19: Anomaly score distribution showing the value assumed by normal and anomalous value

Comparing AUROC and the maximum F1-Score achievable with different models, it is clear that high AUROC corresponds to high F1-Score as visible in Figure 5.20. For ELM-MI, the method with the best F1-Score, the correlation between F1-Score and AUROC is above the average. The explanation regards the different score distributions of other methods and ELM-MI. All score distributions, excluding ELM-MI, can assume any value greater than zero. Instead, the ELM-MI score distribution is limited to 0.5. For all values greater than 0.5, assume that value. This affects the distribution parameter, such as the mean. The consequence is that according to ELM-MI or the other models, the same threshold technique assumes a completely different meaning. In this case, the methods STD of ELM-MI works particularly well, generating a correlation between F1-Score and

AUROC above the average. The results might improve by applying other thresholding techniques to the other method, generating a correlation between F1-Score and AUROC similar to ELM-MI.

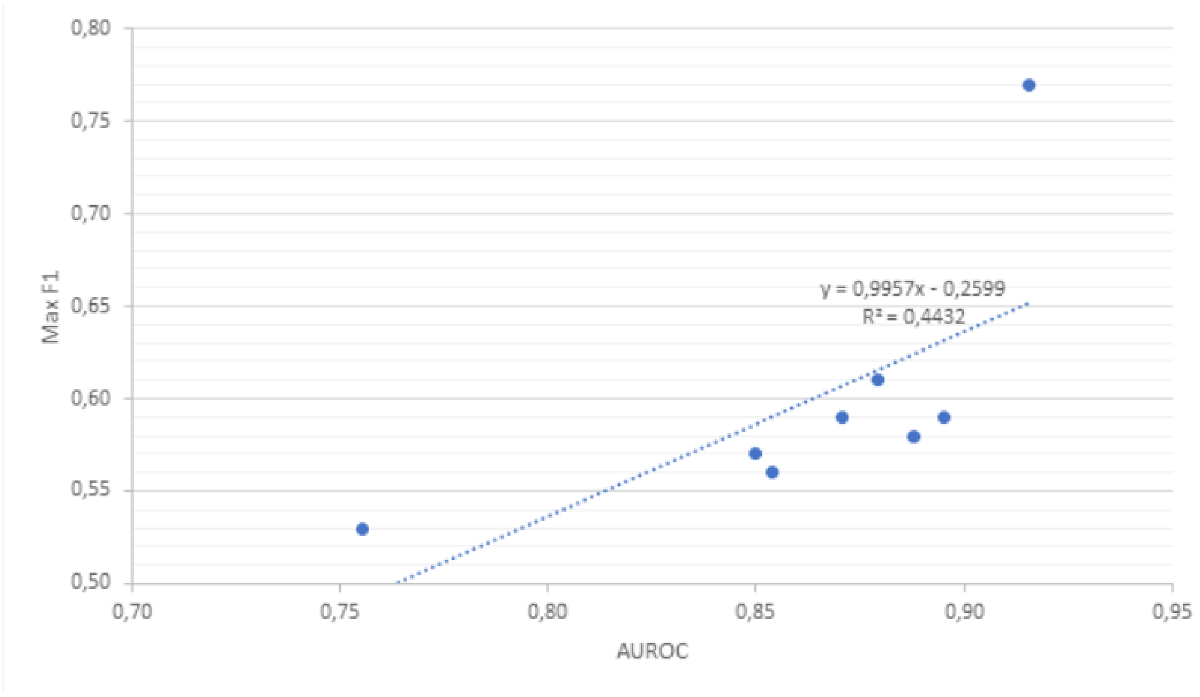


Figure 5.20: Analysis of the relation between the AUROC value and the F1-Score

### 5.3.4. Threshold

This section analyzes how the F1-Score according to evaluating the anomaly score with the four different thresholding techniques, MAD, IQR, STD, and MV. Each threshold technique, excluding MV, also explores the effect of different threshold factors on the threshold and the overall results. IQR, MAD, and STD, since they depend on various thresholding factors, generate several thresholds and results. Figure 5.21 compares the F1-Score obtained with these three methods showing similar results. IQR is the method that produces the lowest variance in the F1-Score, while STD is the most sensitive to the threshold factor having the greater difference between maximum and minimum F1-Score for all methods as visible in Table 5.15.

Contrary to what happens for the SKAB dataset, MAD and STD give different results. This is caused by the difference between the mean and the median of the validation score.

| | VAE | LSTM-AE | VAE+ReEnc | DENSE-AE | USAD | CONV-AE | LSTM | ELM-MI |
|---|---|---|---|---|---|---|---|---|
| Mean | 0.41998 | 0.43112 | 0.00017 | 0.00161 | 0.89105 | 0.10814 | 0.52539 | 0.48218 |
| Median | 0.36782 | 0.36545 | 0.00010 | 0.00131 | 0.64188 | 0.07872 | 0.22578 | 0.48499 |

Table 5.13: Validation score mean and median comparison

Table 5.13 shows that, excluding ELM-MI, the mean and median deviate considerably. This is caused by some values in the validation score that assume a very high value and influence the mean value. This does not happen for the median since it corresponds to the central value of the score increased ordered, which does not suffer the effect of very high values. ELM-MI is the only model that not follows this rule because the score is limited to 0.5 by design [81]. Despite this, MAD and STD generate very different thresholds since:

$$std(s) >> 1.4826 \cdot median(|s - median(s)|)$$

For LSTM and USAD, the MAD method works better than STD, while ELM-MI generates good results only if evaluated with the STD model. The threshold is too small with IQR and MAD to detect anomalies well. Moreover, it is a model very sensitive to the threshold value, and a slight threshold variation corresponds to a significant change in F1-Score as visible by comparing Table 5.15 which contains the F1-Score and the Table 5.16 which contains the threshold for each method. Moreover, the ELM-MI score is limited to 0.5 due to a design choice in [81]. Since having a threshold greater than the score has no sense, the threshold is limited too, as happens for the method STD with $th\_facror = 2$.

Table 5.15 shows that the MV method is ineffective on this dataset, except ELM-MI, where the score, limited to 0.5, does not contain values that deviate consistently from the score mean. As shown in Section 5.3.1, all other validation distributions have some extreme value that makes the threshold too high to find anomalies. This is supported by the results in Table 5.14 where are reported the $N$, correspondent to the number of standard deviations necessary to verify the equation:

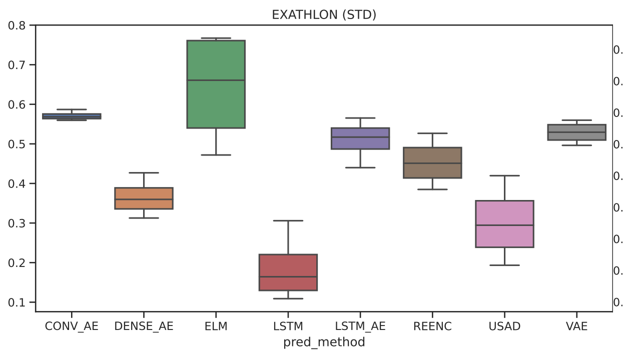
$$max(s) = mean(s) + N \cdot std(s)$$

ELM-MI does not follow this rule, having a $N$ lower than one, because validation and test scores are limited to 0.5 as [81].

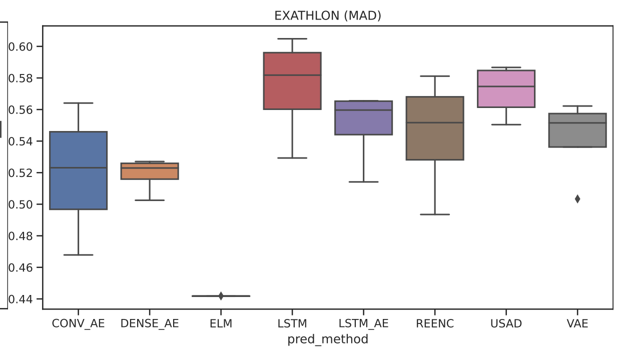
| | VAE | LSTM-AE | ReEnc | DENSE-AE | USAD | CONV-AE | LSTM | ELM-MI |
|---|---|---|---|---|---|---|---|---|
| SKAB | 2.25 | 2.16 | 2.77 | 1.94 | 1.89 | 2.15 | 2.39 | 1.67 |
| EXATHLON | 50.62 | 42.68 | 52.76 | 41.94 | 15.29 | 51.26 | 70.83 | 1.25 |

Table 5.14: Comparison between SKAB and Exathlon of the number of standard deviation needed. that summed to the mean. to reach the same value as the maximum value of the score

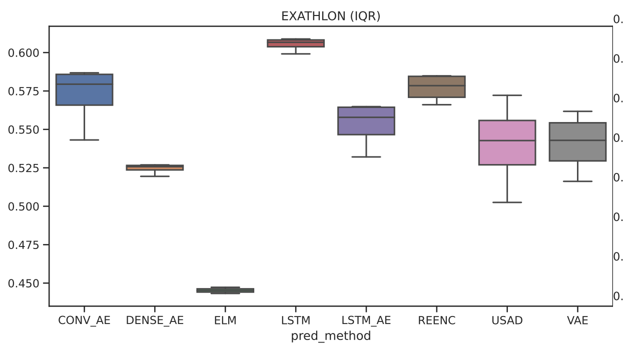
Analyzing Figure 5.21d, which represents the F1-Score behavior according to the thresholding technique, it is noticed that the results on the Exathlon dataset are highly dependent on the threshold, and the model more threshold sensitive is LSTM.



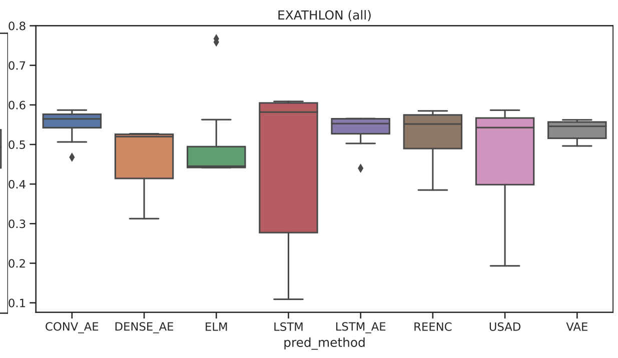
(a) STD threshold technique F1-score results



(b) MAD threshold technique F1-score results



(c) IQR threshold technique F1-score results



(d) Average F1-score results

Figure 5.21: Overall results obtained by computing the threshold in three different ways

| Threshol Method | Th factor | VAE | LSTM-AE | VAE+ReEnc | DENSE-AE | USAD | CONV-AE | LSTM | ELM-MI |
|---|---|---|---|---|---|---|---|---|---|
| IQR | 0.5 | 0.56175 | 0.56483 | 0.56606 | 0.52692 | 0.57218 | 0.54313 | 0.59914 | 0.44326 |
| | 1 | 0.55182 | 0.56428 | 0.58486 | 0.52646 | 0.55036 | 0.57336 | 0.60883 | 0.44446 |
| | 1.5 | 0.53394 | 0.55144 | 0.58442 | 0.52503 | 0.53511 | 0.58548 | 0.60800 | 0.44592 |
| | 2 | 0.51620 | 0.53209 | 0.57252 | 0.51945 | 0.50250 | 0.58682 | 0.60534 | 0.44728 |
| | Δ | 0.04556 | 0.03274 | 0.01879 | 0.00747 | 0.06968 | 0.04369 | 0.00970 | 0.00401 |
| MAD | 0.5 | 0.50336 | 0.51413 | 0.49349 | 0.50250 | 0.58668 | 0.46786 | 0.52928 | 0.44169 |
| | 1 | 0.54721 | 0.55407 | 0.53969 | 0.52036 | 0.58410 | 0.50639 | 0.57048 | 0.44178 |
| | 1.5 | 0.56222 | 0.56553 | 0.56368 | 0.52705 | 0.56512 | 0.53983 | 0.59311 | 0.44178 |
| | 2 | 0.55584 | 0.56523 | 0.58115 | 0.52554 | 0.55042 | 0.56409 | 0.60482 | 0.44187 |
| | Δ | 0.05886 | 0.05140 | 0.08766 | 0.02455 | 0.03625 | 0.09623 | 0.07554 | 0.00018 |
| STD | 0.5 | 0.55985 | 0.56532 | 0.52666 | 0.42688 | 0.41965 | 0.58689 | 0.30587 | 0.47190 |
| | 1 | 0.54469 | 0.53154 | 0.47860 | 0.37618 | 0.33513 | 0.57179 | 0.19188 | 0.75882 |
| | 1.5 | 0.51419 | 0.50264 | 0.42352 | 0.34334 | 0.25385 | 0.56519 | 0.13661 | 0.76720 |
| | 2 | 0.49627 | 0.43994 | 0.38488 | 0.31265 | 0.19333 | 0.55955 | 0.10893 | 0.56279 |
| | Δ | 0.06358 | 0.12538 | 0.14178 | 0.11423 | 0.22633 | 0.02735 | 0.19695 | 0.29530 |
| MAX | | 0.02261 | 0.01592 | 0.11406 | 0.02338 | 0.01670 | 0.02079 | 0.00243 | 0.44172 |
| | minF1 | 0.49627 | 0.43994 | 0.38488 | 0.31265 | 0.19333 | 0.46786 | 0.10893 | 0.44169 |
| | maxF1 | 0.56222 | 0.56553 | 0.58486 | 0.52705 | 0.58668 | 0.58689 | 0.60883 | 0.76720 |
| | ΔTot | 0.06595 | 0.12558 | 0.19998 | 0.21440 | 0.39335 | 0.11903 | 0.49991 | 0.32551 |

Table 5.15: F1-score comparison between different threshold technique and threshold factor

| Threshol Method | Th_factor | VAE | LSTM-AE | VAE+ReEnc | DENSE-AE | USAD | CONV-AE | LSTM | ELM-MI |
|---|---|---|---|---|---|---|---|---|---|
| IQR | 0.5 | 0.64827 | 0.65070 | 0.00018 | 0.00167 | 0.97069 | 0.14112 | 0.53454 | 0.48521 |
| | 1 | 0.78198 | 0.78649 | 0.00022 | 0.00184 | 1.13782 | 0.16934 | 0.67077 | 0.48543 |
| | 1.5 | 0.91569 | 0.92227 | 0.00026 | 0.00200 | 1.30495 | 0.19756 | 0.80701 | 0.48564 |
| | 2 | 1.04939 | 1.05806 | 0.00030 | 0.00217 | 1.47208 | 0.22579 | 0.94324 | 0.48586 |
| | Δ | 0.40112 | 0.40736 | 0.00012 | 0.00050 | 0.50140 | 0.08467 | 0.40871 | 0.00064 |
| MAD | 0.5 | 0.46557 | 0.46505 | 0.00013 | 0.00143 | 0.76627 | 0.09875 | 0.31405 | 0.48500 |
| | 1 | 0.56333 | 0.56464 | 0.00015 | 0.00155 | 0.89065 | 0.11879 | 0.40231 | 0.48500 |
| | 1.5 | 0.66108 | 0.66424 | 0.00018 | 0.00167 | 1.01504 | 0.13882 | 0.49058 | 0.48501 |
| | 2 | 0.75884 | 0.76383 | 0.00021 | 0.00179 | 1.13943 | 0.15885 | 0.57885 | 0.48502 |
| | Δ | 0.29326 | 0.29879 | 0.00008 | 0.00035 | 0.37316 | 0.06010 | 0.26480 | 0.00002 |
| STD | Δ | 0.63607 | 0.74631 | 0.00050 | 0.00281 | 1.73216 | 0.22446 | 2.92662 | 0.48793 |
| | 1 | 0.85216 | 1.06149 | 0.00084 | 0.00402 | 2.57326 | 0.34078 | 5.32785 | 0.49368 |
| | 1.5 | 1.06824 | 1.37668 | 0.00118 | 0.00522 | 3.41437 | 0.45710 | 7.72909 | 0.49944 |
| | 2 | 1.28433 | 1.69187 | 0.00152 | 0.00643 | 4.25547 | 0.57342 | 10.13032 | 0.50000 |
| | Δ | 0.64826 | 0.94556 | 0.00101 | 0.00362 | 2.52332 | 0.34896 | 7.20370 | 0.01207 |
| MAX | | 22.29737 | 27.33589 | 0.03574 | 0.10271 | 26.61404 | 12.03240 | 340.67161 | 0.48500 |

Table 5.16: Threshold value according to different threshold technique and threshold factor

### 5.3.5. Thesis vs. State-of-Art

Relevant considerations must be made about comparing the results obtained with the models implemented in the thesis and the available results. Extensive research does not reveal better results than those included in the paper that introduces and analyzes the dataset [46] [47]. They consider, as shown in Figure 5.22, four different anomaly detection tasks according to when the anomalies are detected:

- **Anomaly Existence** is focused on detecting an anomaly within a ground truth anomaly interval. The duration of the anomaly is not important; the focus is placed on whether at least one anomaly within a real anomaly interval is detected. The metrics results are better than other criteria since it is sufficient that an anomaly matches a real anomaly interval and does not measure whether each anomaly within a real anomaly interval is detected.
- **Range Detection** is focused not only on the existence but also on the precise time range of the anomaly, comparing the anomaly prediction to the real anomaly point by point
- **Early Detection** is focused on minimizing the detection latency, namely the time difference between when the anomaly is detected and the starting time of the corresponding real one.
- **Exactly-Once Detection** is focused on reporting each anomaly instance only once.

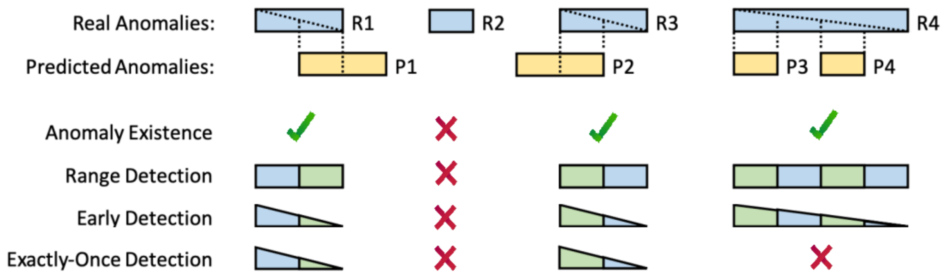


Figure 5.22: Anomaly detection task performed in the state of art paper [46]

| | State of Art | | | My Result | | |
|---|---|---|---|---|---|---|
| **MODEL** | **F1** | **PRECISION** | **RECALL** | **F1** | **PRECISION** | **RECALL** |
| BiGan | 0.17000 | 0.90000 | 0.10000 | - | - | - |
| DENSE-AE | 0.56000 | 0.52000 | 0.68000 | 0.52705 | 0.45037 | 0.63518 |
| LSTM | 0.53000 | 0.59000 | 0.48000 | 0.60883 | 0.59614 | 0.62208 |

Table 5.17: Exathlon result comparison between thesis implementation and state-of-art

The results obtained on this dataset are compared with the one in state of the art, corresponding to the range detection task. Therefore, the metrics are computed by comparing the prediction point-by-point and the ground truth. The references paper applies only three different models to detect anomalies, while the thesis implementation offers a broader analysis using several methods. Moreover, some choices made in the thesis implementation are the same as the referencing paper to obtain comparable results. First, the training, validation, and test splitting are the same. Also, how the threshold is computed is the same and performed on the same dataset part, and it kept the same overlapping window size. Table 5.17 compares the result obtained by reimplementing the model available in state of the art. The first method used in the paper is BiGan [92], which gives poor results with a very low F1-Score and is not analyzed in detail, focusing on methods that work well. Instead, LSTM and DENSE-AE are deeply analyzed, giving different results. Concerning DENSE-AE, despite using the same model configurations and parameters, the results obtained in the thesis implementation are a bit lower. But using more complex autoencoder methods based on convolutional or LSTM layers, the results are improved as deeply analyzed in the following section.

The most relevant difference concerns the LSTM method, where the *F1-Score* is improved by almost ten percentage points. The reason for this difference lies in how LSTM works and how it is implemented. The LSTM method does not work like autoencoders which try to reconstruct input data. Instead, it is used for forecasting, namely taking a sequence of several timestamp samples as input and predicting the following timestamp behavior. So, the main difference between the method implemented in the thesis and the one available in the paper is the length of the overlapping window of the input data processed by the model to predict the following sequence. One uses an overlapping window corresponding to only one timestamp, while the other uses an overlapping window with 40 timestamps. It is decided to use a window of 40 timestamps since it is the same window size used in models based on autoencoder, and so it is possible to compare LSTM and autoencoder on the same input. Moreover, as shown in [72], the importance of the time window in LSTM is crucial because there is a relationship between the time window accuracy of the prediction. Moreover, with more values in input, the model can learn better and consequently make a more accurate prediction. As a check, the LSTM model of the paper has been reimplemented and tested, giving results comparable to the paper ones.

### 5.3.6. Summary Result

Figure 5.23 and Table 5.18 clearly show that the models that work better on this dataset are not based on the autoencoder structure. The reason concerns some limits of the reconstruction-based model, the category to which autoencoders belong. For reconstruction-based models, the scores tend to reduce peaks, while for prediction-based models, such as LSTM, the score, calculated by averaging fewer values, presents more fluctuation. The consequence [108] is that prediction-based models better detect point and short-lived anomalies. Moreover, the dataset contains anomalies of different types. Autoencoders can easily detect anomalies, such as Bursty Input Traces (**T1**), Bursty Input Until Crash

Traces (**T2**), Stalled Input Traces (**T3**), and CPU Contention Traces (**T4**), but gives poor results for anomalies, such as Driver Failure (**T5**), Executor Failure (**T6**) which corresponds to short-lived anomalies. ELM-MI and LSTM, not being reconstruction model based, can detect short-lived anomalies better, giving better F1-Score results. As further evidence to non-reconstruction-model based corresponds the best MAR value. Also, the models that analyze the latent space, such as *VAE+Isof* and *VAE+KNN*, work better than purely autoencoder-based techniques. This happens because the classification between normal and anomalous samples is not performed on an anomaly score but are evaluated with density or distance-based model, which are highly functional for short-lived anomalies.

ELM-MI, by far, is the best approach giving the best F1-Score and a very low False Alarm Rate. The reason regards the different score distributions of other methods and ELM-MI. All score distributions, excluding ELM-MI, can assume any value greater than zero. Instead, the ELM-MI score distribution is limited to 0.5, and all values greater than it assume that value. This affects the distribution parameter, such as the mean. The consequence is that according to ELM-MI or the other models, the same threshold technique assumes a completely different meaning. In this case, the methods STD of ELM-MI works particularly well. In addition, it is evident that the FAR metrics for all the methods are lower and reflects the F1-Score behavior. This happens because few false positives are detected since the expected behavior is reconstructed or predicted precisely.

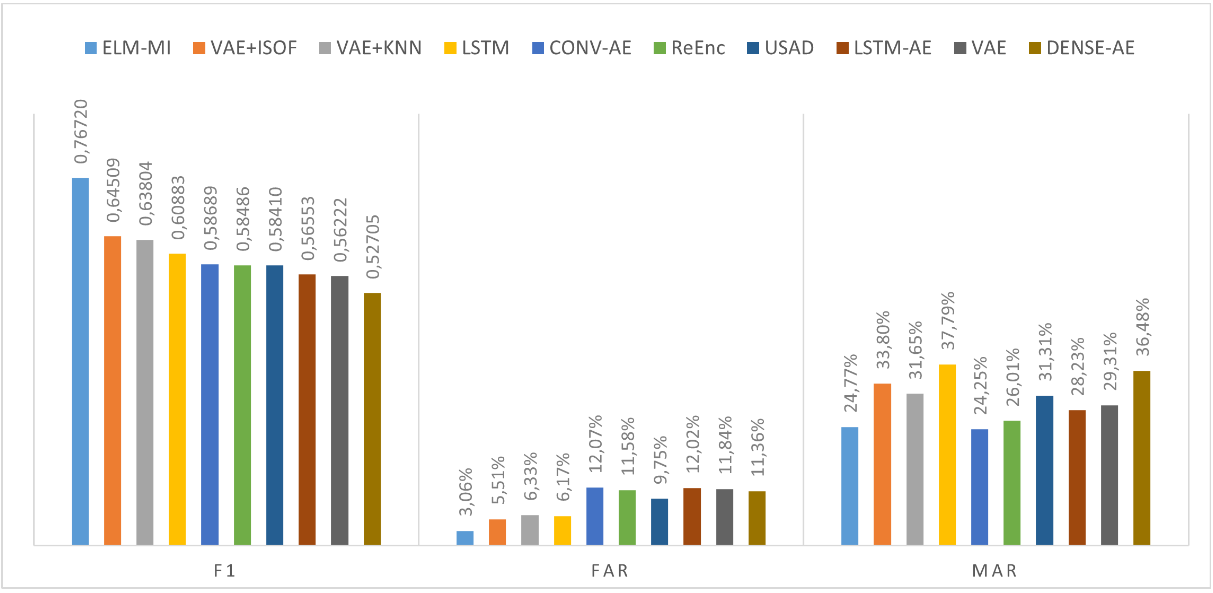


Figure 5.23: Summary results of Exathlon dataset

| MODEL | F1 | PRECISION | RECALL | FAR | MAR |
|---|---|---|---|---|---|
| ELM-MI | 0.767203 | 0.782707 | 0.752301 | 3.06% | 24.77% |
| VAE+ISOF | 0.645091 | 0.629022 | 0.661991 | 5.51% | 33.80% |
| VAE+KNN | 0.638042 | 0.598261 | 0.683560 | 6.33% | 31.65% |
| LSTM | 0.608834 | 0.596141 | 0.622078 | 6.17% | 37.79% |
| CONV-AE | 0.586893 | 0.479022 | 0.757465 | 12.07% | 24.25% |
| ReEnc | 0.584855 | 0.483532 | 0.739900 | 11.58% | 26.01% |
| USAD | 0.584099 | 0.508044 | 0.686934 | 9.75% | 31.31% |
| LSTM-AE | 0.565528 | 0.466579 | 0.717741 | 12.02% | 28.23% |
| VAE | 0.562218 | 0.466685 | 0.706931 | 11.84% | 29.31% |
| DENSE-AE | 0.527047 | 0.450374 | 0.635184 | 11.36% | 36.48% |

Table 5.18: Exathlon summary result

# 6 | Conclusions and Future work

This work performs unsupervised anomaly detection in two multivariate time series, SKAB [52] and Exathlon [46], by processing the data with several inference models and then by applying different thresholds, computed throw several thresholding techniques on a validation set, on the score generated by each model. Therefore, it is analyzed if the anomaly detection results, measured in terms of F1-Score, Recall, Precision, FAR, and MAR, depend more on the inference model or on the thresholding technique.

What emerges is that both, the model and threshold, have to be chosen accurately. Using models which generate bad predictions with low AUROC values, it is not possible to get accurate and precise anomaly detection. On the other hand, also the threshold is important.

Concerning the threshold techniques, they highly depend on the statistical feature of the validation score on which the threshold is computed. If they are calculated on a small validation set and the validation score assumes a nearly uniform distribution, the threshold computed by different techniques is quite similar. Also, the contribution of different $th_{factor}$ is minimal. The result is that the neural network model is the principal choice for better anomaly detection. On the other hand, for huge validation sets, the behavior is the opposite. Being calculated on more value and score distribution containing samples that deviate significantly from the mean, the thresholds assume different values according to the technique used. Moreover, only statistically-based techniques can be used since MV generates inconsistent results. The choice of the threshold factor instead is not so important. It takes values only if the score distribution is limited by a maximum value. The result is that in this type of validation score distribution, the most significant contribution to the F1-Score, Precision, and Recall values corresponds to the choice of the threshold method. This does not mean that the impact of the model is null but lower.

Many different adaptations, tests, and experiments have been left for the future due to the lack of time. Future works concern the analysis of other thresholding techniques like [40], which proposes an approach for detecting anomalies in multivariate telemetry time series using LSTMs and a nonparametric dynamic thresholding method that does not assume a specific underlying distribution of the anomaly scores. The method relies on the standard deviation of the smoothed prediction errors and on a single parameter (z), set experimentally and shown to have little impact on performances. Other future works address applying the threshold technique presented in the thesis to other multivariate datasets with the scope to have further data to determine whether it is possible to establish a relationship between the threshold and the distribution of the testing anomaly score.

# Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] A. Abdulaal, Z. Liu, and T. Lancewicki. Practical approach to asynchronous multivariate time series anomaly detection and localization. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 2485–2494, 2021.

[3] C. C. Aggarwal. *An Introduction to Outlier Analysis*, pages 1–34. Springer International Publishing, Cham, 2017. ISBN 978-3-319-47578-3. doi: 10.1007/978-3-319-47578-3_1. URL `https://doi.org/10.1007/978-3-319-47578-3_1`.

[4] P. J. M. Ali, R. H. Faraj, E. Koya, P. J. M. Ali, and R. H. Faraj. Data normalization and standardization: a technical report. *Mach Learn Tech Rep*, 1(1):1–6, 2014.

[5] M. Alkhayrat, M. Aljnidi, and K. Aljoumaa. A comparative dimensionality reduction study in telecom customer segmentation using deep learning and pca. *Journal of Big Data*, 7:9, 02 2020. doi: 10.1186/s40537-020-0286-0.

[6] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3395–3404, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403392. URL `https://doi.org/10.1145/3394486.3403392`.

[7] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga. Usad: Unsu-

pervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3395–3404, 2020.

[8] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga. Do deep neural networks contribute to multivariate time series anomaly detection? *Pattern Recognition*, 132:108945, 2022.

[9] P. Barson, S. Field, N. Davey, G. McAskie, and R. Frank. The detection of fraud in mobile phone networks. *Neural Network World*, 6(4):477–484, 1996.

[10] M. Belichovski, D. Stavrov, F. Donchevski, and G. Nadzinski. Unsupervised machine learning approach for anomaly detection in e-coating plant. In *2022 IEEE 17th International Conference on Control & Automation (ICCA)*, pages 992–997. IEEE, 2022.

[11] S. M. Bendre. *Sankhyā: The Indian Journal of Statistics, Series B (1960-2002)*, 56(2):305–308, 1994. ISSN 05815738. URL `http://www.jstor.org/stable/25052847`.

[12] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano. A review on outlier/anomaly detection in time series data. *ACM Computing Surveys (CSUR)*, 54 (3):1–33, 2021.

[13] N. Caballé, J. Castillo-Sequera, J. A. Gomez-Pulido, J. Gómez, and M. Polo-Luque. Machine learning applied to diagnosis of human diseases: A systematic review. *Applied Sciences*, 10:5135, 07 2020. doi: 10.3390/app10155135.

[14] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), jul 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL `https://doi.org/10.1145/1541880.1541882`.

[15] A. Chiang, E. David, Y.-J. Lee, G. Leshem, and Y.-R. Yeh. A study on anomaly detection ensembles. *Journal of Applied Logic*, 21:1–13, 2017. ISSN 1570-8683. doi: https://doi.org/10.1016/j.jal.2016.12.002. URL `https://www.sciencedirect.com/science/article/pii/S1570868316301240`.

[16] F. Chollet et al. Keras. `https://keras.io`, 2015.

[17] M. C. Chuah and F. Fu. Ecg anomaly detection via time series analysis. In *Frontiers of High Performance Computing and Networking ISPA 2007 Workshops: ISPA 2007 International Workshops SSDSN, UPWN, WISH, SGC, ParDMCom, HiPCoMB,*

*and IST-AWSN Niagara Falls, Canada, August 28-September 1, 2007 Proceedings 5*, pages 123–135. Springer, 2007.

[18] G. Coelho, L. M. Matos, P. J. Pereira, A. Ferreira, A. Pilastri, and P. Cortez. Deep autoencoders for acoustic anomaly detection: experiments with working machine and in-vehicle audio. *Neural Computing and Applications*, 34(22):19485–19499, 2022.

[19] N. Cohen and Y. Hoshen. Sub-image anomaly detection with deep pyramid correspondences. *arXiv preprint arXiv:2005.02357*, 2020.

[20] J. Cook and V. Ramadas. When to consult precision-recall curves. *The Stata Journal*, 20(1):131–148, 2020.

[21] S. Craw. *Manhattan Distance*, pages 790–791. Springer US, Boston, MA, 2017. ISBN 978-1-4899-7687-1. doi: 10.1007/978-1-4899-7687-1_511. URL `https://doi.org/10.1007/978-1-4899-7687-1_511`.

[22] P. Cunningham and S. J. Delany. k-nearest neighbour classifiers-a tutorial. *ACM computing surveys (CSUR)*, 54(6):1–25, 2021.

[23] R. de Jong. Lecture 1: Stationarity time series. URL `https://www.asc.ohio-state.edu/de-jong.8/note1.pdf`.

[24] L. Delamaire, H. Abdou, and J. Pointon. Credit card fraud and detection techniques: a review. *Banks and Bank systems*, 4(2):57–68, 2009.

[25] D. A. Dickey and W. A. Fuller. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, 74 (366a):427–431, 1979. doi: 10.1080/01621459.1979.10482531. URL `https://doi.org/10.1080/01621459.1979.10482531`.

[26] H. U. Dike, Y. Zhou, K. K. Deveerasetty, and Q. Wu. Unsupervised learning based on artificial neural network: A review. In *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, pages 322–327. IEEE, 2018.

[27] N. Duffield, P. Haffner, B. Krishnamurthy, and H. Ringberg. Rule-based anomaly detection on ip flows. In *IEEE INFOCOM 2009*, pages 424–432. IEEE, 2009.

[28] M. Elfil and A. Negida. Sampling methods in clinical research; an educational review. *Emergency*, 5(1), 2017.

[29] M. Entezami, S. Hillmansen, P. Weston, and M. Papaelias. Fault detection and diagnosis within a wind turbine mechanical braking system using condition mon-

itoring. *Renewable Energy*, 47:175–182, 2012. ISSN 0960-1481. doi: https://doi.org/10.1016/j.renene.2012.04.031. URL `https://www.sciencedirect.com/science/article/pii/S0960148112002728`.

[30] P. Esling and C. Agon. Time-series data mining. *ACM Computing Surveys*, 45(1):12, 2012. doi: 10.1145/2379776.2379788. URL `https://hal.science/hal-01577883`.

[31] T. Fernando, H. Gammulle, S. Denman, S. Sridharan, and C. Fookes. Deep learning for medical anomaly detection–a survey. *ACM Computing Surveys (CSUR)*, 54(7): 1–37, 2021.

[32] F. E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969. doi: 10.1080/00401706.1969.10490657. URL `https://www.tandfonline.com/doi/abs/10.1080/00401706.1969.10490657`.

[33] X. Gu, L. Akoglu, and A. Rinaldo. Statistical analysis of nearest neighbor methods for anomaly detection. *Advances in Neural Information Processing Systems*, 32, 2019.

[34] G. Guo, H. Wang, D. Bell, and Y. Bi. Knn model-based approach in classification. 08 2004.

[35] Y. Guo, X. Cao, B. Liu, and K. Peng. El niño index prediction using deep learning with ensemble empirical mode decomposition. *Symmetry*, 12:893, 2020.

[36] S. Hansen, S. Gautam, R. Jenssen, and M. Kampffmeyer. Anomaly detection-inspired few-shot medical image segmentation through self-supervision with supervoxels. *Medical Image Analysis*, 78:102385, 2022. ISSN 1361-8415. doi: https://doi.org/10.1016/j.media.2022.102385. URL `https://www.sciencedirect.com/science/article/pii/S1361841522000378`.

[37] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[38] D. M. Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[39] N. A. Heard, D. J. Weston, K. Platanioti, and D. J. Hand. Bayesian anomaly detection methods for social networks. 2010.

[40] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 387–395, 2018.

[41] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[42] A. Hyndman, R.J. *G. (2018) Forecasting: principles and practice, 2nd edition.* OTexts: Melbourne, Australia, 2018, OTexts.com/fpp2. Accessed on 13/02/23.

[43] K. Hyun. The prevention and handling of the missing data. *Korean J Anesthesiol*, 64(5):402–406, 2013. doi: 10.4097/kjae.2013.64.5.402. URL `http://ekja.org/journal/view.php?number=7569`.

[44] T. M. Inc. Matlab version: 9.13.0 (r2022b), 2022. URL `https://www.mathworks.com`.

[45] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum. Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[46] V. Jacob, F. Song, A. Stiegler, B. Rad, Y. Diao, and N. Tatbul. Exathlon: A benchmark for explainable anomaly detection over time series. *arXiv preprint arXiv:2010.05073*, 2020.

[47] V. Jacob, F. Song, A. Stiegler, B. Rad, Y. Diao, and N. Tatbul. A demonstration of the exathlon benchmarking platform for explainable anomaly detection. *Proceedings of the VLDB Endowment (PVLDB)*, 2021.

[48] A. Jadon, A. Patil, and S. Jadon. A comprehensive survey of regression based loss functions for time series forecasting. *arXiv preprint arXiv:2211.02989*, 2022.

[49] A. T. Jebb, L. Tay, W. Wang, and Q. Huang. Time series analysis for psychological research: examining and forecasting change. *Frontiers in Psychology*, 6, 2015. ISSN 1664-1078. doi: 10.3389/fpsyg.2015.00727. URL `https://www.frontiersin.org/articles/10.3389/fpsyg.2015.00727`.

[50] J. Jose. Introduction to time series analysis and its applications. 08 2022.

[51] S. Kandanaarachchi. Unsupervised anomaly detection ensembles using item response theory. *arXiv preprint arXiv:2106.06243*, 2021.

[52] I. D. Katser and V. O. Kozitsin. Skoltech anomaly benchmark (skab). `https://www.kaggle.com/dsv/1693952`, 2020.

[53] S. Khan, M. A. Alam, N. S. Ram, K. Mirza, and V. Chowdary. Noise reduction of time-series satellite data using various de-noising algorithms. *Int. J. Tech. Res. Sci*, (3):55–69, 2020.

[54] T. Kieu, B. Yang, C. Guo, and C. S. Jensen. Outlier detection for time series with recurrent autoencoder ensembles. In *IJCAI*, pages 2725–2732, 2019.

[55] A. Kind, M. P. Stoecklin, and X. Dimitropoulos. Histogram-based traffic anomaly detection. *IEEE Transactions on Network and Service Management*, 6(2):110–121, 2009.

[56] W. Koehrsen. Overfitting vs. underfitting: A complete example. *Towards Data Science*, pages 1–12, 2018.

[57] Y. Kou, C.-T. Lu, S. Sirwongwattana, and Y.-P. Huang. Survey of fraud detection techniques. In *IEEE International Conference on Networking, Sensing and Control, 2004*, volume 2, pages 749–754 Vol.2, 2004. doi: 10.1109/ICNSC.2004.1297040.

[58] A. Kulkarni, P. Mani, and C. Domeniconi. Network-based anomaly detection for insider trading. *arXiv preprint arXiv:1702.05809*, 2017.

[59] T. Kurita. Principal component analysis (pca). *Computer Vision: A Reference Guide*, pages 1–4, 2019.

[60] J. H. Lee, K. T. McDonnell, A. Zelenyuk, D. Imre, and K. Mueller. A structure-based distance metric for high-dimensional space exploration with multidimensional scaling. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):351–364, 2013.

[61] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013. ISSN 0022-1031. doi: https://doi.org/10.1016/j.jesp.2013.03.013. URL `https://www.sciencedirect.com/science/article/pii/S0022103113000668`.

[62] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of experimental social psychology*, 49(4):764–766, 2013.

[63] J. Li, H. Izakian, W. Pedrycz, and I. Jamal. Clustering-based anomaly detection in multivariate time series data. *Applied Soft Computing*, 100:106919, 2021.

[64] H. Liang, L. Song, J. Wang, L. Guo, X. Li, and J. Liang. Robust unsupervised anomaly detection via multi-time scale dcgans with forgetting mechanism for industrial multivariate time series. *Neurocomputing*, 423:444–462, 2021.

[65] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich. A survey on long short-term memory networks for time series prediction. *Procedia CIRP*, 99:650–655, 2021. ISSN 2212-8271. doi: https://doi.org/10.1016/j.procir.2021.03.088. URL `https://www.sciencedirect.com/science/article/pii/S2212827121003796`. 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020.

[66] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[67] F. T. Liu, K. Ting, and Z.-H. Zhou. Isolation forest. pages 413 – 422, 01 2009. doi: 10.1109/ICDM.2008.17.

[68] R. M. Liu, S. K. Babanajad, T. Taylor, and F. Ansari. Experimental study on structural defect detection by monitoring distributed dynamic strain. *Smart Materials and Structures*, 24(11):115038, oct 2015. doi: 10.1088/0964-1726/24/11/115038. URL `https://dx.doi.org/10.1088/0964-1726/24/11/115038`.

[69] J. Ma, L. Sun, H. Wang, Y. Zhang, and U. Aickelin. Supervised anomaly detection in uncertain pseudoperiodic data streams. *ACM Transactions on Internet Technology (TOIT)*, 16(1):1–20, 2016.

[70] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, et al. Long short term memory networks for anomaly detection in time series. In *ESANN*, volume 2015, page 89, 2015.

[71] C. Manikopoulos and S. Papavassiliou. Network intrusion and fault detection: a statistical anomaly approach. *IEEE Communications Magazine*, 40(10):76–82, 2002.

[72] K. Martin-Chinea, J. Ortega, J. Gomez-Gonzalez, E. Pereda, J. Toledo, and L. Acosta. Effect of time windows in lstm networks for eeg-based bcis. *Cognitive Neurodynamics*, pages 1–14, 2022.

[73] L. Martí, N. Sanchez-Pi, J. M. Molina, and A. C. B. Garcia. Anomaly detection based on sensor data in petroleum industry applications. *Sensors*, 15(2):2774–2797, 2015. ISSN 1424-8220. doi: 10.3390/s150202774. URL `https://www.mdpi.com/1424-8220/15/2/2774`.

[74] M. S. Mok, S. Y. Sohn, and Y. H. Ju. Random effects logistic regression model for anomaly detection. *expert systems with applications*, 37(10):7162–7166, 2010.

[75] M. Murugesan and S. Thilagamani. Efficient anomaly detection in surveillance videos based on multi layer perception recurrent neural network. *Microprocessors and Microsystems*, 79:103303, 2020.

[76] R. Murugesan, E. Mishra, and A. Krishnan. Deep learning based models: Basic lstm, bi lstm, stacked lstm, cnn lstm and conv lstm to forecast agricultural commodities prices, 07 2021.

[77] T. pandas development team. pandas-dev/pandas: Pandas, Feb. 2020. URL `https://doi.org/10.5281/zenodo.3509134`.

[78] S. H. Park, J. M. Goo, and C.-H. Jo. Receiver operating characteristic (roc) curve: practical review for radiologists. *Korean journal of radiology*, 5(1):11–18, 2004.

[79] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[80] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[81] X. Peng, H. Li, F. Yuan, S. G. Razul, Z. Chen, and Z. Lin. An extreme learning machine for unsupervised online anomaly detection in multivariate time series. *Neurocomputing*, 501:596–608, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.06.042. URL `https://www.sciencedirect.com/science/article/pii/S0925231222007615`.

[82] O. I. Provotar, Y. M. Linder, and M. M. Veres. Unsupervised anomaly detection in time series using lstm-based autoencoders. In *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, pages 513–517. IEEE, 2019.

[83] G. Pu, L. Wang, J. Shen, and F. Dong. A hybrid unsupervised clustering-based

anomaly detection method. *Tsinghua Science and Technology*, 26(2):146–153, 2021. doi: 10.26599/TST.2019.9010051.

[84] B. Quinn. Stationarity and invertibility of simple bilinear models. *Stochastic Processes and their Applications*, 12(2):225–230, 1982. ISSN 0304-4149. doi: https://doi.org/10.1016/0304-4149(82)90045-X. URL `https://www.sciencedirect.com/science/article/pii/030441498290045X`.

[85] M. Raginsky, R. M. Willett, C. Horn, J. Silva, and R. F. Marcia. Sequential anomaly detection in the presence of noise and limited feedback. *IEEE Transactions on Information Theory*, 58(8):5544–5562, 2012.

[86] E. Ramasso, V. Placet, and M. L. Boubakar. Unsupervised consensus clustering of acoustic emission time-series for robust damage sequence estimation in composites. *IEEE Transactions on Instrumentation and Measurement*, 64(12):3297–3307, 2015.

[87] K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox, and P. Gehler. Towards total recall in industrial anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14318–14328, 2022.

[88] P. J. Rousseeuw and C. Croux. Alternatives to the median absolute deviation. *Journal of the American Statistical Association*, 88(424):1273–1283, 1993. doi: 10.1080/01621459.1993.10476408. URL `https://www.tandfonline.com/doi/abs/10.1080/01621459.1993.10476408`.

[89] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft. Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694*, 2019.

[90] A. Saci, A. Al-Dweik, and A. Shami. Autocorrelation integrated gaussian based anomaly detection using sensory data in industrial manufacturing. *IEEE Sensors Journal*, 21(7):9231–9241, 2021.

[91] A. Sahar and D. Han. An lstm-based indoor positioning method using wi-fi signals. pages 1–5, 08 2018. ISBN 978-1-4503-6529-1. doi: 10.1145/3271553.3271566.

[92] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *Information Processing in Medical Imaging: 25th International Conference, IPMI 2017, Boone, NC, USA, June 25-30, 2017, Proceedings*, pages 146–157. Springer, 2017.

[93] A. Siffer, P. Fouque, A. Termier, and C. Largouët. Anomaly detection in streams

with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1067–1075. ACM, 2017. doi: 10.1145/3097983.3098144. URL `https://doi.org/10.1145/3097983.3098144`.

[94] Simplilearn. Normalization vs standardization - what's the difference? "https://www.simplilearn.com/normalization-vs-standardization-article", 01 2023.

[95] K. Singh and S. Upadhyaya. Outlier detection: applications and techniques. *International Journal of Computer Science Issues (IJCSI)*, 9(1):307, 2012.

[96] A. A. Sodemann, M. P. Ross, and B. J. Borghetti. A review of anomaly detection in automated surveillance. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1257–1272, 2012.

[97] V. A. Sotiris, W. T. Peter, and M. G. Pecht. Anomaly detection through a bayesian support vector machine. *IEEE Transactions on Reliability*, 59(2):277–286, 2010.

[98] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2828–2837, 2019.

[99] A. I. Tambuwal and D. Neagu. Deep quantile regression for unsupervised anomaly detection in time-series. *SN Computer Science*, 2:1–16, 2021.

[100] J. Tian, M. H. Azarian, and M. Pecht. Anomaly detection using self-organizing maps-based k-nearest neighbor algorithm. In *PHM society European conference*, volume 2, 2014.

[101] Z. Tian, M. Zhuo, L. Liu, J. Chen, and S. Zhou. Anomaly detection using spatial and temporal information in multivariate time series. *Scientific Reports*, 13(1):4400, 2023.

[102] K. M. Ting, B.-C. Xu, T. Washio, and Z.-H. Zhou. Isolation distributional kernel: A new tool for kernel based anomaly detection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 198–206, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403062. URL `https://doi.org/10.1145/3394486.3403062`.

[103] A. Toshniwal, K. Mahesh, and R. Jayashree. Overview of anomaly detection techniques in machine learning. In *2020 Fourth International Conference on I-SMAC*

*(IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 808–815, 2020. doi: 10.1109/I-SMAC49090.2020.9243329.

[104] G. Van Rossum and F. L. Drake Jr. *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[105] Y. Wang, H. Yao, and S. Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

[106] Z. Wang, Z. Yu, C. L. P. Chen, J. You, T. Gu, H.-S. Wong, and J. Zhang. Clustering by local gravitation. *IEEE Transactions on Cybernetics*, 48(5):1383–1396, 2018. doi: 10.1109/TCYB.2017.2695218.

[107] H. Weytjens and J. De Weerdt. Process outcome prediction: Cnn vs. lstm (with attention). In *Business Process Management Workshops: BPM 2020 International Workshops, Seville, Spain, September 13–18, 2020, Revised Selected Papers 18*, pages 321–333. Springer, 2020.

[108] L. Wong, D. Liu, L. Berti-Equille, S. Alnegheimish, and K. Veeramachaneni. Aer: Auto-encoder with regression for time series anomaly detection. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 1152–1161, 2022. doi: 10.1109/BigData55660.2022.10020857.

[109] M. Wu and C. Jermaine. Outlier detection by sampling with accuracy guarantees. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 767–772, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150501. URL `https://doi.org/10.1145/1150402.1150501`.

[110] S.-y. Xia, Z.-y. Xiong, Y.-g. Luo, Wei-Xu, and G.-h. Zhang. Effectiveness of the euclidean distance in high dimensional spaces. *Optik - International Journal for Light and Electron Optics*, 126, 09 2015. doi: 10.1016/j.ijleo.2015.09.093.

[111] J. Yang, S. Rahardja, and P. Fränti. Outlier detection: how to threshold outlier scores? In *Proceedings of the international conference on artificial intelligence, information processing and cloud computing*, pages 1–6, 2019.

[112] K. Yoshihara and K. Takahashi. A simple method for unsupervised anomaly detection: An application to web time series data. *Plos one*, 17(1):e0262463, 2022.

[113] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar. Adversarially learned anomaly detection. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 727–736, 2018. doi: 10.1109/ICDM.2018.00088.

[114] C. Zhang, S. Li, H. Zhang, and Y. Chen. Velc: A new variational autoencoder based model for time series anomaly detection. *arXiv preprint arXiv:1907.01702*, 2019.

[115] Y. Zhang, N. Meratnia, and P. Havinga. Outlier detection techniques for wireless sensor networks: A survey. *IEEE communications surveys & tutorials*, 12(2):159–170, 2010.

[116] Y. Zhang, W. Liu, Z. Chen, J. Wang, and K. Li. On the properties of kullback-leibler divergence between multivariate gaussian distributions. *arXiv preprint arXiv:2102.05485*, 2021.

[117] M. Zhao and V. Saligrama. Anomaly detection with score functions based on nearest neighbor graphs. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. URL `https://proceedings.neurips.cc/paper/2009/file/996a7fa078cc36c46d02f9af3bef918b-Paper.pdf`.

[118] J. T. Zhou, J. Du, H. Zhu, X. Peng, Y. Liu, and R. S. M. Goh. Anomalynet: An anomaly detection network for video surveillance. *IEEE Transactions on Information Forensics and Security*, 14(10):2537–2550, 2019.

# List of Figures

# List of Tables

# List of Abbrevations

| **Abbreviation** | **Long word/phrase** |
|---|---|
| AE | Autoencoder |
| AI | Artificial Intelligence |
| BigGAN | Big Generative Adversarial Network |
| CE | Centrality |
| CNN | Convolutional Neural Network |
| CONV | Convolutional |
| CV | Computer Vision |
| ELM | Extreme Learning Machine |
| FAR | False Alarm Rate |
| FN | False Negative |
| FP | False Positive |
| GT | Ground Truth |
| IoT | Internet of Things |
| IQR | Inter-Quantile Range |
| ISOF | Isolation Forest |
| KNN | k-Nearest Neighbors |
| LRF | Local resultatnt Force |
| LSTM | Long Short-Term Memory |
| MAD | Median Absolute Deviation |
| MAR | Missing Alarm Rate |
| MI | Mutual Information |
| ML | Machine Learning |

| | |
|---|---|
| PCA | Principal Component Analysis |
| RBK | Radial Basis Function Kernel |
| ReENC | Re-Encoding |
| RNN | Recursive Neural Network |
| SD | Standard Deviation |
| SVM | Support-Vector Machines |
| TN | True Negative |
| TP | True positive |
| USAD | Unsupervised Anomaly Detection |
| VAE | Variational Autoencoder |

# Acknowledgements

Vorrei riservare questo spazio finale della tesi per ringraziare tutte le persone che mi hanno sostenuto ed aiutato nel mio percoso universitario.

Per prima cosa, vorrei ringraziare il mio relatore, Piero Fraternali, per i suoi consigli e per la sua disponibilità. Con la grande passione e dedizione per il suo lavoro, ha fin da subito fatto accrescere in me l'interesse verso gli argomenti trattati nella tesi.

Un ringraziamento particolare va al mio co-relatore, Nicolò Oreste Pinciroli Vago, per tutto il tempo dedicatomi nella scrittura della tesi. Grazie per tutti i consigli forniti e per la prontezza nel rispondere a qualunque richiesta.

Infine un sentito ringraziamento va alla mia famiglia. Grazie per aver sempre sostenuto e appoggiato ogni mia decisione e per avermi dato l'opportunità di studiare ciò che mi appassiona senza avermi mai fatto mancare nulla.