POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Deep learning models for anomaly detection in time series

Supervisor

Prof. Andrea BOTTINO

Candidate

Alessio SICILIANO

December 2021

Abstract

Recent advances in technology allow us to collect a large amount of data over time in diverse fields and the amount of data transferred exceeds the human ability to study it manually. Generally, standard statistical approaches assume that the samples are generated by a specific statistical model and do not scale well with the amount of data and thus automated data analysis becomes necessary. On the other hand, machine learning methods consider the data generation process as a black box and try to learn from the input only. Moreover, in many application fields, such as economic, healthcare and security, there is the need to have fast computations but at the same time receive a reliable result.

Traditional models use supervised machine learning algorithms but, in the context of applications, collecting and annotating such large-scale datasets is difficult, time-consuming or even too expensive, and it requires domain knowledge from experts in the field. Therefore, anomaly detection has been such a great challenge for researchers and practitioners.

Anomaly detection is referred to as the process of detecting anomaly data instances. The definition of an anomaly depends on the task and domain but, most of the time, it is an instance that significantly deviates from the others. In this thesis, the focus is on deep learning models for anomaly detection in time series. In the first part, a general overview of the anomaly detection task is provided and the properties and the definition of the time series are presented. Then, in the second part, various state of the art anomaly detection algorithms are discussed. Moreover, I present two new approaches, along with a comparison with the classic methods. In the third part, experiments carried out with different datasets and different architectures are shown.

Furthermore, I provide some improvements to the presented methods. In detail, the experiments are made with two public datasets and one on damage detection in industrial composite structures. These datasets have different properties in order to show how the discussed methods perform in different situations.

In the end, the results show the ability of the proposed models to detect anomalous patterns in time series from different fields of application while providing structured and expressive data representations.

Acknowledgements

To my loving family and my girlfriend, who supported me during these years.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
artificial intelligence

DAD
deep anomaly detection

AE
auto encoder

VAE
variational auto encoder

LSTM
long short-term memory

NN
neural network

MLE
maximum likelihood estimator

ROC
receiver operating characteristic curve

AUC
area under the curve

CPS
cyber physical system

CPSs

cyber physical systems

SWaT

secure water treatment

GNNs

graph neural networks

GDN

graph deviation network

IQR

inter-quantile range

NIG

Normal Inverse Gamma

MSE

Mean Squared Error

Chapter 1

Introduction

In recent years, advances in technology allowed us to collect a large amount of data over time in diverse fields, exceeding the human ability to study it manually. Generally, standard statistical approaches assume that the samples are generated by a specific statistical model and do not scale well with the amount of data. Moreover, in many application fields, there is the need to have fast computations but at the same time receive a reliable result.

On the other hand, machine learning methods consider the data generation process as a black box and learn from the input only. Hence, automated data analysis becomes necessary.

One of the most important data analysis tasks is the detection of anomalies in the data.

1.1 Definition of anomaly

The definition of an anomaly depends on the task and domain but, most of the time, it is an instance that significantly deviates from the others.

During the years, many authors and researchers have given their definitions of an anomaly. For example, *Hawkins* defines an outlier as an observation that *"deviates so significantly from other observations as to arouse suspicion that it was generated by a different mechanism"*[1].

The anomalies, in all the definitions described in the literature:

1. are points that usually exhibit strange behaviours with respect to the majority samples. However, in certain situations, anomalies can be also samples with normal behaviour that occur at "strange timesteps". In literature, these particular anomalies are called *Contextual anomalies*.

2. have low frequency, in fact, they represent a small subset of the entire data.

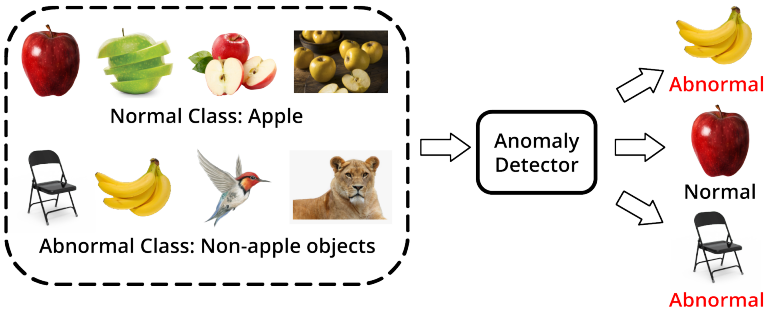

Figure 1.1: Example of an anomaly detection model.

As shown in Figure 1.1, in the case of a model trained to recognize the images of apples (normal class), the anomalies are all the images that do not contain an apple. For example, a banana and a chair are classified as anomalies.

The anomaly is a general concept and thus this term is often used interchangeably with other terms. For example, the word *outlier* is used most in statistical with the meaning of *something that comes from a different distribution* and thus is often associated, in machine learning, to the concept of anomaly.

1.2 Applications

Nowadays, many traditional fields have been more "smart" thanks to the addition of sensors that have made some operations automatic. For this reason, anomaly detection is an increasing area of research because it is widely used in more and more fields.

Bulusu et al. [2] provided the following main applications of anomaly detection:

- **Intrusion Detection**. An intrusion detection system is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. A key challenge for intrusion detection is the huge volume of data and sophisticated malicious patterns.

- **Fraud Detection**. It refers to the detection of fraudulent activities occurring in many domains such as e-commerce, banking, insurance, law enforcement, etc. A good fraud detection system should be able to identify fraudulent transactions accurately and should make the detection possible in real-time.

- **Anomaly Detection in Healthcare and Industrial domains**. Anomaly detection in the healthcare domain tries to detect abnormal patient conditions or instrumentation errors. Since this is a very critical problem, anomaly detection requires a high degree of accuracy. Similarly, in industrial systems like wind turbines, power plants, and storage devices that are exposed to large amounts of stress daily, it is critical to detect any damages as quickly as possible.
- **Anomaly Detection in IoT**. As the number of IoT devices is growing, it is crucial to maintain their safety and security. Due to a large number of sensors, utilizing deep learning approaches to process large amounts of data is a rising trend.

1.3 Challenges

As mentioned above, anomalies are data samples that do not comply with the expected normal behaviour. Hence, a naive approach for detecting anomalies is to define a region in the data space that represents normal behaviour and declare a sample as an anomaly if it does not lie in this region.

However, anomaly detection seems seemingly a simple task but, in reality, *Pang et al.* [3] defined the following challenges that must be faced:

- **Heterogeneous anomaly classes**. For definition, anomalies are irregular, and thus, one class of anomalies may demonstrate completely different behaviour and characteristics from another class of anomalies. For example, in video surveillance, the abnormal events of robbery, traffic accidents and burglary are visually highly different.
- **Rarity and class imbalance**. Anomalies are typically rare data instances and for this reason, it is difficult to collect a large amount of labelled abnormal instances leading to an imbalanced dataset. Moreover, most of the time, it is impossible to collect a full labelled dataset and thus we must proceed in a semi-supervised or unsupervised way which especially often incur in a high number of false positives.
- **Diverse types of anomaly**. Based on the type of applications, the definition of an anomaly changes. For certain domains, a small deviation from the normal behaviour may have far-reaching consequences and thus may be declared as an anomaly. In other applications, the deviation needs to be large for the input to be declared as an anomaly.
- **Unknownness**. Anomalies are points without clear definition and they are associated with many unknowns, e.g., instances with unknown abrupt

behaviours and distributions. They remain unknown until actually occur, such as novel frauds, network intrusions and breakage of a sensor. In other words, we do not know what to expect before they occur.

- **High dimensional data**. Nowadays, most of the data sources are multidimensional with two or more features and thus the problem becomes more complex because the anomaly can occur in each of the features. Complex data require deeper networks and in general, it is more challenging to define the boundaries between the normal instances and the anomalies.
- **Noise-resilient anomaly detection**. Many semi-supervised algorithms assume the labelled training data is clean, which can be vulnerable to noisy instances that are mistakenly labelled as an opposite class label. In such cases, we may use unsupervised methods instead, but this fails to utilize the genuine labelled data. The main challenge is that the amount of noise can differ significantly from datasets and applications therefore we must build a model able to distinguish when it can trust or not the data.

Chapter 2

Time series

2.1 Introduction

The goal of this thesis is to study the problem of anomaly detection in multivariate time series.

In this chapter, the fundamental basis of the time series is discussed. Anomaly detection methods are specific to the type of data. For instance, the algorithms used to detect anomalies in images are different to the approaches used on data streams.

With the ever-growing computational power in recent decades, machine learning approaches increased in popularity for data science tasks such as classification and pattern detection. Therefore, many researchers started to use these machine learning methods to detect anomalies in time series.

2.2 Definition of time series

A common definition of time series is that it is a sequence of data taken at successive equally spaced in time. Thus it is a sequence of discrete-time data.

Time series are used in statistics, signal processing, pattern recognition, finance, weather forecasting, astronomy, communications engineering, and largely in any domain of applied science and engineering which involves temporal measurements.

Time series analysis comprises methods for analyzing time-series data to extract meaningful statistics and other characteristics of the data. The difference between a simple regression task and a time series analysis is that, in the latter case, the model must not only learn the correlation between characteristics but also the correlation with time.

2.3 Univariate and multivariate time series

Researchers and authors have defined two main categories of time series.

Definition 2.3.1 (Univariate time series) *A univariate time series* $X = \{x_t\}_{t \in T}$ *is defined as an ordered set of real-valued observations,* x_t*, where each observation is recorded at a specific* $t \in T \subseteq \mathbb{Z}^+$*.*

Definition 2.3.2 (Multivariate time series) *A multivariate time series* $X = \{x_t\}_{t \in T}$ *is defined as an ordered set of* k*-dimensional vectors, each of which is recorded at specific time* $t \in T \subseteq \mathbb{Z}^+$ *and consists of* k *real-valued observations,* $x_t = (x_{1t}, ..., x_{kt})$*.*

A univariate detection method only considers a single time-dependent variable, whereas a multivariate detection method is able to simultaneously work with more than one variables.

Moreover, the detection method can be univariate even if the input data is a multivariate time series because an individual analysis can be performed on each time-dependent variable without considering the dependencies that may exist between the variables.

In contrast, a multivariate technique cannot be used if the input data is a univariate time series.

Since correlation dependencies between the variables are not considered when applying univariate techniques to each time-dependent variable, we can overcome this problem by preprocessing the time series with a *dimensionality reduction* technique called *PCA*.

In this way, the new features are combinations of the initial input variables and also uncorrelated each other (Figure 2.1).

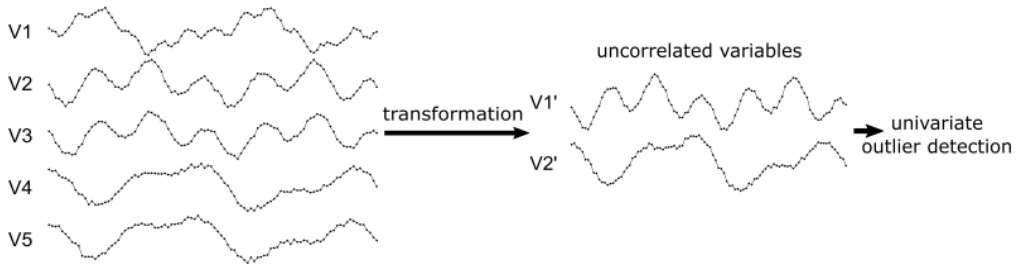

Figure 2.1: Effect of PCA in a multivariate time series (image taken from [4]).

2.4 Autocorrelation

Since a time series is a sequence of values for different timestamps, it could be useful to find the temporal correlation within the same features.

Just as correlation measures the extent of a linear relationship between two variables, autocorrelation measures the linear relationship between lagged values of the same feature of a time series (hence the name autocorrelation).

There are several autocorrelation coefficients, corresponding to each panel in the lag plot. For example, r_1 measures the relationship between y_t and y_{t-1}, r_2 measures the relationship between y_t and y_{t-2} and so on.

The value of r_k can be written as:

$$r_k = \frac{\sum_{t=k+1}^{T} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T} (y_t - \bar{y})^2} \tag{2.1}$$

where T is the length of the time series. The autocorrelation coefficients make up the *autocorrelation function* or ACF.

The autocorrelation plot can be also used to view if the time series has a trend or seasonal behaviour. When data have a trend, the autocorrelations for small lags tend to be large and positive because observations nearby in time are also nearby in size. So the ACF of trended time series tend to have positive values that slowly decrease as the lags increase.

When data are seasonal, the autocorrelations will be larger for the seasonal lags (at multiples of the seasonal frequency) than for other lags.

When data are both trended and seasonal, there is a combination of these effects.

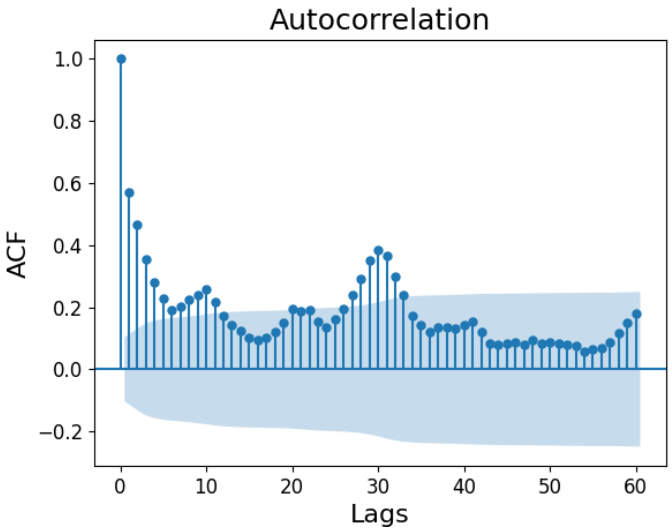

Figure 2.2: Autocorrelation plot.

The autocorrelation plot in Figure 2.2 represents the temporal correlation of the temperature recorded in Barcelona during 2019. It is possible to observe that the correlation between a day and the four preceding days is very strong, this means that the temperature of one day depends on one of the days immediately before.

Moreover, the autocorrelation plot has peaked with lags equal to 10 and 30 (one month) thus for a given day d_t has a strong correlation with the temperature of $d_{t-1:t-4}$ but also with temperature further back in time.

We can conclude by saying that the time series considered as an example is stationary with no associated trend because the autocorrelation decreases quickly for small lags but with a likely seasonal behaviour.

2.5 Trend

Trend is a pattern in data that shows the movement of a series to relatively higher or lower values over a long period of time. In other words, a trend is observed when there is an increasing or decreasing slope in the time series. The trend usually happens for some time, then disappears and it does not repeat.

In Figure 2.3, the antidiabetic drug sales in Australia show a clear and increasing trend of sales during the years.

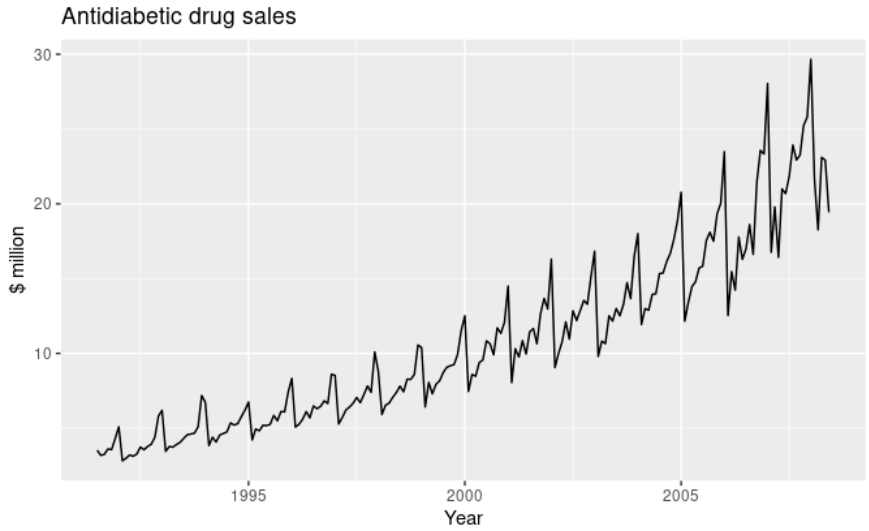

Figure 2.3: Monthly sales of antidiabetic drugs in Australia (image from [5]).

2.6 Seasonality

A seasonal pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week. Seasonality is always of a fixed and known period.

For example, the monthly sales of antidiabetic drugs (Figure 2.4) show seasonality which is induced partly by the change in the cost of the drugs at the end of the calendar year.

In this case, it is clear that there is a large jump in sales in January each year. Actually, these are probably sales in late December as customers stockpile before the end of the calendar year, but the sales are not registered with the government until a week or two later. The graph also shows that there was an unusually small number of sales in March 2008 (most other years show an increase between February and March). The small number of sales in June 2008 is probably due to incomplete counting of sales at the time the data were collected.

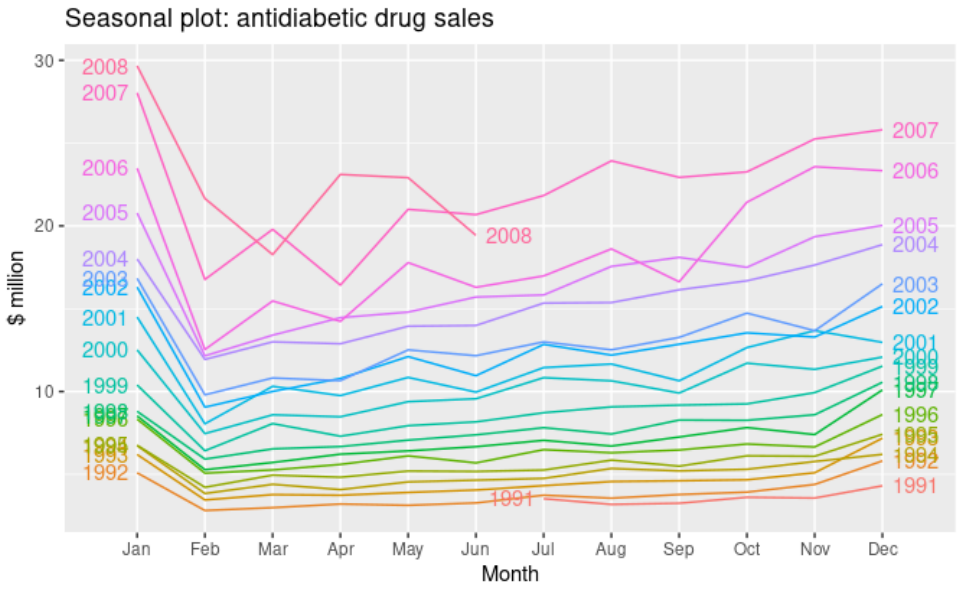

Figure 2.4: Example of seasonality (image taken from [5]).

2.7 Cycles

A cycle occurs when the data exhibit rises and falls that are not of a fixed frequency. These fluctuations are usually due to economic conditions, and are often related to the "business cycle." The duration of these fluctuations is usually at least 2 years.

Cyclic behaviour is quite different from seasonal behaviour. If the fluctuations are not of a fixed frequency then they are cyclic; if the frequency is unchanging and associated with some aspect of the calendar, then the pattern is seasonal.

In general, the average length of cycles is longer than the length of a seasonal pattern, and the magnitudes of cycles tend to be more variable than the magnitudes of seasonal patterns.

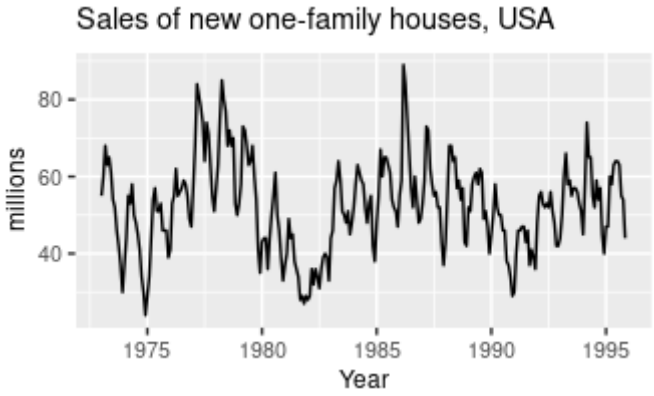

Figure 2.5: Example of a cycle (image taken from [5]).

The monthly housing sales in the USA, in Figure 2.5, show strong seasonality within each year, as well as some strong cyclic behaviour with a period of about 6–10 years.

2.8 Stationarity

Intuitively, a stationary time series is a time series having the same characteristics over every time interval or in other words whose properties do not depend on the time at which the series is observed. Formally, we can express it as follow:

Definition 2.8.1 *X_t is a stationary time series, if $\forall s \in \mathbb{R}$: the distribution of $(x_t, ..., x_{t+s})$ is equal.*

The above definition implies that a stationarity time series $x_1, ..., x_T$ will have the following characteristics:

1. **Constant mean**, thus no trend exists in the time series.
2. The time series has a **constant variance**.
3. There is a **constant autocorrelation** over time.
4. The time series has **no seasonality**, i.e., no periodic fluctuations.

Most of the time series is not stationary but some methods could help to make the data close to the stationarity.

Differencing One of the most used methods is the *differencing* because it can happen that a time series is not stationary but the differences between consecutive observations are. Therefore the time series after the transformation is given as $x'_t = x_t - x_{t-1}$.

Is always better to have stationary time series? Machine learning methods are used when the classical methods fail and better results are needed. It is impossible to know how to best model unknown nonlinear relationships in time series data and some methods may result in better performance when working with non-stationary observations or some mixture of stationary and non-stationary views of the problem.

In conclusion, stationary time series are not always preferred but this is part of the feature engineering/selection when using machine learning methods.

2.9 Decomposition

Time series data can exhibit a variety of patterns, and it is often helpful to split a time series into several components (*trend, seasonality* and *cycles*), each representing an underlying pattern category.

During the decomposition, usually, the trend and cycle are combined into a single trend-cycle component. Hence, a time series can be viewed as a combination of three components: a trend-cycle component, a seasonal component, and a remainder component (containing anything else in the time series).

Often this is done to help improve understanding of the time series, but it can also be used to improve forecast accuracy.

When decomposing a time series, it is sometimes helpful to first transform or adjust the series in order to make the decomposition (and later analysis) as simple as possible.

An *additive* decomposition is when:

$$y_t = S_t + T_t + R_t, \tag{2.2}$$

where y_t is the data, S_t is the seasonal component, T_t is the trend-cycle component, and R_t is the remainder component, all at period t. Alternatively, a multiplicative decomposition would be written as

$$y_t = S_t \times T_t \times R_t. \tag{2.3}$$

The additive decomposition is the most appropriate if the magnitude of the seasonal fluctuations, or the variation around the trend-cycle, does not vary with the level of the time series. When the variation in the seasonal pattern, or the variation around the trend-cycle, appears to be proportional to the level of the time series, then a multiplicative decomposition is more appropriate. Multiplicative decompositions are common with economic time series.

Many time series include trends, cycles and seasonality. When choosing a forecasting method, the first step is to identify patterns in the time series data, and then choose a method that is able to capture those patterns properly.

2.10 Types of Anomalies

In literature, as shown in Figure 2.6, anomalies in time series can be broadly classified into three categories: *point anomalies*, *contextual anomalies* and *collective anomalies*.

In recent years, due to the increase of the complexity of the reality that we want to model, also the complexity of the anomalies is increased. Therefore, it is needed to have tools to analyze such data, learn the patterns and autonomously detect anomalies.

For this reason, Deep anomaly detection (DAD) methods have been shown to detect all three types of anomalies with great success.

Deep learning models can detect anomalies in both univariate and multivariate data whether the anomaly afflicts a single sensor or more time-dependent variables. In other words, in the case of multivariate time series, we can find anomalies in one or more features.

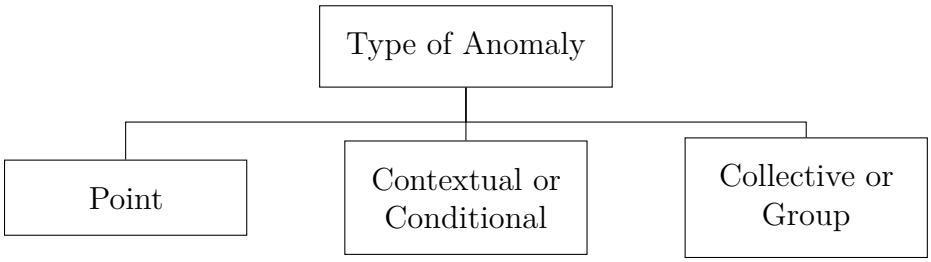

Figure 2.6: Taxonomy of anomalies.

2.10.1 Point Anomalies

The most simple and common anomaly in the area of time series is the *point anomaly*. Point anomalies are points that deviate from the majority of other samples and often represent an irregularity or deviation that happens randomly and may have no particular interpretation.

A naive approach for a given univariate time series can be to consider as an anomaly each point x_t with distance from its expected value higher than a predefined threshold τ:

$$|x_t - \hat{x_t}| > \tau \tag{2.4}$$

where x_t is the observed value and $\hat{x_t}$ is its expected value.

For instance, in Figure 2.7, the daily temperature in Barcelona in 2019 are recorded and the 23^{rd} January seems a point anomaly since it significantly deviates from the rest of the values.

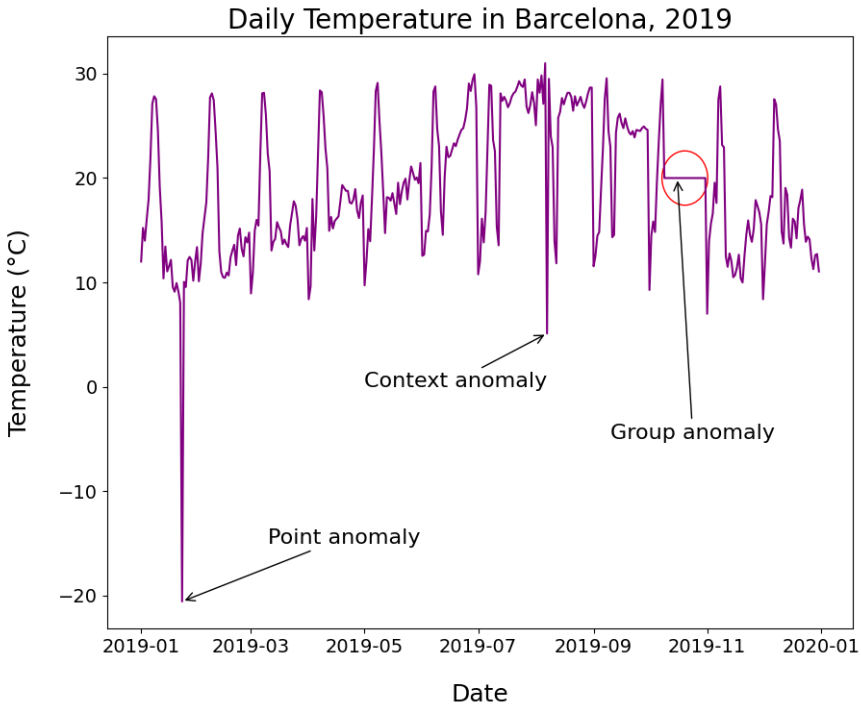

Figure 2.7: Example of the types of anomalies.

2.10.2 Contextual Anomalies

A contextual anomaly is a data instance that could be considered anomalous only in some specific context. These anomalies are the most difficult to recognize because we must train the model in a way that it is able to capture also the context of each timestamp.

Figure 2.7 illustrates the example of a contextual anomaly considering temperature data indicated by a drastic drop in August; this value is not indicative of a normal value found during this specific month and in general in summer. Having a daily temperature of 5°C in winter in Barcelona is normal but the same temperature in summer is regarded as an anomaly.

2.10.3 Collective or Group Anomaly Detection

Anomalous collections of individual data points are known as collective or group anomalies, where each point individually appears as normal while observed in a group exhibit unusual characteristics. Moreover, these sequences can be *periodic subsequence outliers* that are repeated over time. Unlike point outliers where

periodicity is not relevant, periodic subsequence outliers are important in areas such as fraud detection because it might be interesting to discover certain periodic anomalous transactions over time.

In the example of the temperature in Barcelona (Figure 2.7, for two weeks the temperature remains exactly the same and it might probably not seem anomalous because the temperature is reasonable. On the other hand, it is very unlikely that for 15 days the temperature remains perfectly constant thus we can conclude by saying that probably the sensor stopped work for that specific range of dates.

2.11 Type of learning

Anomaly detection is a binary task where a sample can be normal or an outlier. Anomalies are rare and most of the time it is challenging to obtain their labels because we need to break manually the machine and then record the samples.

Based on the data and labels available, it is possible to divide the algorithms into three categories as shown in figure 2.8.

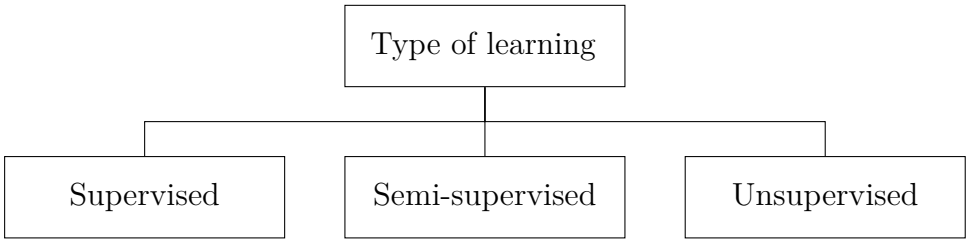

Figure 2.8: Taxonomy based on the type of models.

2.11.1 Supervised

Supervised methods involve training a binary model, using labels of both normal and anomalous data instances.

The supervised techniques usually have higher performance compared to other methods as they use the labelled data from both classes. They can learn the boundary from the labelled training examples and then more easily classify the unseen test data.

However, the anomalies available during the training may not represent the full spectrum of anomalies, supervised approaches may overfit and perform poorly on unseen anomalous data.

Furthermore, these methods are not as popular as semi-supervised or unsupervised methods due to the lack of availability of labels. Moreover, the performance

is sub-optimal because the number of normal samples is far more than the total number of anomaly ones.

2.11.2 Semi-supervised

We refer to the anomaly detection techniques as semi-supervised if they utilize unlabeled contaminated data in addition to labelled instances of the in-distribution class. Since the labels of normal instances are easier to obtain than anomalies, these methods are more widely adopted. The learning phase involves the normal sample to be able to separate outliers. One of the most used semi-supervised models is the autoencoder, where it learns to reconstruct the normal samples during training. With sufficient training samples, the model produces low reconstruction errors on normal points whereas fails on anomalies samples.

Other famous semi-supervised methods are the one-class SVN and GANs that are also widely used in this area of research.

2.11.3 Unsupervised

This last type of learning detects outliers solely based on the properties of the data instances without labels. Unsupervised techniques are quite flexible and broadly applicable as they do not rely on the availability of labels.

These methods learn to inherit characteristics and the boundary between the normal instances and the anomalies ones in an autonomous way. Unfortunately, this flexibility comes at cost of robustness because unsupervised techniques are very sensitive to noise and data corruption and are often less accurate than supervised or semi-supervised algorithms.

Nowadays, in more and more fields, researchers are focusing on unsupervised learning due to its high flexibility and the lack of labels. Therefore, this type of learning is growing and it is adopted more and more.

2.12 Pipeline

A machine learning pipeline is a way to codify the workflow it takes to produce a machine learning model. It consists of multiple sequential steps that span from data extraction and preprocessing to model training and testing.

In the case of anomaly detection, the pipeline can be summarized as shown in Figure 2.9.

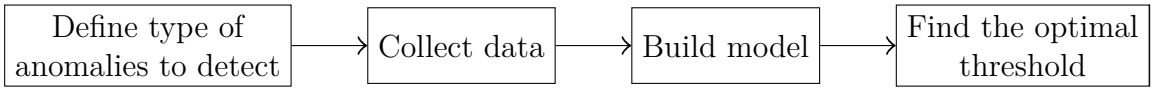

Figure 2.9: Scheme of the pipeline in DAD.

1. Define the type of anomalies to detect The first step is to define the type of anomalies we expect to detect. It is mainly based on domain knowledge and it is very important because each type of outlier has its characteristics and requires different detection methods.

For example, group anomalies require an evaluation method with a high temporal dependency thus in this way it can recognize sequences of anomalies over time. On the other hand, to detect point anomalies the detection algorithm should focus more on single instances instead of sequences.

2. Collect data This phase span from data mining to data preprocessing. Raw data require transformations or modifications because, depending on the nature of the data, there could be missing values at certain time-steps, features with different scales which require scaling or also features that can be removed or added.

Moreover, with data analysis, we can model the data to discover useful information and correlations between the features.

3. Build model Next when the data have been processed and the type of anomalies that we want to identify is clear, we move on to the creation of the model. Depending on the task, we can choose a model from the ones existing in the literature or decide to develop a new model build on top of the knowledge of the domain and the data.

After the definition of the architecture of the model, we train the network with the data collected in the previous step.

4. Find the correct threshold This last step is the most difficult because there are many ways to separate the normal instances from the anomalies ones. The boundary between the normal points and the anomalies is often hard to define due to many factors such as the high dimensionality of input data or the nature of the anomalies.

Almost all researches are focused now on this last step because is the most important in order to detect the anomalies.

2.13 Autoencoders

An auto-encoder is a type of neural network used in unsupervised learning in which the network is composed of an encoder and a decoder sub-models. The encoder forces a compressed representation of the input in smaller dimensions and the decoder attempts to recreate the input from the compressed version provided by the encoder.

Auto-encoders are applied to many problems, from facial recognition, feature detection and data denoising. They represent data within multiple hidden layers by reconstructing the input data, effectively learning an identity function. When trained solely on normal data instances, they fail to reconstruct the anomalous data samples producing a large reconstruction error. These points associated with a high residual error are considered anomalies.

The choice of autoencoder architecture depends on the nature of data, convolutional networks are preferred for image datasets while Long short-term memory (LSTM) based models are able to capture the time dependency in sequential data.

The deep of an autoencoder depends on the dimension of the input data. The more dimensions, the more layers are needed to extract all the relevant information during training.

The type of learning is unsupervised because the model does not require any information about the labels, making it very popular and widely used in literature.

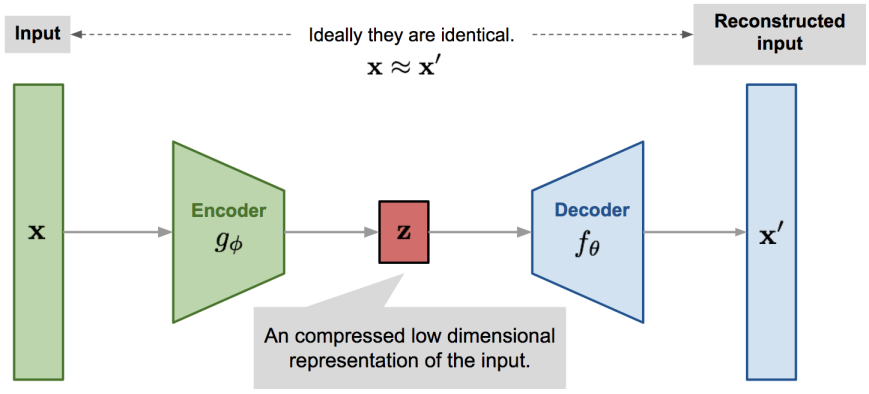

Figure 2.10: Scheme of a generic autoencoder (image taken from [6]).

As shown in Figure 2.10, The input x is compressed by the encoder g_ϕ to a lower dimensional space z and then the decoder f_θ tries to reconstruct the original input. The parameters (ϕ, θ) are learned together to output a reconstructed data sample same as the original input, $x \approx f_\theta(g_\phi(x))$.

There are various metrics to quantify the difference between two vectors. One

of the most adopted metrics is the MSE loss:

$$\mathbb{L}_{AE}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}(x_i - f_\theta(g_\phi(x_i)))^2 \tag{2.5}$$

Autoencoders for dimensionality reduction Auto-encoders with a single layer along with a linear activation function are nearly equivalent to Principal Component Analysis (PCA). While PCA is restricted to a linear dimensionality reduction, autoencoders enable both linear or non-linear transformations. The difference between these two approaches is visualized in the figure 2.11 below.

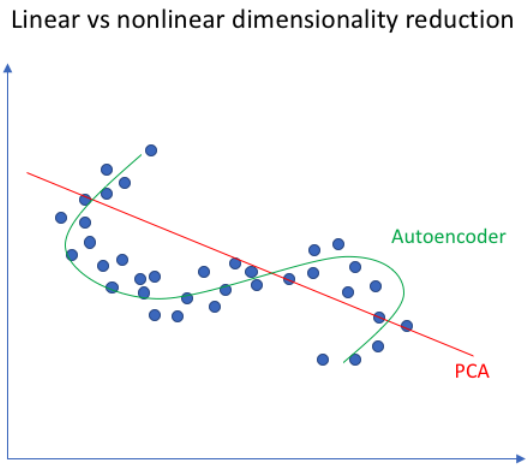

Figure 2.11: Comparison between PCA and an autoencoder (image taken from [7]).

The big deal with autoencoders One reason why they have attracted so many researches and attention is that they have long been thought to be a potential avenue for solving problems without the need for labels.

Autoencoders are not a truly unsupervised learning technique (which would imply a different learning process altogether), they are a self-supervised technique, a specific instance of supervised learning where the targets are generated from the input data.

Although autoencoders are simple and effective architectures for outlier detection, the performance can get degraded due to noisy training data.

Chapter 3

Anomaly detection: state of the art and new proposals

3.1 Introduction

The focus of this thesis is on deep learning models for anomaly detection in multivariate time series. In general, anomaly detection approaches can be categorized in terms of the type of data needed to train the model and the method used to define the boundary between the region with normal instances and the anomaly one.

In this chapter, we are going to discuss the actual state of the art for, respectively, the SWaT and MSL datasets.

Moreover, in the last two sections, we will present two new approaches, each of them with different properties and application contexts.

3.2 A Deep Learning Solution for Anomaly Detection in Industrial Control Systems

Real-world *Industrial Control Systems* (ICSs) are dynamic and operate in noisy environments: these factors may hamper the correct functioning of the detection system because they can "shift" the normal behaviour. In such scenarios, the main challenge is to periodically refresh the detection thresholds.

A common issue of one-class classification mechanisms is the high false alarm rate caused by the progressive evolution of the normal behaviour of the ICS with respect to the initial one, based on which the model was trained. This phenomenon, called *domain shift*, can be caused by changes in the industrial workflow, degradation of devices and communication links over time, installation/removal of devices, updates in the devices' firmware or configuration, or external noise on the communication

channels.

For example, in case of the SWaT dataset, a *domain shift* can be observed in the feature *AIT201* (Analyser Indication Transmitter) where in training set it assumes a range in interval $[251, 272]$ $\mu S/cm$ (micro Siemens per centimetre), while in the test one it ranges in $[168, 267]$ $\mu S/cm$ with a substantial different distribution, as illustrated in Figure 3.1.

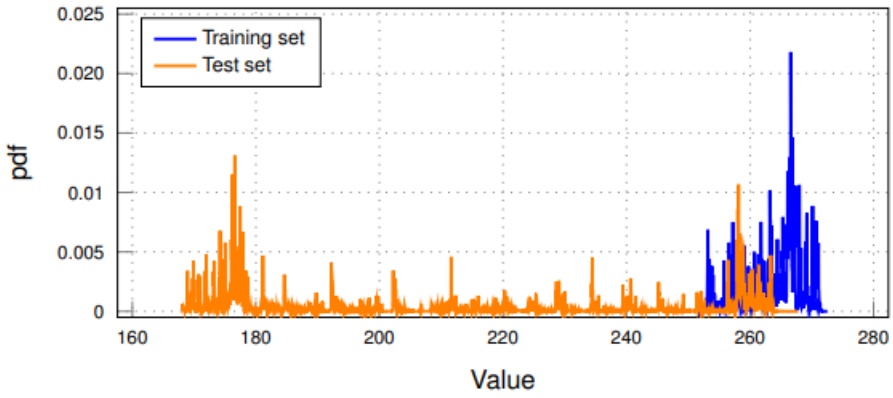

Figure 3.1: Shift of the distribution of sensor *AIT201*.

In summary, a common drawback of available solutions is that they are not flexible enough to quickly and efficiently adapt to changes in the production environment. In a water treatment plant, examples of such changes are: increasing the size of a water tank or replacing a motorised valve with another with different operation modes. Instead, *Abdelaty et all.* [8] have proposed a new approach based on a dynamic threshold that evolves with the system.

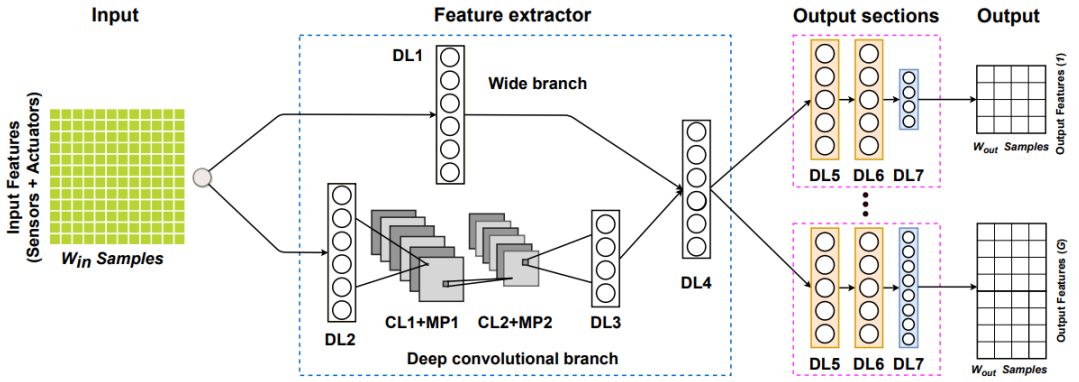

Figure 3.2: Scheme of the architecture.

In Figure 3.2, we can see the network architecture and in particular, the *wide branch* that memorizes and learns the relationships between features in the training set and the *deep branch* that tries to explore relationships that do not exist in the training.

Instead, the output section is split into multiple branches to serve large-scale ICSs where the sensors are controlled by several PLCs, usually specialised on a specific part of the industrial process with specific behaviour and anomaly threshold.

Moreover, the authors of the paper, to prevent the forecasting model to copy the last values of the input time windows, the W_{out} is separated by a time interval called *horizon H*.

The neural network presented above has been trained to minimise the Mean Square Error (MSE) cost function.

TTNN To have a dynamic threshold, the authors have introduced the *Tuning Threshold Neural Network*.

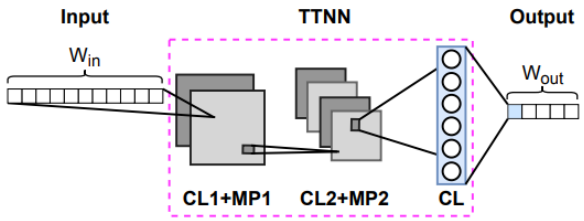

Figure 3.3: Architecture of TTNN. Only the first element of the output is kept.

As input, we have the prediction error MSE as a univariate time series. We use G instances of TTNN, one for each output section of the model, to tune the thresholds $T_g, g \in [1, G]$.

Then, each instance is trained with the prediction error MSE_g measured on a single output section g on the validation set, which only contains benign records, as the training set. The trained model is used in the online system to compute the optimal anomaly threshold T_g using past prediction errors.

In the end, the threshold T_g used for anomaly detection on sensors values collected in time interval $[t, t+s)$ is obtained using past prediction errors computed in time interval $[t_0, t_0 + s + W_{in})$, where $t_0 = t - H - W_{in}$.

Anomaly detection in actuators To detect anomalies in the actuators the authors suggested to create a database to store, during the training, all the combinations of the actuators.

During the evaluation, if a combination has no occurrences in the dataset it is marked as an anomaly.

Anomaly detection in sensors In Figure 3.4, we can observe how the model reports an anomaly.

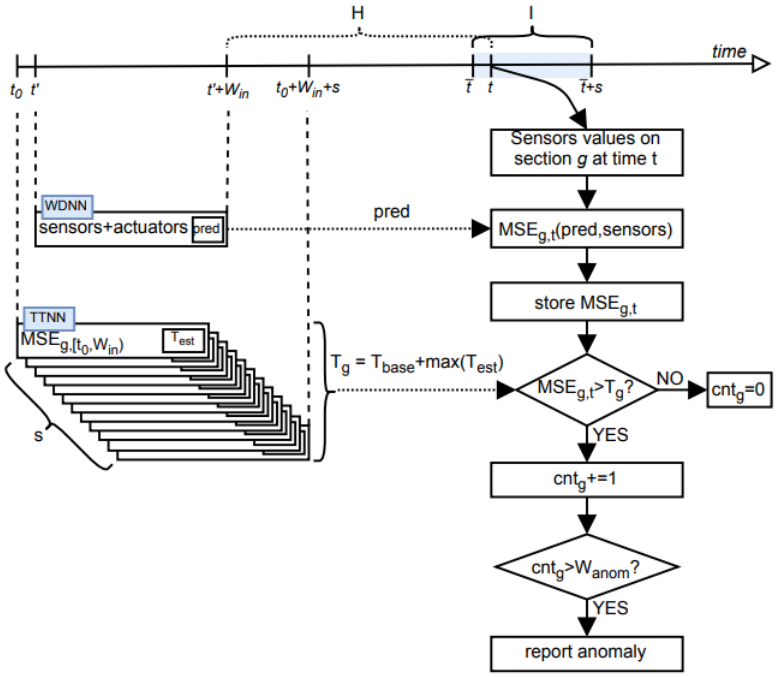

Figure 3.4: Detection process on section g of sensors through WDNN and adaptive thresholds.

To reduce the false positives caused by sudden changes in the underlying physical process, DAICS only reports an anomaly at time t on the sensors if the anomaly condition has been also previously observed for W_{anom} consecutive sampling intervals.

The few-time-steps algorithm The few-time-steps algorithm has been designed to efficiently reconfigure DAICS in a production environment in the case an anomaly is identified by the system and then recognised as a false alarm by the technician. The only assumption is that the technician can recognise false alarms caused by changes in the normal operating condition of the ICS.

In the case of a false positive in the actuators, the algorithm adds that combination to the database when the technician has determined that is normal and that must be treated as such by the system.

Instead, if the false positive is one or more sensors, the output sections of the neural network that have produced the false alarm are updated according to the predefined number of epochs. SGD is employed to fine-tune the output section

through multiple gradient steps and the prediction loss is calculated for the data samples aggregated in a batch of S samples containing the false alarm.

3.3 Multivariate Timeseries Anomaly Detection via Graph Attention Network

Zhao et all. [9] have proposed a novel framework in which they try to model the correlations between different features explicitly, while the temporal dependencies within each time-series are modelled at the same time.

The key ingredients of this method are two graph attention layers, namely the *feature-oriented graph attention layer* and *the time-oriented graph attention layer*. Whereas the first layer captures the causal relationships between multiple features, the time-oriented graph attention layer underlines the dependencies along the temporal dimension.

In addition, they integrate the advantages of a forecasting-based model and a reconstruction-based model for better representations of time series data. The two models can be optimized simultaneously by a joint objective function. The forecasting-based model focuses on single-timestamp prediction, while the reconstruction based model learns a latent representation of the entire time series.

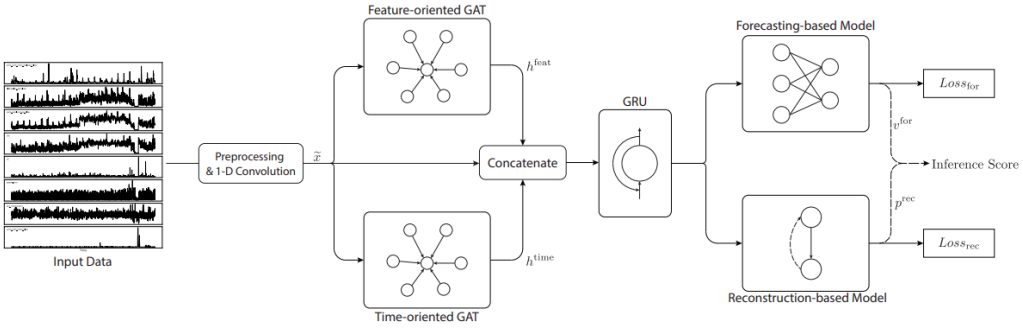

Figure 3.5: The architecture of MTAD-GAT.

In Figure 3.5, it is shown the architecture of the entire model which is composed of the following modules in order:

1. A 1-D convolution layer at the first layer to extract high-level features of each time-series input.

2. The outputs of the 1-D convolution layer are processed by two parallel graph attention (GAT) layers, which underline the relationships between multiple features and timestamps.

3. The output representations from the 1-D convolution layer and two GAT layers are concatenated together, feeding them into a Gated Recurrent Unit (GRU) layer with d1 hidden dimension. This layer is used for capturing sequential patterns in time series.

4. The outputs of the GRU layer are fed into a forecasting-based model and a reconstruction based model in parallel to obtain the final result. The forecasting-based model is implemented as a fully-connected network, and adopt VAE for the reconstruction-based model.

The reason why both forecasting-based and reconstruction-based models are implemented is that they have shown their superiority in some specific situations. The forecasting-based model is specialized for feature engineering of next timestamp prediction, and the construction-based model is good at capturing the data distribution of the entire time series.

Moreover, none of the existing solutions in the literature capture the correlations between multiple features explicitly, which is emphatically addressed in this algorithm to enhance the performance of multivariate time-series anomaly detection.

Generally, given a graph with n nodes, i.e., $\{v_1, v_2, ..., v_n\}$, where v_i is the feature vector of each node, a GAT layer computes the output representation for each node as follows:

$$h_i = \sigma\Big(\sum_{j=1}^{L} \alpha_{ij} v_j\Big) \tag{3.1}$$

where h_i denotes the output representation of node i, which has the same shape with input v_i; σ represents the sigmoid activation function; α_{ij} is the attention score which measures the contribution of node j to node i, where j is one of the adjacent nodes for node i; L denotes the number of adjacent nodes for node i.

The attention score α_{ij} can be computed by the following equations:

$$e_{ij} = LeakyReLU(w^T \dot{(} v_i \oplus v_j)) \tag{3.2}$$

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{j=1}^{L} exp(e_{ij})} \tag{3.3}$$

Here $\oplus$ represents the concatenation of two-node representations, $w \in R^{2m}$ is a column vector of learnable parameters where m is the dimension of the feature vector of each node, and $LeakyReLU$ is a nonlinear activation function.

In detail, in the *feature-oriented graph attention layer* each node represents a certain feature, and each edge denotes the relationship between two corresponding features. Specifically, each node x_i is represented by a sequential vector $x_i = \{x_{i,t} | t \in [0, n)\}$ and there are totally k nodes, where n is the total number of timestamps and k is the total number of multivariate features.

On the other hand, in the *time-oriented graph attention layer* a node x_t represents the feature vector at timestamp t, and its adjacent nodes include all other timestamps in the current sliding window. The output of the feature-oriented graph attention layer is a matrix with shape $k \times n$ where each row is an n dimensional vector representing the output for each node and there are in total k nodes.

During the training process, the parameters from both models are updated simultaneously. The loss function is defined as the sum of two optimization targets, i.e., $Loss = Loss_{for} + Loss_{rec}$, where $Loss_{for}$ denotes the loss function of forecasting-based model and $Loss_{rec}$ denotes the loss function of reconstruction-based model.

$$Loss_{for} = \sqrt{\sum_{i=1}^{k}(x_{n,i} - \tilde{x}_{n,i})^2} \tag{3.4}$$

$$Loss_{recon} = -E_{q_\phi(z|x)}[logp_\theta(x|z)] + D_{KL}(q_\phi(z|x) \,||\, p_\theta(z)) \tag{3.5}$$

The model assigns a *score* to each point according to the formula:

$$score = \sum_{i=1}^{k} \frac{(\hat{x}_i - x_i)^2 + \gamma \times (1 - p_i)}{1 + \gamma} \tag{3.6}$$

where $(\hat{x}_i - x_i)^2$ is the squared error between the forecasting value $\hat{x}_i$ and the actual value x_i, indicating how much the actual value of feature i deviates from prediction; $(1-p_i)$ is the probability of encountering an abnormal value for feature i according to the reconstruction model; k is the total number of features; γ is a hyper-parameter to combine the forecasting-based error and the reconstruction-based probability.

3.4 Deep Evidential Regression

In recent years, uncertainty estimation has had a central role and a high impact on society. Standard neural networks are black-box predictors, from a given input they return an output but we do not know the intermediate steps. Unfortunately, there are some tasks in which we need to know the uncertainty of the prediction, mostly in tasks in which an automatic operation is executed immediately after the prediction. This is due to the fact that a wrong prediction can lead to the execution of a wrong operation that could have potentially high risk in particular tasks such as autonomous driving, medical domain, high imbalanced datasets or data with high bias.

Therefore is very important to calibrate the uncertainty of the prediction to make the model in condition to understand when to trust the prediction and proceed with the execution (low uncertainty) or does not do anything (high uncertainty).

In literature, Bayesian neural networks can output both the prediction and the uncertainty associated with that prediction through assumptions a priori on the weights of the network. In this manner, the model is able to answer "I do not know" when it does not truly know the correct prediction.

This property changes how we train the model because in addition to optimizing the network to output the correct prediction we must check that the uncertainty is coherent with the prediction.

From the human perspective, having the uncertainty of the prediction is very important and makes trusting the model easier because humans know that further operations are executed only if the uncertainty is low.

For example, in the case of autonomous drive, the model should apply the brakes only if it recognizes a pedestrian in the trajectory of the car. This automatic operation must be executed only if the model is very sure that there is a pedestrian otherwise there is a risk of a car accident.

As we will see, Deep Evidential Regression is able to capture the uncertainty along with the prediction in a similar Bayesian way.

Precise uncertainty estimates are useful for recognizing out-of-distribution (OOD) test samples and when the model is likely to fail.

There are two different uncertainties that can be calculated:

1. uncertainty in the input data, called *aleatoric uncertainty*.

2. uncertainty in the prediction, called *epistemic uncertainty*.

In literature, already exists methods able to capture and model the uncertainty. Bayesian networks place probabilistic priors over network weights and use sampling operations to approximate the output variance. On the other hand, there are several disadvantages including the computational cost of sampling during inference and the question of how to choose the right prior distribution.

With Deep Evidential Regression, the prior assumptions are not more on the weights as done in Bayesian NNs, but directly over the likelihood function. Therefore, by training the model to output the hyperparameters of the higher-order evidential distribution, we get simultaneously the prediction along with the epistemic and aleatoric uncertainty without the need for sampling as shown in Figure 3.6.

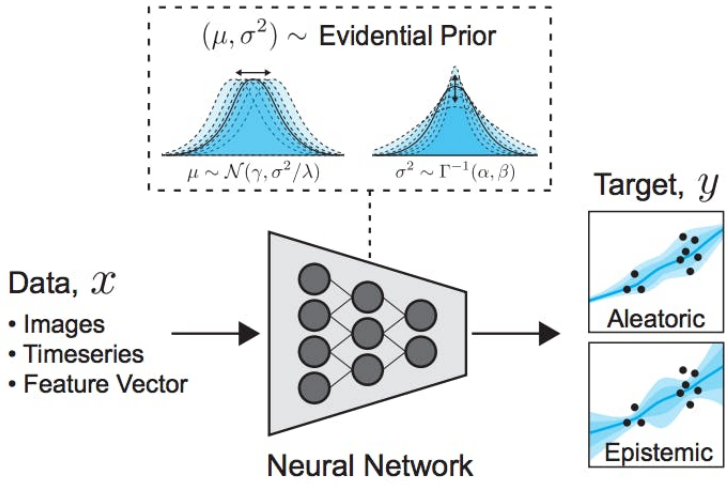

Figure 3.6: General overview of deep evidential regression model.

3.4.1 Evidential uncertainty for regression

In case of regression, we assume our target, y_i were drawn i.i.d from a normal distribution with parameters $\boldsymbol{\theta} = (\mu, \sigma^2)$. With maximum likelihood estimation (MLE), we aim to learn a model to infer $\boldsymbol{\theta}$ that maximize the likelihood of observing our targets, y, given by $p(y_i \,|\, \boldsymbol{\theta})$. We proceed in this way by minimizing the negative log likelihood loss function:

$$\mathcal{L}_i(\boldsymbol{w}) = -\log p(y_i \,|\, \mu, \sigma^2) = \frac{1}{2} log(2\pi\sigma^2) + \frac{(y_i - \mu)^2}{2\sigma^2} \tag{3.7}$$

In learning $\boldsymbol{\theta}$, this function models successfully the uncertainty in the data but not also the predictive epistemic uncertainty.

With Deep Evidential Regression, the problem setup is the same as before but now y_i are drawn i.i.d from a normal distribution with *unknown mean and variance*. This leads to placing a Gaussian prior distribution on the unknown μ and an Inverse-Gamma prior on the unknown σ^2:

$$\begin{gathered} (y_1, \ldots, y_N) \sim \mathcal{N}(\mu, \sigma^2) \\ \mu \sim \mathcal{N}(\gamma,\, \sigma^2 \upsilon^{-1}) \qquad \sigma^2 \sim \Gamma^{-1}(\alpha,\, \beta) \end{gathered} \tag{3.8}$$

We use the network to infer $m = (\gamma, \upsilon, \alpha, \beta)$ where $\gamma \in \mathbb{R}$, $\upsilon > 0$, $\alpha > 1$, $\beta > 0$.

To obtain an approximation for the true posterior, we assume that the two distributions are independent and the estimated distribution can be factorized such that $q(\mu, \sigma^2) = q(\mu)\, q(\sigma^2)$.

Therefore, the approximation takes the form of the Gaussian conjugate prior, the Normal Inverse-Gamma (NIG) distribution:

$$p(\underbrace{\mu, \sigma^2}_{\boldsymbol{\theta}} \,|\, \underbrace{\gamma, \upsilon, \alpha, \beta}_{\boldsymbol{m}}) = \frac{\beta^\alpha \sqrt{\upsilon}}{\Gamma(\alpha)\sqrt{2\pi\sigma^2}} \Big(\frac{1}{\sigma^2}\Big)^{\alpha+1} exp\Big\{ -\frac{2\beta + \upsilon(\gamma-\mu)^2}{2\sigma^2} \Big\} \tag{3.9}$$

In other words, we can interpret this as the *higher-order evidential* distribution on top of the unknown *lower-order* likelihood distribution from which observations are drawn.

3.4.2 Prediction and uncertainty estimation

As said in the previous section, the model tries to infer two uncertainties. The first one is the aleatoric uncertainty, also referred to as statistical or data uncertainty which is representative of unknowns that differ each time we run the same experiment. The epistemic uncertainty describes the estimated uncertainty in the prediction.

Given a NIG distribution we can compute these metrics as:

$$\begin{aligned}
\mathbb{E}[\mu] &= \int_{\mu=-\infty}^{\infty} \mu\, p(\mu)\, d\mu && = \text{mean of normal distribution p}(\mu) \\
\mathbb{E}[\sigma^2] &= \int_{\sigma=0}^{\infty} \sigma^2\, p(\sigma^2)\, d\sigma^2 && = \text{mean of inverse gamma distribution p}(\sigma^2) \\
Var[\mu] &= \int_{\mu=-\infty}^{\infty} \mu^2\, p(\mu)\, d\mu \,-\, (\mathbb{E}[\mu])^2 && = \text{variance of normal distribution p}(\mu)
\end{aligned} \tag{3.10}$$

Then:

$$\underbrace{\mathbb{E}[\mu] = \gamma,}_{\text{prediction}} \qquad \underbrace{\mathbb{E}[\sigma^2] = \tfrac{\beta}{\alpha-1},}_{\text{aleatoric}} \qquad \underbrace{Var[\mu] = \tfrac{\beta}{\upsilon(\alpha-1)}}_{\text{epistemic}} . \tag{3.11}$$

3.4.3 Learning the evidential distribution

The learning task is divided in two distinct parts: (1) acquiring or maximizing model evidence in support of our observations and (2) minimizing the evidence or inflating uncertainty when the prediction is wrong. In other words, we can think of (1) as a way of fitting data to the evidential model while (2) enforces a prior to remove incorrect evidence and inflate uncertainty.

(1) Maximizing the model fit.

$$p(y_i|\boldsymbol{m}) = \frac{p(y_i|\boldsymbol{\theta}, \boldsymbol{m}) p(\boldsymbol{\theta}|\boldsymbol{m})}{p(\boldsymbol{\theta}|y_i, \boldsymbol{m})} = \int_{\sigma^2=0}^{\infty} \int_{\mu=-\infty}^{\infty} p(y_i|\mu, \sigma^2) p(\mu, \sigma^2|\boldsymbol{m})\, d\mu\, d\sigma^2 \tag{3.12}$$

The model evidence is, in general, not straightforward to evaluate since computing it involves integrating out the dependence on latent model parameters. However, in the case of placing a NIG evidential prior on Gaussian likelihood function an analytical solution does exist:

$$p(y_i|\boldsymbol{m}) = St(y_i; \gamma, \frac{\beta(1+\upsilon)}{\upsilon\alpha}, 2\alpha) \tag{3.13}$$

where $St(y; \mu_{St}, \sigma^2_{St}, \upsilon_{St})$ is the Student-t distribution evaluated at y with location μ_{St}, scale σ^2_{St} and υ_{St} degrees of freedom. We denote the loss as the negative logarithm of model evidence:

$$\mathcal{L}_i^{NLL}(\boldsymbol{w}) = \frac{1}{2}log(\frac{\pi}{\upsilon}) - \alpha\, log(\Omega) + \Big(\alpha + \frac{1}{2}\Big)log((y_i - \gamma)^2\upsilon + \Omega) + log\Big(\frac{\Gamma(\alpha)}{\Gamma(\alpha + \frac{1}{2})}\Big) \tag{3.14}$$

where $\Omega = 2\beta(1+\upsilon)$. This loss provides an objective for training the model to output parameters of a NIG distribution to fit the observations by maximizing the model evidence.

(2) Minimizing evidence on errors. Next, we try to minimize evidence on incorrect predictions. In case of a regression task, the minimization task involves a novel evidence regularizer, scaled on the error of the i-th prediction,

$$\mathcal{L}_i^R(\boldsymbol{w}) = |y_i - \mathbb{E}[\mu_i]| \cdot \Phi = |y_i - \gamma| \cdot (2\upsilon + \alpha) \tag{3.15}$$

This loss imposes a penalty whenever there is an error in the prediction and scales with the total evidence of our inferred posterior. In other words, large amounts of predicted evidence will not be penalized as long as the prediction is close to the target.

Summary and implementation details. The total loss, $\mathcal{L}_i(\boldsymbol{w})$, consists of the two loss terms for maximizing and regularizing evidence, scaled by a regularization coefficient, λ,

$$\mathcal{L}_i(\boldsymbol{w}) = \mathcal{L}_i^{NLL}(\boldsymbol{w}) + \lambda\, \mathcal{L}_i^R(\boldsymbol{w}) \tag{3.16}$$

Here, λ trades off uncertainty inflation with model fit. Setting $\lambda = 0$ yields an over-confident estimate while setting λ too high results in over-inflation.

In practice, our NN is trained to output the parameters, m, of the evidential distribution: $m_i = f(x_i, w)$. Since m is composed of 4 parameters, f has 4 output neurons for every target y. We enforce the constraints on $(\upsilon, \alpha, \beta)$ with a $softplus$ activation (and additional $+1$ added to α since $\alpha > 1$). Linear activation is used for $\gamma \in \mathbb{R}$.

For our experiments, we consider a dynamic λ that is updated at each step according to the following formula:

$$\lambda_{new} = \lambda + 10^{-4}\dot{(}\mathcal{L}_i^R - 10^{-2}) \tag{3.17}$$

Doing so, λ is updated proportionally to the changes of the regression loss.

3.5 Graph Neural Network

Most of the existing methods are not able to learn which sensors are related to one another and for this reason, the resulting model has difficulties to caption potential inter-relationships. This limits their ability to detect and explain deviations from such relationships when anomalies occur.

Recently, graph neural networks (GNNs) have shown success in detecting anomalies and at the same time modelling the relationships among the sensors. On the other hand, applying these methods to time series data faces two main challenges:

1. different sensors can have very different behaviours, e.g. one sensor may measure the temperature while another measures the pressure. General GNNs use the same model parameters to model the behaviour of each node and thus face limitations when different sensors have different behaviours.

2. typically GNNs treat the graph structure as input but most of the time relationships between sensors are initially unknown and should be learned along with our model.

Deng et all. [10] have proposed a novel attention-based approach called Graph Deviation Network (GDN), which learns a graph of relationships between sensors and detects anomalies from these patterns.

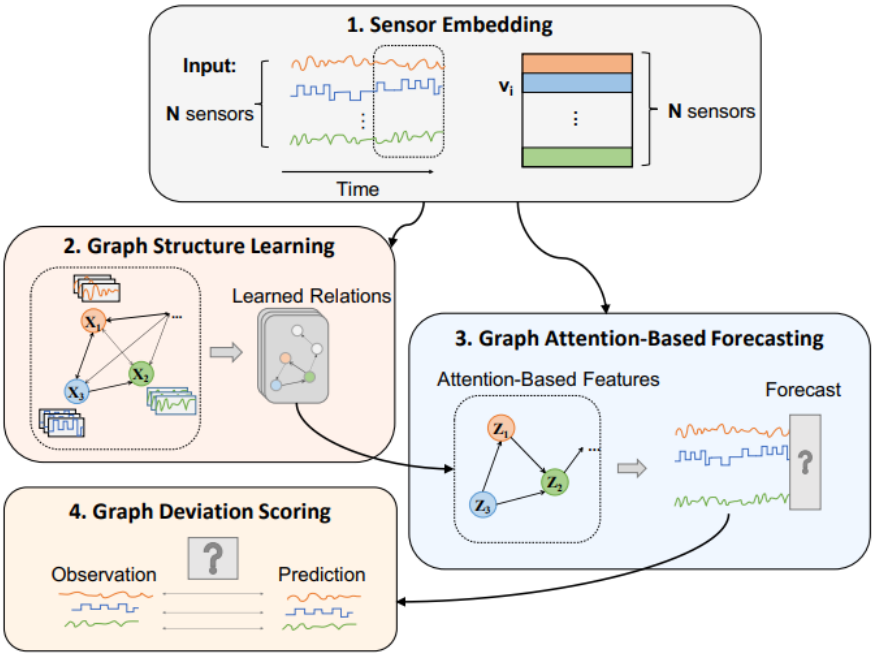

Figure 3.7: Overview of the proposed method.

As shown in Figure 3.7, this method involves four main components:

1. **Sensor Embedding**, which uses embedding vectors to capture the characteristics of each sensor.
2. **Graph Structure Learning** learns the graph data structure.
3. **Graph Attention-Based Forecasting** learns to predict the future value of a sensor based on attention function over its neighbours in the graph.
4. **Graph Deviation Scoring** calculates the deviation error of the predicted value.

This work is very important because provides an explainable model through its embeddings, its learned graph structure and by comparing the predicted and actual behaviour of the sensors.

Since this method separate input test data depending on the forecasting error, the training data is assumed to consist only of normal data.

3.5.1 Embedding

Machine learning models take vectors (arrays of numbers) as input. When working with text or in general with categorical features, the first thing we must do is come

up with a strategy to convert strings to numbers (or to "vectorize" the features) before feeding it to the model.

From the literature, we know the *One Hot Encoding* but this method is very inefficient because, to represent each value/word, we will create a zero vector with length equal to the vocabulary, then place a one in the index that corresponds to the desired value. Therefore, in doing so, we create a very sparse matrix with many zeros.

On the other hand, another interesting approach is to encode each word using a unique vector. Firstly, we need to define the size of the embedding that corresponds to the dimension used for the vector to identify each value.

For example, if we set 2 as embedding size and in our training set we have the following values:

[4, 1, 2, 3]

Once the network has been trained, we can get the weights of the embedding layer, which in this case will be of size (5, 2) and can be thought as the Table 3.1 used to map integers to embedding vectors:

Index	Embedding
0	[1.2, 3.1]
1	[0.1, 4.2]
2	[1.0, 3.1]
3	[0.3, 2.1]
4	[2.2, 1.4]

Table 3.1: Table to map each value to its vector.

So according to these embeddings, our data will be represented as:

[[2.2, 1.4], [0.1, 4.2], [1.0, 3.1], [0.3, 2.1]]

3.5.2 Sensor Embedding

As said in the previous section, different sensors can have very different characteristics and these characteristics can be related in complex ways. Ideally, we would want to represent each sensor in a flexible way that captures the correlation in a dimensional space with a given dimension (an hyperparameter).

$$v_i \in \mathbb{R}^d, \text{ for } i \in \{1, 2, \ldots, N\}$$

The idea behind the embeddings is that sensors with similar values should have a high tendency to be related to one another. In this model, these embeddings are used in two ways:

1. for structure learning
2. to perform attention over neighbors in a way

The embeddings are initialized randomly and then trained along with the rest of the model.

3.5.3 Graph Structure Learning

The main data structure is the **directed graph**, whose nodes represent sensors and whose edges represent dependency relationships between them. The decision to use a direct graph is motivated by the fact that the dependency patterns between sensors is not necessary symmetric. This means that an edge from one sensor to another indicates that the first sensor is used for modelling the behaviour of the second but the contrary is not true.

The adjacency matrix A stores this direct graph, where A_{ij} represents the presence of a directed edge from node i to j.

This method is very flexible because can be applied either to the usual case where we do not have prior information about the connection among sensors or in case where we have some prior information about which edges are plausible.

During training, for each sensor i we represent the set of candidate relations C_i with the prior information.

$$C_i \subseteq \{1, 2, \ldots, N\} \setminus \{i\} \tag{3.18}$$

If there is not a prior information, the candidate relations of sensor i is simply all sensors, other than itself.

For a given node i in order to select its candidate relations, we compute the normalized dot product between node i's embedding vector, and the embeddings of its candidates $j \in C_i$:

$$e_{ji} = \frac{v_i^T \, v_j}{||v_i|| \cdot ||v_j||} \text{ for } j \in C_i \tag{3.19}$$

$$A_{ji} = \mathbb{1}\left\{j \in \text{TopK}(\{e_{ki} \, : \, k \in C_i\})\right\} \tag{3.20}$$

Then, we select the top k similarities, where k is a hyperparameter.

Mean Proposal A disadvantage of this method is the fact that the number of connections per node k is a predefined parameter. In particular, it can happen that sometimes a specific sensor requires more connections (e.g., fundamental sensors in autonomous driving computation) whereas others instead are less connected (e.g., peripheral sensors in autonomous driving). This algorithm fixes the number of

connections of each node without considering the nature and the importance of the sensors.

Therefore, I would propose another approach in order to change how the edges are created. First of all, we replace the $TopK$ function from 3.20 with the mean operation.

$$A_{ji} = \mathbb{1}\{j \in \{e_{ji} \geq mean(e_{ki} : k \in C_i)\}\} \tag{3.21}$$

In this way, for each node, we first compute the mean among all the candidates' similarities and then we select only those with a similarity greater than the calculated mean. This brings two advantages, (1) it facilitates the system to be more dynamic without constraints during the learning phase and (2) it removes one hyperparameter.

3.5.4 Graph Attention-Based Forecasting

In this components, we implement a forecasting-based approach, where we forecast the expected value for each sensor based on the past values. This mechanism allows the user to easily identify the sensors which deviate from their expected behaviour. Moreover, it can be useful to compare the expected value of each sensor and the observed one to understand why the model has recognized a sensor as anomalous.

In order to capture the relationships between sensors, we introduce a graph attention-based feature extractor to mix the node's information with its neighbors based on the learned structure as follows:

$$z_i^{(t)} = \mathrm{ReLU}\Big(\alpha_{i,i} W x_i^{(t)} + \sum_{j \in \mathcal{N}(i)} \alpha_{i,i} W x_j^{(t)}\Big) \tag{3.22}$$

where $x_i^{(t)} \in \mathbb{R}^w$ is node i's input feature, $\mathcal{N}(i) = \{j \mid A_{ji} > 0\}$ is the set of neighbors of that node obtained from the learned adjacency matrix A, $W \in \mathbb{R}^{d \times \omega}$ is a trainable weight matrix which applies a shared linear transformation to every node, and the attention coefficients $\alpha_{i,j}$ are computed as:

$$\begin{aligned} g_i^{(t)} &= v_i \oplus W x_i^{(t)} \\ \pi(i,j) &= \mathrm{LeakyReLU}\Big(a^T \Big(g_i^{(t)} \oplus g_j^{(t)}\Big)\Big) \\ \alpha_{i,j} &= \frac{exp(\pi(i,j))}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} exp(\pi(i,k))} \end{aligned} \tag{3.23}$$

where $\oplus$ denotes concatenation. Therefore, $g_i^{(t)}$ is the concatenation between the sensor embedding v_i and the corresponding transformed feature $W x_i^{(t)}$ and a is a vector of learned coefficients for the attention mechanism. After the feature extractor, we have the representations for all N nodes $\{z_1^{(t)}, \ldots, z_N^{(t)}\}$.

For each $z_i^{(t)}$, we element-wise multiply it with the corresponding embedding v_i and use the results across all nodes as the input of stacked fully-connected layers with output dimensionality N, to predict the vector of sensors values at time step t:

$$\hat{s}^{(t)} = f_\theta\Big(\Big[v_1 \circ z_1^{(t)}, \ldots, v_N \circ z_N^{(t)}\Big]\Big) \tag{3.24}$$

where $\hat{s}^{(t)}$ is the model predicted output. During the training, the model tries to minimize the following loss function:

$$L_{MSE} = \frac{1}{T_{train} - \omega} \sum_{t=\omega+1}^{T_{train}} \left\| \hat{s}^{(t)} - s^{(t)} \right\|_2^2 \tag{3.25}$$

which is the Mean Squared Error between the predicted output $\hat{s}^{(t)}$ and the observed data $\hat{s}^{(t)}$.

3.5.5 Graph Deviation Scoring

Given the learned data structure and the forecastings, we want to detect and explain anomalies. To do this, the model computes individual anomalousness scores for each sensor and also combines them into a single score for each time step. In this way, we know when anomalies occur and which sensor is affected.

The error value at time t for the sensor i can be computed:

$$\mathrm{Err}_i(t) = |s_i^{(t)} - \hat{s}_i^{(t)}| \tag{3.26}$$

As different sensors can have very different characteristics, their deviation values may also have very different scales. To prevent the deviations of one sensor being dominant over the other sensors, we perform a robust normalization of the error of each sensor:

$$a_i(t) = \frac{\mathrm{Err}_i(t) - \tilde{\mu}_i}{\tilde{\sigma}} \tag{3.27}$$

where $\tilde{\mu}_i$ and $\tilde{\sigma}_i$ are the median and inter-quantile range (IQR) across time ticks of the $\mathrm{Err}_i(t)$ values. The reason why we use the median and IQR instead of mean and standard deviation is that they are more robust against anomalies. Then, to compute the overall score at time tick t, we aggregate over sensors using the max function:

$$A(t) = \max_i a_i(t) \tag{3.28}$$

Moreover, often some values are not perfectly predicted and result in sharp spikes in error values even when this behaviour is normal, thus we use a simple moving average of length three to generate the smoothed scores $A_s(t)$.

Finally, a time tick t is labelled as an anomaly if $A_s(t)$ exceeds a fixed threshold, equals to the max of $A_s(t)$ over the validation data.

Chapter 4

Experiments

4.1 Introduction

In this chapter, we are going to discuss the most used preprocessing techniques and all the metrics that we use in the experimental section. After these theory definitions, we implement the algorithms presented in the previous chapter describing the obtained results.

4.2 Preprocessing

Data scaling is a recommended pre-processing step when working with many machine learning algorithms. Machine learning models learn a mapping from input variables to an output variable. The scale and distribution of the data drawn from the domain may be different for each variable. Input variables may have different units (e.g. feet, kilometres, and hours) that, in turn, may mean the variables have different scales.

Differences in the scales across input variables may increase the difficulty of the problem being modelled. An example of this is that large input values (e.g. a spread of hundreds or thousands of units) can result in a model that learns large weight values. A model with large weight values is often unstable, meaning that it may suffer from poor performance during learning and sensitivity to input values resulting in higher generalization error. Moreover, if a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

4.2.1 Data Normalization

A common transformation in machine learning is the *normalization* which is a rescaling of the data from the original range so that all values are within the new range $[min, max]$.

$$z = \frac{X - X_{min}}{X_{max} - X_{min}} * (max - min) + min \tag{4.1}$$

In general, the new range is $[0, 1]$ and the good practice usage with normalization and other scaling techniques is as follows:

1. Fit the scaler using available training data. For normalization, this means the training data will be used to estimate the minimum and maximum observable values.

2. Apply the scale to training data. This means we can use the normalized data to train our model.

3. Apply the scale to testing data. This means we can prepare new data in the future on which we want to make predictions.

In general, in the case of outliers with abnormal values (too high or too low), this transformation is not used because it is sensitive to outliers.

4.2.2 Standardization

Another widely used transformation is the *standardization* in which we scale the data in order to have zero mean and unit variance. This can be thought of as subtracting the mean value or centring the data as shown in 4.2.

$$z = \frac{x - \mu}{\sigma} \tag{4.2}$$

Like normalization, standardization can be useful, and even required in some machine learning algorithms when our data has input values with differing scales.

Standardization assumes that our observations fit a Gaussian distribution (bell curve) with a well-behaved mean and standard deviation. We can still standardize our data if this expectation is not met, but we may not get reliable results.

Furthermore, standardization maintains useful information about outliers and makes the algorithm less sensitive to them in contrast to min-max scaling, which scales the data to a limited range of values. Many machine learning classifiers, such as SVN, require standardization in order to achieve better results. On the other hand, classifiers based on distance metric prefer normalization to reduce the distances.

As for the normalization, also in this case we calculate the mean and standard deviation only with training data. Then, we apply the transformation both to the training and testing set.

4.2.3 Quantile transformation

Raw data input variables may have a highly skewed or non-standard distribution. This could be caused by outliers in the data, multi-modal distributions, highly exponential distributions, and more.

Many machine learning algorithms prefer or perform better when numerical input variables and even output variables in the case of regression have a standard probability distribution, such as a Gaussian (normal).

The quantile transform provides an automatic way to transform a numeric input variable to have a different data distribution, which in turn, can be used as input to a predictive model.

This method transforms the features to follow a normal distribution. Therefore, for a given feature, this transformation tends to spread out the most frequent values. It also reduces the impact of (marginal) outliers: this is therefore a robust preprocessing scheme.

The transformation is applied to each feature independently. First, an estimate of the cumulative distribution function of a feature is used to map the original values to a uniform distribution. The obtained values are then mapped to the desired output distribution using the associated quantile function. Features values of new/unseen data that fall below or above the fitted range will be mapped to the bounds of the output distribution. This transform is non-linear and thus it may distort linear correlations between variables measured at the same scale but renders variables measured at different scales more directly comparable.

From the statistics, we know that a quantile function is the inverse of the cumulative probability distribution (CDF). A CDF is a function that returns the probability of a value at or below a given value.

4.3 Evaluation Metrics

For definition, anomalies are rare events and this leads to an imbalanced dataset where we have few abnormal instances and thus we need to choose the right evaluation metric to take into account this property.

We consider the following metrics to evaluate a model:

- the model performance metrics: precision, recall and f1 score;
- the prediction confusion matrix;

- the ROC AUC Score;
- the PR AUC Score;

Let's start with some definitions:

- **TP**: samples for which the prediction is positive and the true label is positive.
- **FP**: samples for which the prediction is positive but the true label is negative.
- **TN**: samples for which the prediction is negative and the true label is negative.
- **FN**: samples for which the prediction is negative but the true label is positive.

The importance of these terms depends on the situation we are facing. For example in the case of medicine, a false-positive patient, as well a false negative one, means a wrong medicine therapy. Therefore, we wanted to analyze results using the confusion matrix to know where and when our model fails.

The most common metrics used in machine learning are:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.4}$$

$$F_1\, score = \frac{2 \times precision \times recall}{precision + recall} \tag{4.5}$$

$$False\, Positive\, Rate = \frac{FP}{FP + TN} \tag{4.6}$$

In this case, the accuracy is not taken into account because since the dataset is highly imbalanced this metric does not help.

4.3.1 ROC curve

A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability and corresponds to the area under the curve. It tells how much the model is capable of distinguishing between classes.

Higher the AUC, the better the model is at predicting 0 classes like 0 and 1 classes as 1. By analogy, the higher the AUC, the better the model is at distinguishing between normal samples and anomalies ones.

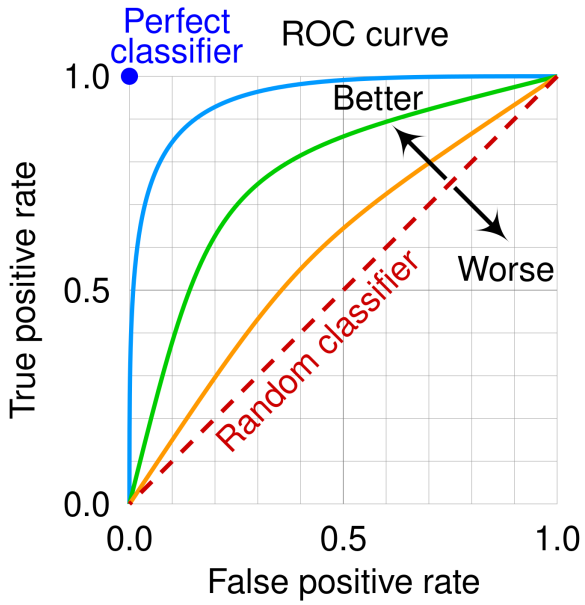

Figure 4.1: ROC curve.

As shown in figure 4.1, an excellent model has AUC near to the 1 which means it has a good measure of separability. When AUC is 0.5, it means the model has no class separation capacity.

The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.

When we have a very imbalanced dataset, we should pay attention to this score because the false positive rate for highly imbalanced datasets is pulled down due to a large number of true negatives. We should use it when we care equally about positive and negative classes. ROC AUC can be optimistic on severely imbalanced classification problems with few samples of the minority class.

4.3.2 PR curve

A PR curve is simply a graph with precision values on the y-axis and recall values on the x-axis. In other words, the PR curve contains TP/(TP+FN) on the y-axis and TP/(TP+FP) on the x-axis.

As shown in figure 4.2, it is desired that the algorithm should have both high precision, and high recall. However, most machine learning algorithms often involve a trade-off between the two measures. A good PR curve has a greater AUC (area under the curve). In the figure 4.2, the classifier corresponding to the blue line has better performance than the classifier corresponding to the green line.

The intuition is the following: since PR AUC focuses mainly on the positive

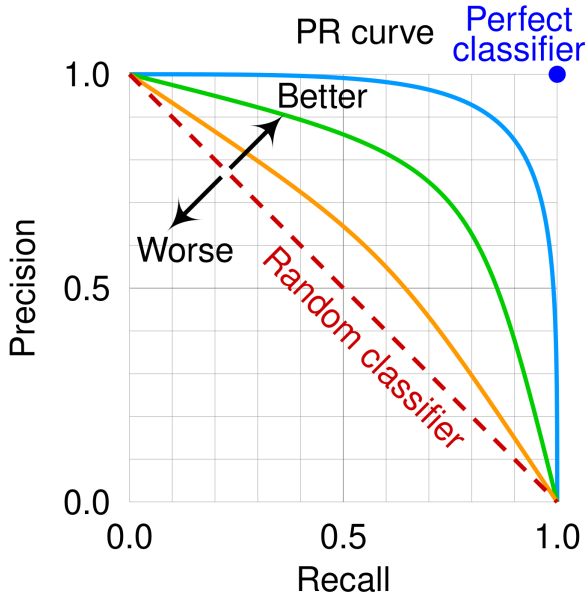

Figure 4.2: Caption

class (precision and recall) it cares less about the frequent negative class. We can say that we use PR-Curve when we want to focus more on the positive class.

4.4 Datasets and experiments

After the data analysis and methods explanation, the next step is to test the algorithms with the described datasets. Since each dataset has its characteristics, it is interesting to study the performance of each proposed method in different situations to present its strengths and weaknesses.

For example, the SWaT has point and group anomalies with a limited duration, whereas the dataset of the strain gauge of a suspension arm has a break-out point. This means that the system works until a certain timestep and then it breaks and from that point, all the samples are anomalies.

In literature, there are mainly two approaches based on reconstruction or forecasting. Due to this difference of the datasets, both *reconstruction-based models* and *forecasting-based models* are implemented.

Forecasting-based models predict the actual value of the next timestamp in a deterministic manner, which is sensitive to the randomness of the time series. On the other hand, reconstruction models alleviate this problem by learning a distribution of stochastic variables, which is more robust to perturbations and noises.

As in any task that involves time series, the input data is split into time windows, where their length is a hyperparameter. Instead, the stride is set to 1 because only *online models* are considered.

For a better comparison, all the experiments are made with the same learning rate ($1 \cdot 10^{-3}$), batch size (64), Adam as optimizer and a dropout layer with a rate of 0.2.

Moreover, we have added two callbacks to reduce the learning rate with a rate 0.1 if there is no improvement in the validation loss for five epochs and stop the training if for ten epochs the validation loss does not decrease.

Furthermore, when discussing several algorithms, it is useful to compare a *baseline* model, defined as the simplest method in the literature, with the actual state of the art algorithm. In this way, the results obtained from the experiments are comparable and considerations about the performance are possible.

4.4.1 SWaT

SWaT is a fully operational scaled-down water treatment plant with a small footprint, producing 5 gallons/minute of doubly filtered water. This testbed replicates large modern plants for water treatment such as those found in cities. Its main purpose is to enable experimentally validated research in the design of secure and safe CPSs. SWaT has six main processes corresponding to the physical and control components of the water treatment facility as shown in Figure 4.3.

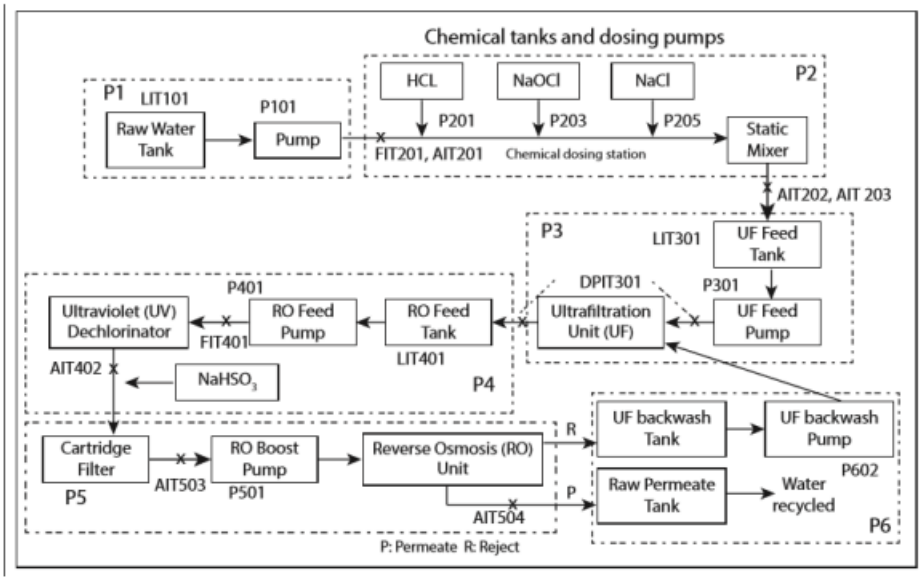

Figure 4.3: Scheme of the CPS.

Attack scenarios SWaT consists of six stages where each stage contains a different number of sensors and actuators. Based on attack points in each stage, the attacks are divided into four types.

1. *Single Stage Single Point (SSSP)*: A single-stage single point attack focuses on exactly one point in a CPS.

2. *Single Stage Multi-Point (SSMP)*: A single-stage multiple point attack focuses on two or more attack points in a CPS but on only one stage.

3. *Multi Stage Single Point (MSSP)*: A multi-stage single point attack is similar to an SSMP attack except that now the SSMP attack is performed on multiple stages.

4. *Multi Stage Multi-Point (MSMP)*: A multi-stage multi-point attack is an SSMP attack performed in two or more stages of the CPS.

A total of 36 attacks were launched during the data collection process. The duration of the attack is varied based on the attack type. A few attacks, each lasting ten minutes, are performed consecutively with a gap of 10 minutes between successive attacks. Some of the attacks are performed by letting the system stabilize before a subsequent attack. Some of the attacks have a stronger effect on the dynamics of the system and cause more time for the system to stabilize. Simpler attacks, such as those that affect flow rates, require less time to stabilize. In general, attacks lasted between a few minutes to an hour.

Attack category	Number of attacks
SSSP	26
SSMP	4
MSSP	2
MSMP	4

Table 4.1: Number of attacks per category.

Data collection process The data collection process lasted for a total of 11 days. SWaT was functioning non-stop 24 hours/day, during the entire 11-day period. SWaT was run without any attacks during the first seven of the 11-days. Attacks were launched during the remaining four days.

The data collection process starts from an empty state of SWaT. This initialization was deemed necessary to ensure that all the tanks are filled with unfiltered water and not pre-treated.

All the data was logged continuously once every second. As the data collection process started from an empty state, it took about 5 hours for SWaT to stabilize and the samples during this time are removed.

To speed up the training process, the original data samples are down-sampled to one measurement every 10 seconds by taking the median values. The resulting label is the most common label during the 10 seconds.

In table 4.2 are summarized the size of both training and testing sets. Moreover, from the statistics, we notice that we have enough data and anomalies in the test set (about 1/10).

Number of dimensions	Training set size	Testing set size	Anomaly ratio (%)
55	47.520	42.833	11.71%

Table 4.2: Statistics of the dataset used in experiments.

Data analysis and challenges In general, the datasets of CPSs are more complicated than the others because they represent a system in continuous evolution. The correlations and the distributions of the features are mutable because the system degrades over time.

As described in Section 3.2, some features are constants in the training set but not in the testing one.

From the data analysis, we have 37 non-constants features in the training set and their distribution can be observed in Figure 4.4.

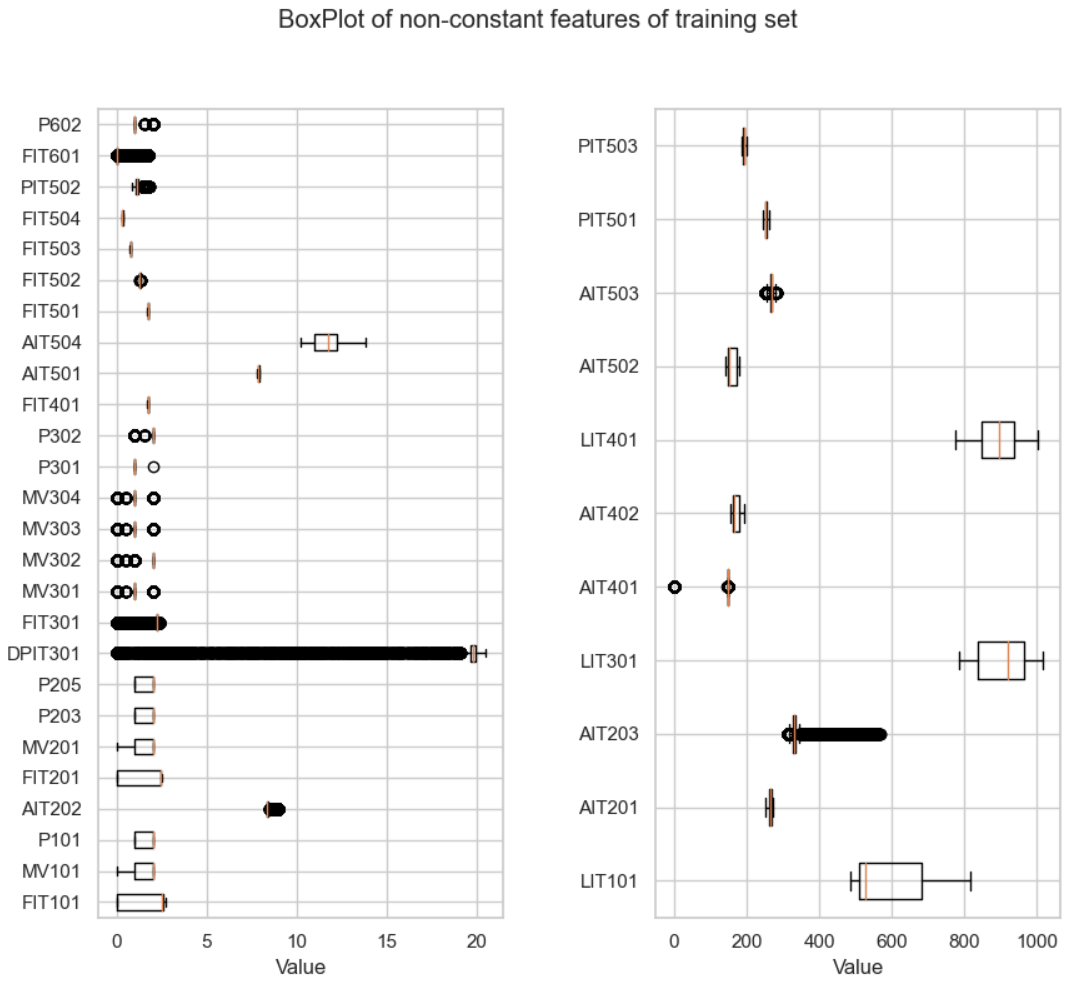

Figure 4.4: Boxplot of the features.

Detrending In many cases, it is advisable to train a model with a stationary time series but most of the time the input data is not. For this reason, one possible choice is to consider as input $x_t - x_{t-1}$. In this way, the distribution now is more similar to the Gaussian and the time series is stationary.

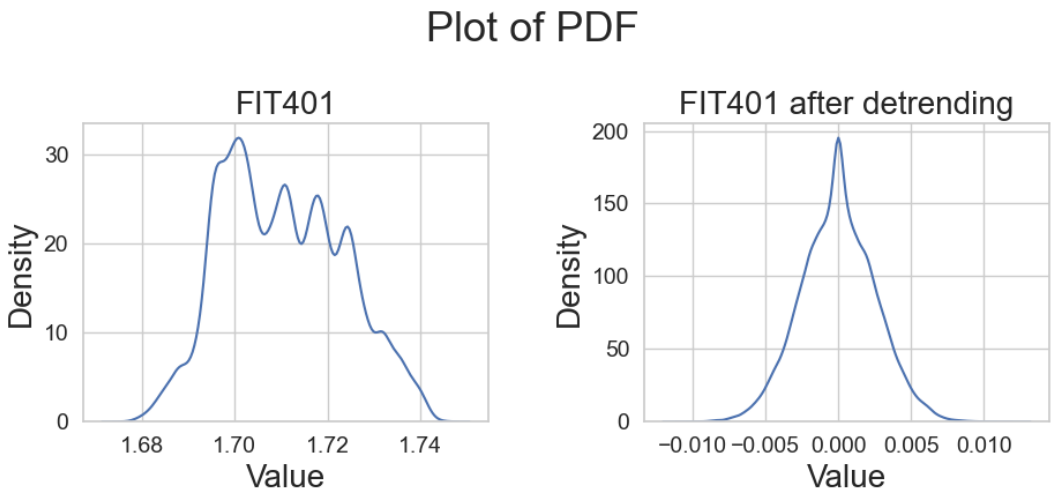

Figure 4.5: Effect of the detrending operation.

As shown in Figure 4.5, we can notice how the feature "FIT401" has a distribution far from the normal one before the transformation and then after the detrending this distance is minimized.

Baseline

Based on the original paper [11], we choose a *forecasting-based model* in which we predict the values of the next timestep. Therefore, we have to change the architecture as shown in Figure 4.6.

Figure 4.6: Scheme of the forecasting model with time windows of size T and N features.

The neural network is composed of two LSTM layers followed by a Dense layer with the output dimension equal to the number of features we want to predict. In our simple implementation, our model learns to predict only one timestep at a

time.

From the explanation paper, we know that all the anomalies affect a number of sensors between 1 and 4 therefore we define a threshold for each feature and in case of an error higher than the threshold we consider that sample as an anomaly.

As we consider the error between our predictions and the true values as the threshold, the size of the time window may have an impact on the model performance. To study this impact, we train the same model with two different sizes of time windows. From the theoretical point of view, with a larger time windows, the model can learn more long temporal patterns.

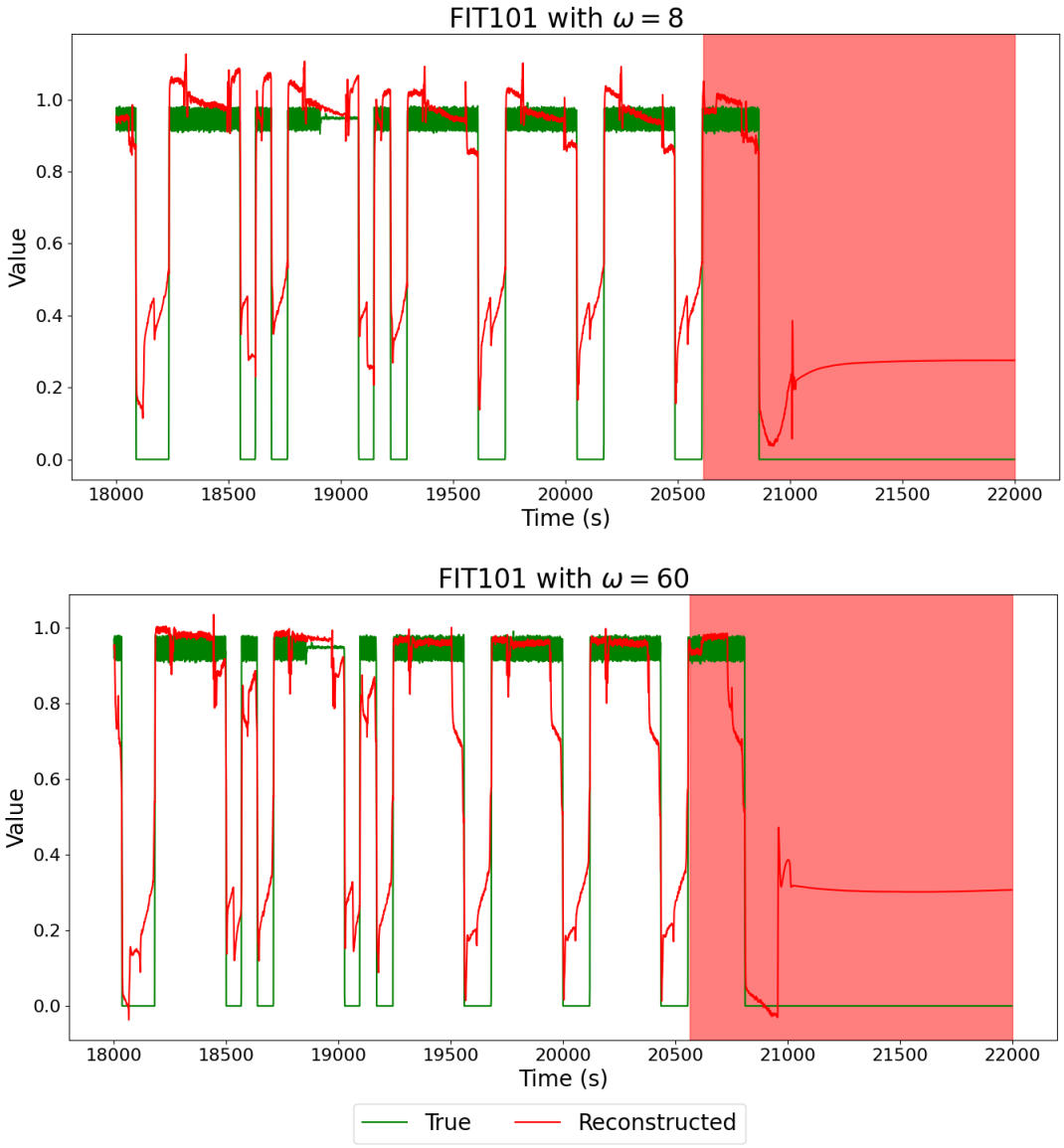

Figure 4.7: Comparison of the reconstruction with different values of w.

As shown in Figure 4.7, the same sequence is reconstructed very well except for the correctly wrong anomaly one. In general, the performance grows with the size of the time window until a certain value. Models with large windows require more time to train and thus in general we stop increasing this parameter until we find reasonable scores.

We can plot the confusion matrix in order to show how the model distribute the predictions.

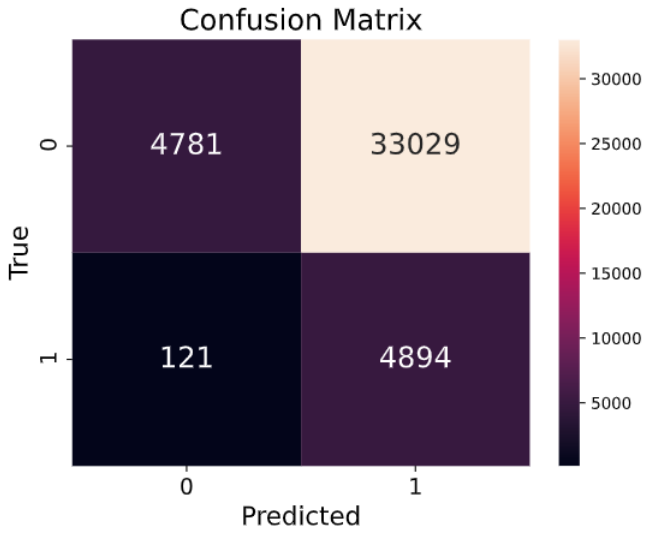

Figure 4.8: Confusion matrix with $w = 8$.

In Figure 4.8, the confusion matrix shows that the number of *false positives* is very high and thus we expect a low precision. The reason is that some features are always bad reconstructed and consequently the samples will be considered always anomalies.

In Table 4.3, we made a comparison of scores with different values of w and, as supposed theoretically, the performance is higher with a larger window.

w	F1 score	Precision	Recall	ROC AUC	PR AUC
8	24.40%	13.98%	96.47%	56.59%	13.42%
60	**25.20%**	**14.78%**	**97.16%**	**58.96%**	**13.89%**

Table 4.3: Mean of scores after 5 runs.

In particular, the PR AUC is very low in both models due to the high number of *false positives*.

Deep Evidential Regression

As described in the previous section, the Deep Evidential Regression is suitable for univariate regression, where the model has to predict only one value.

However, in Section 2, we have discussed that it is impossible to study multivariate time series in a univariate way. Therefore, we train one network for each feature as we did with the MSL dataset and for each feature we store the estimated

entropy. It is important to highlight that as input we pass the entire number of features whereas the output consists of only one. In this way, the model can better capture the dependency among the features.

Then, after all the training, we consider the maximum entropy between the features for each sample and we select the threshold to maximize the ROC score.

We replace the Dense layer of the previous architecture with a new *DenseNormalGamma* layer provided by the author of the original paper [11]. In this way, the new Dense layer has four times the number of original output because for each output feature it provides the estimation of the four variables $\mu, \gamma, \alpha, \beta$.

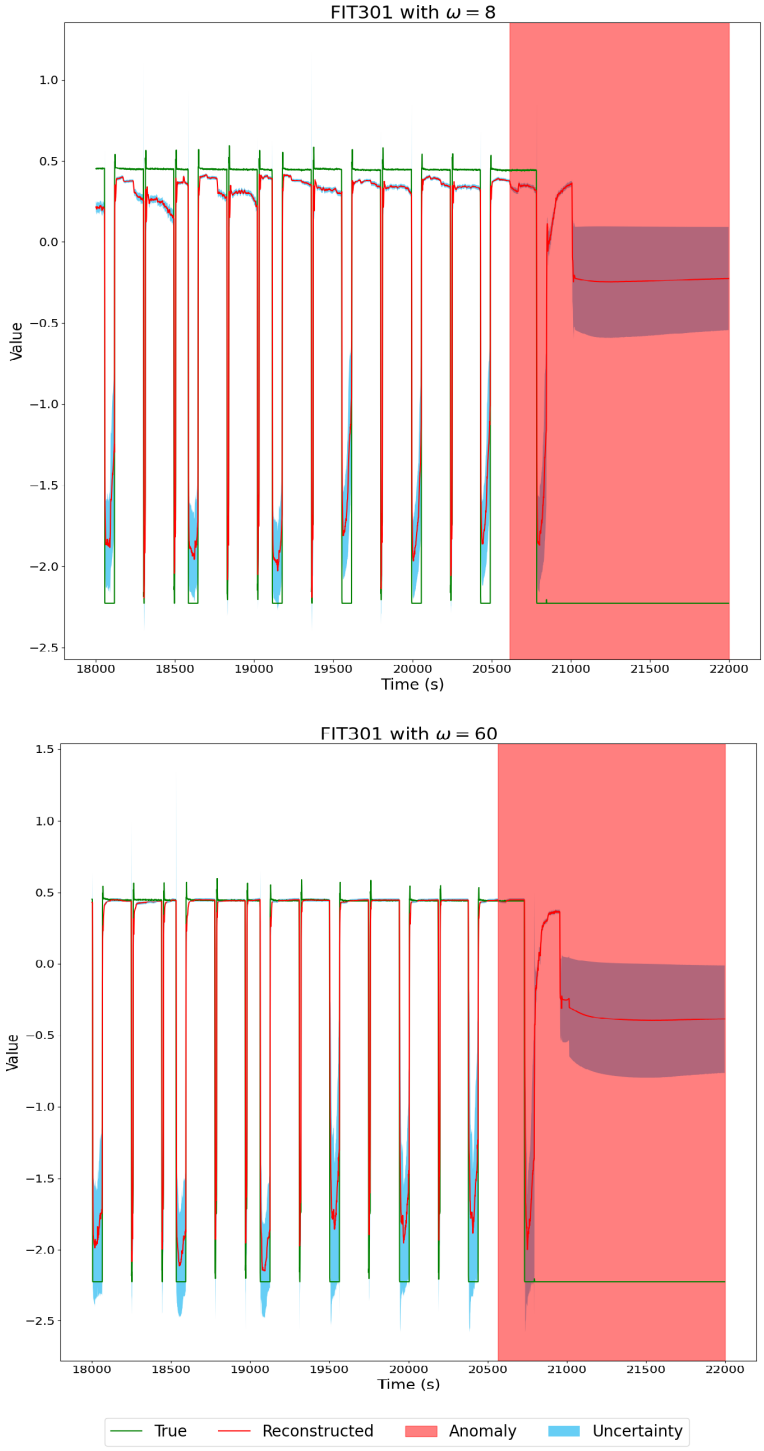

Figure 4.9: Comparison of the same portion of data with different w.

In Figure 4.9, we can notice how both the reconstructions are similar but the one with a larger size of the window has smaller errors and its uncertainty is more coherent with the true values.

Inside the anomaly region, both models correctly miss estimating the uncertainty because it does not include the true values. The reason is that since the anomaly is something "unexpected" the models are not able to estimate the uncertainty and thus that region is predicted as an anomaly.

Detrending As described in the previous section, a possible method to make the distribution of the features closer to the normal one is the *detrending* operation. Doing so, since our method is based on the assumption that the predictable variable is normally distributed, we expect high results.

F1 score	Precision	Recall	ROC AUC	PR AUC
19.74%	11.92%	57.33%	50.57%	11.83%

Table 4.4: Mean of scores after 5 runs.

Unfortunately, as shown in Table 4.4, the performance are very bad because with the detrending operation all the features are stationary and thus harder to predict since they do not depend on time anymore.

A possible solution - Quantile Transformation The goal is to transform our data without change the stationary of the time series. One of the possible preprocessing operation available on scikit-learn is the *Quantile Transformation*.

w	F1 score	Precision	Recall	ROC AUC	PR AUC
8	54.43%	43.56%	72.84%	**80.83**%	35.98%
8 (with *QuantileTrans.*)	58.82%	50.44%	70.53%	80.67%	39.03%
60	**60.61**%	**60.05**%	**77.05**%	80.77%	**42.88**%
60 (with *QuantileTrans.*)	53.92%	42.77%	73.13%	80.03%	34.41%

Table 4.5: Mean of scores after 5 runs.

As shown in Table 4.5, the model with a larger time window performs better. We can notice how with the *Quantile Transformation* the performance is increased with $w = 8$. The behaviour has changed with a larger time window where the best model is obtained with *Standard Scaler* as suggested by the authors of the paper.

Overall, this method goes beyond the baseline even though it has been extended to the multivariate case.

Graph Neural Network

In the previous experiments with other methods, we removed the 13 constants features because they do not bring any information. Instead, in the paper, the authors trained the model with the entire set of sensors and actuators thus we first try this setting to reproduce their result and then set a sort of baseline for further experiments.

As already discussed, with this *forecasting-based* algorithm, we have to tune the following hyperparameters:

- k: denotes the number of connections (edges) for each node. The value of k can be chosen according to the desired sparsity level. A higher value means that the graph is very dense in which all nodes are linked to each other. In case of low value, the number of edges decreases and this is the way it is more sparse.
- *embeddings dimension*: dimension of the embedding space.
- *time window length (w)*: size of the time window.

First of all, we reproduce the scores obtained with the suggested parameters from the authors of the paper with all the features as shown in Table 4.6.

k	Embeddings dimension	out layers inter. dim	size of time window
15	64	128	5

Table 4.6: Hyperparameters suggested by the paper.

With these parameters, we were able to reproduce the results from the paper and we obtained the confusion matrix shown in Figure 4.10 in which we notice that the model is very accurate. It is able to identify correctly almost all normal samples and a high number of anomalies maintaining the number of miss-classified samples low.

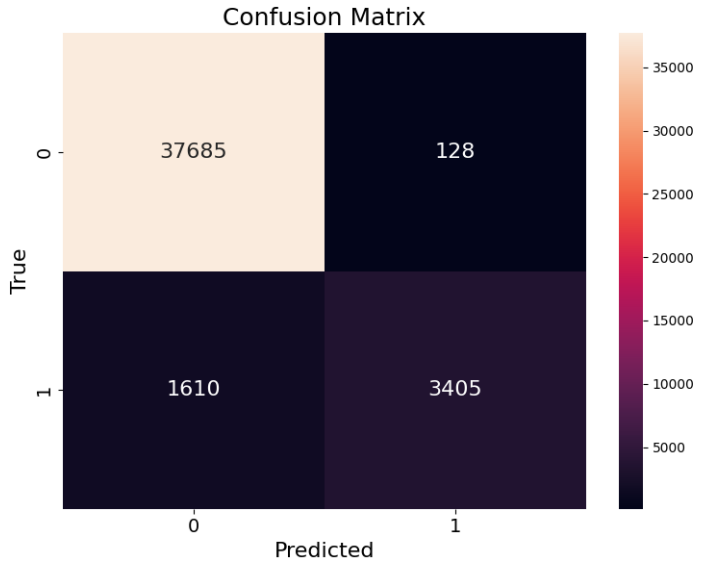

Figure 4.10: Confusion matrix with the suggested parameters.

Since some features are constants in the training set but not in the testing one, the model fails to reconstruct them. In Figure 4.11, we plot the comparison between the reconstructed sequences and the real ones of the two features constant in the training set.

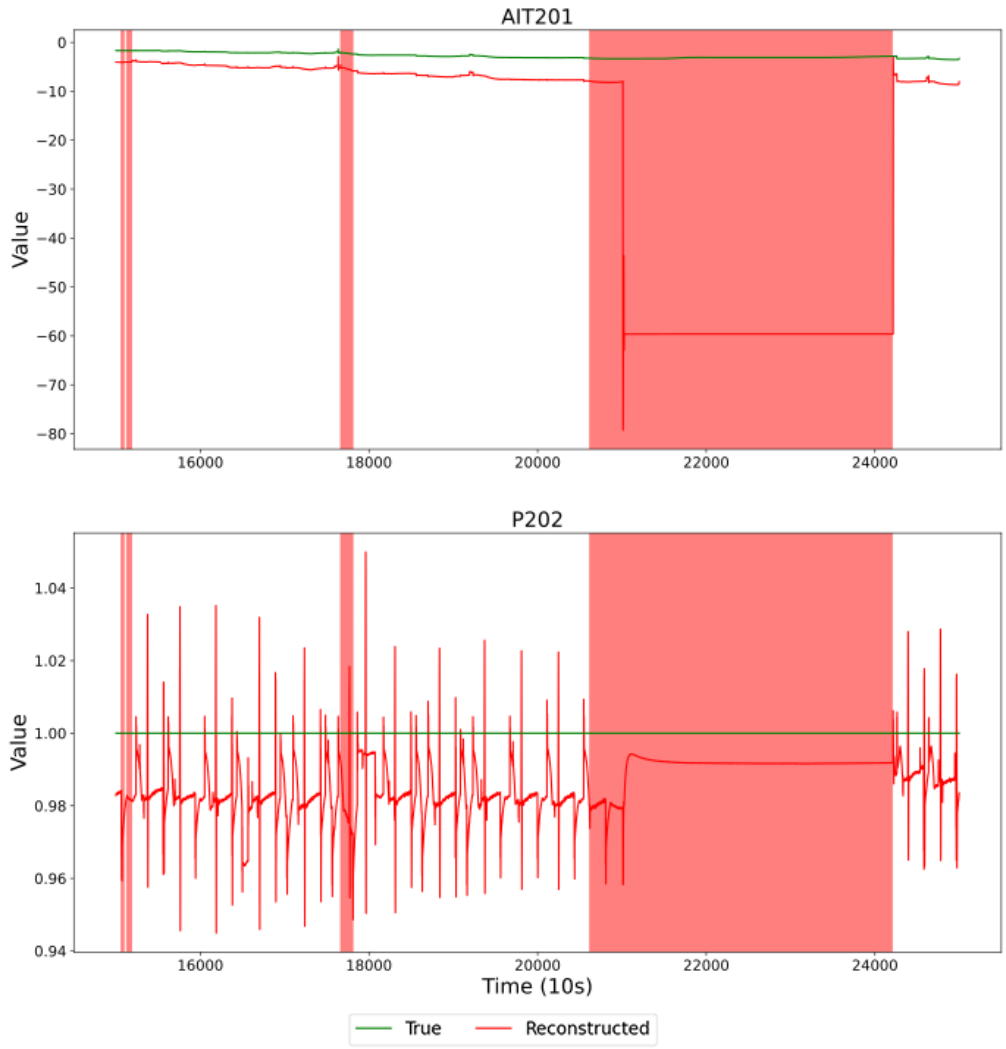

Figure 4.11: Example of reconstruction of two different sensors.

The model makes errors in both features and the reconstruction is very bad because the true signals are almost constant to certain values but the network is not able to capture and learn this behaviour but instead, it tries to approximate them with dynamic behaviours.

For this reason, the next experiment is to remove these features reducing the dimension of the dataset to 37 in order to speed up the training process considering only features that bring information.

With this new setting, from the confusion matrix in Figure 4.12 we can observe how this new model identifies fewer FPs and FPs than the previous one. This proves that the feature selection has improved the training process and the model

seems more flexible.

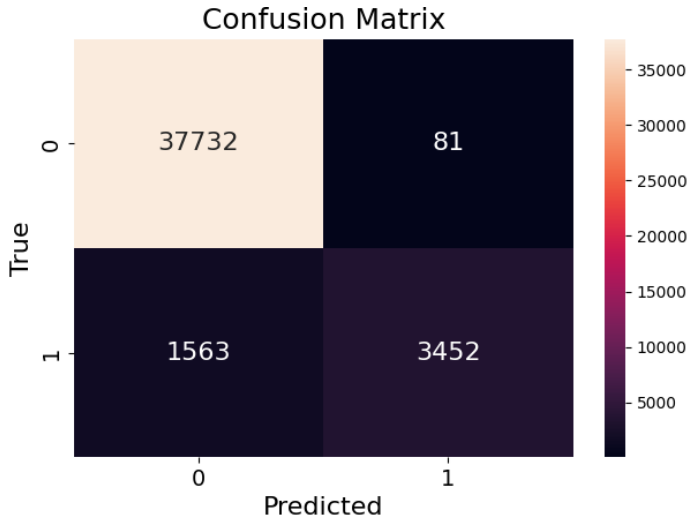

Figure 4.12: Confusion matrix with feature selection.

Moreover, we can compare the reconstruction of a feature with both models. In Figure 4.13, we notice that the feature *MV101* is badly reconstructed by the first model (the one with all features) and the true values (green line) are far from the reconstructed ones. For what concerning this feature, we can say that the model has failed to understand the correct behaviour.

Instead, the model with the reduced number of features has performed well and it approximated correctly the true values and badly the anomaly ones.

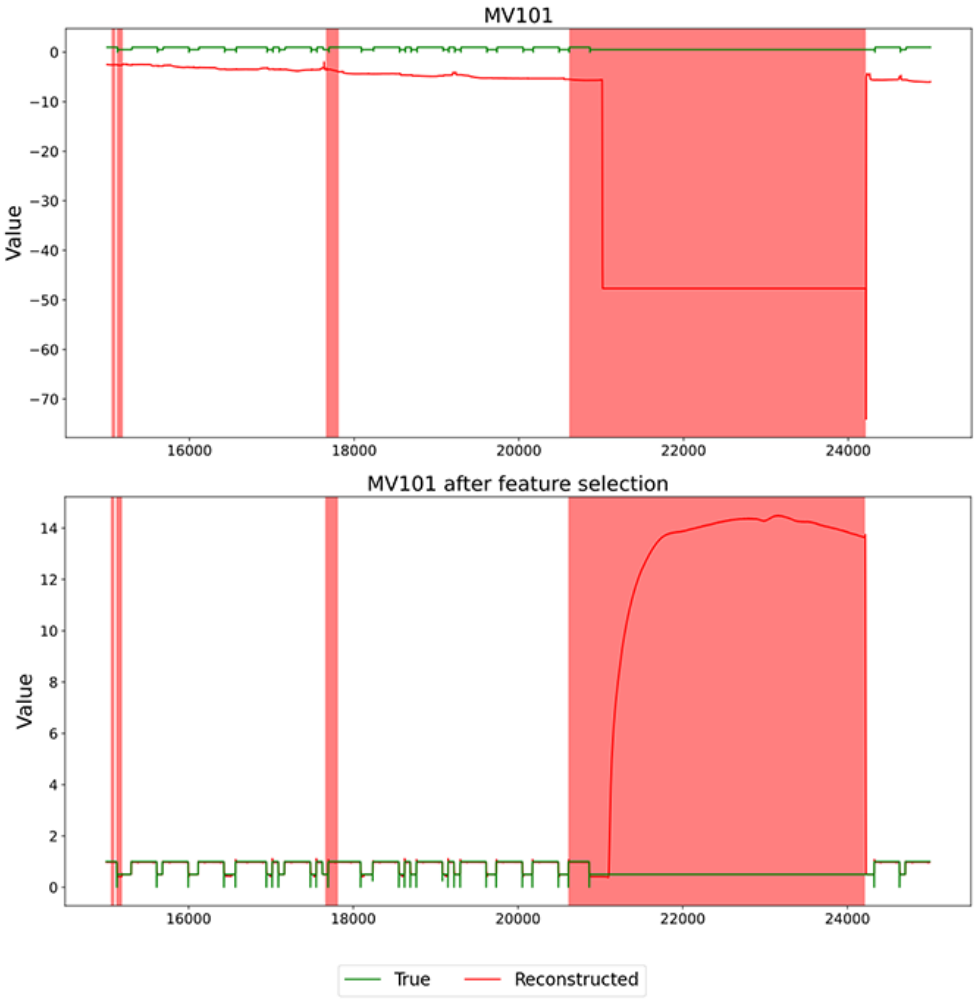

Figure 4.13: Reconstruction of a portion of the same feature (*MV101*) before and after the feature selection.

Then, in Table 4.7 we compare the scores obtained with both models.

Method	F1 score	Precision	Recall	ROC AUC	PR AUC
With all features	79.82%	**97.82**%	67.51%	83.65%	69.83%
Drop constants features	**81.10**%	97.55%	**69.41**%	**84.59**%	**71.28**%

Table 4.7: Scores obtained with the suggested hyperparameters after 5 runs before and after the feature selection.

There is a slight increase of the f1 score (+2%) and recall (+2%) whereas the precision remains stable around 97.5% and for this reason, in the next experiments we proceed with the feature selection.

After these experiments, we also analyze the influence of each hyperparameter and its impact on the model's performance. The first hyperparameter that we test is the k.

As shown in Figure 4.14, with both settings the reconstruction of that feature is very accurate. With $k = 10$, the reconstruction data present some high spike values mostly inside the anomaly sequence.

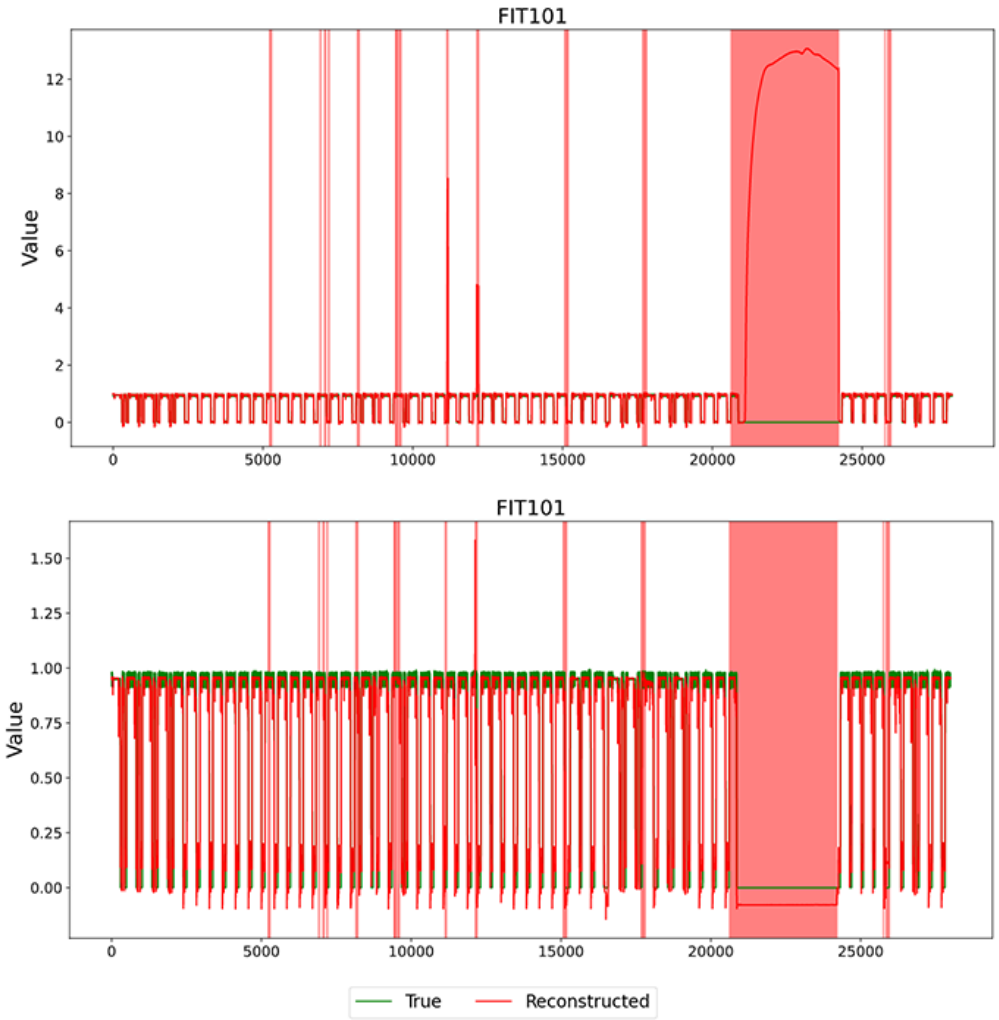

Figure 4.14: Comparison of the reconstruction of the same portion of input data with $k = 10$ (top) and $k = 20$ (bottom).

Increasing the value of k seems to impact only the reconstruction of the anomaly sequence. In fact, with $k = 20$ the reconstruction error of the anomaly sequence is lower than the first case.

In Table 4.8, we notice the scores with different values of k. During my experiments with a static value of this hyperparameter, I found a high deviation of the scores thus we can conclude that by doing more runs the differences of the scores decreases.

Method	F1 score	Precision	Recall	ROC AUC	PR AUC
$k = 10$	81.31%	97.22%	69.87%	84.80%	71.45%
$k = 20$	80.50%	96.29%	69.16%	84.40%	70.19%
Mean proposal	**81.75**%	**97.29**%	**70.16**%	**85.40**%	**71.86**%

Table 4.8: Mean scores with different k after 5 runs.

My proposal performs slightly better than the models with a predefined value of k. With this new setting and the same hyperparameters of previous experiments, the learned graph is composed of 37 nodes (features) and the edges: min=2 and max=32. This proves that each sensor has its behaviour and requires a different number of connections.

The second hyperparameter that we study is the size of the embeddings. The embeddings are necessary to connect nodes to each other and for this reason, the next experiment is dedicated to studying the influence of the variation of the dimension of the embeddings space.

From the literature, it does not exist a correct formula or an always-good value for the size of embedding for each task. Instead, we need to tune this parameter by doing more trials and then studying the performance to select the best value.

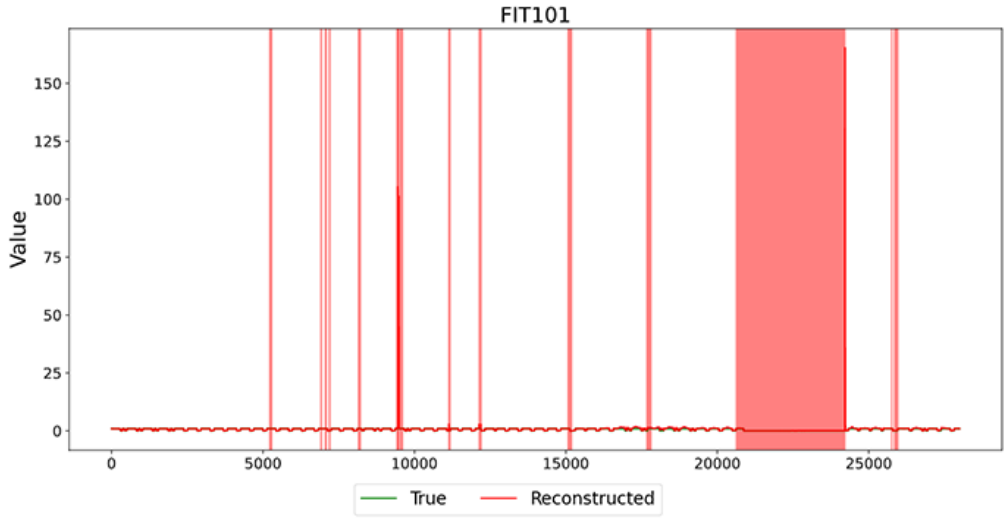

Figure 4.15: Reconstruction of the first feature with $dim = 32$.

In Figure 4.15, it is shown the reconstruction with $dim = 32$. As with the previous hyperparameters, the reconstruction is still good with low errors but some points, inside the anomaly sequences, presents very high peaks in the reconstruction.

Since the reconstruction is almost the same with $dim = 128$, for simplicity, we avoid plotting it and we can conclude by saying that both the proposed embeddings dimensions can be considered as equivalent and valid solutions from the reconstruction and performance points of view as shown in Table 4.9 where the model with a smaller embedding size performs slightly better.

Method	F1 score	Precision	Recall	ROC AUC	PR AUC
$dim = 32$	**80.74**%	97.15%	**69.11**%	**84.42**%	**70.74**%
$dim = 128$	80.62%	**97.55**%	68.71%	84.24%	70.69%

Table 4.9: Mean scores with different size of embeddings after 5 runs.

The last hyperparameter that we can study is the size of the time window. The authors of the original paper decided to choose a small value because the model does not implement temporal learning and to speed up the training process. As described in the introduction of this section, increasing the size of the time window increase the ability of the model to capture long temporal dependency and in some situation, it translates into better performance.

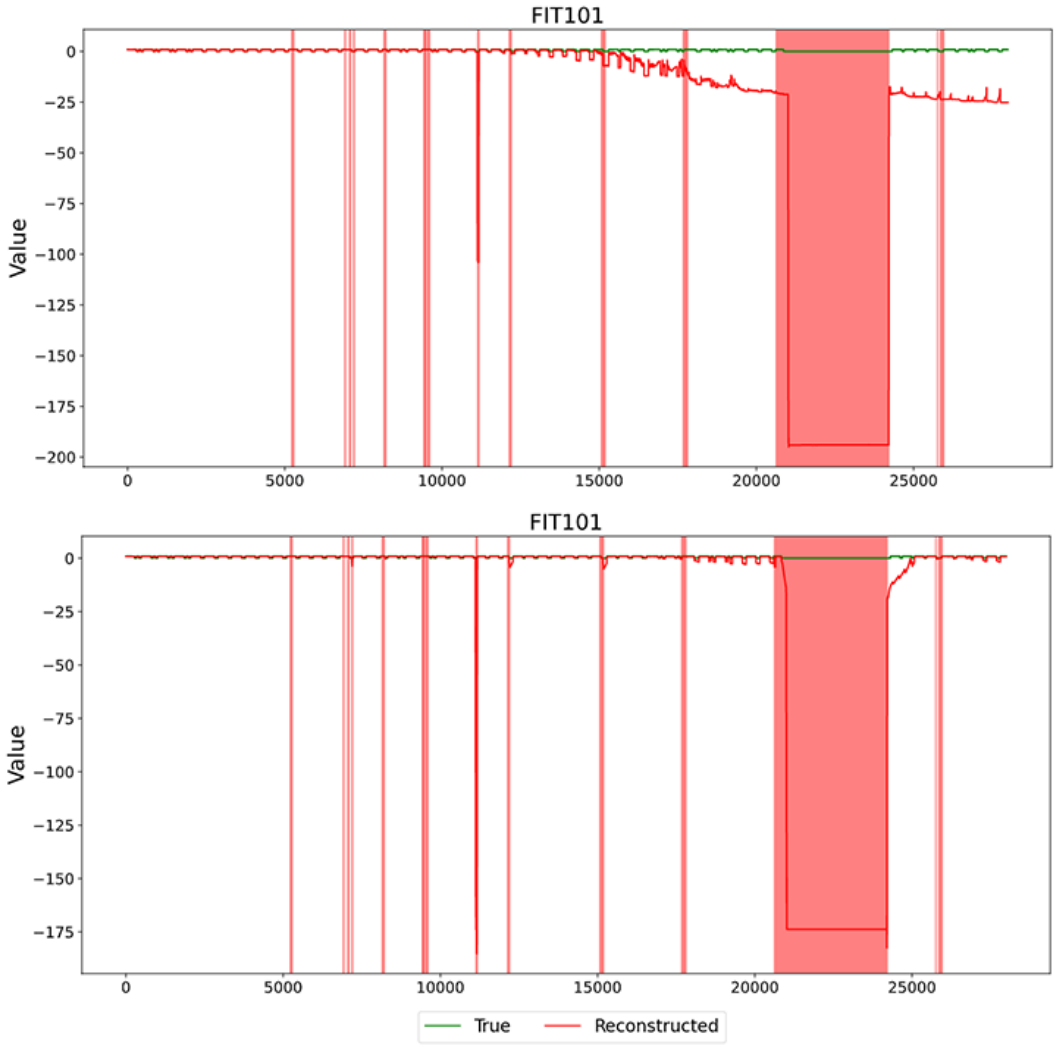

Figure 4.16: Reconstruction of a portion of the input data with $w = 10$(top) and $w = 20$ (bottom).

In Figure 4.16, the reconstruction error of the first model ($w = 10$) is increasing from sample #15000 where the separation between the true signal and the output of the network grows.

Instead, this problem is solved in the second model with a larger time window ($w = 20$) where all the normal sequences from the beginning to the end are reconstructed perfectly.

In Table 4.10, the results obtained are summarized as the average of five runs.

Method	F1 score	Precision	Recall	ROC AUC	PR AUC
$w = 10$	**80.58**%	**97.55**%	**68.70**%	**84.24**%	**70.67**%
$w = 20$	80.10%	96.93%	68.26%	83.99%	69.88%

Table 4.10: Mean scores with different size of time window after 5 runs.

The results show that different sizes of the time window do not improve the scores obtaining almost the same performance on the test set. This means that increasing the size of the input sequences and consequently the training time is not worth it because the performance remains almost the same.

We can conclude by saying that the disadvantage of this method is that it does not include a network to learn the temporal correlations as in MT-GAT [9].

Consequentially, increasing the size of the window does not improve the performance.

Conclusions

In this section, we have discussed the experiments with the SWaT dataset. From the nature of the dataset, we know that there is a *domain shift* and thus the distribution of the features change with time.

Consequently, traditional methods that try to approximate the distribution of the input as the *Deep Evidential Regression* are not suitable in environments where the features are in evolution. However, methods like the *Quantile Transformation* have shown success in increasing the performance.

Method	F1 score	Precision	Recall
Baseline	25.20%	14.78%	97.16%
Deep Evidential Regression	60.61%	60.05%	77.05%
Graph Neural Network	81.75%	97.29%	70.16%
State of the art (DAICS)	88.92%	91.85%	86.16%

Table 4.11: Comparison of the results between the proposed approaches and the state of the art.

In Table 4.11, there is a comparison between the scores obtained with the proposed models and the state of the art algorithm. The advantages of *DAICS* algorithm are:

1. *Dynamic threshold*. This dataset presents a *domain shift* and thus requires a threshold that adapts itself with the distribution of the input data and its changes.

2. *Reconfiguration in case of false positives.* It is possible to reconfigure DAICS in a production environment in the case an anomaly is identified by the system and then recognised as a false alarm by the technician.

My second proposed approach (*Graph Neural Network*) presents some interesting points such as a graph architecture and interpretability of the results via sensor embeddings.

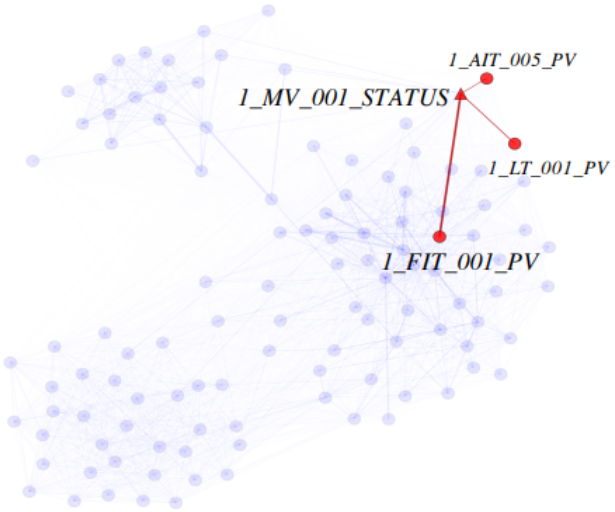

Figure 4.17: Force-directed graph layout with attention weights as edge weights, showing an attack. The red triangle denotes the central sensor identified by our approach, with highest anomaly score. Red circles indicate nodes with edge weights larger than 0.1 to the central node.

In Figure 4.17, the edges in the learned graph provide interpretability by indicating which sensors are related to one another. Moreover, the attention weights further indicate the importance of each of a node's neighbours in modelling the node's behaviour.

4.4.2 Mars Science Laboratory rover (MSL)

Figure 4.18: Image of the curiosity rover.

This dataset is a set of data about the state of the Curiosity Rover sent on Mars in a mission to study if the planet can host life. This subsystem consists of the X-Band subsystem for direct communication with Earth and the UHF subsystem for communications with Mars relay orbiters, including the Mars Reconnaissance Orbiter (MRO), Odyssey (ODY), Maven (MVN), Mars Express (MEX), and the Trace Gas Orbiter (TGO).

From the Telemanom paper [12], we know that this dataset is divided into different channels and thus we need to process each channel in a univariate manner, training and evaluating the model to each channel.

Moreover, all data has been anonymized with regard to time and all telemetry values are pre-scaled between (-1,1) according to the min/max in the test set. Channel IDs are also anonymized, but the first letter gives indicates the type of channel (P = power, R = radiation, etc.).

Model input data also includes one-hot encoded information about commands that were sent or received by specific spacecraft modules in a given time window. The distribution of these values is depicted in Figure 4.19.

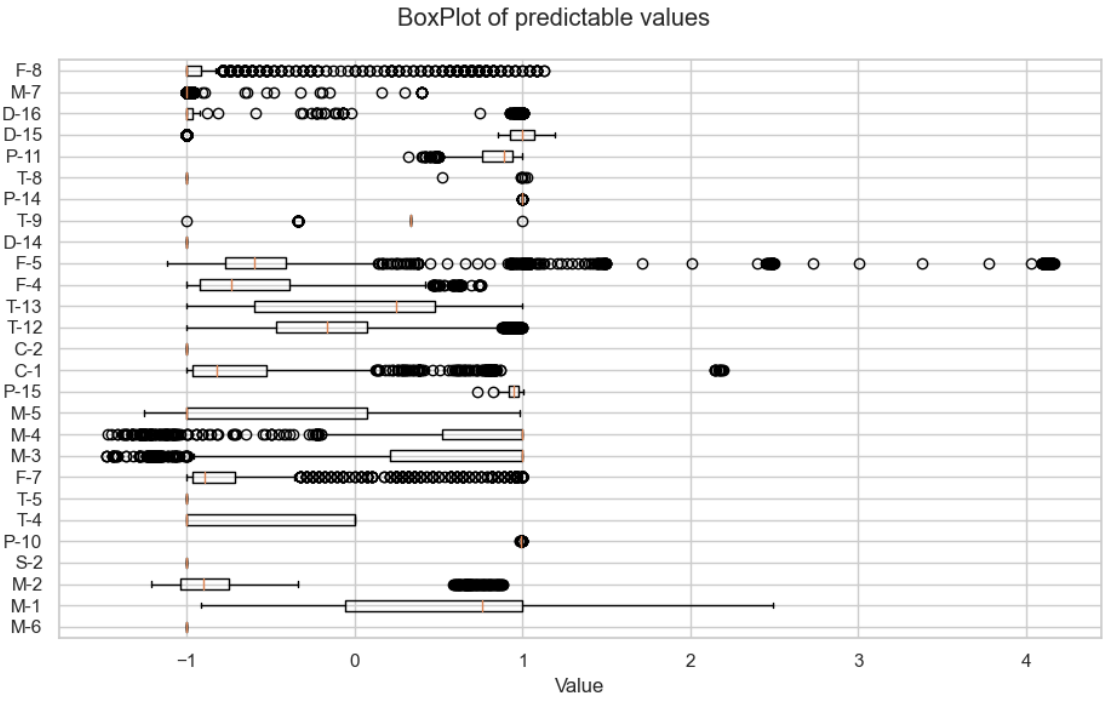

Figure 4.19: Boxplot of the first feature of the training set.

Along with the available data, it is provided with a file with the details of anomalies channel by channel. From this file, we know that two types of anomalies are detected: *point* and *contextual* anomalies.

In table 4.12, we notice that the number of channels is 27 with a total number of 36 anomaly sequences.

Total anomaly sequences	36
Point anomalies (% tot.)	19 (53%)
Contextual anomalies (% tot.)	17 (47%)
Unique telemetry channels	27
Telemetry values evaluated	66,709

Table 4.12: Summary of the dataset.

Labels adjustment Since the anomalies are in sequence, the authors suggest that is enough to recognize at least one point of that anomaly sequence. Only one true positive is recorded even if portions of multiple predicted sequences fall within a labelled sequence. If no predicted sequence overlaps with a positively labelled sequence, a false negative is recorded for the labelled sequence. For all

predicted sequences that do not overlap a labelled anomalous region, a false positive is recorded.

Baseline

With the first missions on Mars, NASA started to release some public datasets about the state of their rovers. One of these is the MSL, studied initially by some employees of the NASA centre in order to recognize and prevent anomalies in the system. They collected the results and the description of the proposed model in the original paper [12].

Furthermore, many advanced experiments are performed with this dataset in the years and the actual at the state of the art is the *Multivariate Time-series Anomaly Detection via Graph Attention Network* [13]. The authors have proposed both *reconstruction* and *forecasting* models and they found out that the model that best fits the characteristics of this dataset is the one based on the reconstruction of the input data.

Their motivation is based on the fact that the dataset is highly unpredictable with stationary features and they demonstrate how the reconstruction-based model outperforms the forecasting one.

Therefore, we proceed with the training channel by channel of the autoencoder, described previously in Figure 4.26, with size of the time window from the original paper ($w = 250$).

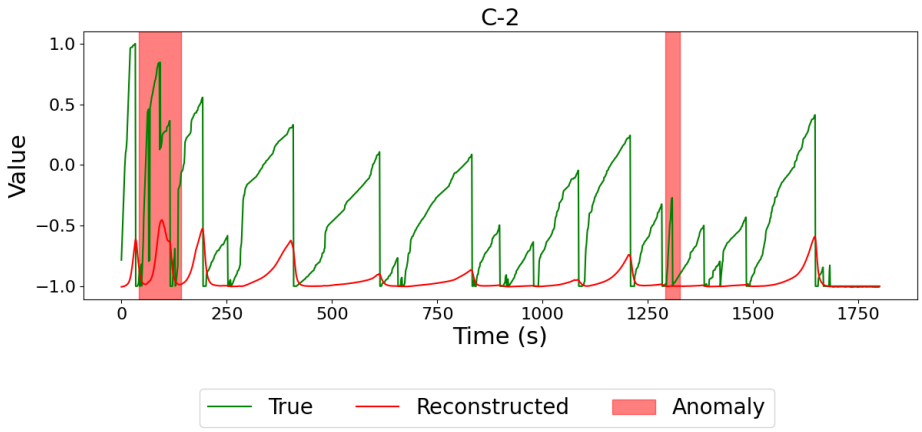

Figure 4.20: Comparison between the real values and the reconstructed ones.

In Figure 4.20, the comparison between the real values and the predicted ones with $w = 250$ of feature *C-2* shows that the network has learnt the general shape of the input data but the error is still high.

For each channel, we select the threshold as the point that maximizes the area under the ROC curve.

In Table 4.13, we can see the number of *true positives*, *false positives* and *false negatives* for each channel.

Channel	True Positives	False Positives	False Negatives
M-6	1	0	0
M-1	0	0	1
M-2	1	0	0
S-2	1	0	0
P-10	1	0	0
T-4	1	0	0
T-5	1	0	0
F-7	0	1	3
M-3	0	0	1
M-4	0	0	1
M-5	1	0	0
P-15	1	0	0
C-1	1	0	1
C-2	2	14	0
T-12	0	0	1
T-13	0	0	2
F-4	1	4	0
F-5	1	0	0
D-14	2	0	0
T-9	2	1	0
P-14	1	1	0
T-8	0	0	2
P-11	1	0	1
D-15	0	0	1
D-16	0	0	1
M-7	1	0	0
F-8	0	0	1
	20	21	16

Table 4.13: Predictions for each channel with $w = 250$. Channels with the worst performance are highlighted in red.

In summary, in almost all channels, the anomaly sequences are recognized. On the other hand, in Table 4.13 are highlighted the three channels with very bad scores. In particular, the model fails to reconstruct the channel *C-2* and thus the number of *false positives* is high. If the model had reconstructed better those three features the scores would have been even higher.

Then, we sum all *false positives*, *false negatives* and *true positives* and we obtain the final scores shown in Table 4.14.

F1 score	Precision	Recall
50.08%	44.74%	57.22%

Table 4.14: Final scores after 5 runs.

Deep Evidential Regression

With this dataset, we proceed again in a univariate way, training and evaluating each channel separately using time windows of size 250 as suggested by the paper [12].

Unlike the baseline, this method is based on forecasting thus it is interesting to study this dataset with another configuration.

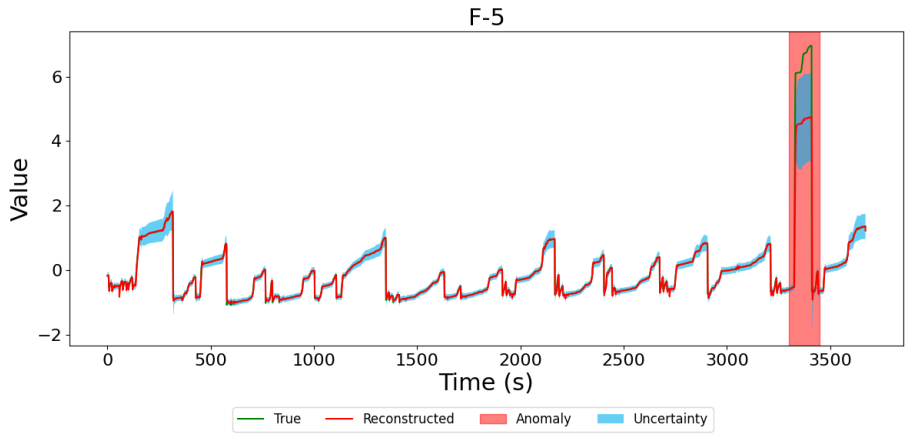

Figure 4.21: Reconstruction of the channel *F-5*.

In Figure 4.21, we observe how the model has perfectly reconstructed the channel *F-5* and, in correspondence with the anomalous sequence, we can see how the uncertainty of the model is higher.

Channel	True Positives	False Positives	False Negatives
M-6	1	0	0
M-1	0	0	1
M-2	1	0	0
S-2	0	0	1
P-10	0	0	1
T-4	1	0	0
T-5	0	0	1
F-7	3	0	0
M-3	0	0	1
M-4	0	0	1
M-5	0	0	1
P-15	1	0	0
C-1	1	0	1
C-2	1	0	1
T-12	0	1	0
T-13	0	0	2
F-4	0	0	1
F-5	1	0	0
D-14	1	0	1
T-9	1	0	1
P-14	1	0	0
T-8	0	0	2
P-11	2	0	0
D-15	1	0	0
D-16	1	0	0
M-7	1	0	0
F-8	1	0	0
	19	1	16

Table 4.15: Predictions for each channel with $w = 250$.

In Table 4.15, we can see how the number of *false negatives* is very high meaning that the model has reconstructed perfectly also some of the anomaly sequences. This is because those anomalies do not present changes in their distributions.

In the end, in Table 4.16, we can see the final scores. There is a huge improvement in the scores obtained with the baseline. The recall score is low due to the high number of *false negatives*.

F1 score	Precision	Recall
70.16%	96.48%	54.90%

Table 4.16: Final scores after 5 runs.

Graph Neural Network

Proceeding as before, we try first with the default parameters used for the SWaT dataset using a time window of size 250.

During the experiments, I found that by changing the parameters the reconstructions produced did not change. Therefore, for simplicity, we will only discuss the scores obtained.

Both models produce the same reconstruction plot so it is useless to plot it and the only difference is on the number of testing labels.

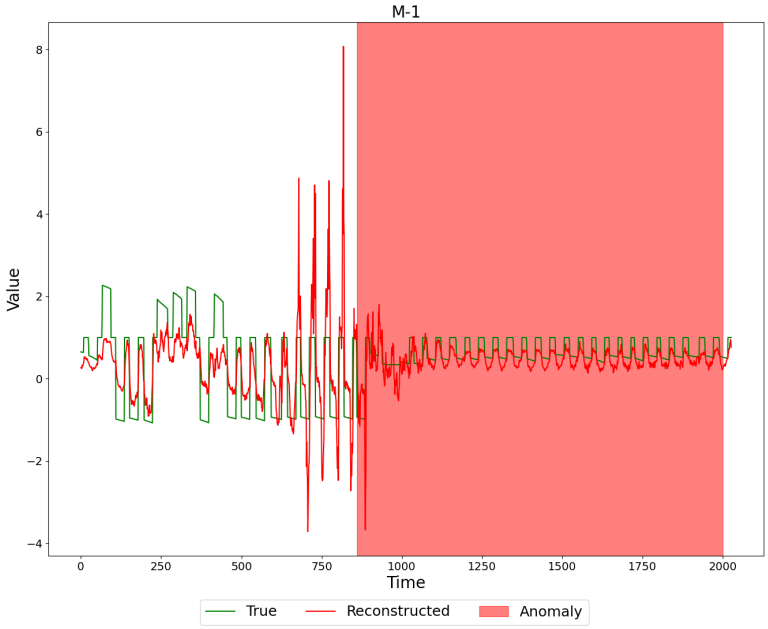

Figure 4.22: Reconstruction of the channel *M-1*

As shown in Figure 4.22, the model is not able to reconstruct well the input and it approximates only the general shape of the data.

More in detail, in Table 4.17, it is possible to notice that all channels are correctly predicted and, unlike the previous methods, the number of *false positives* and *false negatives* is very low.

Channel	True Positives	False Positives	False Negatives
M-6	1	0	0
M-1	1	0	0
M-2	0	0	1
S-2	1	0	0
P-10	1	0	0
T-4	1	0	0
T-5	1	0	0
F-7	3	0	0
M-3	1	0	0
M-4	0	0	1
M-5	1	0	0
P-15	1	0	0
C-1	2	0	0
C-2	1	1	1
T-12	1	0	0
T-13	2	0	0
F-4	0	0	1
F-5	1	0	0
D-14	2	0	0
T-9	1	1	1
P-14	1	0	0
T-8	1	0	1
P-11	1	0	1
D-15	0	0	1
D-16	1	0	0
M-7	1	0	0
F-8	1	0	0
	28	2	8

Table 4.17: Predictions for each channel with $w = 250$.

In Table 4.18, there are the scores obtained with the default parameters.

w	F1 score	Precision	Recall
250	83.08%	93.00%	75.00%

Table 4.18: Mean scores after 5 runs.

With this model, we have obtained the highest results.
Then, we try to change the number of connections per node (k).
In Table 4.19, we can summarize the results obtained with different values of k.

Value of k	F1 score	Precision	Recall
$k = 10$	80.29%	**100.00**%	67.48%
$k = 30$	80.65%	96.00%	69.00%
Mean proposal	**85.71**%	**100.00**%	**75.00**%

Table 4.19: Mean scores after 5 runs.

Changing the value of k does not improve the performance of the model. The only difference is the balance between *false positives* and *false negatives*. For example, with a lower value of k the model does not recognize *false positives*.

With my *mean approach*, the scores are higher and in particular *f1 score* and *recall*. Removing the constraints on the graph structure, each node has from 10 to 40 connections for almost all channels. This proves that trying to make each node the same as the others do not work.

Then, we can proceed in our experiments by changing the size of the embedding.

Value of *dim*	F1 score	Precision	Recall
$dim = 32$	**87.97**%	97.00%	**81.00**%
$dim = 128$	83.87%	**100.00**%	72.00%

Table 4.20: Mean scores after 5 runs.

In Table 4.20, we notice how changing this parameter the performance decrease a lot.

The last hyperparameter that we can change is the size of the time window. Due to the lack of samples, the expectation is that by increasing this parameter, the performance will also increase because the number of predicted labels is lower.

Value of w	F1 score	Precision	Recall
$w = 60$	81.97%	**100.00**%	69.00%
$w = 250$	**83.08**%	93.00%	**75.00**%

Table 4.21: Mean scores after 5 runs.

In fact, in Table 4.21, we notice how the model with $w = 250$ outperforms the first one but it does not mean that the second model performs better.

In other words, removing the few normal points ad the beginning of the testing set will result in a final set with ever more anomalies than normal points.

Conclusions

This dataset is very hard to model and many experiments in literature are performed on this data. Moreover, the channels have very different distributions because each channel models a different physical system.

Furthermore, due to how the dataset was collected, different features with different meanings are put together. For example, the dataset includes one-hot encoded information about commands that were sent or received by specific spacecraft modules in a given time window.

Due to the nature of the data, *reconstruction-based* models are reported to get better performance. Unfortunately, the two proposed methods are both based on *forecasting* and, since the data are stationary, we could not get high scores.

Method	F1 score	Precision	Recall
Baseline	50.08%	44.74%	57.22%
Deep Evidential Regression	70.16%	96.48%	54.90%
Graph Neural Network	87.97%	97.00%	81.00%
State of the art (MTAD-GAT)	90.84%	87.54%	94.40%

Table 4.22: Comparison of the results between my proposal approaches and the state of the art.

Nonetheless, as shown in Table 4.22, with *Graph Neural Network* the performance is high. The results of the baseline model are quite satisfactory when compared to the ones obtained in Telemanom [12].

Both of the proposed models have a precision score higher than the state of the art, meaning that the number of *false positives* is lower.

Deep Evidential Regression fails to detect some anomalies because those samples do not present any shift distribution with respect to the normal ones and thus they are reconstructed well.

The state of the art, *MTAD-GAT*, implements two different graphs to learn the correlations between the features and the temporal one. Moreover, the authors have proposed a combination of both *reconstruction* and *forecasting* contributions.

This function makes the model applicable in various contexts since it just requires modifying the parameter γ that controls the trade-off between the two contributions.

4.4.3 Suspension arm

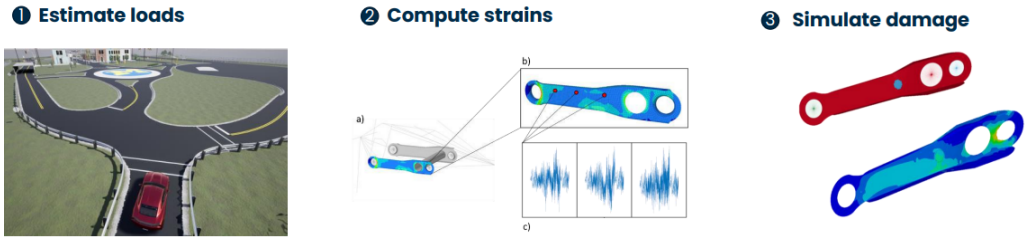

Figure 4.23: Scheme of the suspension arm. (image taken from [14]).

This private dataset, provided by AddFor, represents the simulate damage of a suspension of a car in a simulate environment with sampling rate of 3Hz as shown in Figure 4.23. The data has been already preprocessed with the *MinMaxScaler* in range [0, 1] therefore we can directly train the model with the provided data.

In Figure 4.24, we can notice that all the 30 features are float with no missing or wrong values.

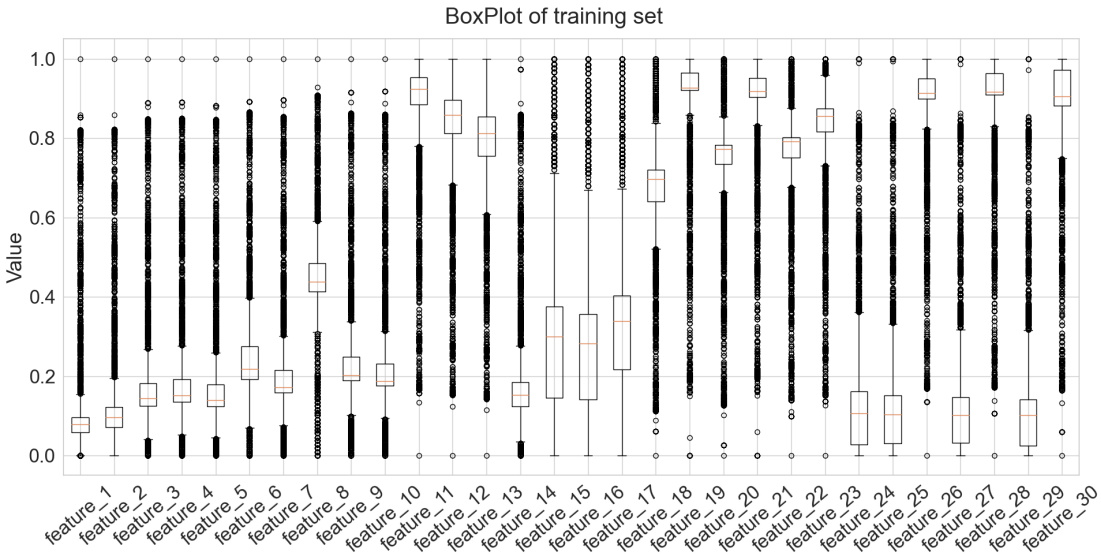

Figure 4.24: Boxplot of training set.

Some sensors have distributions completely different from the normal one and this, as said before, can make the train more difficult for the model. In Figure 4.25, we can see some of those sensors with strange distributions.

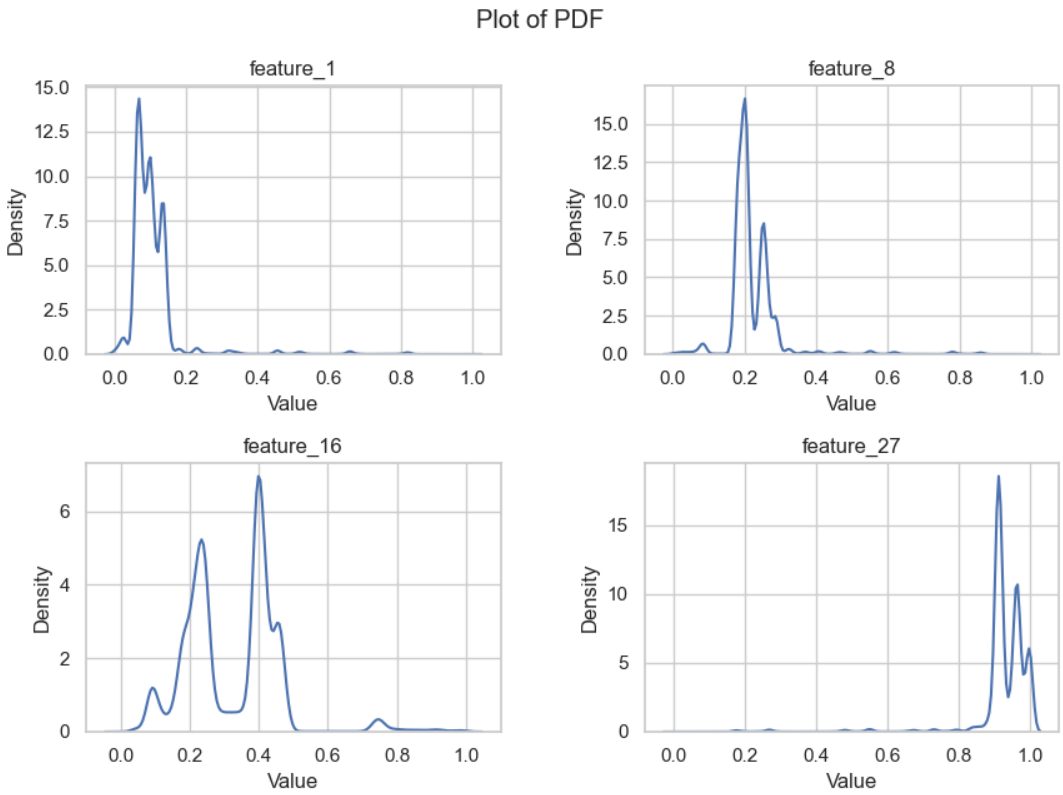

Figure 4.25: Plot of the distributions of some features.

Moreover, as shown in Table 4.23, we know that we have enough data to train and validate the model and the number of anomalies is far greater than the normal samples.

At the end, after the adfuller test, we conclude saying that the timeseries is stationary thus we do not apply the detrending.

Number of dimensions	Training set size	Validation set size	Testing set size	Anomaly ratio (%)
30	56.114	190.475	61.647	71.55%

Table 4.23: Summary of the dataset.

Since we have a large validation set (around 190k samples), we use this set to pick the best parameters and for the threshold selection.

Baseline

As described in the previous section, this dataset is composed of 30 features and the data available have already been normalized in the range [0, 1] thus any scaling operation is not needed.

From the discussed characteristics of this dataset, the best approach is to implement a *reconstruction-based* model and for this reason the initial model is an auto-encoder based on LSTM layers so the network can capture the temporal dependency of the input data.

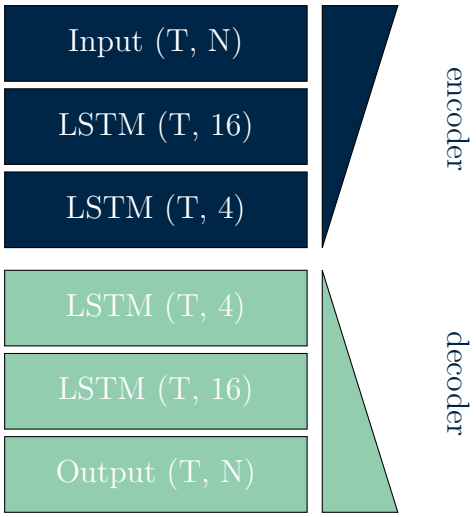

Figure 4.26: Scheme of the proposed auto-encoder with T time instants and N features.

In Figure 4.26, the network's architecture is shown and it can be seen that the model takes as input a multivariate time series with N features in T time instants. The encoder, composed of two LSTM layers, encodes the time series to a dimension of 4. Then, the 1x4 vector is repeated w times, with w the number of instants in each window. The final layers decode the repeated encoded vector to re-built the input time series.

As said in Section 2.12, after the training, the last step in the pipeline is to define the way to calculate the threshold. A very trivial method is to compute the MSE with the reconstruction error of the last point of each time window and then plot the ROC curve to choose a threshold that maximizes the area under the curve of the validation data samples. The goal is to maximize the ROC score since other metrics are sensible to the imbalance of the classes. Then, the MSE over the test samples is calculated and the previously calculated threshold is used to divide the normal sample from the anomaly one.

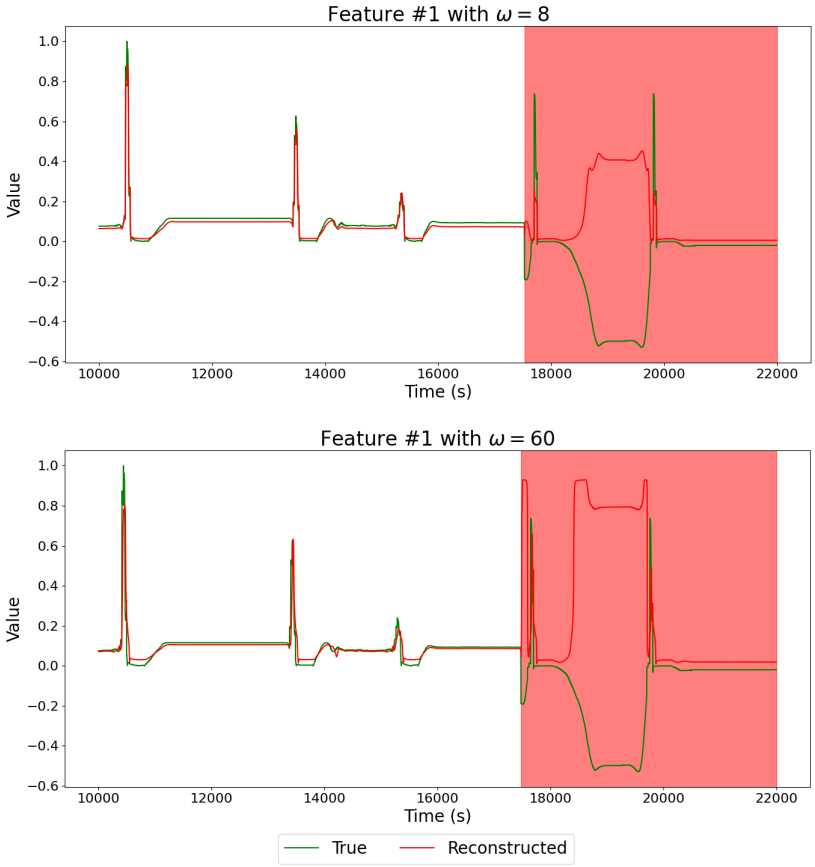

Figure 4.27: Portion of reconstruction of the first two features of the breaking point of the suspension with $w = 8$.

In Figure 4.27, there is a comparison of the same portion of input data with different values of the hyperparameter w. In both reconstructions, the AE performs very well at approximating the shape of the input data for that specific feature whereas it correctly fails with the anomaly sequence.

w	F1 score	Precision	Recall	ROC AUC	PR AUC
8	**94.90%**	**99.50%**	**90.79%**	**92.51%**	**98.87%**
60	93.42%	99.26%	88.23%	89.97%	98.48%

Table 4.24: Mean of scores on validation set after 5 runs with different w.

From the scores in Table 4.24, the model with a smaller dimension time window performs slightly better and this can be motivated by the fact that the dataset has a low temporal dependency. Therefore, for further experiments, only this setting is

considered.

From what concerning the method to pick the best threshold, two possible ways can be described. In general, since the anomalies can be in even one sensor at a time and consequentially the average reconstruction error would be low, some authors pick one threshold for each feature based on reconstruction error on the validation set.

Threshold	F1 score	Precision	Recall	ROC AUC	PR AUC
Multiple	**97.86**%	95.83%	**100.00**%	72.50%	95.83%
Single	94.90%	**99.50**%	90.79%	**92.51**%	**98.87**%

Table 4.25: Mean of scores on validation set after 5 runs with different threshold selection.

As shown in Table 4.25, in this dataset, since the features are homogeneous and the anomaly is on more than one feature it is better to proceed to select a global threshold based on the average of the reconstruction error (MSE). Therefore, considering the ROC curve in Figure 4.28, the best threshold is the point closer to the point $(0, 1)$ or, in other words, the one that maximizes the area under the curve.

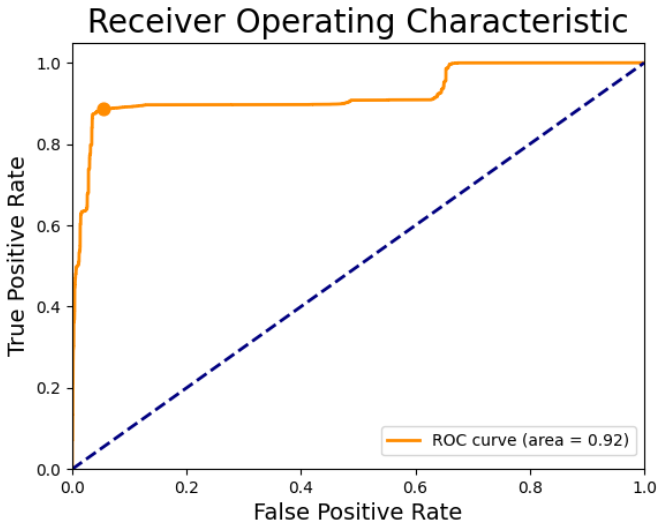

Figure 4.28: ROC curve of validation set.

Finally, in Table 4.26, we can observe the scores of the model with the best settings.

F1 score	Precision	Recall	ROC AUC	PR AUC
89.40%	87.81%	91.16%	79.60%	86.37%

Table 4.26: Final scores after 5 runs.

Deep Evidential Regression

As did for the SWaT dataset, we train one model for each feature because Deep Evidential Regression is suitable for univariate regression only, where the model has to predict only one value.

We implement the same architecture shown before with the new *DenseNormalGamma* layer.

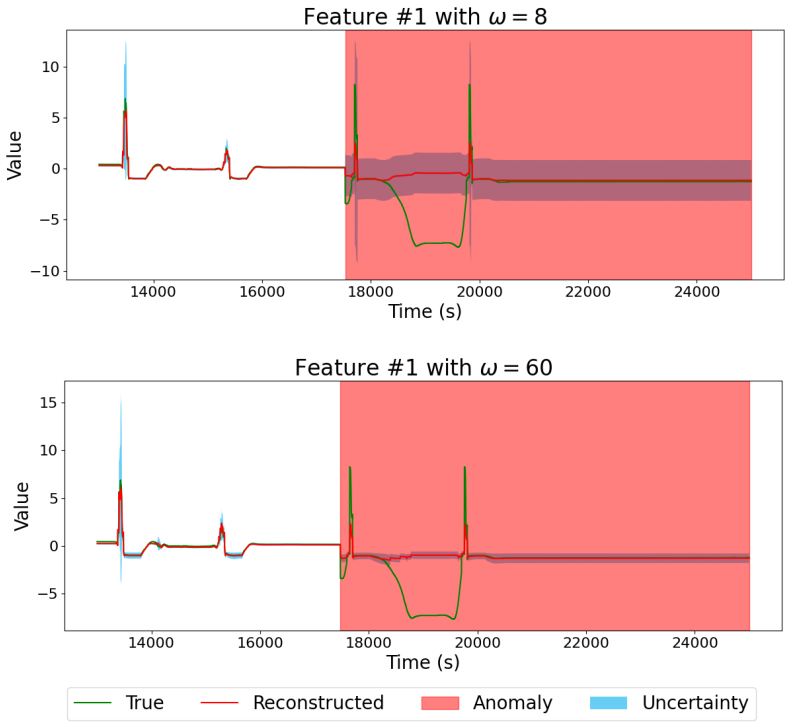

Figure 4.29: Comparison of the reconstruction of the same portion of data with different w.

In Figure 4.29, we can observe how with both models the reconstruction error is very low and then with a smaller time window the model has captured correctly the anomaly region by increasing the uncertainty in that area.

On the other hand, with $w = 60$ the model is able to reconstruct well also the anomaly region with low uncertainty. This behaviour is the demonstration

of the overfitting occurring during the training. In other words, the model does not estimate correctly the uncertainty because it does not recognize that region as anomalous.

Quantile Transformation As did for SWaT dataset, we try also with the *Quantile Transformation* with the goal to reshape the distribution of the features. In previous experiments, this transformation had improved the performance with $w = 8$ thus we test also a model with this transformation.

Figure 4.30: Reconstruction of a portion of data after *Quantile Transformation*.

In Figure 4.30, we observe how the reconstruction of the model with the *Quantile Transformation* presents high uncertainty in the abnormal region whereas the normal samples are correctly reconstructed. Despite this plot, the performance is almost comparable with the one of the model with *Standard Scaler*.

w	F1 score	Precision	Recall	ROC AUC	PR AUC
8	**98.52**%	**99.23**%	**97.82**%	**94.12**%	**99.08**%
8 (with *QuantileTrans.*)	97.40%	98.56%	96.57%	87.49%	98.06%
60	93.67%	99.04%	88.86%	88.99%	98.33%

Table 4.27: Final scores on validation set after 5 runs.

As shown in Table 4.27, the model that performs better is the one with a smaller time window as discussed before.

Since the company has provided also a validation set, we proceed following the standard procedure in machine learning. We test the hyperparameters on the

validation set and then we pick the best combination to get the final results on the testing set.

To select the correct threshold, we consider again the ROC curve and the point that maximizes the area under the curve.

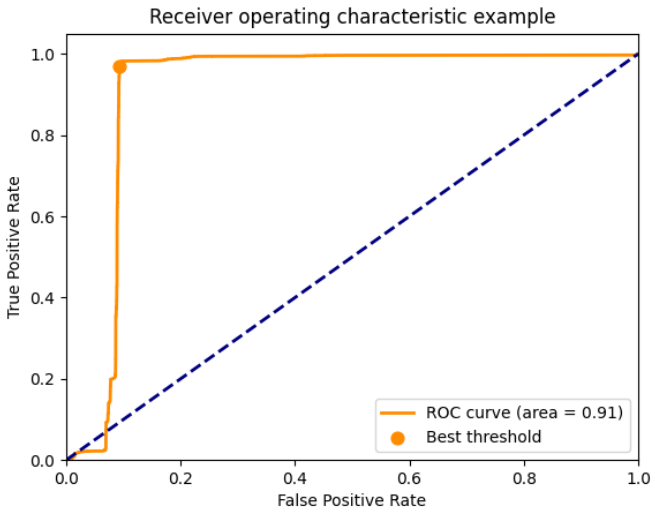

Figure 4.31: ROC curve.

In Figure 4.31, we can observe the ROC curve and the point represents the best threshold that we will use on the testing set.

Then, in Table 4.28, we show the final results on the testing set with the best combination found ($w = 8$).

F1 score	Precision	Recall	ROC AUC	PR AUC
92.13%	87.20%	97.66%	80.78%	86.83%

Table 4.28: Final scores after 5 runs with $w = 8$.

We can notice an improvement in the already good performance of the baseline model. In addition to the detection of anomalies, we get also the uncertainty associated with each point making this method very suitable in this context.

Graph Neural Network

Firstly, as did before, we define the baseline scores using the suggested hyperparameters of the SWaT dataset and then we change one parameter at a time to study how the performance changes.

Since the company has provided the validation set, we find the best hyperparameters on that set and then we evaluate the best combination on the testing set.

In Figure 4.32, we can see the reconstruction of the first feature.

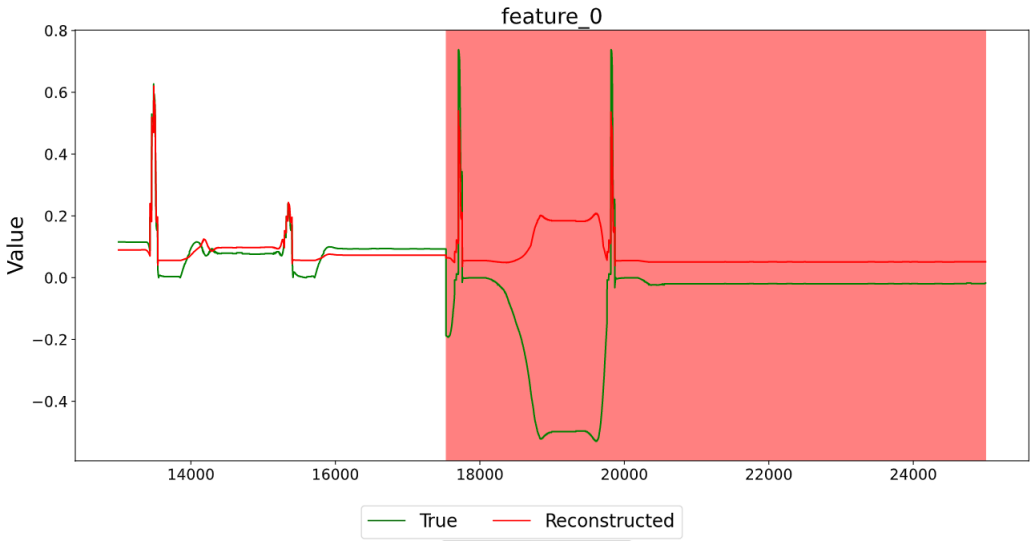

Figure 4.32: Reconstruction of a portion of the first feature.

The reconstruction of the normal sequence is very accurate whereas the anomaly one presents many reconstruction errors. This good behaviour is reflected also in the final scores as shown in Table 4.29 where we reach very high performance.

F1 score	Precision	Recall	ROC AUC	PR AUC
89.71%	87.82%	100.00%	82.41%	87.82%

Table 4.29: Mean scores on the testing set after 5 runs.

In particular, we can notice that the recall is 100% thus there are not *false negatives*.

Then, we can proceed with changing the value of k. In Figure 4.33, we can observe the comparison of the reconstruction of the same portion of input data of the first feature with different values of k.

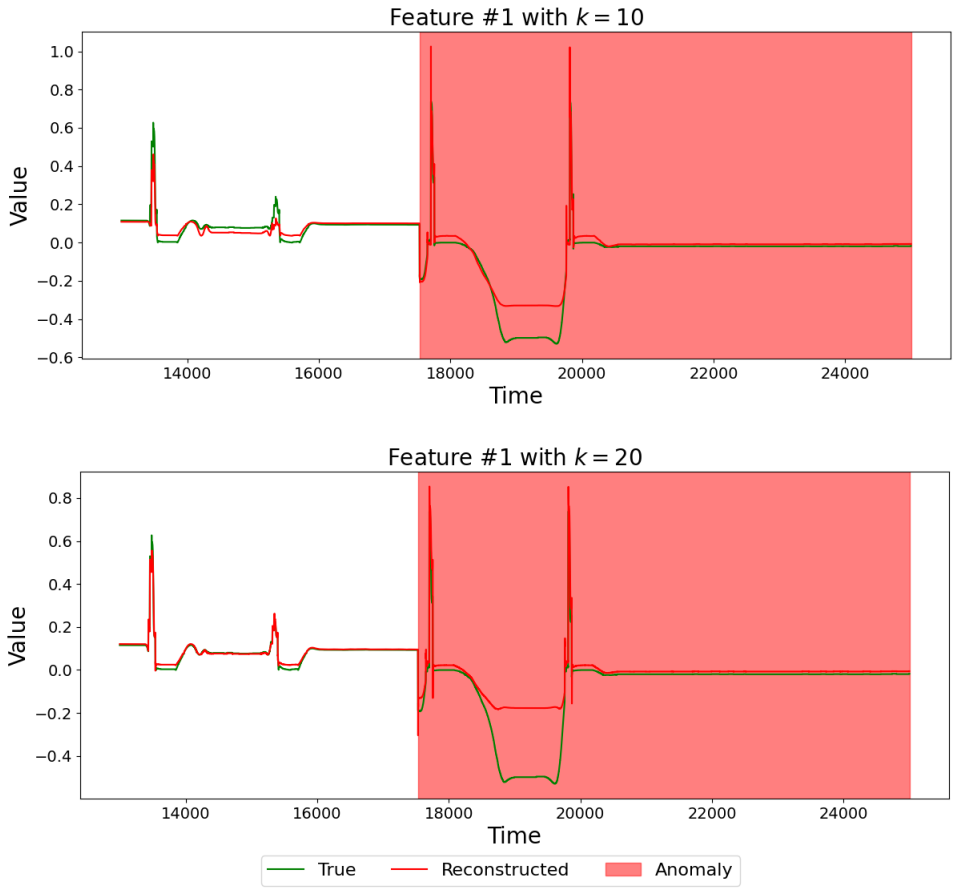

Figure 4.33: Comparison of reconstruction of the same portion of data with different values of k.

As in the case of SWaT, a high value of the number of connections per node k increases the capacity of the model to better reconstruct the input for both normal and anomaly sequences.

In Table 4.30, we have the comparison between a couple of static values of k and my implementation with a dynamic k. We notice how with my method the performance is overall higher. The reason is that the sensors of this dataset are more correlated than the SWaT dataset.

k	F1 score	Precision	Recall	ROC AUC	PR AUC
$k = 10$	97.57%	95.26%	**100.00%**	81.54%	97.99%
$k = 20$	98.69%	97.97%	99.41%	92.49%	99.06%
Mean k	**98.91%**	**98.27%**	99.21%	**92.56%**	**99.55%**

Table 4.30: Mean scores after 5 runs changing the k parameter.

Then, we can analyze how the predictions and performance change with different sizes of the embeddings.

In Figure 4.34, a portion of the second feature is reconstructed with different embeddings dimensions.

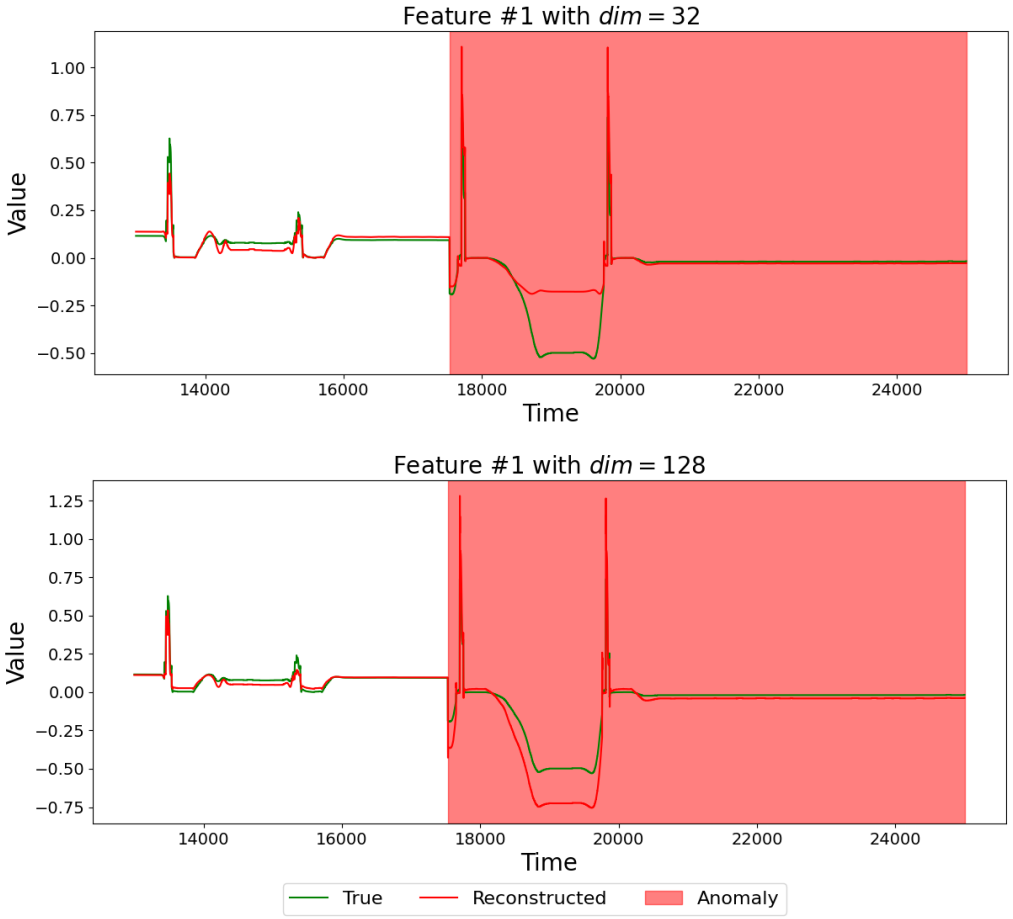

Figure 4.34: Comparison of reconstruction of the same portion of data with different values of *dim*.

With a high embeddings size, the true signal is reconstructed better as in previous experiments. In general, these tests have proved that the size of the embedding is the key of this method and the optimal value let the model minimize the reconstruction error. In fact, in Table 4.31, the performance with $dim = 128$ are by far higher than the one with $dim = 32$.

size	F1 score	Precision	Recall	ROC AUC	PR AUC
$dim = 32$	96.18%	92.64%	100.00%	85.73%	98.94%
$dim = 128$	**98.17**%	**94.56**%	100.00%	**91.16**%	**98.90**%

Table 4.31: Mean scores after 5 runs changing the embeddings size.

The last parameter that can help the model to increase the performance is the size of the time window. As did earlier, we try to increase that value in order to study the behaviour of the network with different input dimensions.

From the architecture of the model, we know that it does not implement the time correlation among the variables thus we expect that by changing this value the performance remains stable. To demonstrate this, we can see in Table 4.32 how the scores with $w = 10$ are much higher than the model with a larger window.

w	F1 score	Precision	Recall	ROC AUC	PR AUC
$w = 10$	**96.63**%	**92.65**%	**100.00**%	**91.62**%	**98.65**%
$w = 20$	94.69%	89.60%	100.00%	88.85%	94.86%

Table 4.32: Mean scores after 5 runs changing the size of the time window.

We can plot also the confusion matrix as shown in Figure 4.35. With a smaller time window, the number of false positives is half of the model with a larger window.

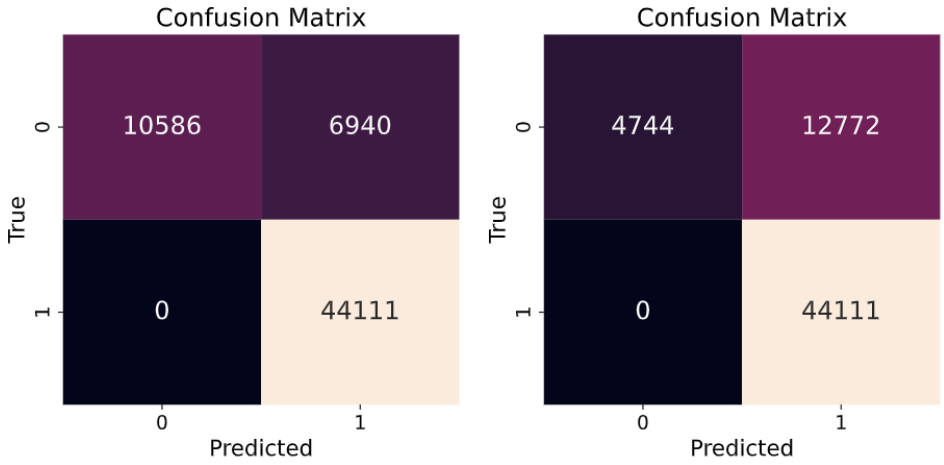

Figure 4.35: Confusion matrix with $w = 10$ (left) and $w = 20$ (right).

As the last experiment, we combine all the best hyperparameters found and observe the scores with this final combination on the testing set.

Hyperparameters	F1 score	Precision	Recall	ROC AUC	PR AUC
default	89.71%	87.82%	**100.00**%	82.41%	88.82%
$w = 10$ $Mean\ k$ $dim = 128$	**92.03**%	**88.23**%	98.77%	**82.87**%	**90.23**%

Table 4.33: Final scores on testing set.

In Table 4.33, we summarize the results obtained with the model with the default hyperparameters and the one with the best combination found on the validation set. The latter model performs slightly better though the scores on the test set are lower than the ones on the validation set.

Conclusions

Since it is an industrial dataset, the data present a *domain shift*. In fact, the distribution of the validation set is slightly different from the one of the testing sets. This can be proven by the different scores obtained with these two sets of data.

Method	F1 score	Precision	Recall
Baseline	89.40%	87.81%	91.16%
Deep Evidential Regression	92.13%	87.20%	97.66%
Graph Neural Network	92.03%	88.23%	98.77%

Table 4.34: Comparison of the results of the proposed approaches.

In Table 4.34, we can notice how both proposed methods perform well on the unseen test data.

Since in this dataset we have a breakout point, the scores are higher than the previous datasets.

With the *Graph Neural Network*, it is possible to learn the correlation between the features and, especially in industrial domains, this can be very important in order to find which sensor is damaged.

At the same time, the contribution of the *Deep Evidential Regression* is very important to estimate the uncertainty of the predictions.

Chapter 5

Conclusions

In this thesis, we have presented two new approaches for anomaly detection in multivariate time series, in particular in the industrial domain. These methods try to solve some challenges specific to this field.

In Chapter 3, we have described first the *Deep Evidential Regression*. The contribution of this method is the capacity of estimating the uncertainty of the predictions without sampling during the training. Instead of placing priors distributions on network weights, as is done in Bayesian NNs, the authors have placed priors directly over the likelihood function. By training a neural network to output the hyperparameters of the higher-order evidential distribution, a grounded representation of both epistemic and aleatoric uncertainty can then be learned without the need for sampling and no additional training time. In the industrial domain, the goal is to recognize anomalies in real-time with a high degree of certainty.

The *Deep Evidential Regression* is designed for the detection of anomalies in images or univariate regression. During the thesis, we have extended the application domains to the multivariate time series and provided an evaluation system based on the entropy of the data.

The second proposed method is the *Graph Neural Network* that is based on a graph network. The model can capture and learn the correlations among all the sensors providing interpretability of the model through the graph structure and the embeddings. In particular, the graph edges and attention weights provide interpretability by indicating which sensors are related to one another. Moreover, the attention weights further indicate the importance of each of a node's neighbours in modelling the node's behaviour. The number of connections for each node is predefined as a hyperparameter but, during the experiments, we have also provided a possible implementation based on a dynamic number of connections per node observing an increase in the performance. The reason is that each sensor may require a different number of connections due to its nature. For example, an important sensor requires more connections than a peripheral one.

For each public dataset, we have also presented the state of the art algorithms describing their characteristics.

The *DAICS* [8] proposes an adaptive threshold that changes with the distribution of the input data trying to solve the problem of the *domain shift* due to the deterioration of components.

The *MTAD-GAT* [9] implements two different graph networks to learn both the correlations among the features and the temporal ones. Moreover, since in literature both *reconstructed-based* and *forecasting-based* models have been proposed, this method implements a combination of both the models. By doing so, the algorithm can be applied in many more situations.

In conclusion, uncertainty estimation and the explanation of the anomalies have a very significant societal impact because humans will inevitably become increasingly trusting in a model's predictions.

Bibliography

[1] D. Hawkins. *Identification of Outliers*. Chapman & Hall, 1980 (cit. on p. 1).

[2] Saikiran Bulusu, Bhavya Kailkhura, Bo Li, Pramod K. Varshney, and Dawn Song. *Anomalous Example Detection in Deep Learning: A Survey*. 2021. arXiv: 2003.06979 [cs.LG] (cit. on p. 2).

[3] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. «Deep Learning for Anomaly Detection». In: *ACM Computing Surveys* 54.2 (Apr. 2021), pp. 1–38. ISSN: 1557-7341. DOI: 10.1145/3439950. URL: http://dx.doi.org/10.1145/3439950 (cit. on p. 3).

[4] Ane Blázquez-Garcıa, Angel Conde, Usue Mori, and Jose A. Lozano. «A Review on Outlier/Anomaly Detection in Time Series Data». In: *ACM Comput. Surv.* 54.3 (Apr. 2021). ISSN: 0360-0300. DOI: 10.1145/3444690. URL: https://doi.org/10.1145/3444690 (cit. on p. 6).

[5] Robin John Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. English. 2nd. Australia: OTexts, 2018 (cit. on pp. 9–11).

[6] Lilian Weng. *From Autoencoder to Beta-VAE*. https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html. [Online]. 2018 (cit. on p. 18).

[7] Jeremy Jordan. *Introduction to autoencoders.* https://www.jeremyjordan.me/autoencoders/. [Online]. 2018 (cit. on p. 19).

[8] Maged AbdelAty, Roberto Doriguzzi Corin, and Domenico Siracusa. «DAICS: A Deep Learning Solution for Anomaly Detection in Industrial Control Systems». In: *CoRR* abs/2009.06299 (2020). arXiv: 2009.06299. URL: https://arxiv.org/abs/2009.06299 (cit. on pp. 21, 91).

[9] Hang Zhao et al. *Multivariate Time-series Anomaly Detection via Graph Attention Network*. 2020. arXiv: 2009.02040 [cs.LG] (cit. on pp. 24, 63, 91).

[10] Ailin Deng and Bryan Hooi. «Graph Neural Network-Based Anomaly Detection in Multivariate Time Series». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021 (cit. on p. 31).

[11] Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. «Deep evidential regression». In: *Advances in Neural Information Processing Systems* 33 (2020) (cit. on pp. 47, 51).

[12] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. «Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding». In: *arXiv preprint arXiv:1802.04431* (2018) (cit. on pp. 65, 67, 69, 75).

[13] Hang Zhao et al. *Multivariate Time-series Anomaly Detection via Graph Attention Network*. 2020. arXiv: 2009.02040 [cs.LG] (cit. on p. 67).

[14] Alberto Ciampaglia, A Santini, and G Belingardi. «Design and analysis of automotive lightweight materials suspension based on finite element analysis». In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 235.9 (2021), pp. 1501–1511. DOI: 10.1177/0954406220947457. eprint: https://doi.org/10.1177/0954406220947457. URL: https://doi.org/10.1177/0954406220947457 (cit. on p. 76).