POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Deep Learning-based Unsupervised Anomaly Detection on Energy Consumption Data

Supervisors

Prof. Paolo GARZA

Marco GALATOLA

Candidate

Carla Maria MEDORO

July 2024

Summary

Time series anomaly detection is the process of identifying deviations from expected patterns within sequential data points over time. It is a fundamental task in all those scenarios where analysing and understanding temporal trends is essential. This field leverages advanced statistical and machine learning techniques to detect irregularities, spikes or unusual patterns in time series data.

Nowadays, it has become of great importance to perform anomaly detection on energy consumptions, in order to identify and prevent a wasteful use of energy for a better management of LECs in EU.

This thesis investigates state-of-the-art Deep Learning approaches applied on energy consumption data in a completely Unsupervised manner. The dataset used for the development of the chosen models is taken from the Large-scale Energy Anomaly Detection competition hosted by Kaggle. The methods explored work as to reconstruct or forecast the normal behaviour of the input time series; then, anomaly detection is performed by classifying as anomalies all those output data points which differ from the expected values more than a certain threshold.

The goal of the thesis is to understand the applicability of these methods in absence of annotated data. In many real-life scenarios, in fact, datasets are not provided with indications of whether the contained observations are to be considered as anomalies or not. This makes the anomaly detection task harder. Without labels, in fact, one has no way of identifying and removing anomalous data points from the dataset which is going to be used to train the models: they will thus learn to recognize abnormal patterns in data as normal energy consumption behaviour.

This work explores various Deep Learning techniques and thresholding methods in order to enhance the development of more resilient and effective models for practical applications in energy management.

Ai miei genitori,
che mi hanno spronato e supportato in tutti questi anni

Table of Contents

List of Tables

List of Figures

Acronyms

AI

Artificial Intelligence

LEC

Local Energy Community

DATA CELLAR

Data Hub for the Creation of Energy communities at the Local Level and to Advance Research on them

EU

European Union

GDPR

General Data Protection Regulation

IQR

Inter-Quartile Range

STL

Seasonal-Trend Decomposition using LOESS

MSTL

Multiple Seasonal-Trend Decomposition

XGBoost

eXtreme Gradient Boosting

ARIMA

AutoRegressive Integrated Moving Average

LSTM

Long-Short Term Memory

Bi-LSTM

Bidirectional Long-Short Term Memory

CNN

Convolutional Neural Network

NN

Neural Network

DeepAnT

Deep-learning approach for Anomaly detection

GNN

Graph Neural Networks

GCN

Graph Convolutional Netowrk

GAT

Graph Attention Mechanism

ReLU

Rectified Linear Unit

AE

AutoEncoder

VAE

Variational Autoencoder

GAN

Generative Adversarial Networks

SGD

Stochastic Gradient Descent

AD

Anomaly Detection

STAD-GAN

Self-Training based Anomaly Detection with GAN

DNN

Deep Neural Network

USAD

UnSupervised Anomaly Detection

TP

True Positive

FP

False Positive

FN

False Negative

FC

Fully Connected Layer

KL

Kullback-Leibler loss

LEAD

Large-scale Energy Anomaly Detection

ROC

Receiver Operating Characteristic curve

AUC

Area Under the ROC curve

TPR

True Positive Rate

FPR

False Positive Rate

SWaT

Secure Water Treatment

MSE

Mean Squared Error

MAE

Mean Absolute Error

CTIC

Technological Center for Information and Communication Foundation

Chapter 1

Introduction

1.1 Problem setting: energy anomaly detection

Anomaly detection is the task of identifying patterns in data that do not conform to the usual, expected behaviour. This is of great interest in many real-life applications and scenarios, like healthcare, cybersecurity, industry and robotics, where detecting for example heart palpitations, intrusions, production faults or system defects is extremely important to ensure the correction and eventually even the prevention of these problems.

A field in which such task is acquiring more and more importance is the one of energy consumption. Nowadays, people are concerned with the impact that their everyday activities have on the environment and on sustainability. Even such an important entity as the United Nations listed among its seventeen Sustainable Development Goals the importance of ensuring access to affordable, reliable, sustainable and modern energy for all citizens, of making cities and human settlements sustainable and of ensuring sustainable consumption and production patterns.

It is in this setting that the DATA CELLAR project came to life: the goal is that of creating a federated energy dataspace that will support the creation, development and management of Local Energy Communities (LECs) in EU.

1.2 DATA CELLAR

DATA CELLAR, Data Hub for the Creation of Energy communities at the Local Level and to Advance Research on them, is a project funded by the European Commission that will last three and a half years and that involves the collaboration of several organizations in different European countries, Italy being one of them. It aims to support the energy management of Local Energy Communities through Artificial Intelligence techniques: a big step towards saving energy and achieving global sustainability. LECs, in fact, have being recognised by the European Union as the main protagonist in driving the EU's energy transition.

LECs are citizen-driven energy actions, involving citizens, small businesses and local authorities, that contribute to the clean energy transition, advancing energy efficiency within local communities, by producing, managing and consuming their own energy. They comprise both energy producers and distributors, and consumers: their goal is to enable access to renewable energy and information regarding how to enhance the energy efficiency of buildings, in order to improve the control over their energy bills.

The main objective of DATA CELLAR is that of creating an energy dataspace that will support the creation, development and management of LECs in the EU. The data space population will be facilitated via an innovative rewarded private metering approach, with a focus on an easy onboarding and interaction, guaranteeing a smooth integration with other EU energy data spaces, providing to LEC stakeholders services and tools for developing their activities [1].

In this setting, a great importance has been given to analysing methodologies to perform anomaly detection on energy consumption data, to try to identify and prevent a wasteful use of energy. This can be particularly useful not only from an environmental point of view, but also from an economical one, as wasting energy is directly linked with a waste of money both for the energy providers and the consumers, who will have to pay higher bills not reflecting their true, normal, energy consumption.

The data employed to design anomaly detection applications is often times collected by smart meters present in buildings of any kind, which record information like the energy consumption and communicate it to both the consumer and the energy providers for monitoring the correct functioning of the system and for customer

[1] https://datacellarproject.eu/

billing. Smart meters typically collect data periodically, for example every hour, thus allowing the users to observe the evolution of the consumptions over a period of time.

1.3 Objective of this thesis

The work of this thesis has been done in collaboration with LINKS Foundation, and contributes in a very small way to the big project of DATA CELLAR.

To perform anomaly detection, there are typically several methodologies which can be taken into consideration, and their distinction is mainly based on whether the dataset one works with is annotated with labels indicating the abnormality of data points or not.

In fact, if the dataset is labeled, one can decide to take advantage of Supervised approaches and pose the task as a classification one, using AI models to categorize data into "anomaly" and "not-an-anomaly" classes.

However, in many real-life scenarios this is not possible, as often times the data has no indications of whether each measurement is anomalous or not. So how can we efficiently perform anomaly detection in this setting? This thesis aims at investigating whether it is feasible to employ Unsupervised approaches, that do not need labels but rather leverage the true target of the analysis, the energy consumption, to perform anomaly detection.

A big challenge therefore arises: if the data is not annotated, one has no way of knowing which measurements can be considered as anomalies, unless one decides to use statistical methods like the Inter-Quartile Range (IQR), assuming that all the values which do not fall within the IQR are anomalies, and therefore eliminate these data points from the training set. This would mean, though, simplifying too much the problem: anomalies, in fact, are not only abnormally high or low values, but can also take on values in the normal range for the energy measurements, therefore such an approach would be counter-productive.

Since there is no way to safely remove potentially anomalous data points, if labels are not available, this means that AI models need to be trained with data that contains also anomalies, even though usually in a very small percentage with respect to the normal measurements (2-5%), which makes therefore the task of anomaly detection more challenging: the models employed would learn to reproduce also the anomalous points, as they are interpreted as the normal behaviour of the phenomenon in observation. This is the challenge that we will address in this thesis: trying to assess the feasibility of Unsupervised, Deep Learning, models for the task of anomaly detection, when such models need to be trained on data containing anomalies.

1.4 Outline of this Thesis

The thesis is organized as follows.

Chapter 2 presents an overview of the main methodologies to perform anomaly detection: after a presentation of the type of data we are going to deal with, the chapter focuses on explaining the different approaches to the task. The focus will mainly be on Unsupervised methodologies, in particular forecasting-based and reconstruction-based models, but a brief description of how to tackle the problem in the Supervised domain will also be provided.

Chapter 3 will propose an in-depth description of the problem, from the analysis of the dataset used to perform all the experiments, to the description of the chosen techniques for the task of anomaly detection.

Chapter 4 will present the experimental results obtained with each of the methods, a comparison of all of them on another dataset and their generalization capabilities on the real data provided within the DATA CELLAR project.

Finally, Chapter 5 will contain the final conclusions that can be drawn from this project, analysing the successes and failures of the chosen methodologies and assessing what could be possibly done in the future, to further tackle such problem.

Chapter 2

Related Works

The task of anomaly detection has increasingly gained importance in many different fields. Much attention has been invested in research, to try to efficiently and effectively tackle the problem of identifying unusual patterns in data. There are many surveys that give an overview of the main characteristics of the task and the preferred models, like [1] and [2].

Anomaly detection can be applied on images and tabular data. In particular, in this thesis we are going to work with time series data.

2.1 Time Series Data

A time series is a sequence of data naturally ordered with time.

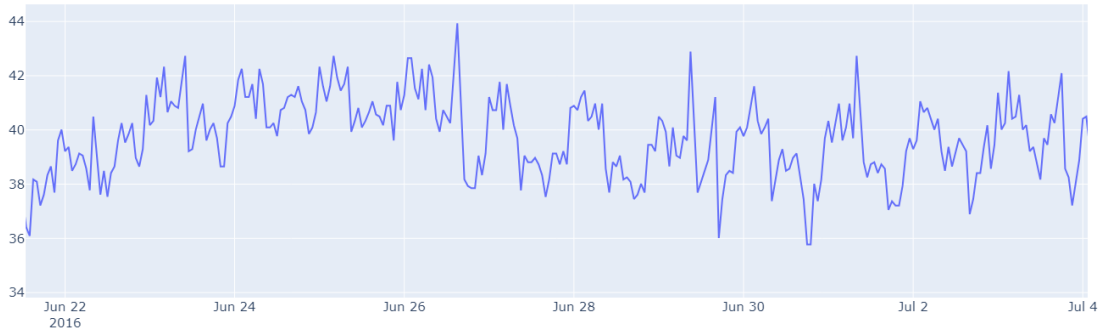


Figure 2.1: Example of a portion of a Time Series taken from the LEAD dataset

It is typically denoted as $X = \{X_i\}_{1 \leq i \leq N}$, where N indicates the number of points constituting the time series and X_i is a n-dimensional vector characterized by n possible features. A time series is said to be univariate if $n = 1$: this means that it tracks the evolution over time of a single feature, like the energy consumption

of a building in the specific case of this thesis. If $n > 1$, the time series is said to be multivariate, meaning that several features depending on time are taken into account, like the evolution of the air temperature, of the wind speed and many others: here such features are not only dependent on their own past values, but also on those of the other variables present in the dataset.

Time series analysis is extremely important in any field. One of the most interesting analysis that can be performed is the Seasonal-Trend Decomposition, which aim is that of identifying the different components of the evolution over time of a phenomenon. A time series can be defined as such: $X_t = T_t + S_t + R_t$, in the case of an additive decomposition, or $X_t = T_t \cdot S_t \cdot R_t$, for the multiplicative decomposition. The former is appropriate if the magnitude of the seasonal fluctuations or the variation of the trend does not vary with the level of the time series; it assumes that the influence of each component is independent of each other. The latter, instead, is appropriate when the seasonal factor increases or decreases as the trend changes; it assumes an influence of the components on each other. The main components are:

- Trend component T_t: it indicates the long-term movement or direction of the evolution over time of the phenomenon; it is an essential component to identify upward, downward or stable behavior of a time series
- Seasonal component S_t: it indicates a pattern which happens at fixed and known frequencies, i.e. daily, weekly, monthly
- Residual component R_t: it is what remains after all the other components have been removed from the original time series

Note that decomposing a time series can prove to be particularly useful for the task of anomaly detection because the residual component is the one that contains noise and can therefore contain anomalies.

One technique to decompose a time series is the Seasonal-Trend decomposition using LOESS (STL), proposed by [3], which applies recursively a locally weighted regression (loess) smoother to obtain the different components of a time series.

The loess regression curve $g(x)$ is a smoothing of $y = f(x)$ that can be computed for any value of x along the scale of the independent variable. To obtain $g(x)$ one considers a positive integer, q: the q values of x_i that are closest to x are selected and given a neighborhood weight $v_i(x)$ based on their distance to x, so that those which are closest are going to have the highest weights. Then, one fits a polynomial of degree d to the data to obtain the smoothed function. As q increases, $g(x)$ becomes smoother. If the underlying pattern in the data has a gentle curvature one can take $d = 1$, otherwise $d = 2$ is better. Each observation (x_i, y_i) has a

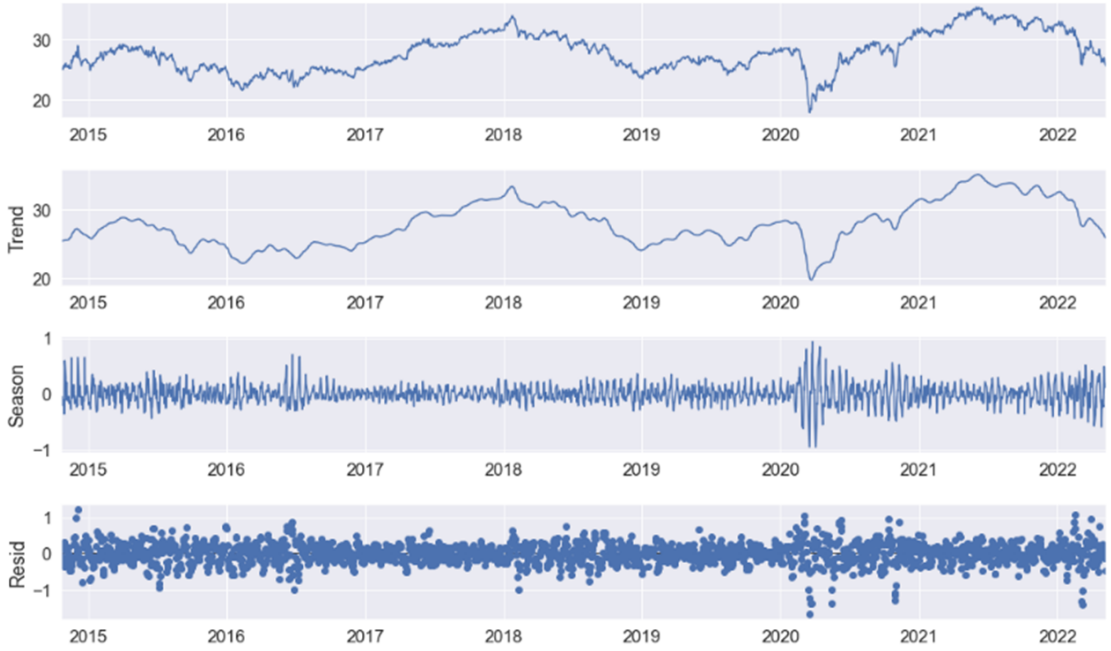


Figure 2.2: STL Decomposition of a Time Series

weight ρ_i that indicates the reliability of it with respect to the others: this can be incorporated in the loess smoothing by using $\rho_i \cdot v_i(x)$ as the weights for the local least-squares fitting.

In the case of very difficult time series, especially those that contain multiple seasonal patterns, a decomposition using the STL method might not produce optimal results: [4] proposed the Multiple STL Decomposition (MSTL) algorithm which initially identifies the number of distinct seasonal patterns present in the time series, which can be interlaced one with another, then applies the STL algorithm iteratively on each of the identified seasonal frequencies.

2.2 Anomalies in Time Series Data and methodologies

As previously stated, what becomes particularly interesting when analysing time series is to identify unexpected behaviours with respect to the normal evolution of a feature. These patterns are often known with the name of anomaly and they can be of three main different kinds:

- Point anomalies: these are the simplest types of anomalies, that refer to individual data instances that can be considered anomalous with respect to the normal behaviour
- Contextual anomalies: in this case a data instance is considered as anomalous due to its being in a specific context, which can also be defined by a series of attributes present in the dataset
- Collective anomalies: here, a collection of data points, which taken singularly would not constitute an anomaly, is anomalous with respect of the entire dataset due to their simultaneous occurrence

In the following Figure 2.3 we can appreciate these three types of anomalies.

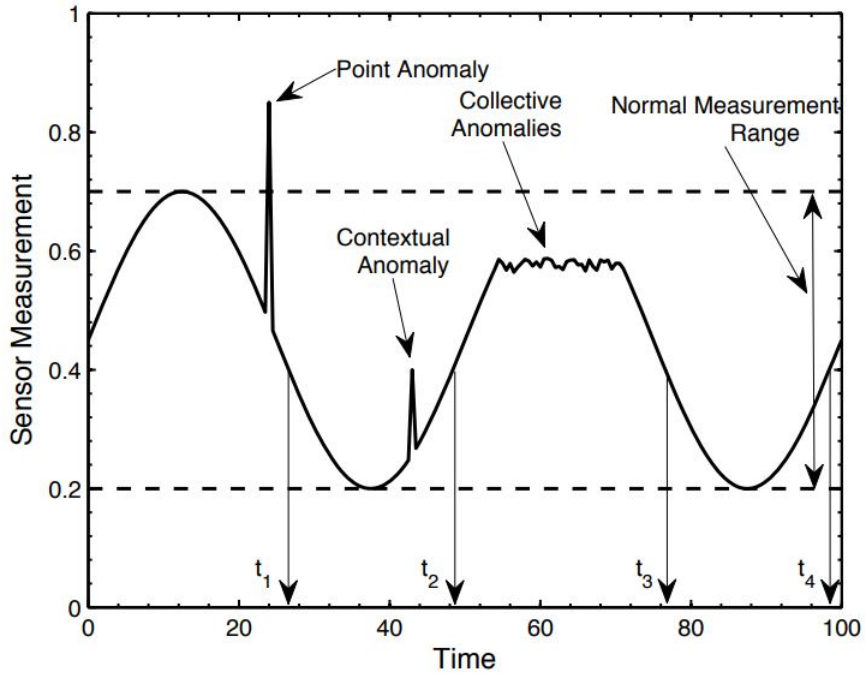


Figure 2.3: Example of Point, Contextual and Collective anomalies

It is immediate to notice the differences between the different anomalies: a point anomaly is what we would usually define as an outlier, meaning a data point whose value is much higher or in some cases even much lower than those which are typical of the time series in observation. A contextual anomaly, instead, is a data point whose value alone would actually be considered normal, but given the context it is not, as for example it does not follow smoothly the evolution of the time series, but may be an abrupt change. Finally, collective anomalies are a series of points which are simultaneously considered as abnormal: from Figure 2.3 it is evident to notice how they represent an unusual prosecution of the sensor measurement, which instead of continuing the sinusoidal-like curve gets truncated for a period of time.

To identify anomalies in a time series there are different methodologies, depending on whether the dataset is provided with labels, indicating if a data point is to be considered anomalous or not. The main approaches are:

- Supervised Approach: this technique relies on the presence of labels in the dataset. The task if formulated as a classification problem, where the aim is that of categorizing each data point as either anomalous or not. One challenge of this methodology is the imbalance often present in the dataset among anomalies and non-anomalies: in fact, in most real-case scenarios, the anomalies typically constitute very small percentages of the data, around 2%, therefore resampling techniques must be taken into consideration to balance the dataset and avoid having classifiers which are biased towards the negative (normal) class.
- Self-Supervised Approach: this technique assumes that the dataset has labels only for the normal class; typically, one would build a model to train on the normal class and apply it on the test data to identify possibly anomalous points
- Unsupervised Approach: this technique does not rely on the presence of labels in the dataset, but rather exploits the true target of the analysis to create, in many cases, reconstructing or forecasting-based tasks to obtain predictions. Once a model has been trained, it is applied on a test set and the difference between the input and the output of such model is compared against a predefined threshold to revert back to the anomaly detection task, by classifying each data point as anomalous or not based on the reconstruction or forecasting error.

Finding anomalies in time series data is not a trivial task: as stated in Chapter 1, the way one decides how to tackle the problem depends mainly on the presence or absence of labelled data.

2.3 Supervised Approaches

When an annotated dataset is available, the problem is formulated as a classification task. Given the dataset, and all its features, a model is chosen in order to associate to each data point a label, usually 0 to indicate a "normal" point or 1, in case of an anomaly.

Often, a supervised approach reveals to be quite successful, leading to high performances, but it can still face some challenges. In particular, in many real-life scenarios, the anomalies that can be present in a dataset are way fewer than the normal data points, typically around 2%-5%, therefore a resampling is needed, otherwise any model, having such a discrepancy in the number of examples per class in the dataset, would tend to classify almost every data point as belonging to the majority class. If instead one proceeds with resampling, choosing whether it is better to undersample the majority class or oversample the minority one allows to create a balanced dataset, thus helping the model in performing a fair evaluation of each data point.

There are many different models that can be used, but recently one that has gained increasingly more relevance is eXtreme Gradient Boosting, which is able to reach very satisfactory results in this setting.

2.3.1 eXtreme Gradient Boosting

XGBoost ([5]) combines the predictive power of several base learners, i.e. regression trees, into a single model: at each iteration, a learner is built by taking into consideration the residual error with respect to the prediction made at the previous time-step, paying attention that, while it is important that such error is minimized, the learner should not overfit on the data.

Thus, we can formulate the additive training as such: for each input instance x_i, at a time-step t, the prediction is obtained as $\hat{y}_i^t = \hat{y}_i^{t-1} + f_t(x_i)$, where f_t is the learner built at time-step t; while training, the goal is to minimize the objective function

$$\sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \sum_{k} \Omega(f_k), \tag{2.1}$$

where Ω is the regularization term that penalizes the complexity of the algorithm in order to avoid overfitting, often expressed as $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda||w||^2$. The final prediction over each data point is given by a combination of all the predictions obtained from the K different learners:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \tag{2.2}$$

By minimizing Equation (2.1), we greedily add the tree to the configuration that most improves the model. To quickly optimize the objective, one could leverage second-order approximation, by considering the first and second derivative of the loss with respect to the previous prediction $\hat{y}^{t-1}$, respectively indicated as g_i and h_i. Thus, the new objective function to minimize at each time-step is:

$$\sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) \tag{2.3}$$

XGBoost in this setting will provide us with one of two possible results: it will output a 1 if the data value is to be considered as an anomaly, a 0 otherwise.

2.4 Unsupervised Approaches

As stated before, the absence of annotated data precludes the use of Supervised approaches, which instead rely heavily on the presence of labels. In this setting, one needs to employ Unsupervised methodologies which can work directly with the real target of the analysis, as is in the case of this thesis the energy consumption measured for each building by a smart meter.
Mejri et al.in [6] explain how one can identify mainly five Unsupervised paradigms: clustering-based, density estimation-based, distance-based, forecasting-based and reconstruction-based. The last two methodologies are the ones this thesis investigates about, as we work with Deep Learning models.

2.4.1 Forecasting methods

These approaches work by predicting future states given the previous observations. Typically, in many methods which perform forecasting, the time series given for training are divided into smaller sections by using a sliding window of a certain length and with a certain stride. Such created windows constitute the input for the models, which need to predict a certain number of future steps with respect to the last observation of each window.

To perform anomaly detection, one takes into account the distance between the input and the forecasted time series and if this exceeds, for a specific data point, a predefined threshold, then this is flagged as an anomaly.

Traditional methods are based on autoregression-based models, like AutoRegressive Integrated Moving Average (ARIMA), which have been progressively replaced by Deep Learning models like LSTMs, which are able to model short and long-term temporal dependencies.

Long-Short Term Memory - LSTM

LSTMs were firstly introduced by Hochreiter and Schmidhuber in [7] as an improvement upon RNNs, to solve the problems of vanishing gradient and gradient clipping in sequence-to-sequence (seq2seq) tasks.
An LSTM cell, as the one depicted in Figure 2.4, comprises three gates:

- Forget gate, which implements a sigmoid layer to decide which information to forget in the long-term, by considering the current input, X_t, and the short-term memory represented by the hidden state h_{t-1} passed through the cells of the architecture

$$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f) \tag{2.4}$$

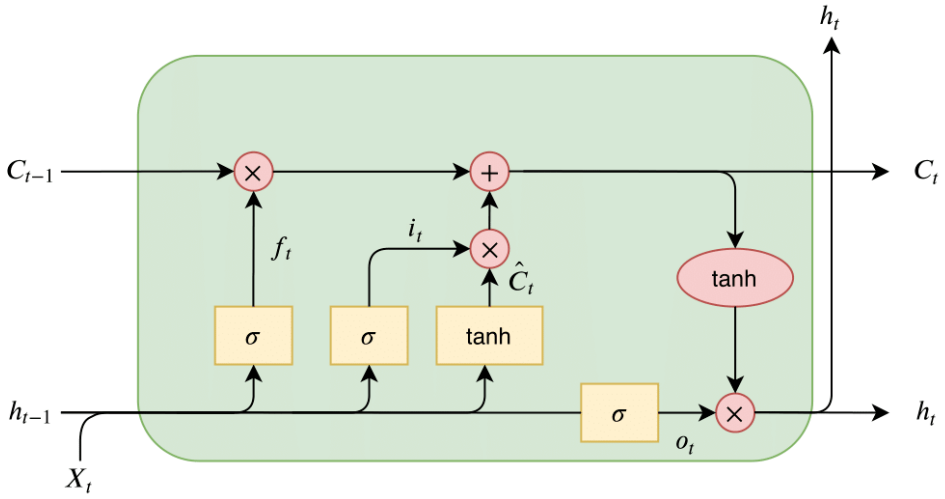


Figure 2.4: Architecture of an LSTM cell

- Input gate, which decides which information to store in the internal cell state C_t: through a sigmoid layer it decides the values to update, while through an hyperbolic tangent it creates a vector of new candidate values to add to the state

$$i_t = \sigma(W_i \cdot [h_{t-1}, X_t] + b_i) \tag{2.5}$$

$$\hat{C}_t = tanh(W_C \cdot [h_{t-1}, X_t] + b_C) \tag{2.6}$$

- Output gate, which decides which parts of the cell state are going to be the output of the cell

$$o_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o) \tag{2.7}$$

The internal cell state, representing the long-term memory, is updated as follows:

$$\mathcal{C}_t = i_t \cdot \hat{\mathcal{C}}_t + f_t \cdot \mathcal{C}_{t-1} \tag{2.8}$$

The final output from the cell, corresponding to the hidden state h_t, which represents the short-term memory, is filtered with the internal cell state as $h_t = o_t \cdot tanh(\mathcal{C}_t)$.

When using an LSTM model to perform anomaly detection, one usually inputs windows of a specific length and predicts the next data point in the sequence: the predictions are then compared to the ground truth next timestamps and if the distance is above a certain threshold for a certain observation, that would be flagged as an anomaly.

Convolutional Neural Networks

Other than LSTMs, also CNNs can be used as a forecasting module in the anomaly detection framework, as proposed by Munir et Al. in [8] (DeepAnT).

CNNs are widely used in Computer Vision and NLP, but have been adapted to handle sequential data, like time series.

A CNN consists of a sequence of layers (convolutional, pooling, fully connected layers). Each convolutional layer firstly performs the convolution operation resulting in linear activations: in the case of images, this implies convolving, sliding, a filter over an image and computing dot products; the convolution of a filter over all spatial locations results into an activation, or feature, map s. A series of filters can be applied, each generating their own activation map. Then on each of them a non-linear activation function is often applied. The convolution is defined as follows:

$$s(t) = (x * w)(t) \tag{2.9}$$

where s is a weighted average of the function $x(\tau)$, the input, at the timestamp t, and $w(-\tau)$ is the weight, the kernel.

A one dimensional convolution is defined as:

$$s(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)w(t-\tau) \tag{2.10}$$

The output of a convolutional layer is then passed through a pooling layer, where a pooling function summarizes the output at a certain location based on its neighbors. A typically used pooling function is the **max-pooling** operation, which outputs the maximum activation in a defined neighborhood. Such function is applied on all the feature maps, separately.

Fully connected layers are characterized by neurons such that each is connected to all the other neurons from a following layer.

Munir et Al. propose the architecture shown in Figure 2.5 based on a CNN to perform anomaly detection:

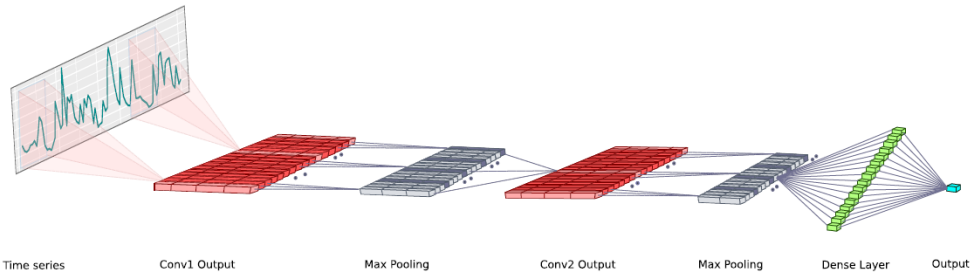


Figure 2.5: DeepAnT architecture, as described in [8]

Here there are two convolutional layers, each followed by a max-pooling layer. The input layer has w input nodes, where w is the window size: in fact, to use a CNN for forecasting, one needs to prepare the data in a way which is similar to that needed for the LSTM. The time series is, thus, divided into overlapping windows, and labels are obtained by considering the next value in the sequence for each window. Then, the so called forecasting horizon is defined, meaning the number of timestamps that need to be predicted. The final part of the network consists of a fully connected layer, whose output is the network prediction for the next timestamp.

Then, to perform anomaly detection one needs to compute in some way the distance between the output of the network and the labels: in the case of DeepAnT, the Euclidean distance is used as a measure of discrepancy, as the anomaly score. Finally, by using a threshold, one can categorize each data point as anomaly or not-an-anomaly.

2.4.2 Reconstruction methods

When one needs to deal with time series consisting of a large number of data points, forecasting-based approaches like LSTMs may reveal themselves to be ineffective: this can happen because of rapidly and continuously changing time series, which make it harder to perform accurate predictions, and therefore anomaly detection. An alternative approach would be that of using Autoencoders, to perform a reconstruction of an original sequence once encoded in a latent space.

Specifically, an Autoencoder, like the one depicted in Figure 2.6, is made of two parts, an encoder and a decoder. The input of an encoder is a sequence of values obtained by applying a sliding window over the training dataset: such structure will try to capture the most relevant aspects of the input and represent it in a lower-dimensional space. The decoder, instead, takes as input the output of the encoder, and does the opposite operation, meaning it tries to reconstruct the original sequence by starting from a latent representation of it. The Autoencoder is trained in order to minimize the loss between the reconstruction and the ground truth, as the goal would be that of building a model which, ideally, would be able to output a sequence resembling as much as possible the core characteristics of the input of the encoder.

This kind of models can be employed in anomaly detection, because traditionally one assumes that the reconstruction error associated with anomalies is much higher than the one associated with the "normal" data points, as the trained Autoencoder should learn the latent space of the normal samples. There are different ways to compute the reconstruction error, the most used being the Mean Squared Error: given the input $X = \{x_i\}, i = 1...N$ and the reconstruction $\hat{X} = \{\hat{x}_i\}, i = 1...N$

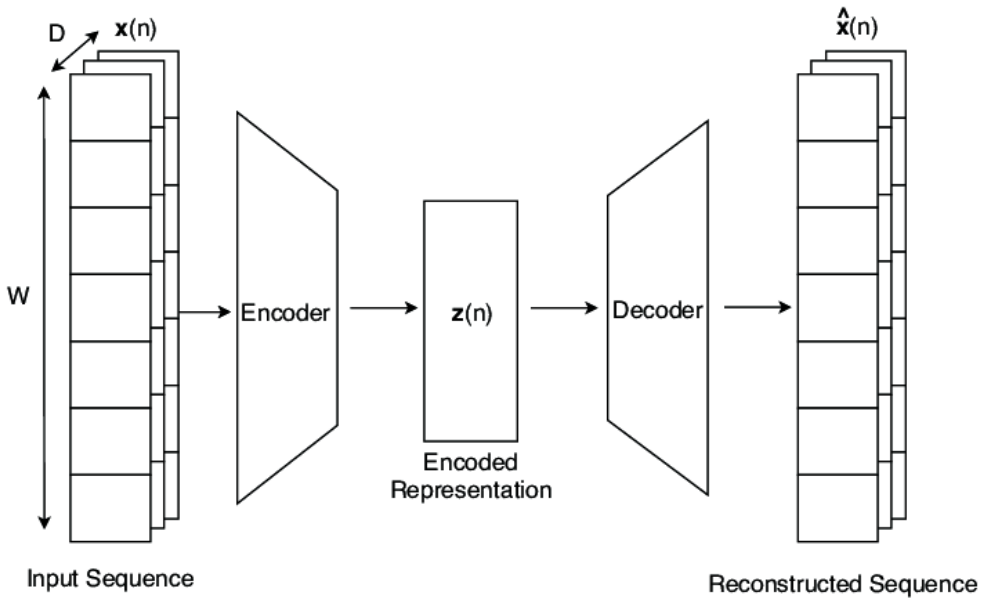


Figure 2.6: Autoencoder architecture

the reconstruction error is computed as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}|x_i - \hat{x}_i|^2 \tag{2.11}$$

Then, by defining a specific threshold, it should be easy to detect anomalies.

Autoencoders for anomaly detection can be built in different ways, employing either linear or convolutional or LSTM layers, like the one proposed by Provotar et Al. in [9].

Hybrid Methodologies

It is not uncommon to simultaneously leverage the advantages of forecasting and reconstruction-based models.

Kardi et Al. in [10] propose an approach which employs a traditional LSTM network to predict the next hour sample in the input sequence, paired up with an LSTM Autoencoder which is useful to learn the features of normal consumption. It leverages the Exponential Moving Average as a threshold and identifies both local anomalies, i.e. when a single metric crosses the threshold, and global ones, i.e. when the mean loss of all metrics crosses such threshold.

Jiang et Al., in [11], propose to combine several Deep Learning methods to perform anomaly detection, by combining CNN, Bidirectional LSTM (Bi-LSTM) and attention mechanisms. In particular, CNNs are used as they can extract higher-order features from the input data; Bi-LSTM can acquire contextual information on the time series by combining it in forward and backward directions; an attention mechanism is used to assign different weights to the hidden units of a neural network, in order to make a hidden layer focus more on a specific portion of information in the data. Anomaly detection is then performed by setting a threshold value which is 3σ above the predicted value, where σ is the standard deviation of electricity consumption on the day before the actual one.

Yang et Al., in [12], propose an anomaly detection framework in which the time series is treated as a signal, and Fourier transforms are applied to obtain a decomposition in trend, seasonal and residual series, based on different frequency components. Each component is then handled with a different method:

- Trend Series: since it contains the most important components in the original time series, it should be predicted as accurately as possible, employing an LSTM network to perform predictions

- Seasonal Series: as it contains more components with different frequencies, and its waveform changes rapidly, to accurately perform a prediction Yang et Al. decided to implement a model based on CNN and an attention mechanism. In particular, the model contains an attention layer, a CNN to capture features of the weighted component map resulting from the previous layer, and finally an LSTM layer which outputs the final prediction

- Residual Series: given its chaotic pattern, a reconstruction-based method is chosen based on a Gaussian distribution

In the end, the prediction or reconstruction of each component is then summed to obtain the overall reconstruction of the input series, then a reconstruction error is computed and compared to a threshold to determine which data points can be considered anomalous.

2.4.3 Generative methods

Generative models aim at generating new samples from the same distribution of the training data. In particular, they aim at learning a model probability function that approximates the one of the data, and use such learned distribution to sample new data points.

Generative modeling is often formulated as a density estimation problem:

- Explicit density estimation, by explicitly defining and solving for the model probability distribution; this is the case of Variational Autoencoders
- Implicit density estimation, by learning a model that can sample from a probability distribution without explicitly defining it; this is the case of Generative Adversarial Networks

Generative Adversarial Networks

GANs, introduced by Goodfellow et Al. in [13], are a type of generative models that consists of a generator G and a discriminator D. The former is typically a Neural Network that takes as input a noise vector and it is trained to learn the probability distribution of the training data and generate samples following it. The latter is a Neural Network which needs to classify whether the samples from G are fake or real; its goal is that of understanding the underlying probability distribution of the real data and distinguish it from the samples generated by G. The training of such model falls within the framework of adversarial learning, and it is performed until the discriminator is fooled about half of the time.

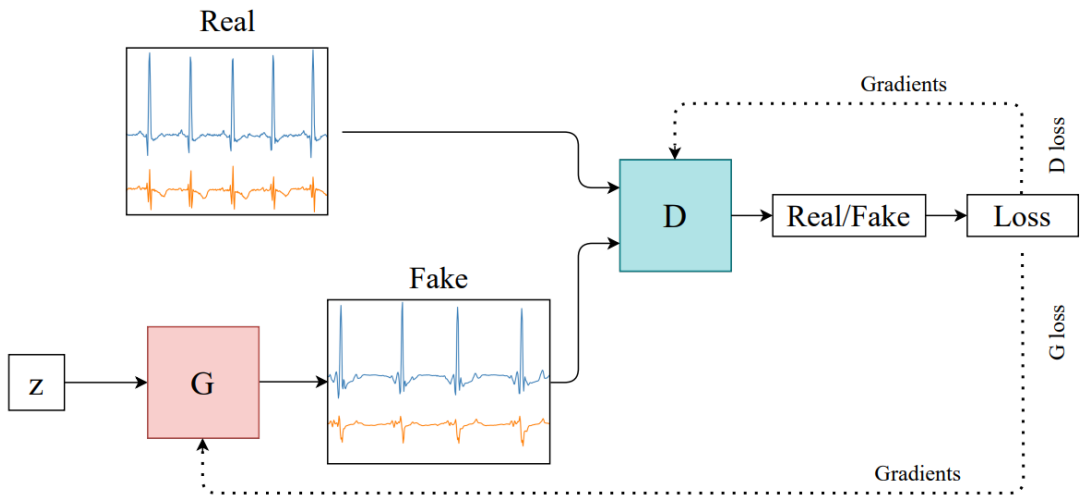


Figure 2.7: Generative Adversarial Network applied on time series data, as shown by [14]

In particular, given the generator's distribution $p_g(x)$, we define a prior on input noise variables $p_z(z)$; $G(z;\theta_g)$ is the mapping with parameters θ_g to the data space. The discriminator outputs a scalar $D(x;\theta_d)$, which indicates the probability that the data point x came from the data, rather than was generated by G. The discriminator is trained as to maximize the probability of assigning the correct label to real and fake (generated) samples, while at the same time the generator is

trained to actually fool the discriminator, by minimizing the difference between the generated samples and the real training data.
This could be seen as a two-player min-max game with the following value function:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \quad (2.12)$$

[13] proposes the following minibatch SGD (Stochastic Gradient Descent) for GANs:

Algorithm 1 GAN training. The number of steps k to apply to the discriminator is a hyperparameter

for *n_iterations* **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

GANs can be used for anomaly detection because if they are trained on a dataset containing exclusively normal samples, it can identify anomalies as those data points that the generator poorly reconstructs. In particular, as stated in [15], the objective is to master a representation of normal data and determine data instances that differ from it. The network is trained as to generate a feature extractor, often the generator, which extracts a representation of normal data sample used to reconstruct it: the reconstruction error is going to be used to signal the existence of anomalies. In alternative, the output of the discriminator can be used as anomaly score: in fact, the sequences it labels as fake with high confidence are likely to be anomalous.

It is not common to find GAN-based networks applied to time series data, due to the complex temporal correlations which pose significant challenges to generative modeling.

Geiger et Al. in [16] propose an unsupervised AD approach built on GANs, known as TadGAN, whose architecture is shown in Figure 2.8. The model comprises two

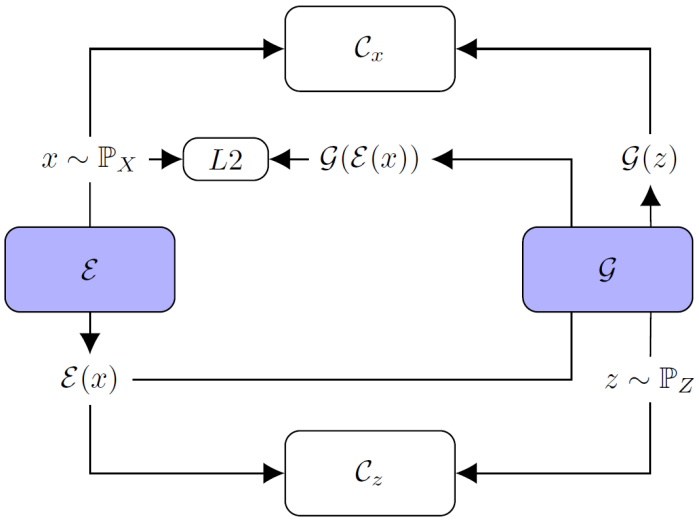


Figure 2.8: TadGAN architecture

generators: $\mathcal{E}$, which acts as an Encoder that maps the time series sequences into the latent space; $\mathcal{G}$, which serves as a Decoder outputting a reconstruction for the input time series. Additionally, there are two adversarial *Critics*, acting as discriminators. C_x aims at distinguishing between the real time series sequences from X and the ones generated by $\mathcal{G}(z)$; C_z measures the goodness of the mapping into the latent space, by distinguishing between random latent samples $z \sim \mathbb{P}_Z$ and the encoded ones.
The reconstruction errors and the *Critic* outputs $C_x(x)$ are then used to compute the anomaly score for each data point in the dataset, by taking into consideration their z-scores. Finally, a threshold is employed to categorize the samples as anomalies or normal points.

Zhang et Al. in [17] propose a Self-Training based Anomaly Detection with GAN model called STAD-GAN. The model is based off a generator-discriminator structure for adversarial learning, where the former aims to capture the normal data distribution, while the latter aims at amplifying the reconstruction error of abnormal data to recognize them. Such architecture is optimized with a self-training teacher-student framework, where a teacher model generates pseudo-labels which are going to be used to obtain a refined dataset to train the student model.

The input dataset contains time series which are sequenced by applying a sliding

window. Such sequences are passed as input of the generator, which consists of an Encoder-Decoder-Encoder Network, as depicted in Figure 2.9. The two

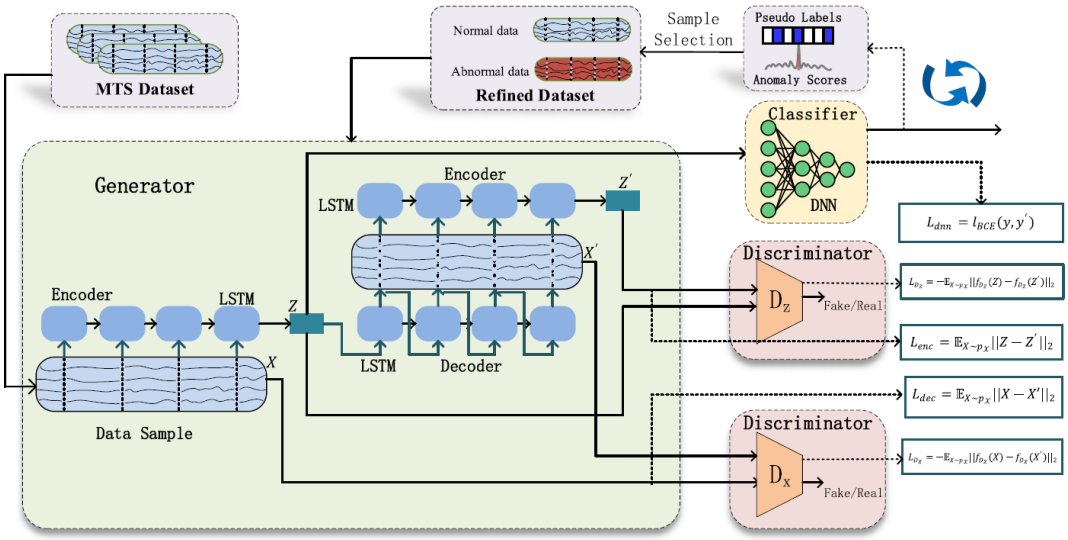


Figure 2.9: STAD-GAN architecture

encoders present the same structure, without sharing the model parameters. The second encoder works in order to minimize the difference between the two latent representations, $Z = G_{E_1}(X)$ and $Z' = G_{E_2}(D(z))$: if X and X' are different, their latent representations will be far away from those of normal samples.

The architecture comprises two discriminators, D_Z and D_X. The former aims at distinguishing as much as possible between the two latent representations, while the generator aims to make them indistinguishable, by learning to generate latent representations which capture normal patters. The latter tries to separate the inputs of the two encoders, X (original dataset) and $X' = D(G_{E_1}(X))$ (output of the decoder), to distinguish between the generated data and the original one, by amplifying the reconstruction error of abnormal data points, to make them more separable from normal patterns, while the generator aims at making them indistinguishable to D_X.

Note that, the latent representation Z embeds the feature information of the original time series and is then fed to a DNN classifier, which classifies each data point as anomalous or not.

The GAN training is embedded within a self-training process, characterized by two separate models, teacher and student. The teacher works in a completely unsupervised manner: the output of the DNN classifier is treated as pseudo-labels

and an anomaly score is computed. Based on the latter, a refined dataset is created, containing a certain percentage of the top normal samples (those that are associated to the lowest anomaly scores) and of the top abnormal one (those associated, instead, to the highest anomaly scores). This will constitute the input dataset for training a student model, which aim is that of minimizing the classification loss on the pseudo-labels, which will constitute the ground truth labels in this case. Then the student model becomes the teacher model and the training procedure iterates until the pseudo-labels no longer change, or if a number of maximum iterations have been made.

By combining self-training with adversarial learning to train the teacher-student model, the feature representation in the latent space becomes more discriminative.

Audibert et Al. propose in [18] a method whose learning is inspired by GANs, by implementing an adversarial training of an encoder-decoder architecture to learn how to amplify the reconstruction error in correspondence of anomalous data points. The complete anomaly detection procedure is depicted in Figure 2.10.

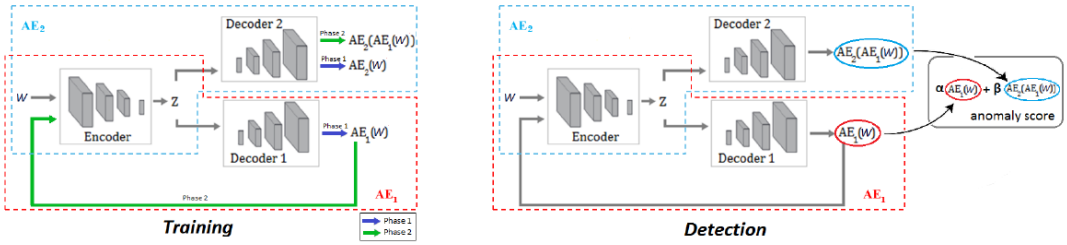


Figure 2.10: UnSupervised Anomaly Detection: training and detection phases

The training procedure involves an encoder E and two decoders, D_1 and D_2, which can be also viewed as two autoencoders AE_1 and AE_2, sharing the same encoder.

First of all, the two AE are trained in order to learn a reconstruction of the input time series. Subsequently, they are trained in an adversarial way: AE_2 aims at distinguishing the real data from the reconstruction outputted by AE_1, while this is trained in order to fool the second Autoencoder.

Given an input sequence W, AE_1 wants to minimize the difference between such input and the output of AE_2, while AE_2 wants to maximize this difference. The training objective can be formulated as:

$$\min_{AE_1} \max_{AE_2} ||W - AE_2(AE_1(W))||_2 \tag{2.13}$$

In particular, the weights of the encoder-decoder architecture are updated according

to the following two losses:

$$\mathcal{L}_{AE_1} \leftarrow \frac{1}{n}||W - AE_1(W)||_2 + (1 - \frac{1}{n})||W - AE_2(AE_1(W))||_2 \tag{2.14}$$

$$\mathcal{L}_{AE_2} \leftarrow \frac{1}{n}||W - AE_2(W)||_2 - (1 - \frac{1}{n})||W - AE_2(AE_1(W))||_2 \tag{2.15}$$

At inference time, instead, a test dataset is passed as input to the network and the output is an anomaly score, which can be defined as a weighted sum of the reconstruction error performed by the first Autoencoder and the one of the second Autoencoder, but considering as reconstruction the one that started from the output of AE_1. The anomaly score is thus defined as:

$$\mathcal{A}(\hat{W}) = \alpha||\hat{W} - AE_1(\hat{W})||_2 + \beta||\hat{W} - AE_2(AE_1(\hat{W}))||_2 \tag{2.16}$$

where $\alpha + \beta = 1$, which are used to parametrize the trade-off between false positives (FP) and true positives (TP). $\alpha < \beta$ is a high detection sensitivity scenario, as the number of TPs is increased, as well as that of FPs; instead, $\alpha > \beta$ is a low detection sensitivity scenario, where the number of TPs is reduced as well as that of FPs.

Variational Autoencoders

A Variational Autoencoder (VAE), introduced in [19], is a probabilistic graph model which combines Autoencoders with variational inference to learn complex data distributions.

Figure 2.11 depicts an example of the VAE architecture, by also proposing a comparison with a standard Autoencoder.

The main difference with respect to autoencoders is that a VAE does not encode the input as single points, but rather as a probability distribution over the latent space. The probabilistic encoder comprises two separate Fully Connected layers through which passes what in a simple AE would be considered as the latent vector: one of these two layers returns the mean μ of the latent probability distribution, while the other returns the logarithmic variance $log(\sigma^2)$. These two quantities are used to perform the reparameterization trick, useful to ensure seamless propagation of gradients: the input for the probabilistic decoder is computed as per following equation

$$z = \mu + \sigma * \epsilon \tag{2.17}$$

where ϵ corresponds to random noise, extracted from a Standard Gaussian.

More in details, consider having a dataset $X = \{x^{(i)}\}_{i=1}^{N}$, which is generated from the prior distribution of $z \sim p_{\theta^*}(z)$, by exploiting the conditional distribution

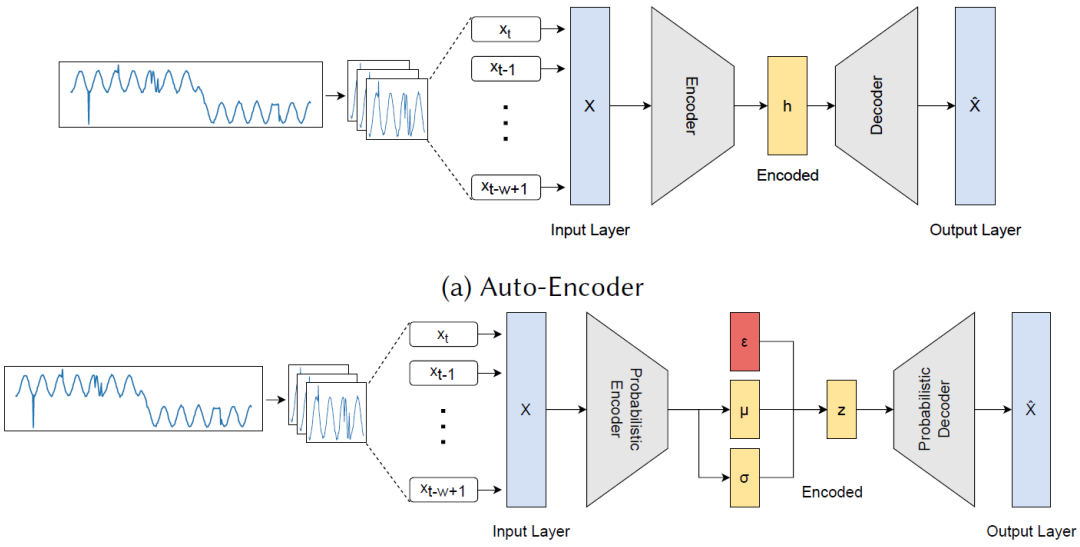


Figure 2.11: Comparison of AE and VAE architecture

$p_{\theta^*}(x|z)$. To estimate the true parameters θ^*, one chooses a Gaussian prior and represents the conditional distribution with a neural network; the model should be trained in order to maximize the likelihood of the training data, according to the following equation:

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz \tag{2.18}$$

Here is where the problem becomes intractable, as it is not possible to compute $p_\theta(x|z)$ for every z, and the posterior density $p_\theta(z|x)$ is intractable as well. Therefore, the goal becomes to learn a distribution $q_\theta(x|z)$, which approximates the true posterior: this allows us to derive a lower bound on the data likelihood which is indeed tractable and can be optimized.

Maximizing the likelihood lower bound implies maximizing the following equation:

$$\mathbb{E}_z[logp_\theta(x^{(i)}|z)] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) \tag{2.19}$$

The second term corresponds to the Kullback-Leibler divergence, and computes the divergence between the prior, Standard Gaussian distribution, and the approximate posterior, which is based on the mean and standard deviation obtained by passing the direct output of the encoder through the two FC layers above mentioned. This acts as a regularization loss, which penalizes the posterior for being too far from

the prior. It is often computed per the following formula:

$$\frac{1}{2}\sum_{j=1}^{N}(1 + log(\sigma_j^2) - \mu_j^2 - \sigma_j^2) \tag{2.20}$$

The first term, instead, corresponds to the reconstruction error. Based on the type of data one deals with, $p_\theta(x|z)$ can be either a multivariate Bernoulli, or a multivariate Gaussian.

VAEs can prove themselves to be useful within the anomaly detection framework because they are able to identify anomalies thanks to their ability to learn the underlying distribution of the data and reconstruct the input samples.

2.5 Anomaly Detection in Energy Consumption

The methodologies described up until now have been used in different fields, with some approaches tested on different kinds of data.

[11] deals with energy consumption data, by implementing a Deep Learning architecture, based on the usage of Convolutional, bidirectional-LSTM layers and the attention mechanism.

Also [10] proposes an architecture to perform Anomaly Detection on electricity consumption data. In particular, an LSTM is used to predict the next hour sample, while an LSTM Autoencoder is then used to learn the features of normal consumption. Aside from electricity measurements, the models were also fed other features, like time-related features, indications regarding the weather, and the average consumption over periods of time.

Several approaches have been investigated within this setting, some encompassing Deep Learning and Machine Learning approaches, others using statistical methods. A couple of examples are proposed, just to give an idea of how the task has been tackled for energy consumptions in literature.

Hollingworth et Al. in [20] propose a methodology to combine the predictive power of ARIMA and LSTMs. ARIMA is a statistical method which consists in fitting a linear equation on stationary data. It can also be applied on non-stationary data, after differencing the time series a certain amount of times in order to obtain its stationarity. In this work, the authors propose a framework in which the data is firstly passed through an ARIMA model, then through the LSTM in order to minimize erroneal detections of anomalies and increase the effectiveness of the task.

Zhang et Al. in [21] propose an architecture based on Graph Neural Networks (GNN), which model multivariate time series by considering each feature as a node in the graph. In particular, in this work they combine two versions of GNN: Graph Convolution Networks (GCN) and Graph Attention Networks (GAT). A GCN is characterized by a graph convolution layer which uses information of the edges to aggregate nodes information, thus creating a new node representation: it models the topological relationships between the features characterizing the time series and extracts the correlations among them. The GAT is used to give priorities to more important relationships between such features. In this setting, a multivariate time series is considered as a complete graph and the GAT learns the interrelationships between the different features of the time series, thus allowing an improved anomaly detection over energy consumption data.

Lastly, Fahim et Al. in [22] propose a Machine Learning approach to the task, by employing a Support Vector Regression model to build an energy consumption profile; then, residuals are computed during the anomaly detection phase in order to find variations between the actual and predicted values of energy consumption.

Chapter 3

Methodology

3.1 Problem Setting

As stated in Chapter 1, the objective of this thesis is to investigate the feasibility of employing Deep Learning Unsupervised approaches for the task of anomaly detection when the training dataset contains anomalies.

Traditionally, in fact, such models are trained on normal data, reflecting the normal evolution over time of the energy consumption, and then they are tested on a dataset which contains anomalies. In such context, given that the models have learned the patterns which correspond to normality, they would not be able to reproduce anomalies at inference time, thus enabling an easy identification of them.

In many cases the dataset used to test models on the task of anomaly detection is divided in a "normal" training set and a test set which contains anomalies, which account usually for $2 - 5\%$ of the dataset. In our specific case, this is not true: DATA CELLAR will provide data from the energy consumption of the LECs which not only is not annotated with labels, but also potentially contains anomalies. Therefore, if one wants to train models on the provided data, it is fundamental to take into account the presence of anomalies in the training set.

One could argue, though, that even in absence of annotations, it could still be possible to identify anomalies in the dataset and remove them from the training set. Statistical methods like the Inter-Quartile Range are useful to identify outliers in the data distribution. In particular, mild outliers can be identified as those points which belong outside of the following interval:

$$[Q_1 - 1.5 \cdot (Q_3 - Q_1), Q_3 + 1.5 \cdot (Q_3 - Q_1)] \tag{3.1}$$

where Q_1 is the first quartile, so the value corresponding to the 25th percentile of the data and Q_3 the third quartile, corresponding to the 75th percentile.
Extreme outliers are the points belonging outside of the following interval:

$$[Q_1 - 3 \cdot (Q_3 - Q_1), Q_3 + 3 \cdot (Q_3 - Q_1)] \tag{3.2}$$

One could identify outliers in the dataset thanks to this method and apply an appropriate strategy to make the time series represent the normal energy consumption. The problem is, though, that this would tackle only one type of anomaly, but datasets often comprise also contextual and collective ones, which would still be left in the time series data by such method, thus enabling a model to eventually only improve in correspondence of point anomalies.

Therefore, we are going to leave all the anomalies in the data and work with anomalous data points also when training the model.

3.2 Dataset

The dataset used to train, validate and test the models is the one presented in the LEAD - Large-scale Energy Anomaly Detection Kaggle competition [1]. It was extracted from the dataset used for the Great Energy Predictor III competition hosted by the ASHRAE organization and presented in [23].

It comprises electricity measurements taken by smart meters present in 200 buildings, which collect data with an hourly frequency for the whole year of 2016. Each data point is further enriched with information from 57 features: some of them refer to the building characteristics, some to atmospheric and meteorological conditions and others are generated features, as either aggregation of features already present in the dataset or particular encodings of those. For the purpose of this thesis it was decided to keep only a few of them, excluding in particular the generated ones, as they could be always re-generated, in such a way as to render the code generalisable, to be able to use it with few modifications on the data that DATA CELLAR is going to provide.
Among the most interesting features to retain for the task we have:

- *building_id*: a unique identification number for each building present in the dataset
- *timestamp*: the temporal indication of the energy measurement collected
- *meter_reading*: the energy consumption collected by the smart meters, which constitute the real target of our analysis
- *anomaly*: this column contains the labels which characterize each point as an anomaly (1) or not (0). Note that, given that the entire work takes place in an Unsupervised setting and given the premises stated in the previous paragraph, the information represented by this column is not used in any way during the development, training and testing of each model, but exclusively to assess the performances in terms of the usual metrics of Precision, Recall, F1 and ROC-AUC score
- *site_id*: a unique identifier for the city area in which the building is sited
- *primary_use*: an indication of the type of building, whether it is a school, an office, a church, a residential building, ...
- *square_feet*

[1] https://www.kaggle.com/competitions/energy-anomaly-detection

- *year_built*: the year in which the building was built
- *floor_count*: an indication of how many floors are contained in the building
- *air_temperature*: in Celsius degrees
- *cloud_coverage*: fraction of the sky obscured by clouds on average when observed from a particular location
- *dew_temperature*: temperature at which the water vapor in air at constant barometric pressure condenses into liquid water at the same rate at which it evaporates
- *precip_depth_1_hour*: an indication of at what depth liquid precipitation would cover a horizontal surface in an observation period if nothing could drain, evaporate or percolate from this surface
- *sea_level_pressure*
- *wind_direction*
- *wind_speed*
- *is_holiday*: an indication of whether the day on which a measurement was taken was a holiday or not

When dealing with time series it is often useful to perform feature engineering and enrich the dataset with temporal and energy-related information. Typically, the following features are particularly interesting:

- lag features: each data point is enriched with information regarding energy consumptions which correspond to previous or successive timestamps with respect to the current one. It could be useful to consider for each data point the consumption relative to 1, 24, 168 hours prior and following
- differencing features: for each data point compute the difference between its energy consumption measurement and a lagged one
- rolling features: always for the energy consumption, consider sliding a window over the values and compute statistics for each of them, like the standard deviation, the mean and median
- temporal features: extract temporal information from the timestamp, like the month, the day and the weekday. The year is not relevant as all measurement belong to 2016

- trigonometric features: apply sinusoidal functions over time-related features in order to obtain a continuous representation of time

All the feature engineering operations are performed after imputing the missing values, which are present exclusively in the *meter_reading* column, and the missing timestamps.

For what concerns the former, a custom made technique has been employed. The basic idea is that given a missing measurement, one can take the first previous and following values and average them. If this happens at the beginning of the year, there is no previous value that can be taken for the imputation, therefore in this case the average value over the consumption of a specific building is taken instead of the previous value. Similarly if the missing data point is at the end of the year.

Note that if there are multiple sequential missing values, after the imputation of the first one in the sequence, for the others the previous imputed values are used to compute the average, so as to continue the trend.

A limitation of the imputation of missing consumptions is linked to the fact that there are some buildings which completely lack of measurements for a considerable amount of time, in some cases even five months consecutively: this implies that the imputation will result into a straight line near the average value of the series.

Another important step is making sure that all the time series present in the dataset have the same length, so $24 * 366 = 8784$ measurements, considering that the smart meters collect data every hour and that 2016 is a leap year. The time series are made evenly-timestamped and a simple forward fill strategy is used to impute the corresponding missing values.

As stated before, the measurements present in the dataset are taken from 200 different buildings, each belonging to a specific category. Table 3.1 shows the 12 primary uses that the dataset contains and the corresponding number of buildings for each of them.

Primary Use	*Number of buildings*
Education	80
Office	62
Entertainment/public assembly	23
Lodging/residential	13
Public services	7
Healthcare	5
Services	3
Manufacturing/industrial	2
Parking	2
Food sales and service	1
Religious worship	1
Other	1

Table 3.1: Distribution of buildings per primary use

It is evident to notice how the majority of buildings are related to education (schools, universities) and offices. It becomes interesting to visually inspect the time series present in the dataset in order to understand if the fact that a certain building belongs to a specific category of use implies that all those which belong to the same one have similar characteristics.

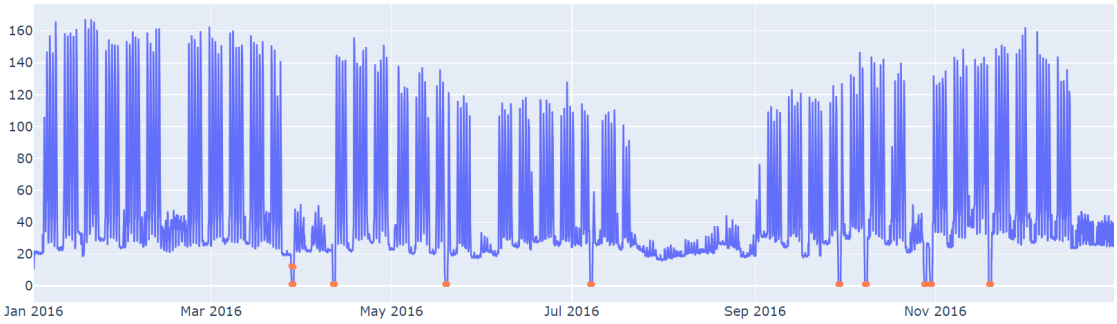
Figure 3.1 shows two examples of buildings belonging to the "Education" category.

The top subfigure, (3.1a) shows an evident weekly pattern for most of the year, which is understandable given the nature of the building: in fact, typically, a school or a university is open during the first five days of the week and closed during the weekend, so it is not a surprise to notice higher consumptions within the week with respect to Saturdays and Sundays. Moreover, there are some cases in which the measurements taken from the smart meters have a very low value, in correspondence of holidays, like Easter and Christmas, or of the summer break.

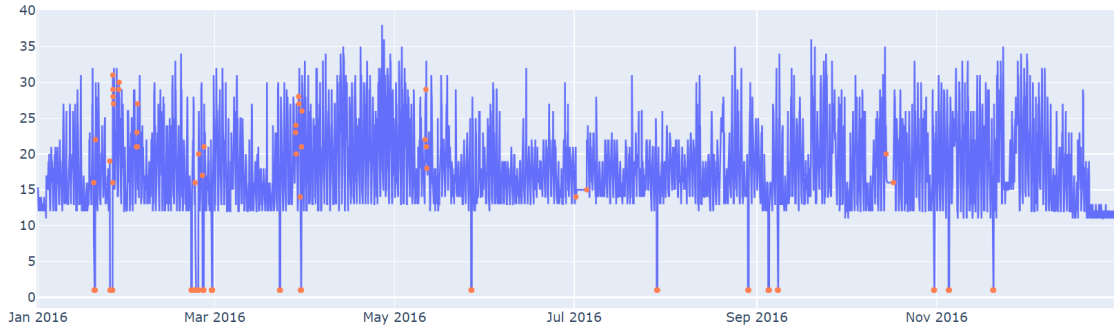
The bottom subfigure, (3.1b), instead, shows a completely different pattern in the evolution of the energy consumption throughout the year, as no evident weekly seasonality is present, which would give even a visual indication of the type of building it refers to.

This is particularly interesting to notice and to take into account, as one would expect Deep Learning models to behave similarly within a specific category, in terms of performances, as they would be able to recognize typical patterns. In reality this is not always the case, as not all buildings of the same kind express the same patterns in terms of the energy consumption, thus leading potentially to

significant differences in the performances of the models for the anomaly detection task.



(a) Educational building (id: 730), with weekly seasonal pattern



(b) Educational building (id: 925), without any evident pattern

Figure 3.1: Comparison of two time series representing the energy consumption of two different educational buildings. The red dots represent the anomalies as annotated in the dataset.

As one typically does for training any model, the dataset needs to be split in train, validation and test set. In this case, we proceeded firstly to obtain the training data and testing data, by leveraging the *building_id*. In particular, 3.3 explains how the division has been made:

$$\begin{cases} train - val \leftarrow \text{building\_id mod } 5 < 4 \\ test \leftarrow \text{building\_id } \mod 5 == 4 \end{cases} \tag{3.3}$$

Then, in order to obtain training and validation set, *train-val* was divided by considering putting 80% of the data in the training set and the remaining in the validation one: in particular, in order to make sure that no building had part of its measurements in one set and part in another, the $80 - 20$ split was performed based on the *building_id* unique values. This resulted in using 129 buildings for training the models, 33 for validating and 38 for testing.

Finally, the data has been prepared in a similar way for all the methods chosen for the task (here presented in the following section).

Considering separately the energy consumption measurements for each building, a sliding window has been applied over the time series: the set of windows obtained as such is going to constitute the input of each model. Such architectures are going to be firstly trained on the training set, and validated, then the test set is going to be used to assess its performances for the task of anomaly detection. In the next section, I am going to go more into depth about the entire process.

3.3 Methods

The following Figure 3.2 represents the basic pipeline followed in order to train the different models to perform anomaly detection.

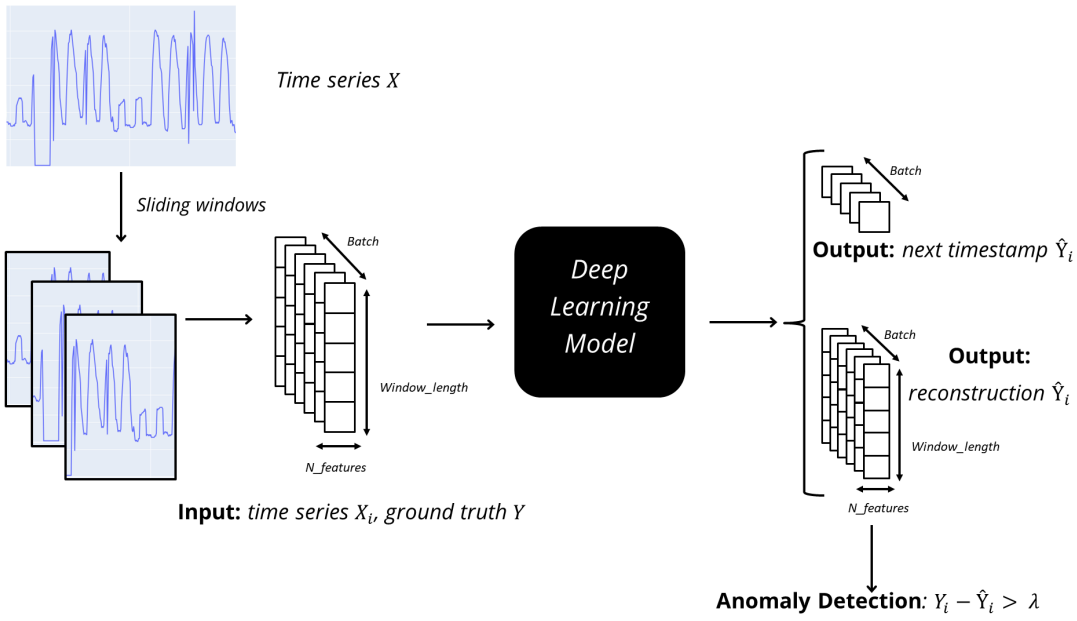


Figure 3.2: Workflow representation

The input for each model chosen is a sequence of sliding windows over the time series of each building. At training time, training and validation set are employed: in both cases, the corresponding loss used is the Mean-Squared Error: given the input ground truth $Y = \{y_i\}, i = 1...N$ and the output of each model $\hat{Y} = \{\hat{y}_i\}, i = 1...N$, either in the form of a sequence of predicted timestamps or of a sequence of reconstructions, the loss is computed as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|^2 \tag{3.4}$$

In alternative, one could choose to use the Mean Absolute Error (MAE):

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{3.5}$$

At inference time, instead, the test set is passed to the models. The resulting output is then compared with a ground truth, that differs based on the chosen model, and their difference is subject to the anomaly detection process: often, if

this is bigger than a predefined threshold for a specific data point, then this will be labelled as anomalous.

Chapter 2 presented an overview of the main methodologies that can be used for performing anomaly detection in the Unsupervised setting. In particular, three main possibilities were pointed out: forecasting, reconstruction and generative methods. In this thesis we investigate all these three approaches, both in a univariate setting and in a multivariate one. In the former case, exclusively the real target of our analysis, the energy consumption value, was taken into consideration. In the latter, almost 20 features were used: some referred to the environmental conditions, like the air temperature and the sea level pressure; some were temporal informations regarding the weekday, the hour and the month of the measurements, encoded in a continuous form; also lagged features were taken into consideration, alongside with their difference with respect to the target; finally, the residual component of each time series was used as additional information that could be useful for the model to have.

3.3.1 Forecasting method

For what concerns the forecasting approach, LSTMs were used in the following way: after having obtained the windows from the time series, each of them was passed through the LSTM network in order to make it predict a value corresponding to the following timestamp.

Architecture	***Input Dimension***	***Output Dimension***
LSTM(n_features, 32)	(128, 168, n_features)	(128, 32)
ReLU	(128, 32)	(128, 32)
Dropout(rate = 0.2)	(128, 32)	(128, 32)
Linear(32, n_features)	(128, 32)	(128, n_features)

Table 3.2: Architecture of the LSTM model with indications regarding the dimensions of the inputs and outputs of each layer.

The LSTM network employed consists of a single layer and 32 cells. Given an input sequence in the form $(batch\_size, window\_lenght, num\_features)$, where *batch_size* was chosen as 128 for all experiments and *window_length* as 168 for forecasting, i.e. the equivalent of one week measurements, the LSTM layer would produce an output of size $(batch\_size, 32)$. This would be then passed through a Fully Connected layer, to reduce the dimensionality from 32 to the number of features of the input sequence.

As ground truth, the values corresponding to the next timestamp for each feature were taken into consideration: the forecast made by the model was then compared with this ground truth and the resulting forecasting error was used to determine whether a point was to be considered as anomalous or not, based on a predefined threshold.

3.3.2 Reconstruction method

In this setting the Autoencoder was taken into consideration to tackle the anomaly detection task from another point of view.

When using this kind of model, the goal is that of obtaining a reconstruction of the input sequence, which consists of a set of windows sliced from the time series. Theoretically, an Autoencoder is able to learn the most important characteristics of the input, thus enabling a reconstruction which should not be accurate in presence of anomalies, as they are deemed as non-fundamental features of the input.

The training of this model is based on the reconstruction error itself, so the difference between the input sequence and the output of the decoder. This same error can also be used at inference time as an anomaly score: given the nature of the model, the score should be higher in presence of anomalies, thus enabling to easily identify them thanks to the definition of a threshold.
Three different architectures were chosen for the autoencoder:

1. LSTM
2. Convolutional
3. Linear

The input is in the form $(batch\_size, window\_lenght, num\_features)$ for all the three types of Autoencoders. *batch_size* was chosen as 128, as stated before, and *window_length* as 72, i.e. the equivalent of half-a-week measurements.

Autoencoders are typically built on LSTM layers, if they have to be applied to perform reconstruction of time series. The main reason is that, given the temporal nature of this kind of data, LSTMs are able to not only take into account the short-term dependencies between the data points, but also the long-term ones. Thus, they are able to perform a reconstruction which not only takes into account the measurements that were collected immediately before the one that is currently being considered, which have in general a big impact in determining the evolution

of the energy consumption, but also those that were collected much before and that could still be useful especially in presence of patterns within the sequence.

In this case, the Encoder is characterized by a single LSTM layer with 32 cells; as output, the last hidden state is taken into consideration and passed to the Decoder, which comprises always a single LSTM layer with 32 cells, and a Fully Connected layer which reduces the dimensionality of the sequence to the number of features of the input.

Architecture	***Input Dimension***	***Output Dimension***
ENCODER		
LSTM(n_features, 32)	(128, 72, n_features)	(128, 32)
Dropout(rate = 0.2)	(128, 32)	(128, 32)
DECODER		
Repeat Vector	(128, 32)	(128, 72, 32)
LSTM(32, 32)	(128, 72, 32)	(128, 72, 32)
Dropout(rate = 0.2)	(128, 72, 32)	(128, 72, 32)
Linear(32, n_features)	(128, 72, 32)	(128, 72, n_features)

Table 3.3: Architecture of the LSTM Autoencoder model with indications regarding the dimensions of the inputs and outputs of each layer. Note that for the multivariate experiments, we asked the model to both output a single feature and a number of features equal to the input one

An alternative could be that of building Autoencoders with Convolutional layers, which are able to consider the local patterns characterizing the input sequence, without handling explicitly the temporal information present in them.

The Encoder is here characterized by three convolutional layers, which apply half of the filters of the previous layer, starting from 32. The Decoder is built using three transposed convolutional layers, which apply a number of filters in the opposite order, to be able to obtain an output with the same dimensionality as the original input.

Finally, one could decide to use Linear layers to build an Autoencoder. These layers are often useful when data is characterized by linear relationships and learn the global patters present in the input sequence. They are not particularly effective with complex data and they do not handle in any way or form the temporal information, thus limiting to work in many cases as a dimensionality reduction mechanism.

The Encoder is based on three linear layers, which have an output dimensionality which is always half that of the previous layer, starting from 32. As before, the Decoder consists of three linear layers acting on the dimensionalities of the latent

Architecture	***Input Dim.***	***Output Dim.***
ENCODER		
Conv1D(n_features, 32, kernel = 7, padding = 3, stride = 2)	(128, n_feats, 72)	(128, 32, 36)
ReLU	(128, 32, 36)	(128, 32, 36)
Dropout(rate = 0.2)	(128, 32, 36)	(128, 32, 36)
Conv1D(32, 16, kernel = 7, padding = 3, stride = 2)	(128, 32, 36)	(128, 16, 18)
ReLU	(128, 16, 18)	(128, 16, 18)
Conv1D(16, 8, kernel = 7, padding = 3, stride = 2)	(128, 16, 18)	(128, 8, 9)
ReLU	(128, 8, 9)	(128, 8, 9)
DECODER		
Conv1DTransposed(8, 16, kernel = 7, padding = 3, stride = 2)	(128, 8, 9)	(128, 16, 18)
ReLU	(128, 16, 18)	(128, 16, 18)
Dropout(rate = 0.2)	(128, 16, 18)	(128, 16, 18)
Conv1DTransposed(16, 32, kernel = 7, padding = 3, stride = 2)	(128, 16, 18)	(128, 32, 36)
ReLU	(128, 32, 36)	(128, 32, 36)
Conv1DTransposed(32, n_features, kernel = 7, padding = 3, stride = 2)	(128, 32, 36)	(128, n_feats, 72)
Sigmoid	(128, n_feats, 72)	(128, n_feats, 72)

Table 3.4: Architecture of the Convolutional Autoencoder model with indications regarding the dimensions of the inputs and outputs of each layer. The dimensions refer to the case where $n\_feats = 1$.

vector in the opposite way with respect to the encoder.

Tables 3.3, 3.4 and 3.5 propose and in-depth description of the architectures of the three models, with the indications of the dimensionality of the input and output of each element constituting the model.

It is evident how some regularization techniques were used. Dropout is particularly useful to avoid overfitting, as during training it randomly shuts down some of the neurons present in the neural networks, according to a certain probability p: this implies that the corresponding connections are associated to weights which are not involved in the updates of the backpropagation process, as the related neurons are not active.

Architecture	*Input Dim.*	*Output Dim.*
ENCODER		
Linear(72*n_feats, 36*n_feats)	(128, 72*n_feats)	(128, 36*n_feats)
ReLU	(128, 36*n_feats)	(128, 36*n_feats)
Linear(36*n_feats, 18*n_feats)	(128, 36*n_feats)	(128, 18*n_feats)
ReLU	(128, 18*n_feats)	(128, 18*n_feats)
Linear(18*n_feats, 9*n_feats)	(128, 18*n_feats)	(128, 9*n_feats)
ReLU	(128, 9*n_feats)	(128, 9*n_feats)
DECODER		
Linear(9*n_feats, 18*n_feats)	(128, 9*n_feats)	(128, 18*n_feats)
ReLU	(128, 18*n_feats)	(128, 18*n_feats)
Linear(18*n_feats, 36*n_feats)	(128, 18*n_feats)	(128, 36*n_feats)
ReLU	(128, 36*n_feats)	(128, 36*n_feats)
Linear(36*n_feats, 72*n_feats)	(128, 36*n_feats)	(128, 72*n_feats)
Sigmoid	(128, 72*n_feats)	(128, 72*n_feats)

Table 3.5: Architecture of the Linear Autoencoder model with indications regarding the dimensions of the inputs and outputs of each layer.

Activation functions like the ReLU and Sigmoid can be employed in order to regularize the output of each layer, adding non-linearity and constricting the values to be within a specific interval.

3.3.3 Generative Method

Instead of using a GAN, we decided to implement a GAN-like approach, always based on adversarial learning, like USAD, proposed in [18].

We followed the original paper implementation. Both the Autoencoders are characterized with Linear layers which in the Encoder reduce the dimensionality of the input sequence iteratively through the use of three layers which reduce it of half the size every time, while the two Decoders comprise linear layers which increase the dimensionality in the opposite direction.

The architecture of the Encoder and the two Decoders is the same as the one depicted in Table 3.5.

3.3.4 Thresholds Definition

All the methods presented up until now are not technically predisposed to perform anomaly detection.

The outputs of each of these approaches need to be compared with the corresponding ground truth, which can be either the set of subsequent timestamps with respect to the input windows in the forecasting case, or the input itself in the other two settings. Usually one considers their difference. Then, a threshold needs to be defined in order to identify the anomalous points, as follows:

$$\begin{cases} 1 & \text{if } anomaly\_score > \lambda \\ 0 & \text{if } anomaly\_score \leq \lambda \end{cases} \tag{3.6}$$

where λ is the defined threshold. So, if the anomaly score, i.e. the difference between outputs and ground truths, is bigger than the threshold for a specific data point, then this would be labelled as anomalous (1), otherwise as normal (0).

There are different ways to define both the anomaly score and the threshold.

In many cases, the anomaly score can be computed as the absolute difference between the two quantities to compare, as the relative difference, or even as the MSE.

For what concerns the thresholds, several types have been tested, some resulting more effective in performing anomaly detection than others.

The first kind of threshold that one can use is a percentile value of the absolute loss distribution computed over the validation data.

The absolute error only considers the difference between the predictions or reconstructions and the actual values. It could be useful to notice that the difference between the true energy consumption and the one outputted by the models needs to be related to the magnitude of the consumption. In fact, an absolute error of $10kwh$ can have a different impact in the analysis, based on whether for example it reflects a situation in which between prediction and ground truth there is a $1:2$ ratio or instead a case in which the prediction is $150kwh$ and the ground truth $140kwh$.

This is why a second threshold has been proposed, to take into account the possibility of scaling the error with respect to the expected values of the consumption.

Another possibility is to directly use a percentile of the distribution over the relative loss of the validation set.

We argued previously in this work that the IQR, a statistical technique to detect outliers, was useless for us in order to remove the anomalies from the training set and train the model on "normal" data. We decided though that it could still be interesting to use it in order to determine a threshold for the task. Firstly, we thought of computing the IQR over the relative loss distribution of the validation set and then compare it with respect to the relative loss of the test data points. Then,

the idea was to try an approach which took into account the different buildings present in the test set: we defined a separate threshold for each building based on the IQR computed over the test relative loss distribution, and then performed anomaly detection as before. In light of this, a threshold was tried as a weighted average between the type of threshold just described and one computed as the IQR over the entire test set.

Finally, we employed an exponential weighted moving average (EWMA) over the test relative loss, computed the difference between the EWMA and the relative loss for each point and compared it with the IQR of this same difference. The EWMA is often used as a smoothing technique for time series data: it leverages the fact that the most recent data points have a greater impact on the next values, with respect to previous ones.

In line with the original experiments performed by Audibert et Al. in [18], the anomaly detection task was also assessed by using as anomaly score the MSE between the ground truth and the outputs of the models, or a weighted average of the MSE of the ground truth and the output of the first Autoencoder and that computed with respect to the second Autoencoder, having as input the output of the first one, in case of USAD. Then such value was compared to a threshold which could be either defined by finding a percentile over the distribution of the anomaly scores, or by considering the thresholds used to compute the Receiver Operating Curve (ROC) and selecting the one corresponding to where the difference of the Equation 3.7 is non-zero.

$$\text{sign}\left(\text{tpr} - (1 - \text{fpr})\right) \tag{3.7}$$

where tpr is the true positive rate and fpr the false positive one.

Note that for these experiments, the anomaly score was compared with respect to ground truth labels which were obtained by considering a sliding window over the original labels present in the dataset: if in a specific window there was at least one anomaly, then that whole sequence was to be considered as anomalous, and was indicated as such in the ground truth labels.

3.3.5 Comparison with SWaT

To properly understand the goodness, or badness, of the chosen models, it was decided to perform a comparative study on the SWaT (Secure Water Treatment) dataset, which has been used by the authors of USAD.

Goh et Al. present this new dataset in [24]. SWaT is a scaled down version of an industrial water treatment plant, consisting of several tanks in which water is collected: the data regarding this process was collected in two modes, normal and attacked. The system run for 11 days: for the first seven, it operated normally, without any faults, while during the remaining days cyber and physical attacks were launched. A cyber attack is an attack that is transmitted through a communication network with an intention to cause some economic harm, while a physical attack involves a physical component in order to disrupt the state of the system.

The data was recorded thanks to several sensors and actuators, which are responsible for turning the water pump on or off, present in the water tanks. 946722 samples were collected through 11 days, characterized by 51 features, each being the measurements taken by the different sensors or actuators.

The following two Figures, 3.3 and 3.4, represent two features in the SWAT dataset.

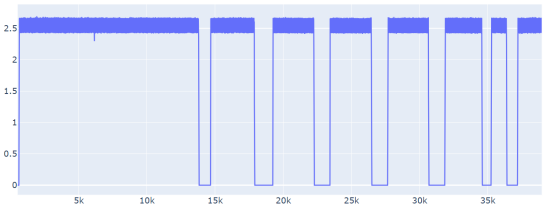


Figure 3.3: Example of time series in SWAT - FIT101

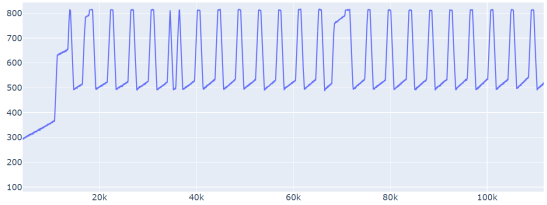


Figure 3.4: Another time series in SWAT - LIT101

It is evident to notice how the nature of the data is very different from the energy consumptions characterizing our dataset. The proposed time series seem to be constituted by less oscillating values, which could result in better performances.

Using such dataset for comparison can be useful for two main reasons: assessing the correct functioning of the models and evaluating the difference in the performances between the two different ways to set up the anomaly detection task. In fact, given that this dataset has a clear distinction between normal data and anomalous one, the Unsupervised models can be used in the traditional way, training on the

first set and testing on the one that contains also anomalies.

The models have been slightly adapted to the SWaT dataset, as shown in the following tables, 3.6, 3.8, 3.9, 3.7, but the overall architecture has been kept the same, to make the results as comparable as possible. Note, moreover, that given the nature of the features in the dataset, the experiments in this case were mainly conducted in a multivariate setting, as it does not make much sense to consider a univariate one, given that each feature represents the evolution over time of a measurement taken from different sensors and actuators.

Architecture	***Input Dimension***	***Output Dimension***
LSTM(51, 32)	(7919, 12, 51)	(128, 32)
ReLU	(128, 32)	(128, 32)
Dropout(rate = 0.2)	(128, 32)	(128, 32)
Linear(32, 51)	(128, 32)	(128, 51)

Table 3.6: Architecture of the LSTM model for the SWaT dataset, which was divided into windows of 12 measurements.

Architecture	***Input Dim.***	***Output Dim.***
ENCODER		
Linear(51*12, 51*6)	(7919, 51*12)	(7919, 51*6)
ReLU	(7919, 51*6)	(7919, 51*6)
Linear(51*6, 51*3)	(7919, 51*6)	(7919, 51*3)
ReLU	(7919, 51*3)	(7919, 51*3)
Linear(51*3, 12*40)	(7919, 51*3)	(7919, 12*40)
ReLU	(7919, 12*40)	(7919, 12*40)
DECODER		
Linear(12*40, 51*3)	(7919, 12*40)	(7919, 51*3)
ReLU	(7919, 51*3)	(7919, 51*3)
Linear(51*3, 51*6)	(7919, 51*3)	(7919, 51*6)
ReLU	(7919, 51*6)	(7919, 51*6)
Linear(51*6, 51*12)	(7919, 51*6)	(7919, 51*12)
Sigmoid	(7919, 51*12)	(7919, 51*12)

Table 3.7: Architecture of the Linear Autoencoder model for SWaT

Architecture	***Input Dimension***	***Output Dimension***
ENCODER		
LSTM(51, 32)	(7919, 12, 51)	(7919, 32)
Dropout(rate = 0.2)	(7919, 32)	(7919, 32)
DECODER		
Repeat Vector	(7919, 32)	(7919, 12, 32)
LSTM(32, 32)	(7919, 12, 32)	(7919, 12, 32)
Dropout(rate = 0.2)	(7919, 12, 32)	(7919, 12, 32)
Linear(32, 51)	(7919, 12, 32)	(7919, 12, 51)

Table 3.8: Architecture of the LSTM Autoencoder model for SWaT

Architecture	***Input Dim.***	***Output Dim.***
ENCODER		
Conv1D(51, 32, kernel = 7, padding = 3, stride = 2)	(7919, 51, 12)	(7919, 32, 6)
ReLU	(7919, 32, 6)	(7919, 32, 6)
Dropout(rate = 0.2)	(7919, 32, 6)	(7919, 32, 6)
Conv1D(32, 16, kernel = 7, padding = 3, stride = 2)	(7919, 32, 6)	(7919, 16, 3)
ReLU	(7919, 16, 3)	(7919, 16, 3)
Conv1D(16, 8, kernel = 7, padding = 3, stride = 2)	(7919, 16, 3)	(7919, 8, 2)
ReLU	(7919, 8, 2)	(7919, 8, 2)
DECODER		
Conv1DTransposed(8, 16, kernel = 7, padding = 3, stride = 2)	(7919, 8, 2)	(7919, 16, 3)
ReLU	(7919, 16, 3)	(7919, 16, 3)
Dropout(rate = 0.2)	(7919, 16, 3)	(7919, 16, 3)
Conv1DTransposed(16, 32, kernel = 7, padding = 3, stride = 2)	(7919, 16, 3)	(7919, 32, 6)
ReLU	(7919, 32, 6)	(7919, 32, 6)
Conv1DTransposed(32, 51, kernel = 7, padding = 3, stride = 2)	(7919, 32, 6)	(7919, 51, 12)
Sigmoid	(7919, 51, 12)	(7919, 51, 12)

Table 3.9: Architecture of the Convolutional Autoencoder for SWaT

3.4 Use case: DATACELLAR

DATA CELLAR has provided access to some data through an API, which was developed by the project participant CTIC in order to gather data from an energy community in the Asturias region in Spain [25]. This is going to be used to apply the developed models also on real-life data and to perform a qualitative evaluation of these, given the absence of annotations regarding anomalous data points.

The collected data represent energy consumption measurements collected also in this case every hour by smart energy meters placed in residential buildings. Unlike the LEAD dataset, the measurements, reported in KWH, do not span an entire year, but rather a single month during 2021, 2022, 2023 and 2024. The data have been collected for the same buildings throughout the different years.

The features which are present here are the following:

- *_id*: a unique identifier for each measurement; given that this does is not useful in any way it will not be considered during the experiments
- *user_id*: the equivalent of *building_id* in LEAD; it represents a unique identifier for the user to which the smart meter belongs to
- *local_date_str*: an indication of the actual date of the measurement with respect to the current location
- *datetime*: the indication of the hour, day, month and year at which the measurement was collected, according to the Greenwich time zone
- *data*: the energy consumption measurement collected at each timestamp

Let's analyze the data regarding 2021.

They have been collected for the whole month of July over 7 different buildings. Each of the seven time series does not present neither missing values for what concerns the measurements, nor missing dates, therefore the preprocessing steps in this case will start directly with the definition of windows and the corresponding normalization.

The following Figure 3.5 shows three examples of time series provided by DATA CELLAR.

Figure 3.5: Examples of three time series from 2021

We notice how such time series are different with respect to the LEAD dataset: in fact, the measurements reflect consumptions which values are quite close to 0 kwh, reaching at most a value of $2.8kwh$ within the whole dataset. Moreover, the presence of spikes brings us to thinks that those could be point anomalies, but given that the data is not annotated this is just a supposition.

Because of this, it is not possible within this thesis to assess the performances

of the models for the downstream task of anomaly detection. Therefore, the only possibility is that of assessing how good or bad are the trained models to generalize to unseen data and how they are able to perform the forecasts or reconstructions over real data values.

Chapter 4

Experimental Results

Several experiments have been made to test out the capabilities of the chosen methods to perform anomaly detection in a completely unsupervised setting.

In all the experiments, firstly the models were trained on sequences of windows extracted from the time series, which were appropriately normalized within the interval $[0, 1]$. In all the cases the MSE was used as a loss, aside from the LSTM Autoencoder, for which a MAE loss was used. Then, the testing was performed on a different set, as explained in the previous Chapter, and the anomaly detection task was performed by defining different kinds of thresholds. Some post processing operations were performed to take into account the initial missing values: given that the missing energy consumption measurements always corresponded to non-anomalous points, it made sense to change the class of each of these data points to normal when performing anomaly detection, so as to not have them influence negatively the performances.

All the experiments have been performed on the LINKS Foundation internal servers, using a single GPU, NVIDIA GeForce RTX 2080 Ti. The training of the models took on average 40 minutes, in the univariate setting, and 70 minutes in the multivariate one. The testing was in all cases quite short, reaching a maximum of 10 minutes.

4.1 Evaluation Methods

To evaluate the performances of the models, the labels present in the dataset were leveraged and the metrics employed are the typical metrics one would use for a classification task. All the results that are going to be presented in the next chapter refer to the metrics computed over the anomaly class.

Precision: as the ratio between the number of data points correctly classified as anomaly and the total number of points classified as such.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

Where TP is the number of True Positives, the ones correctly classified as the positive (anomaly) class, and FP is the number of False Positives, those that were erroneously classified as anomalies, when they were normal points.

Recall: as the ratio between the number of data points correctly classified as anomalies and the total number of data points actually belonging to the positive class.

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

Where FN is the number of False Negatives, the data points which have been erroneously assigned to the normal class, when they are actually anomalies.

F1: an harmonic mean between Precision and Recall

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.3}$$

ROC-AUC: a Receiver Operating Characteristic (ROC) curve is a graph showing the performance of a binary classification model at all classification thresholds. It takes into consideration the True Positive Rate (TPR), also known as Recall for the anomaly class, which considers the fraction of correct classifications over the total number of positives present in the dataset, and the False Positive Rate (FPR), which considers the fraction of elements incorrectly classified as belonging to the positive class, over the total number of elements belonging to the negative class. AUC is the Area Under the ROC curve, which could be interpreted as the probability that the model ranks a random positive example more highly than a random negative one.

Notation

For the sake of simplicity, the following notation is going to be used to present the results of the different experiments:

- Anomaly Score - ROC: the results are based on the usage of the MSE between the ground truth and the output of the models. Such predictions are compared with the anomaly labels, which are obtained by considering the original labels in windows: if a window contains at least one anomaly, then the entire window is going to be considered as anomalous. Then, a threshold is determined based on the ROC between the predictions and the obtained labels
- Anomaly Score - Perc: in this case, the threshold is determined by considering a percentile, typically the 93rd or 80th, over the distribution of the anomaly scores
- Method 0: the threshold is determined based on the absolute loss on the validation set
- Method 1: the threshold is based on the normalized absolute loss with respect to the expected value of the prediction
- Method 2: the threshold is based on the relative loss of the validation set
- Method 3: the threshold is based on the IQR over the validation set
- Method 4: the threshold is based on the IQR over the test set, divided by building
- Method 5: the threshold is based on the Exponential Weighted Moving Average over the test set
- Method 6: the threshold is based on the weighted combination of the IQR over the entire test set, and the one computed per building

4.2 Univariate Setting

First of all, the models were trained in a univariate setting: the target of the analysis, the energy consumption, has been used without additional features, as the exclusive input for all the models. All the models have been trained for 40 epochs.

The following five Tables, 4.1, 4.3, 4.4, 4.5 and 4.2, present the results for each of the methodologies.

Threshold	***Prec***	***Rec***	***F1***	***ROC-AUC***
Anomaly Score - ROC	**0.31**	**0.51**	**0.39**	0.5139
Anomaly Score - Perc	0.31	0.32	0.32	0.5129
Method 0	0.17	0.08	0.11	0.5362
Method 1	0.12	0.51	0.19	0.7107
Method 2	0.22	0.47	0.30	0.7131
Method 3	0.15	0.49	0.23	0.7130
Method 4	0.23	0.47	0.31	0.7164
Method 5	0.11	0.47	0.18	0.6909
Method 6	0.26	0.47	0.33	**0.7199**

Table 4.1: Univariate experiments for LSTM

Threshold	***Prec***	***Rec***	***F1***	***ROC-AUC***
Anomaly Score - ROC	0.21	**0.58**	**0.31**	0.5783
Anomaly Score - Perc	**0.27**	0.33	0.29	0.5767

Table 4.2: Univariate experiments for USAD

Threshold	***Prec***	***Rec***	***F1***	***ROC-AUC***
Anomaly Score - ROC	0.26	**0.65**	**0.38**	0.6502
Anomaly Score - Perc	**0.54**	0.23	0.32	0.5961
Method 0	0.28	0.15	0.19	0.5688
Method 1	0.18	0.49	0.27	0.7193
Method 2	0.24	0.47	0.32	0.7181
Method 3	0.16	0.50	0.24	0.7188
Method 4	0.29	0.47	0.36	0.7202
Method 5	0.13	0.43	0.20	0.6801
Method 6	0.32	0.47	0.38	**0.7252**

Table 4.3: Univariate experiments for LSTM Autoencoder

Threshold	***Prec***	***Rec***	***F1***	***ROC-AUC***
Anomaly Score - ROC	**0.25**	**0.63**	**0.35**	0.6280
Anomaly Score - Perc	0.30	0.37	0.33	0.5987
Method 0	0.16	0.06	0.09	0.5268
Method 1	0.20	0.47	0.28	0.7129
Method 2	0.24	0.46	0.31	0.7121
Method 3	0.16	0.50	0.24	0.7172
Method 4	0.18	0.50	0.27	**0.7215**
Method 5	0.11	0.52	0.18	0.7092
Method 6	0.20	0.49	0.29	0.7211

Table 4.4: Univariate experiments for Convolutional Autoencoder

Threshold	***Prec***	***Rec***	***F1***	***ROC-AUC***
Anomaly Score - ROC	0.28	**0.67**	0.40	0.6700
Anomaly Score - Perc	**0.37**	0.46	**0.41**	0.6556
Method 0	0.26	0.11	0.16	0.5528
Method 1	0.17	0.51	0.26	0.7236
Method 2	0.25	0.48	0.33	0.7231
Method 3	0.15	0.53	0.24	0.7284
Method 4	0.26	0.51	0.34	**0.7361**
Method 5	0.13	0.47	0.20	0.6972
Method 6	0.24	0.50	0.32	0.7322

Table 4.5: Univariate experiments for Linear Autoencoder

It is evident to notice how the best performing model is the Linear Autoencoder, followed shortly by the LSTM Autoencoder. The worst model is USAD.

The results here reported are computed for the entire dataset: it becomes interesting to assess how the performances vary among the different buildings, evaluating the average performance and on which building the models work best or worst.

For sake of simplicity, the proposed results are obtained by considering the last thresholding method (6), aside from USAD, for which we used the anomaly score.

We can see how the average performance is higher than the one we obtained by considering the anomaly detection task over the entire dataset. This can be explained by the following table, 4.6, as it is evident how each method can work

Model	***Prec***	***Rec***	***F1***	***ROC-AUC***
LSTM	0.41	0.54	**0.43**	0.7525
LSTM Autoencoder	0.40	0.53	**0.43**	0.7525
Convolutional Autoencoder	0.37	0.56	0.40	0.7560
Linear Autoencoder	0.38	**0.57**	**0.42**	**0.7646**
USAD	**0.54**	0.35	0.30	0.5919

Table 4.6: Average performances for each model, based on method 6 for threshold

better on certain buildings, rather than others.

On average, the models chosen are able to reach a F1 score of 0.41 and at most a ROC-AUC score of 0.76. It thus becomes interesting to understand how the models perform with respect to the different kinds of anomalies that are present in the dataset.

Table 4.7 and 4.8 show the performances respectively with respect to proper outliers and contextual anomalies. The distinction was made based on the IQR: all the anomalies whose value fell outside of this range were considered as outliers, the others as contextual.

Model	***Prec***	***Rec***	***F1***	***ROC-AUC***
LSTM	0.73	0.91	0.81	0.9527
LSTM Autoencoder	0.72	0.92	0.80	0.9572
Convolutional Autoencoder	**0.75**	0.90	**0.82**	0.9477
Linear Autoencoder	0.69	**0.92**	0.79	**0.9600**
USAD	0.41	0.40	0.41	0.6766

Table 4.7: Performances of the models with respect to Point Anomalies (Outliers)

Model	***Prec***	***Rec***	***F1***	***ROC-AUC***
LSTM	0.11	0.25	0.15	0.6080
LSTM Autoencoder	0.14	0.24	0.17	0.6076
Convolutional Autoencoder	0.08	0.27	0.13	0.6135
Linear Autoencoder	0.10	**0.28**	**0.25**	**0.6206**
USAD	**0.19**	0.27	0.22	0.5785

Table 4.8: Performances of the models with respect to Contextual Anomalies

As was actually to be expected, the models perform way better on point anomalies rather than on contextual. Despite being trained on anomalous data, the first kind of uncommon behaviours in data is still quite easy to recognize: this is due to the fact that the forecast or reconstruction may not be as good in correspondence of those points as it is instead at modelling contextual anomalies, which are learnt as normal behavior and as such not recognized as often.

Let's now assess the performances of the single buildings.

For example, building 439 is the one on which all the methodologies perform worse, regardless of the threshold. From Figure 4.1 it is evident to notice how anomalies are mainly contextual: in fact, the ground truth anomalies (in green) do not correspond to extremely high or low consumption measurements, but rather to consumptions which were not in line with previous ones. In this case, the reconstruction is quite precise with respect to the energy consumptions, but the difference is such that with the chosen thresholding method many anomalies are identified which are not truly present in the dataset. The performances are: 0.02 of Precision, 0.07 of Recall, 0.03 of F1 score and 0.3754 of ROC-AUC score. This method identifies 2674 anomalies, over a ground truth number of 567.

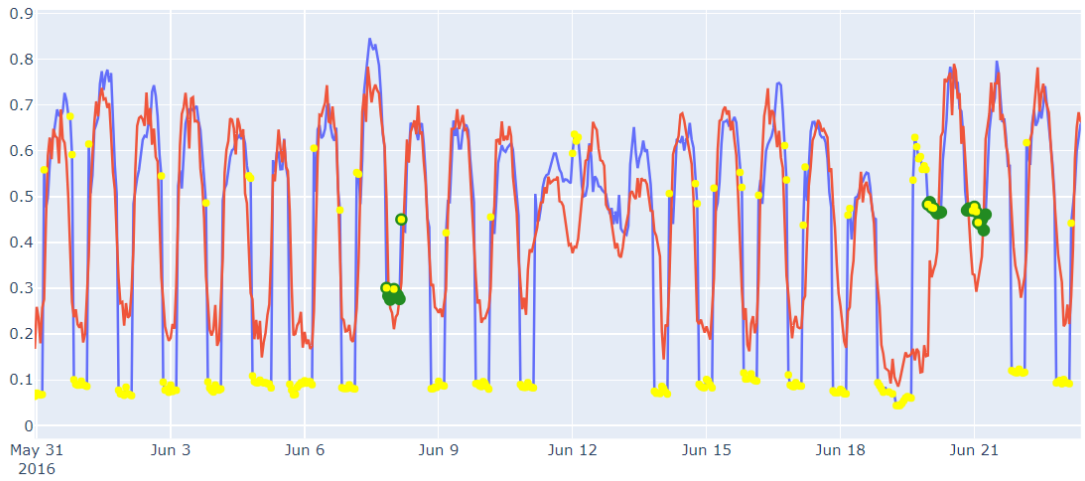


Figure 4.1: Example of reconstruction of the LSTM Autoencoder on a portion of the time series related to building 439. In red the reconstruction, in green the true anomalies, as labelled in the dataset, in yellow the identified anomalies according to thresholding method 3

Unlike the case of the worst performing building, which is the one just presented in almost all the experiments, there is no building which achieves the highest

performances on all models. In some cases it is building 739, in others 994.

To just provide an example of good performances, it could be interesting to analyze building 889, of which a portion is presented in Figure 4.2.

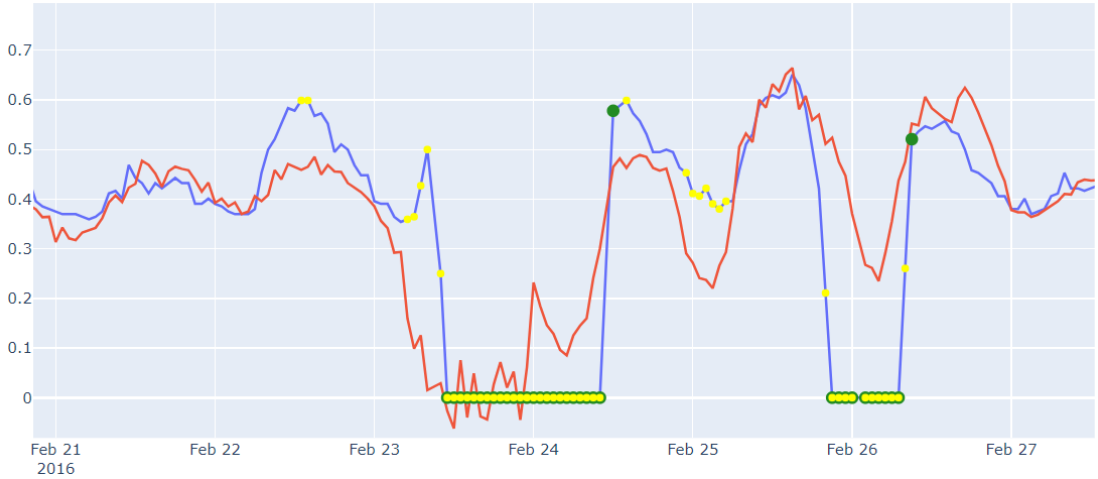


Figure 4.2: Example of reconstruction of the LSTM Autoencoder on a portion of the time series related to building 884. In red the reconstruction, in green the true anomalies, in yellow the identified ones according to thresholding method 3

On this building, the LSTM Autoencoder reaches 0.71 of Precision, 0.88 of Recall, 0.78 of F1-score and 0.9366 of ROC-AUC score. The Figure above is an example of how the model is able to identify the majority of the anomalies, even though the reconstruction is such that other non-anomalous points are flagged as anomalies. Despite this, one could still argue that the identification of such abnormal behaviors in the time series depends also on how an anomaly is defined technically: for example, in this case, the data points between February 23 and 24, which have been flagged as anomalies, could be accepted as such given that they correspond to a rapid change in the energy consumption that then lead to a 0 kwh measurement that lasted almost one day.

It becomes interesting to evaluate the performances for the different *primary_use* present in the test set. Tables 4.9, 4.10, 4.11 and 4.12 show the metrics computed considering method 6 for the threshold definition for each of the models.

Primary_use	***Prec***	***Rec***	***F1***	***ROC-AUC***
Education	0.20	0.64	0.31	0.7959
Entertainment/public assembly	0.14	0.44	0.21	0.6791
Lodging/residential	**0.41**	**0.65**	**0.50**	**0.8086**
Office	0.11	0.33	0.17	0.6166
Parking	0.03	0.47	0.06	0.6166
Public services	0.32	0.52	0.40	0.7416

Table 4.9: Performances of the models with respect to the category of each building in the test set with the LSTM Autoencoder

Primary_use	***Prec***	***Rec***	***F1***	***ROC-AUC***
Education	0.34	**0.67**	0.45	**0.8231**
Entertainment/public assembly	0.17	0.39	0.23	0.6642
Lodging/residential	**0.36**	0.64	**0.46**	0.7996
Office	0.16	0.32	0.22	0.6388
Parking	0.05	0.57	0.09	0.6924
Public services	0.24	0.51	0.32	0.7308

Table 4.10: Performances of the models with respect to the category of each building in the test set with the Linear Autoencoder

Primary_use	***Prec***	***Rec***	***F1***	***ROC-AUC***
Education	0.28	0.65	0.39	0.8089
Entertainment/public assembly	0.16	0.40	0.23	0.6659
Lodging/residential	0.32	**0.61**	**0.42**	**0.7829**
Office	0.12	0.30	0.17	0.6229
Parking	0.04	0.41	0.07	0.6113
Public services	**0.33**	0.5	0.4	0.7340

Table 4.11: Performances of the models with respect to the category of each building in the test set with the Convolutional Autoencoder

Primary_use	***Prec***	***Rec***	***F1***	***ROC-AUC***
Education	**0.35**	**0.65**	**0.46**	**0.8116**
Entertainment/public assembly	0.21	0.39	0.27	0.6709
Lodging/residential	0.34	0.61	0.44	0.7855
Office	0.15	0.28	0.20	0.6198
Parking	0.04	0.38	0.08	0.6165
Public services	0.29	0.42	0.34	0.6949

Table 4.12: Performances of the models with respect to the category of each building in the test set with the LSTM (forecasting)

While for the LSTM and Convolutional Autoencoder, the kind of buildings with the highest overall performances are the Residential ones, the Linear Autoencoder works well also in the case of Educational buildings, while the LSTM model used in forecasting works best on this last kind, which is followed shortly by residential constructions.

4.2.1 Models Visual Comparison

It is interesting to understand how the different models perform in terms of their respective reconstructions or forecastings.

For this purpose, we are going to assess their capabilities on building 889, which constitutes one of the time series on which the models perform better.

The LSTM Autoencoder identifies 190 anomalies, over a ground truth of 167: it achieves 0.75 of Precision, 0.86 of Recall, 0.80 of F1 score and 0.9254 of ROC-AUC score. From Figure 4.3 it is evident to see how the reconstruction is more or less precise: it does not reconstruct exactly the measurements which correspond to what we would consider as outlier sequences o points, which is actually good for the task of anomaly detection. It has some difficulties in identifying contextual anomalies, as the reconstruction error is not as high as in other cases.

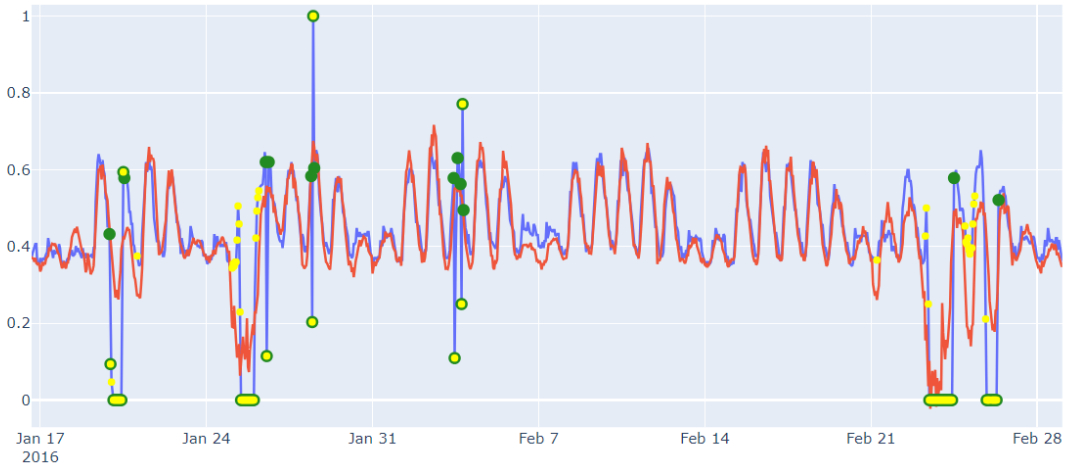


Figure 4.3: Reconstruction of LSTM Autoencoder on building 889, using thresholding method 3. In red the reconstruction, in green the true anomalies, as labelled in the dataset, in yellow the identified anomalies.

The Convolutional Autoencoder identifies 191 anomalies, achieving 0.80 of Precision, 0.91 of Recall, 0.85 of F1 and 0.9528 of ROC-AUC score. Figure 4.4 shows a reconstruction which is almost perfect: there are only some spikes which this model is not able to reproduce perfectly. Note that in this case, even though with a reconstruction as good one would think that the model would not identify almost any anomaly, this is not the case due to the thresholding method: even if the reconstruction error is extremely small, anomalies are still detected due to the presence of the threshold which is defined on the distribution of the relative error

computed on the test set.

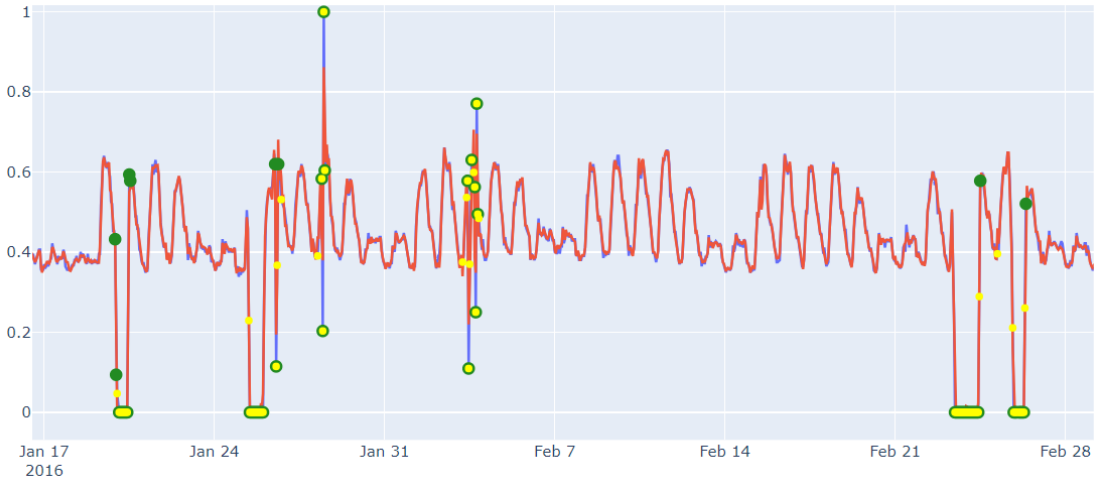


Figure 4.4: Reconstruction of Convolutional Autoencoder on building 889, using thresholding method 3. In red the reconstruction, in green the true anomalies, as labelled in the dataset, in yellow the identified anomalies.

The Linear Autoencoder identifies 209 anomalies, with a 0.71 Precision, 0.89 Recall, 0.79 F1 and 0.9396 ROC-AUC score. The reconstruction shown in Figure 4.5 seems to be a compromise between the two previously presented: it is quite precise, except in correspondence of extremely low or high values.

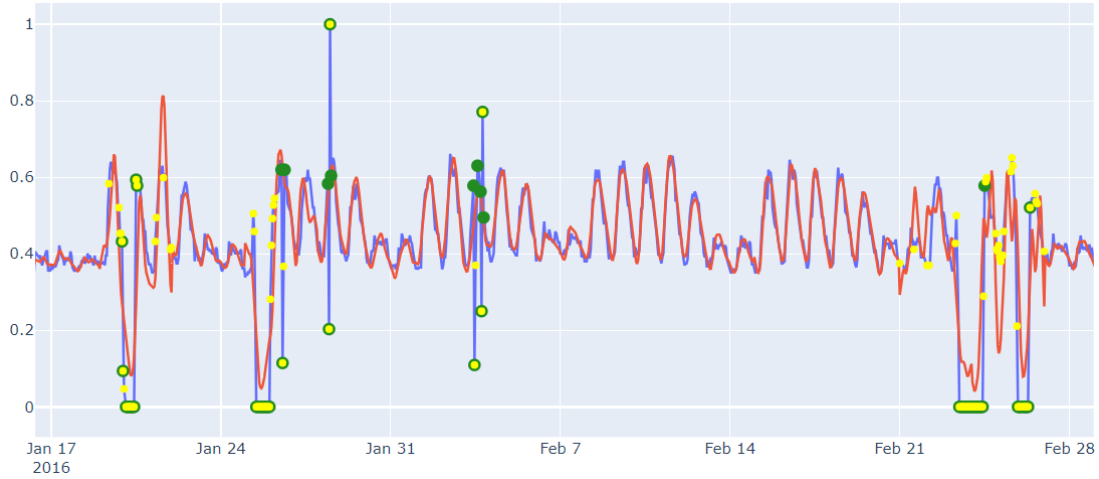


Figure 4.5: Reconstruction of Linear Autoencoder on building 889, using thresholding method 3. In red the reconstruction, in green the true anomalies, as labelled in the dataset, in yellow the identified anomalies.

Finally, the LSTM model identifies with its forecasting task 170 anomalies, reaching a Precision of 0.87, a Recall of 0.89, a F1 score of 0.88 and a ROC-AUC score of 0.9418. Figure 4.6 presents a forecast which is quite good, but is characterized by some noise especially when the real consumptions assume a value equal to 0: the LSTM model produces an oscillating output in this case, not modelling precisely the measurement, but it is still able to identify those points as anomalies thanks to the thresholding method.

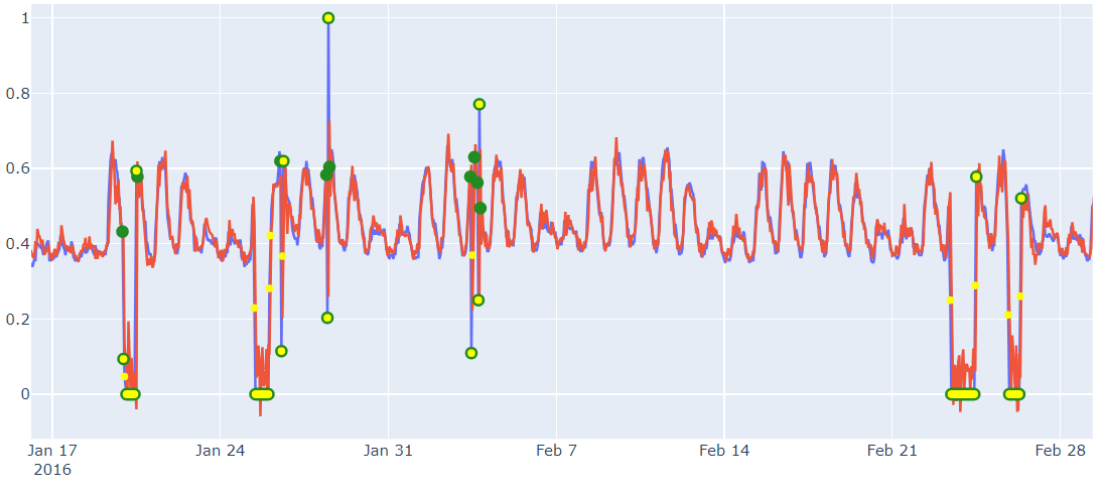


Figure 4.6: Reconstruction of LSTM model on building 889, using thresholding method 3. In red the reconstruction, in green the true anomalies, as labelled in the dataset, in yellow the identified anomalies.

One could decide, in order to try to avoid such noise in the forecast, to perform a progressive substitution of the predicted anomalies: when the model predicts an anomaly in correspondence of a certain data point, the value predicted is substituted to the original energy consumption measurement, so as to take such prediction into consideration when applying the model to the following window. This smooths out the forecast, as shown in Figure 4.7, avoiding the noise which characterizes the original output of the LSTM model.

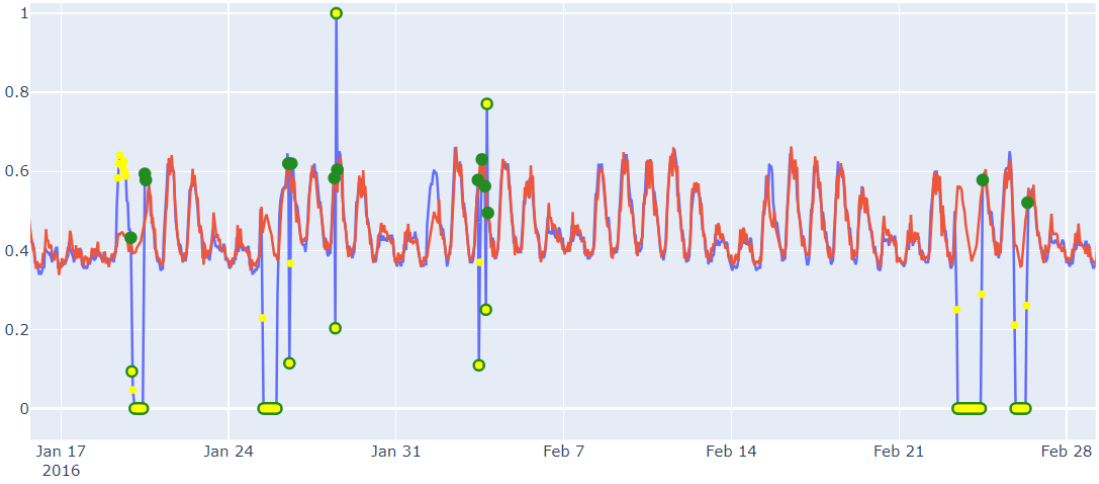


Figure 4.7: Reconstruction of the "corrected" LSTM model on building 889, using thresholding method 3. In red the reconstruction, in green the true anomalies, as labelled in the dataset, in yellow the identified anomalies.

The main downside of this practice is the fact that the process becomes quite lengthy, because it is no longer possible to forecast the entire time series and then perform anomaly detection, but after forecasting the next time stamp for a specific window, it it necessary to immediately evaluate whether that point, based on a predefined threshold, is to be considered as anomalous or not in order to eventually perform the substitution which is going to be employed when considering the next time window.

4.3 Multivariate Setting

To have a comprehensive view of the task, other trials have been performed by considering additional features with respect to the energy consumption. Note that in this case the performances were assessed using the anomaly score as exclusive metric, unless the models were built in such a way as to output a single value, corresponding to the energy consumption, thus using the other features as additional information, instead of outputting a prediction/forecast for all the input features.

For what concerns the Convolutional and LSTM Autoencoder, the experiments were conducted as to output, on one hand, a number of predictions or reconstructions equal to the number of input features, and on the other hand as to output a single feature, the one corresponding to the energy consumption. Note that in the latter case, the Tables below propose all the thresholding methodologies, while in the former exclusively the ones based on the anomaly score.

Tables 4.13 and 4.14 show the performances of the LSTM Autoencoder reconstructing multiple features.

Threshold	***Prec***	***Rec***	***F1***	***ROC-AUC***
Anomaly Score - ROC	0.19	**0.55**	**0.29**	0.5531
Anomaly Score - Perc	**0.25**	0.11	0.15	0.5235

Table 4.13: Multivariate experiments for Convolutional Autoencoder: the model outputs a number of features equal to those in input

Threshold	***Prec***	***Rec***	***F1***	***ROC-AUC***
Anomaly Score - ROC	0.24	**0.62**	**0.34**	0.6160
Anomaly Score - Perc	**0.37**	0.16	0.22	0.5532
Method 0	0.15	0.06	0.09	0.5277
Method 1	0.19	**0.47**	0.28	0.7129
Method 2	**0.21**	0.47	0.29	0.7130
Method 3	0.14	0.50	0.23	0.7164
Method 4	0.19	0.49	0.27	0.7219
Method 5	0.10	**0.52**	0.17	0.7084
Method 6	0.20	0.49	**0.23**	**0.7221**

Table 4.14: Multivariate experiments for Convolutional Autoencoder: the model outputs one feature

Tables 4.15 and 4.16 show the performances of the LSTM Autoencoder reconstructing multiple features.

Threshold	*Prec*	*Rec*	*F1*	*ROC-AUC*
Anomaly Score - ROC	0.22	**0.59**	**0.32**	0.5873
Anomaly Score - Perc	**0.36**	0.15	0.21	0.5499

Table 4.15: Multivariate experiments for LSTM Autoencoder: the model outputs a number of features equal to those in input

Threshold	*Prec*	*Rec*	*F1*	*ROC-AUC*
Anomaly Score - ROC	0.22	**0.59**	**0.32**	0.5947
Anomaly Score - Perc	**0.39**	0.17	0.24	0.5596
Method 0	0.20	0.14	0.17	0.5635
Method 1	0.06	**0.56**	0.11	0.6781
Method 2	0.19	0.46	0.27	0.7045
Method 3	0.17	0.47	0.25	0.7069
Method 4	0.31	0.45	0.37	0.7151
Method 5	0.14	0.41	0.20	0.6736
Method 6	**0.34**	0.45	**0.38**	**0.7142**

Table 4.16: Multivariate experiments for LSTM Autoencoder: the model outputs one feature

For what concerns, instead, the other three methods, the LSTM (Forecasting), the Linear Autoencoder and USAD, the experiments were performed by asking the models to output a reconstruction for all the features.

Threshold	*Prec*	*Rec*	*F1*	*ROC-AUC*
Anomaly Score - ROC	**0.31**	**0.51**	**0.39**	0.5144
Anomaly Score - Perc	**0.31**	0.32	0.32	0.5122

Table 4.17: Multivariate experiments for LSTM: the model outputs a number of features equal to those in input

Threshold	*Prec*	*Rec*	*F1*	*ROC-AUC*
Anomaly Score - ROC	0.21	**0.58**	**0.31**	0.5784
Anomaly Score - Perc	**0.35**	0.11	0.15	0.5470

Table 4.18: Multivariate experiments for Linear Autoencoder

Threshold	***Prec***	***Rec***	***F1***	***ROC-AUC***
Anomaly Score - ROC	**0.20**	**0.56**	**0.29**	0.5565
Anomaly Score - Perc	**0.20**	0.09	0.12	0.5108

Table 4.19: Multivariate experiments for USAD

It is evident to notice how the multivariate setting does not allow a great and significant improvement upon the univariate one: depending on the thresholding method used and on the metric, the numerical results can be slightly higher or lower with respect to those the models were able to reach in the univariate setting.

4.4 Comparison with SWaT

Up until now, the models have been able to achieve an F1-score of at most 0.41. In order to understand if this is due to too simple models or if it depends on the dataset, which may be formed by difficult time series to handle for the anomaly detection task, or even on the difficulty of the task, it was decided to assess the capabilities of such models on the SWaT dataset, which was one of those used by the authors of USAD ([18]) to evaluate their proposal.

Model	***Prec***	***Rec***	***F1***	***ROC-AUC***
LSTM	0.21	0.66	0.32	0.6590
LSTM Autoencoder	0.23	0.68	0.35	0.6832
Convolutional Autoencoder	0.23	0.69	0.35	0.6876
Linear Autoencoder	0.22	0.67	0.33	0.6697
USAD	**0.26**	**0.72**	**0.38**	**0.7160**

Table 4.20: Performances of each model on SWaT based on the anomaly score. The threshold is based on the ROC.

Model	***Prec***	***Rec***	***F1***	***ROC-AUC***
LSTM	**0.99**	0.57	0.72	0.7844
LSTM Autoencoder	0.97	0.56	0.71	0.7777
Convolutional Autoencoder	0.86	0.49	0.63	0.7405
Linear Autoencoder	**0.99**	0.57	0.72	0.7831
USAD	**0.99**	**0.62**	**0.76**	**0.8087**

Table 4.21: Performances of each model on SWaT based on the anomaly score. The threshold is based on the 93rd percentile.

Tables 4.20 and 4.21 shows the results obtained on this dataset. Note that the metrics used are anomaly score based and that the experiments were performed in the multivariate setting, as deemed more significant given the nature of the dataset.

It is evident to see how the models created by me are able to achieve good results on this dataset. This can be explained according to two main reasonings.

First of all, the task, while still being Unsupervised, has been posed in a different way: for SWaT, in fact, a training dataset containing exclusively normal measurements has been used to train the model, which would then be able to recognize

in an easier way anomalies, given that it has not learned them while training. In our case, instead, we are passing as input of the models a dataset which contains normal measurements as well as anomalous ones: this implies that the model learns to recognize anomalies too, as if they were normal energy consumption values, thus at inference time it is able to forecast or reconstruct them, making it very difficult, if not impossible in certain cases, to correctly identify anomalies, as the output models them perfectly.

Secondly, the dataset is quite different from LEAD: as shown in Chapter 3, the time series are not characterized by rapidly changing values and, from a visual inspection, they seem to be easier to learn for a model with respect to the time series present in the LEAD dataset.

4.5 Use Case: DATA CELLAR

In this section I am going to show the capabilities of generalization of the created models on the DATA CELLAR dataset.

Note that since no labels are available, an assessment of their performances for the anomaly detection task on real data cannot be performed. What one can do, though, is look at how well or bad the models are able to perform predictions or forecasts on the time series present in this setting.

Given the nature and structure of the data, presented in Chapter 3, the experiments were carried out in the univariate setting, as no additional features were provided by DATA CELLAR itself.

Note, nonetheless, that performing multivariate experiments could still be doable, if one considers as additional attributes exclusively those that can be derived from either the timestamp of the measurements (so, temporal and trigonometric features) or the energy consumption values themselves, like lagged features.

All experiments have been performed by taking the models previously trained on LEAD in the univariate setting: the DATA CELLAR measurements were divided in windows of the same size as those obtained over the LEAD dataset, then they were normalized in the same way.

Before showing how the different models were able to perform for their original tasks of reconstruction or forecasting on such data, it is important to take some things into account.

First of all, the data provided by DATA CELLAR refers to one type of building, mainly residential: this was among the top 4 main "primary uses" of the LEAD dataset, thus the models were able to see some examples of this kind, even though in proportion to the other kinds of buildings this contained still a very low number of examples.

Secondly, the data presents measurements which assume values in a very narrow interval, on average among the four years present, between $0kwh$ and $2.5kwh$: this could potentially be an issue, because the measurements present in LEAD, instead, are characterized by higher values. Though, the normalization applied on the data before testing the models could help in smoothing out this difference and could still enable the tried methodologies to generalize on such real-life data.

Figure 4.8 shows how the LSTM Autoencoder is able to reconstruct upon the DATA CELLAR time series.

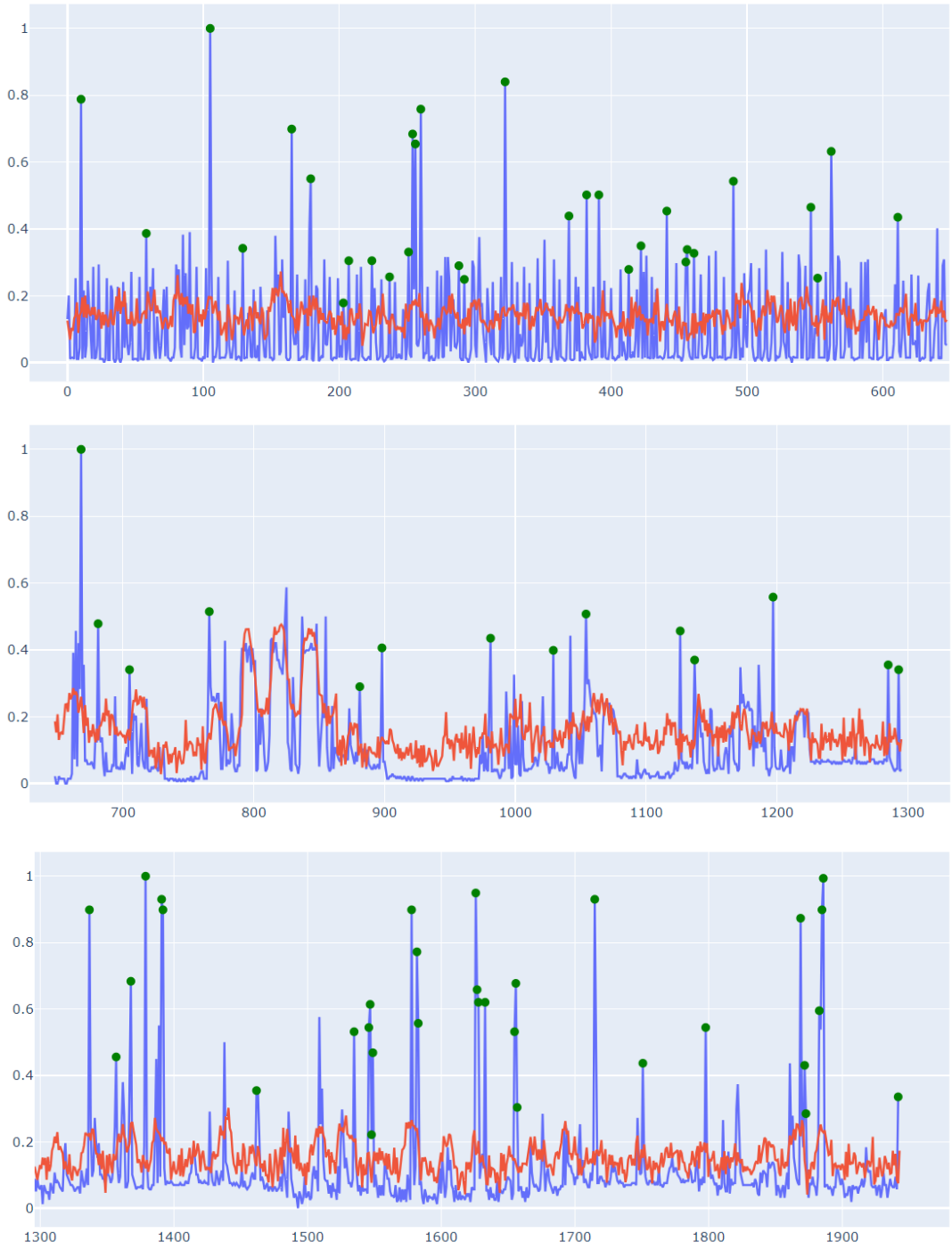


Figure 4.8: Examples of reconstructions of three time series from 2021 using LSTM Autoencoder. In blue the original input, in red the reconstruction, in green the anomalies as identified with Method 3

These reconstructions were obtained by applying the models trained on LEAD on the DATA CELLAR data. We can see that overall the output of the LSTM Autoencoder does not capture precisely the evolution over time of the consumptions: it works better in the second and third case, where despite not being able to produce an almost exact reconstruction of the input, it is still able to follow the overall trend.

Just as a comparison, the following Figures 4.9, 4.10 and 4.11 show how the reconstructions/forecasts vary based on the model used for what concerns the second building above presented.

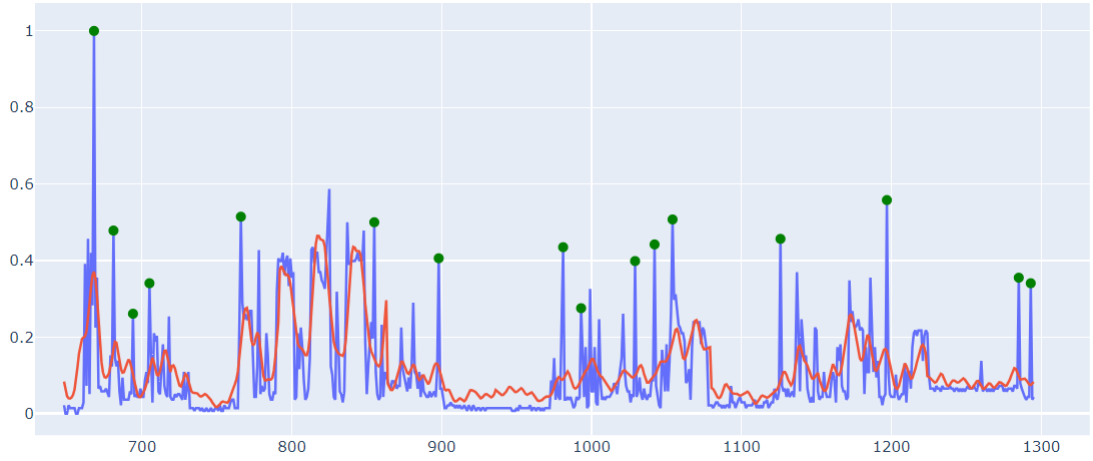


Figure 4.9: Reconstruction of the second building with the Linear Autoencoder

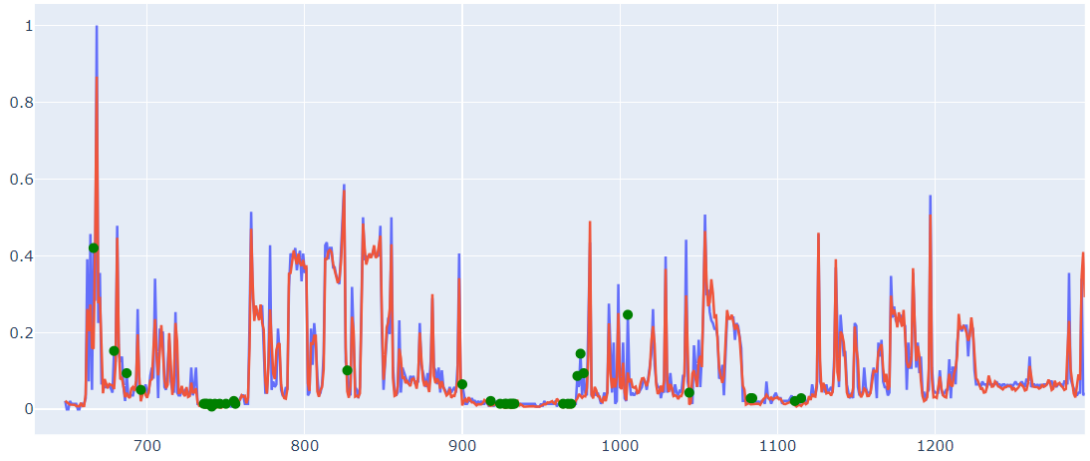


Figure 4.10: Reconstruction of the second building with the Convolutional Autoencoder

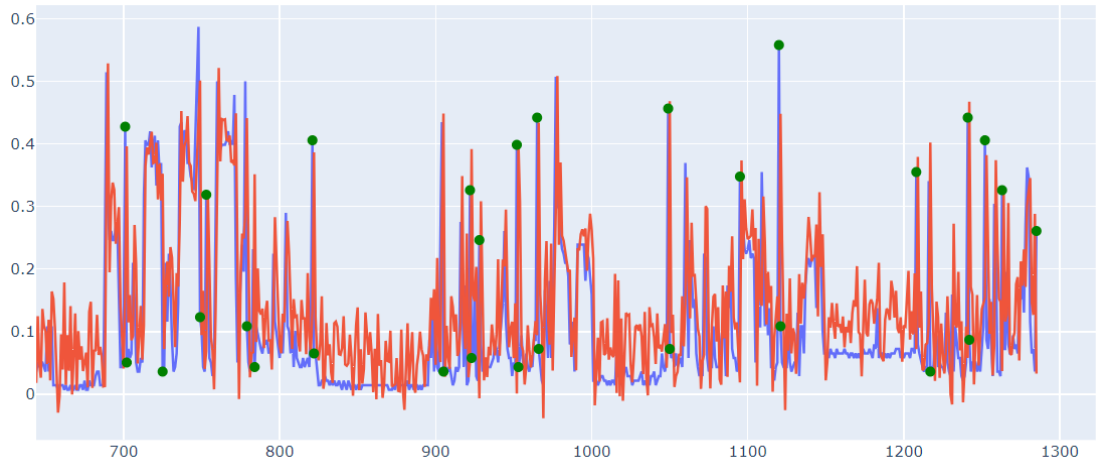


Figure 4.11: Reconstruction of the second building with the LSTM Model (forecasting)

With respect to the reconstruction proposed by the LSTM Autoencoder in Figure 4.8, the one made by the Convolutional Autoencoder is the most precise, as it follows the evolution of the consumptions almost perfectly. This could be an issue though for the anomaly detection task, as the error may be such that with some of the thresholds I developed also points which are not potentially anomalies would be labelled as such. The Linear Autoencoder produces a good approximation of the consumptions, while still not being very precise, but just following the trends.

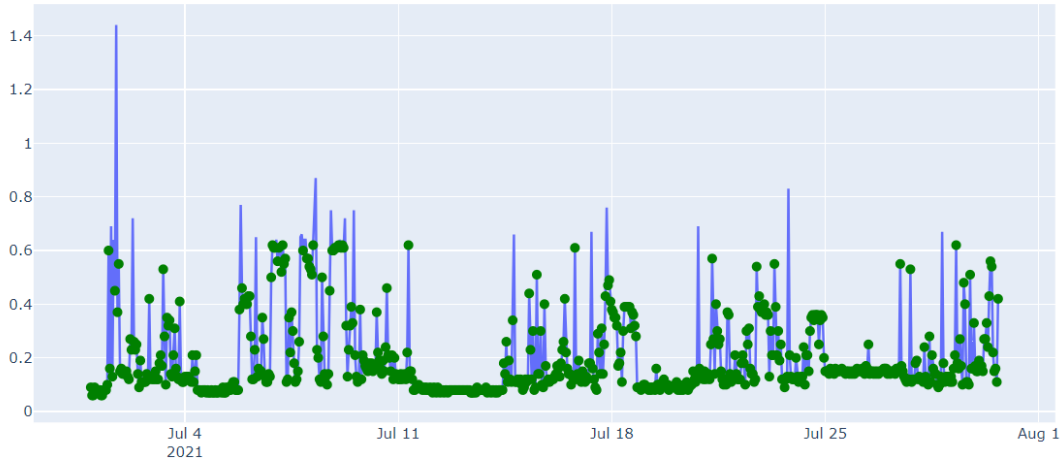
The LSTM model, instead, produces a very noisy forecast of the time series, in some cases more precise, in others not.

One could always argue that, given that the LEAD dataset is technically annotated, a XGBoost classification model could have been trained in a Supervised setting and then tested on the data provided by DATA CELLAR.

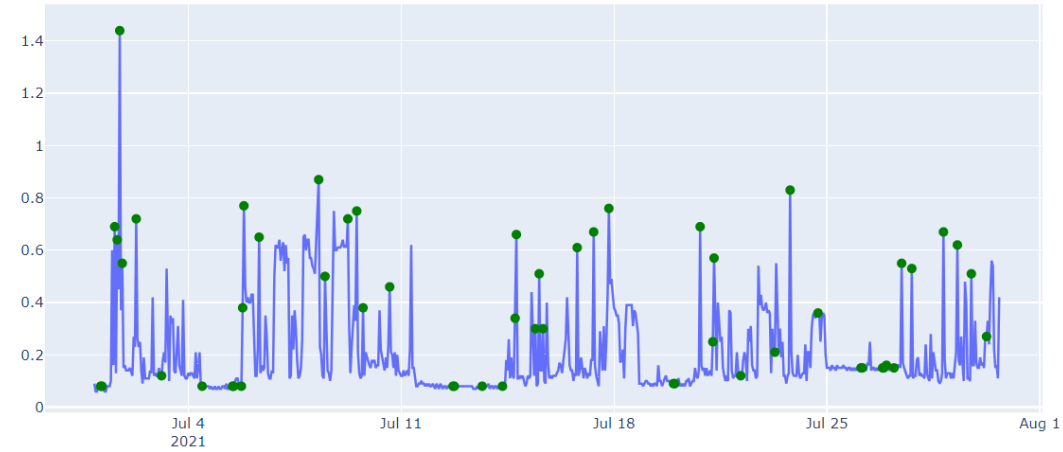
XGBoost, in fact, has proven to be quite effective in identifying the anomalies present in the LEAD dataset, as shown by the winners of the competition. Instead of exploring Unsupervised methodologies, as was done with this thesis, one could have just used such model which proved to be able to reach high results.

The problem that arises with this approach is the complete lack of generalisation: if in fact one performs a comparative experiment in a univariate setting, considering exclusively the energy consumption measurements, it becomes noticeable how XGBoost overfits on the specific dataset, identifying too many anomalies on the data provided by DATA CELLAR. This can be easily explained by the fact that the consumptions have different scales within the datasets: the ones of DATA

CELLAR are very close to 0, and XGBoost has likely learned that measurements equal or around that value are anomalies, thus resulting in almost all the data points to be labelled as such.



(a) Anomaly Detection with univariate XGBoost



(b) Anomaly Detection with multivariate XGBoost

Figure 4.12: Comparison of the anomaly detection performed by XGBoost if trained in a univariate or multivariate setting

Figure 4.12a shows how XGBoost performs on the second building presented: we can see how over 750 data points, around 691 are flagged as anomalies, the majority being the observations corresponding to values very close to zero, which the Unsupervised approaches consider instead as normal points.

The performances can be improved if one trains XGBoost in a multivariate setting,

considering also lagged features and the differences with respect to the current measurement. In this case, the model identifies less anomalies than in the univariate setting.

Figure 4.12b shows the improvement over the second building: this time, XGBoost identifies 54 anomalies, which is definitely better than before, but still the generalisation is quite poor with respect to Unsupervised approaches.

Chapter 5

Conclusions

5.1 Conclusions

In this thesis I tackled the problem of performing anomaly detection on energy consumption data in the case in which the dataset is unlabelled and contains anomalies.

The only way of handling the task in this case is using Unsupervised approaches, which do not rely on the presence of labels, but are traditionally used with a training dataset which contains only normal instances, thus allowing the models to be able to easily identify anomalies, given that they did not learn how to reconstruct or forecast them.

Within this work I assessed the possibility of using those same models when the training data contains anomalies. This makes the task of anomaly detection harder, as shown also by the results obtained: all the models are trained also with anomalous data, which means that they become able to learn those values as if they were normal consumptions. This is why finding the right threshold for identifying the anomalies becomes a very important task: the threshold is the main element responsible for a good performance of a model, as with the right one we can still be able to identify the anomalous points.

We proved that the Unsupervised models are able to work extremely well with point anomalies: despite, in fact, seeing the abnormal behavior of the energy consumptions during the training, the reconstructions or forecasts are usually less precise in presence of outliers, so extremely high or low values, thus enabling us to achieve satisfactory results for the downstream task of anomaly detection. Instead, the proposed approaches have higher difficulties in correctly identifying contextual anomalies, as they are characterized numerically by values which fall within the

normal range for the time series, and are thus harder to identify.

We proved, thanks to the comparison with another dataset, that it is not a problem of the models and their architectures, but rather of the task itself, which is harder than the way it is tackled traditionally, and of the dataset, which contains rapidly changing time series that can make it harder for the models to be able to follow the evolution over time of the measurements.

Finally, we assessed the capabilities of the trained models of generalizing over the real data provided by DATA CELLAR: it is in fact of great interest to identify anomalies in energy consumption measurements, to avoid wasting resources and money and to allow a better management of the LECs that are present in the EU, which aim is that of becoming more and more sustainable.

The analysis performed on the DATA CELLAR data was a qualitative one, as labels were not present: the models previously trained on LEAD dataset were used, at inference time, on this data and the respective reconstructions and forecasts were analyzed, to understand their capabilities of generalizing on new, unseen data. Such models were still able to work quite well, despite the differences between the real data and the LEAD dataset.

5.2 Future Works

The task of anomaly detection over time series is of great interest in many different fields, as it becomes increasingly more important to be able to identify abnormal behaviours in the data.

For what concerns the case of energy consumption data, there are many other approaches which could be explored, to understand whether it is feasible to handle at training time anomalous data in such a way that it becomes possible to achieve good performances for the anomaly detection task.

One possibility could be to further explore the world of GANs, which in this thesis were used via a model which reproduced their adversarial learning, while not being a proper Generative Adversarial Network.

I believe, though, that on this particular dataset the Deep Learning Forecasting, Reconstruction and Generative approaches could have reached their limit. It could be interesting to assess the capabilities of clustering approaches, which aim is that of creating clusters in which divide points which behave normally from those which constitute anomalies.

Finally, one could further explore thresholding methods, which are a very important part of the anomaly detection task: in fact, thresholds are those that allow us to identify whether a data point is to be considered as an anomaly or not, by evaluating the difference between the output of the model and the ground truth. In this work, several possibilities were tried, but it could be interesting to explore the possibility of using more dynamic approaches, as to better take into account changes for example within a specific window of observation, and create a more refined anomaly detection task.

Acknowledgements

First of all, I would like to thank my tutors at LINKS Foundation, Marco Galatola and Stefano Bergia, for giving me the opportunity of contributing even in a very small way to a European project like that of DATA CELLAR and for following my work, week by week, discussing ideas about how to approach the task in different ways to create resilient and efficient models.

A special thank you to Professor Garza, who was kind enough to accept being my academic tutor and to give me feedback regarding my work.

I want to thank my university colleagues, and friends, with whom I spent two amazing years and thanks to whom I grew so much as a person. It was truly an honor to get to know people like you and to share the hardships of the Master's Degree with you.

Finally, I would like to thank my parents, who were always there to support me and even tried to give me suggestions whenever I felt I was not going forward with my thesis work, even though they are not specialists in this field.

Thank you dad for listening to me and trying to understand which suggestions I could need to work on this project.

Thank you mom for being there for me whenever I needed to vent about the experiments not working.

This work is for you two.

Bibliography

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. «Anomaly detection: A survey». In: 41 (July 2009) (cit. on p. 6).

[2] Zahra Zamanzadeh Darban, Geoffrey I. Webb, Shirui Pan, Charu C. Aggarwal, and Mahsa Salehi. *Deep Learning for Time Series Anomaly Detection: A Survey*. 2024. arXiv: `2211.05244 [cs.LG]` (cit. on p. 6).

[3] Robert B. Cleveland, William S. Cleveland, and Irma Terpenning. «STL: A Seasonal-Trend Decomposition Procedure Based on Loess». In: 6 (Mar. 1990), p. 3 (cit. on p. 7).

[4] Kasun Bandara, Rob J Hyndman, and Christoph Bergmeir. *MSTL: A Seasonal-Trend Decomposition Algorithm for Time Series with Multiple Seasonal Patterns*. 2021. arXiv: `2107.13462 [stat.AP]` (cit. on p. 8).

[5] Tianqi Chen and Carlos Guestrin. «XGBoost: A Scalable Tree Boosting System». In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016 (cit. on p. 11).

[6] Nesryne Mejri, Laura Lopez-Fuentes, Kankana Roy, Pavel Chernakov, Enjie Ghorbel, and Djamila Aouada. *Unsupervised Anomaly Detection in Time-series: An Extensive Evaluation and Analysis of State-of-the-art Methods*. 2023. arXiv: `2212.03637 [cs.LG]` (cit. on p. 13).

[7] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-term Memory». In: *Neural computation* 9 (Dec. 1997), pp. 1735–80 (cit. on p. 13).

[8] Mohsin Munir, Shoaib Ahmed Siddiqui, Andreas Dengel, and Sheraz Ahmed. «DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series». In: *IEEE Access* 7 (2019), pp. 1991–2005 (cit. on p. 15).

[9] Oleksandr I. Provotar, Yaroslav M. Linder, and Maksym M. Veres. «Unsupervised Anomaly Detection in Time Series Using LSTM-Based Autoencoders». In: *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*. 2019, pp. 513–517 (cit. on p. 17).

[10] Mohammad Kardi, Tarek AlSkaif, Bedir Tekinerdogan, and João P. S. Catalão. «Anomaly Detection in Electricity Consumption Data using Deep Learning». In: *2021 IEEE International Conference on Environment and Electrical Engineering and 2021 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*. 2021, pp. 1–6 (cit. on pp. 17, 27).

[11] Haipeng Pan, Zhongqian Yin, and Xianzhi Jiang. «High-Dimensional Energy Consumption Anomaly Detection: A Deep Learning-Based Method for Detecting Anomalies». In: *Energies* 15.17 (2022) (cit. on pp. 18, 27).

[12] Tianyang Lei, Chang Gong, Gang Chen, Mengxin Ou, Kewei Yang, and Jichao Li. «A novel unsupervised framework for time series data anomaly detection via spectrum decomposition». In: *Knowledge-Based Systems* 280 (2023), p. 111002. ISSN: 0950-7051 (cit. on p. 18).

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML] (cit. on pp. 19, 20).

[14] Eoin Brophy, Zhengwei Wang, Qi She, and Tomas Ward. *Generative adversarial networks in time series: A survey and taxonomy*. 2021. arXiv: 2107.11098 [cs.LG] (cit. on p. 19).

[15] Umer Saeed et al. *Highlights Generative Adversarial Networks-enabled Anomaly Detection Systems: A Survey Generative Adversarial Networks-enabled Anomaly Detection Systems: A Survey*. Sept. 2023 (cit. on p. 20).

[16] Alexander Geiger, Dongyu Liu, Sarah Alnegheimish, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. *TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks*. 2020. arXiv: 2009.07769 [cs.LG] (cit. on p. 21).

[17] Zhijie Zhang, Wenzhong Li, Wangxiang Ding, Linming Zhang, Qingning Lu, Peng Hu, Tong Gui, and Sanglu Lu. «STAD-GAN: Unsupervised Anomaly Detection on Multivariate Time Series with Self-training Generative Adversarial Networks». In: *ACM Trans. Knowl. Discov. Data* 17.5 (Feb. 2023). ISSN: 1556-4681 (cit. on p. 21).

[18] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. «USAD: UnSupervised Anomaly Detection on Multivariate Time Series». In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. Association for Computing Machinery, 2020, pp. 3395–3404 (cit. on pp. 23, 42, 44, 68).

[19] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML] (cit. on p. 24).

[20] Keith Hollingsworth, Kathryn Rouse, Jin Cho, Austin Harris, Mina Sartipi, Sevin Sozer, and Bryce Enevoldson. «Energy Anomaly Detection with Forecasting and Deep Learning». In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, pp. 4921–4925 (cit. on p. 27).

[21] Zhe Zhang, Yuhao Chen, Huixue Wang, Qiming Fu, Jianping Chen, and You Lu. «Anomaly detection method for building energy consumption in multivariate time series based on graph attention mechanism». In: *PLOS ONE* 18.6 (June 2023), pp. 1–23 (cit. on p. 27).

[22] Muhammad Fahim and Alberto Sillitti. «An Anomaly Detection Model for Enhancing Energy Management in Smart Buildings». In: *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. 2018, pp. 1–6 (cit. on p. 28).

[23] Manoj Gulati and Pandarasamy Arjunan. *LEAD1.0: A Large-scale Annotated Dataset for Energy Anomaly Detection in Commercial Buildings*. 2022. arXiv: 2203.17256 [cs.LG] (cit. on p. 31).

[24] Jonathan Goh, Sridhar Adepu, Khurum Junejo, and Aditya Mathur. «A Dataset to Support Research in the Design of Secure Water Treatment Systems». In: Oct. 2016 (cit. on p. 45).

[25] Andrés García Mangas. *Data Cellar LEC API*. CTIC Technology Centre, 2024 (cit. on p. 48).