Unsupervised Time Series Outlier Detection

David Chaves

Computer Science, Database Technology, DT104F21, 2021-06

Master Thesis

Copyright © Aalborg University 2021

Composed by the author using LaTeX.

Computer Science
Aalborg University
http://www.cs.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Unsupervised Time Series Outlier Detection

Theme:
Database Technology

Project Period:
Spring Semester 2021

Project Group:
DT104F21

Participant:
David Chaves

Supervisors:
Tung Kieu
Bin Yang

Copies: 1

Page Numbers: 49

Date of Completion:
June 4, 2021

Abstract:

The continuous advance in processes digitalization has led to an extended availability of sensor-based devices that aims to improve and complete tasks efficiently in many domains. The breakthrough of using digital methods is driven by the massive production of data in form of time series, which allows steady monitoring of operations to take action when some relevant condition is reached, for instance, anomalies or outliers. Achieving that needs to develop automatic methods with no supervision since the available data is vast. The problem is addressed in this work, considering robust state-of-the-art methodologies and infrastructure with extensive experiments using real-world data sets from different domains.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Contents

Preface

Nowadays, the ubiquitous presence of digital devices and cut-edge technologies presents new and thrilling challenges for using them to handle and solve problems that otherwise are unmanageable. A recurrent exponent is sensor-based data, which is very valuable to evaluate procedures continuously, but requires the help of automatic processing because the data produced is massive. Thus, the current work tackles the problem of processing data in form of time series using no supervision for detecting atypical conditions recognized as outliers.

This work presents a model based on state-of-the-art deep learning techniques, including sequence-to-sequence processing for capturing temporal dependencies and ensemble modeling. An ensemble uses several basic models to achieve better performance, following the idea of the wisdom of crowds. Then, the ensemble is implemented using a diversity criterion to enhance its performance and using sharing knowledge and convolutional networks to reduce the running time.

The model is efficiently evaluated using a detailed schema that maximizes the resources used within a framework. The structure facilitates a fair comparison with several competitive baselines. This work makes contributions to amplify the framework scope including new tools and design the schema to the systematic evaluation for all models. Then, using that infrastructure, extensive experiments are performed over real-world time series from diverse domains, allowing a trustworthy evaluation with state-of-the-art methodologies. Finally, the work shows insights over the model efficacy to improve the accuracy and reduce the execution time.

Aalborg University, June 4, 2021

Chapter 1

Introduction

The availability of digital devices in the everyday life is currently widespread among many fields including numerous industries, scenarios, and applications. Accomplish the condition where many people around the world could access a mobile phone, or including digital features in industrial machinery is feasible, became possible as part of continuous miniaturization and optimization of these technologies. Thus, when the electronic circuits are smaller and they have a low energy consumption, their possibilities of use are expanded to a great range of settings.

The sensors, which are devices that monitor continuously a specific activity, have been very beneficial for the development of new technologies. They are enhanced in terms of capabilities and are smaller and efficient, which allows their implementation inside consumer products and industry-related machines, such as mobile phones and scientific instruments. Therefore, they have a very extended scope including tasks such as counting the steps of a phone user or monitoring a pond water level, allowing the design of new innovative applications.

On a relatively small scale, the readings from a sensor could be interpreted and analyzed by people. For example, a weather station with five measuring instruments and updates every minute would be checked periodically by a meteorologist in charge of a small number of stations. Even so, as the precision of the instruments is increased, with several updates per second, and the number of sensors is larger, considering more variables, the monitoring task becomes unmanageable for being supervised by operators. For that reason, the development of algorithms that allows processing big amounts of data with no supervision is becoming highly desirable, since they perform the monitoring tasks and reduce a repetitive workload for the experts of the field.

The readings associated with sensors are large time-ordered sequences of data recognized as time series. Then, a particular activity could be monitored at the same time evaluating different variables. For instance, a weather station has sensors for precipitation and wind, so it is possible to analyze both readings together

as they occur at the same time. In that case, the time series is multivariate, while the scenario with only one is called univariate.

Other scenarios that use time series involve settings with high-frequency updates, which represent several readings for each time unit. For example, the financial market indicators are usually represented by time series since they are driven by timestamped transactions around the world. Other cases involve usage metrics for products and services, such as the server workload during a day or data flow through a network.

Independently of the field, time series usually involved the same kind of analysis. For example, identifying when the monitoring conditions resemble a known scenario, or when they are changing significantly concerning the normal operation. The first case is recognized as time series classification and helps, for instance, to identify activities such as running, walking, or sleeping for a person wearing a heart rate monitor. Then, the second scenario is called outlier detection, which supports the tasks for recognizing anomalous conditions that need to take some type of action. For example, detecting low blood glucose levels with a wearable would prevent that a person loses consciousness if an action is taken on time.

Outlier detection analysis is becoming a very important topic in recent years in connection with the growing availability of time series data and the intrinsic difficulty of processing it on time. Regardless of the scenario, identifying anomalous conditions promptly is very critical, since it helps to prevent inconvenient situations where an important system can be out-of-service or a health problem could arise. Therefore, the models designed to address this problem should focus on working completely unsupervised and providing high levels of accuracy, supporting the prevention of major issues.

In recent years, with the parallel thrive of powerful hardware for processing deep learning models, the outlier detection subject is getting great attention since this kind of model is effective for identifying anomalies in an unsupervised setting. Thus, to the extent that new deep learning techniques are continuously presented for several domains, it is very suitable to design new models addressed particularly for processing time series in an outlier detection setting.

The current work presents a model that considers state-of-the-art deep learning techniques inspired from different fields to address the outlier detection problem. Then, the model is incorporated into an under-development framework that considers many baselines to be evaluated fairly and homogeneously. The complete system is implemented in a database-friendly setting which contributed to efficient processing, faster performance evaluation, and straightforward results reporting. The complete scope for this work is shown as follows:

1.1 Outlier Detection Model

Following the effectiveness achieved by autoencoder-based techniques for unsupervised outlier detection [1], this work presents a model considering the autoencoder principle with the next improvements:

- Convolutional sequence-to-sequence: inspired by language processing and translation works [2], the method processes time-series considering temporal dependency between observations in a given sequence. Thus, one point is evaluated by its value and its relative position across the sequence. Also, as the method is convolutional, it takes advantage of Graphics Processing Units (GPUs) for a faster execution than previous methods that relied upon recurrent and low-paralleled calculations.
- A diversity-driven ensemble: the model introduces the use of diverse ensembles [3] for the outlier detection setting. Therefore, the selection of single models is guided by how relevant the new model is for the current setting, a fact that previously depended on random choices.
- Transfer parameters between models: derived from the ensemble modeling, to reduce the execution time, a portion of the knowledge for a single model is shared to the new one, reducing the processing workload since it is not necessary to re-learn all the parameters.

The model is evaluated against several baselines using an extensive in-development framework with real-world data sets, demonstrating its effectiveness in terms of accuracy and execution time.

1.2 Evaluation Framework

The model and the baselines are evaluated using an outlier detection framework that is maintained by several developers. As part of this work, the following features are included in the system, allowing to use them to evaluate all applicable models:

- Time-series processing for linear baselines: linear outlier detection methods are not designed for managing time series and their particularities, such as temporal dependency. Thus, a method for preprocessing the data is introduced to make fair comparisons concerning the deep-based techniques.
- Time-series data augmentation: to derive more cases for evaluating the methods using the same data sets, an augmentation process was deployed to included artificial shifts and slopes inside the time series, in order to evaluate the robustness of the methods in relatively different scenarios.

- Validation process for autoencoder-based methods: inspired by the work of [4] for evaluating their model parameters, a similar process was included in the framework for choosing the parameters for autoencoder-based methods. It helps to reduce the uncertainly in unsupervised models since it is possible to define a criterion for selecting parameters.
- Data management: to improve the efficiency of the complete evaluation process, a database engine was included in the framework to support how the input data and the results are distributed across several servers. Thus, the data access is centralized, reducing space consumption and facilitating model reporting and evaluation.

1.3 Execution Procedure

The outlier detection framework is supported by a database engine and a group of tools that are organized in a procedure to maximize the available resources and are prepared for future scalability. It includes the following components:

- Code repository for homogeneous distribution of sources between different execution servers.
- Centralized data sets manager for automatic data provisioning according to the experiment requirements.
- Database engine for storing the results derived from the experiments, facilitating the reports design.
- Reporting tools directly connected to the database to evaluate each model performance.

1.4 Contributions

As a summary, the current work makes the following contributions:

- It presents a deep-learning model for outlier detection that outperforms the state-of-the-art methodologies in terms of accuracy and running time.
- It introduces a group of techniques for enhancing the evaluation of outlier detection methods via a standard framework.
- It deploys a schema for evaluating outlier detection methods focusing on the efficient use of storage and computation resources.

1.5 Document Organization

The document is organized as follows. Chapter 2 introduces some necessary concepts for a comprehensive understanding of the developed method. Chapter 3 presents the model for outlier detection, evaluating its novelty concerning the current available techniques. Chapter 4 outlines the model inside the evaluation framework and the contributions to this system, while Chapter 5 presents the overall implementation for maximizing the available resources. Then, the model is evaluated and compared to several baselines in Chapter 6, which results are discussed in Chapter 7. Finally, related work is reviewed at Chapter 8 and Chapter 9 provides conclusions and future work.

Chapter 2

Preliminary Concepts

The concepts examined in the following sections are important to understand the model since they support the ideas that contribute to its development. They are reviewed including some examples that help to illustrate them.

2.1 Time Series

Usually, they are recognized as sequences of data ordered by time. Formally, a time series $T = \langle t_1, t_2, \ldots, t_C \rangle$ is a sequence of C observations, where each data point $t_i \in R^D$ is a D-dimensional vector. If $D = 1$, T is a *univariate* time series. If $D > 1$, T is a *multivariate* time series.

In Fig. 2.1 an example for a *multivariate* time series is shown, where each color represents a different variable, so each t_i is represented by a two-dimensional vector. As many variables are included, the dimensions for the series are increased.

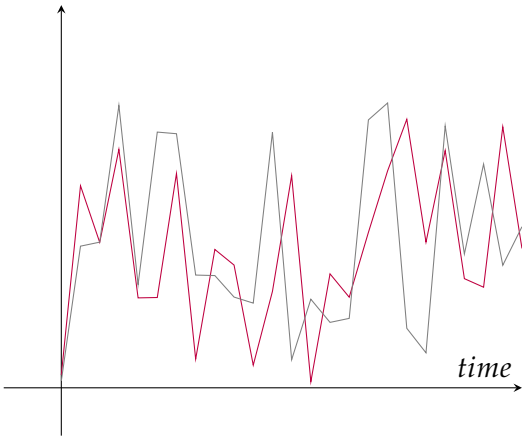

Figure 2.1: Multivariate time series example.

2.2 Unsupervised Learning

It consists of a type of algorithm that learns representations from the data with no requirement of labels. In contrast, the supervised setting uses labeled data, which requires the intervention of humans. For example, a classifier for tagging pictures needs to be trained using examples where the images have a label, previously assigned for an operator. In a similar unsupervised setting, the algorithm will not assign tags to the pictures, but instead, it will able to define groups for the images that share some characteristics. Fig. 2.2 shows an example, where the supervised model is trained with tagged birds and dogs, and then, it is evaluated with an unknown example, so the model assigns a label. In the unsupervised case, the model uses unlabelled examples, so it aims to classify them between unnamed groups.

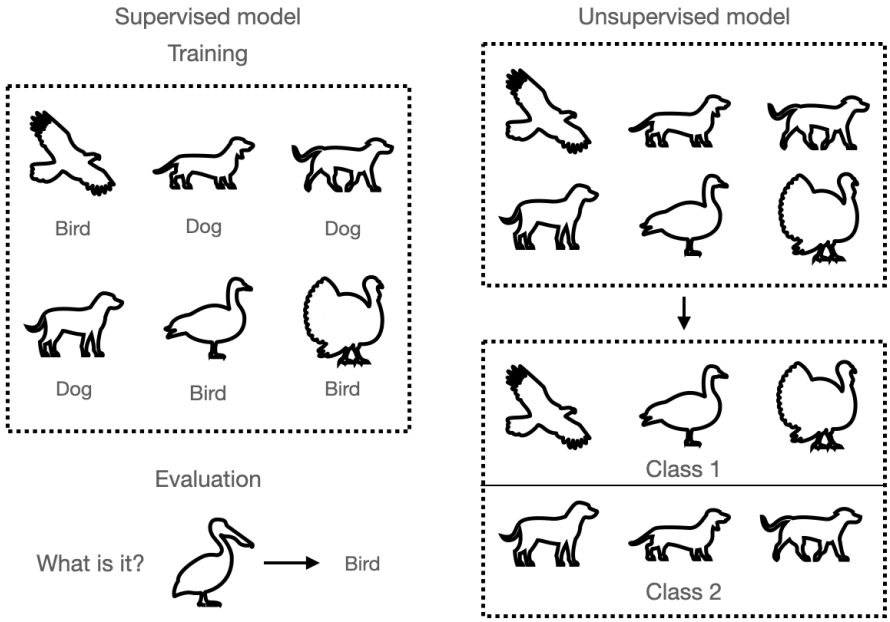

Figure 2.2: Supervised and unsupervised models example.

Unsupervised algorithms are particularly relevant because they do not require people who assign labels to the data. Thus, they could manage extensive amounts of data and provide insights with no intervention. Also, they can identify facts that people could miss, such as subtle differences between similar examples. For those reasons, the development of unsupervised algorithms is a topic of continuous interest, coupled with the fact the available data in many fields is steadily growing. Thus, it is likely that processing those scenarios becomes unmanageable being necessary to automatize algorithms as much as possible.

2.3 Autoencoder

It is a type of algorithm consisting of two parts, encoder, and decoder. They are used to compress an input into a small representation and then try to decompress it to resemble the original data. Formally, the first part encodes a D-dimensional input $\mathbf{x}$ into an intermediate and compact vector $\mathbf{h} \in R^M$, where $M < D$. Then, the decoder takes the intermediate vector $\mathbf{h}$ and produces an output vector $\hat{\mathbf{x}} \in R^D$ that aims to resemble the input. The goal is producing an output $\hat{\mathbf{x}}$ with very little difference to the input $\mathbf{x}$.

In Fig. 2.3 an example is shown considering a picture. In the encoding phase, the image is compressed to a small representation, and using it the decoder tries to reproduce the original input. Even so, it is noticeable that during the process some information is lost, for instance, an output with no fill.

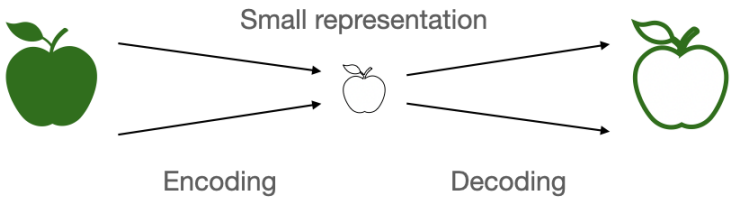

Figure 2.3: Autoencoder example.

The gap of missing data between the input and output is a valuable insight for autoencoder modeling because some features could be learned using that property. For example, the model could compress only the most relevant parts for the input, dismissing relatively uncommon facts. The attribute is especially useful for detecting outliers because if the model is reconstructing the most relevant segments, it is likely that it will miss anomalous points since they are unusual.

Using autoencoders to reconstruct time series for identifying atypical points was introduced by [1] and it is illustrated with an example in Fig. 2.3. For that case, an original input x is reconstructed by the output $\hat{x}$, which only outlines a general trend for the series. Then, comparing the input to its reconstruction, it is possible to delimit outliers as the differences at those points are possibly higher.

2.3.1 Outlier scores

For time series $T = \langle t_1, t_2, \ldots, t_C \rangle$, the computation of an outlier score $OS(t_i)$ for each observation t_i is a value such that the higher $OS(t_i)$ is, the more likely is that the observation t_i is an outlier. Therefore, the difference between the input and output in autoencoders, asserted as reconstruction error, is a suitable property to be used as outlier score.

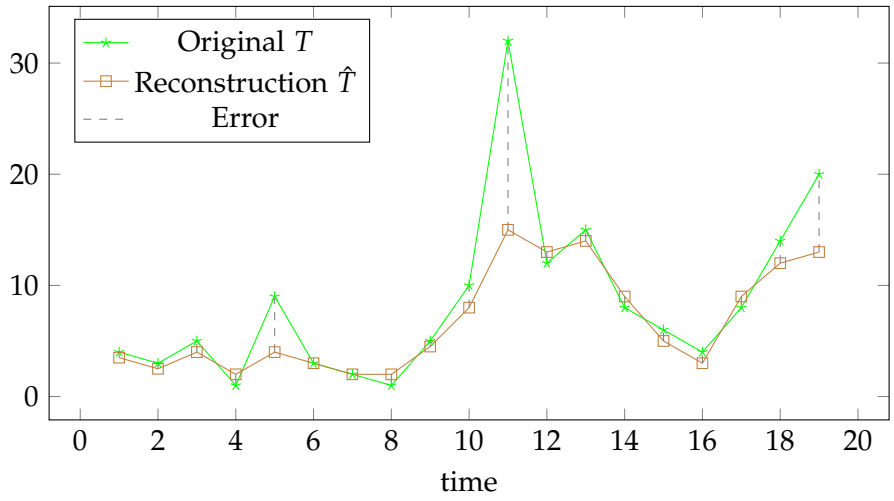

Figure 2.4: An autoencoder reconstruction example for time series.

2.3.2 Outlier

Considering a time series $T = \langle t_1, t_2, \ldots, t_C \rangle$, an outlier is an observation $t_i \in T$, where t_i has an score $OS(t_i)$ which is larger than a specific threshold ϵ. Thus, considering the Fig. 2.3 and a higher ϵ, it is likely that the observation t_{11} is considered as the only outlier. Then, using a lower ϵ is possible that also the timestamps t_5 and t_{19} are classified as anomalous points.

2.4 Sequence-to-Sequence Analysis

It is a technique commonly used for text processing and language translation, which consists of processing the data in smaller sequences, such as sentences. Thus, a sequence input from one domain could be processed as a whole and get an output that reproduces it in another domain. For example, in Fig. 2.5 a translation is illustrated, where the sentence in English is processed completely, so the output in French will reproduce the same meaning, which has fewer words.

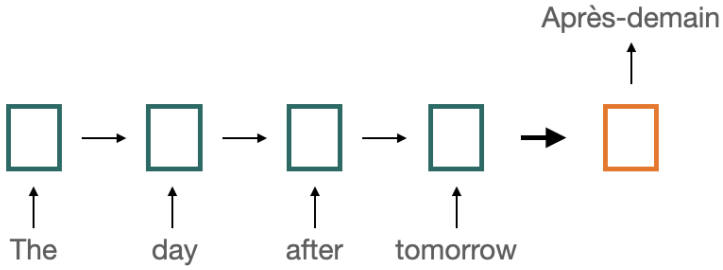

Figure 2.5: English to French translation example.

Processing sequences completely provide a context, which is a valuable insight for reproducing it in other domains since the surroundings for a particular observation are recognized. For instance, in language translation, if the sequence is processed word by word, the context is lost, so the sentence meaning could be completely different. For time series, this type of processing helps to identify data properties such as temporal dependency and trends.

In terms of execution, sequence-to-sequence techniques usually are recurrent tasks, since each observation requires the precedent ones to build a context. Therefore, they have a reduced degree of parallelism, since each sequence requires to be executed as a whole.

2.5 Ensembles

It corresponds to an extensively used technique in machine learning tasks where single models are joined to build a single one that points to be the optimal representation for all cases. In some implementations, an ensemble can act as a voting system, where single models are trained independently to achieve a result and then, the overall result is considered as the final output.

Fig. 2.6 shows an example where four classification models try to predict a label for a picture. In independent conditions, two models failed to assign the labels, while using the ensemble it is possible to get the correct answer, even when some models are incorrect. Thus, single model output is somehow unreliable in comparison to an extensive overview, a reason why the ensemble models are useful in unsupervised settings, where the uncertainty is higher.

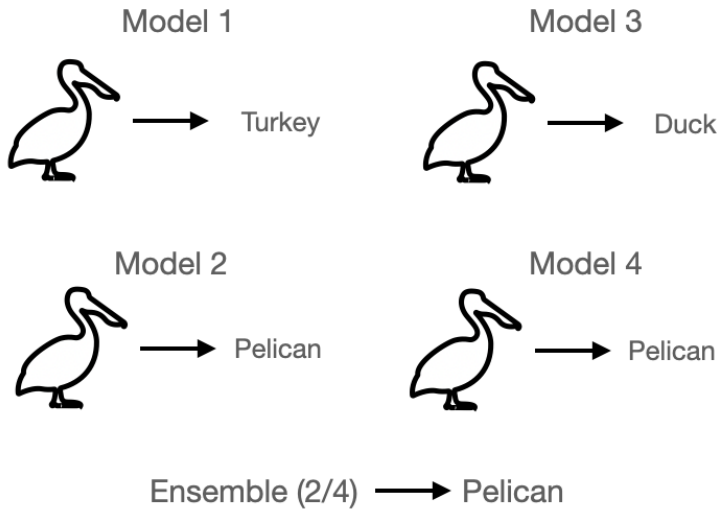

Figure 2.6: Ensemble as a voting system example.

Chapter 3

Outlier Detection Model

Current deep learning methodologies used for outlier detection mainly rely on autoencoder-based methods, since they established a frame for unsupervised modeling. Thus, the model reviewed in this work, called `CAE-Ensemble`, uses this type of neural network, introducing recent techniques that help to improve the overall performance in terms of accuracy and execution time. They are detailed in the following sections and are based on the concepts reviewed in the previous chapter. The complete model was developed as part of a previous project [5], and it is currently in process for publication [6].

3.1 Convolutional Sequence-to-Sequence

As an alternative for the recurrent Sequence-to-Sequence shown before, the approach presented by [2] establishes a different way for managing positional data in sequences of text. Thus, it addresses the drawback for recurrent methodologies concerning the parallel execution of each sequence, since it is possible to complete the computation without losing the positional information.

Redefining the example shown in Fig. 2.5, in Fig.3.1, a translation case is shown, where each word now has an associated position, running through a parallel process with no dependency for the result from previous stages.

Using that approach, each sequence could be processed in parallel, since the position information is maintained during the complete process. Also, it enables to use of convolutional operations, taking advantage of specialized hardware for faster processing. To achieve that, it is necessary to examine how the position is managed, which is detailed next.

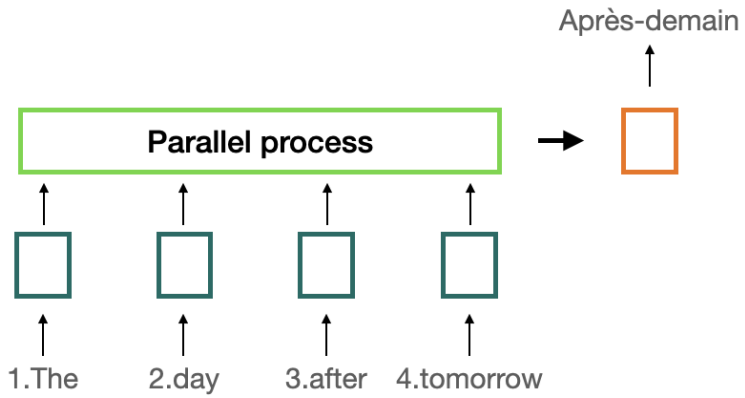

Figure 3.1: Parallel English to French translation example.

3.1.1 Positional input

On text processing, the words are transformed into vector representations, which are easier to compute. Similarly, in time series, all the observations $t_i \in R^D$ are considered together as a vector representation, which is analogous to a word into a sequence. Thus, the computation for both cases is very similar since they could be represented in the same way with no distinction for their original domain.

Considering a multi-dimensional time series $T = \langle t_1, t_2, \ldots, t_C \rangle$, represented in a distributional space as $V = \langle v_1, v_2, \ldots, v_C \rangle$ where each $v_i \in R^{D'}$ is a vector representation in D' dimensions for the original input. Then, for the same sequence, it is possible to specify the absolute position $M = \langle 1, 2, \ldots, C \rangle$ for each observation in an embedded representation $P = \langle p_1, p_2, \ldots, p_C \rangle$ where each $p_i \in R^{D'}$.

Next, both vector representations that incorporate the actual data and the position information, could be joint as the sum $x = \langle v_1 + p_1, v_2 + p_2, \ldots, v_C + p_C \rangle$, which constitutes the input for the convolutional computation. The transformation is shown in Fig. 3.2, where a three-dimensional time series is represented in a vector space, including the relative position in the series.

The process continues similarly to the text processing case, where a Convolutional Neural Network (CNN) autoencoder processes the input x and returns its reconstruction $\hat{x}$, which maintains its positional information. Thus, each observation will be appointed considering its relative position inside the original series.

3.2 Diversity-Driven Ensemble

The ensemble modeling is limited by the fact that it is not possible to make distinctions between singles models. Thus, for the example shown in Fig. 2.6, there is no mechanism to weight better the models that classify correctly in detriment to the inexact cases. For that case, each model has the same voting influence regardless

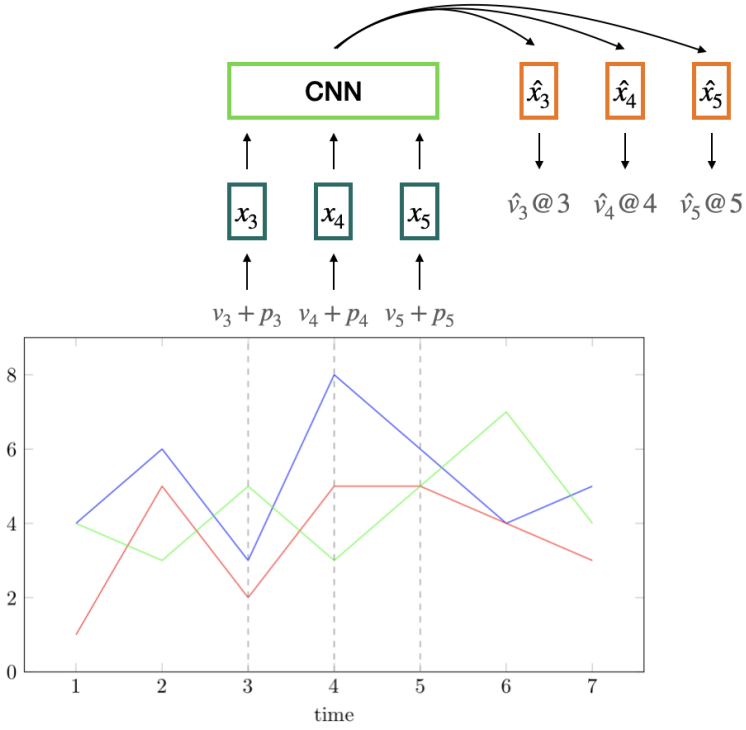

Figure 3.2: Convolutional sequence-to-sequence in time series.

of its actual output.

To address this problem, the work presented by [3] defines a method using the lost function to discern some differences between single models, allowing a better selection to improve the unified output. It is a useful approach, since it provides a guide for choosing sequential ensemble members, a task that usually relied on random assignations. Thus, a similar process was derived to use a diversity measurement to choose single models in an unsupervised setting, for outlier detection.

3.2.1 Diversity measurement

It is a recognized fact that ensembles with diverse members are more accurate [7], meaning that single models with different structures will benefit the overall output. Therefore, if the most diverse models are identified, it is possible to join them achieving a better performance than basic models and regular ensembles. Thus, the proposed model defines a diversity measure that allows distinguishing the models according to their degree of diversity.

The measurement considers a model N_t and defines its degree of diversity by computing by the mean difference between it and the current ensemble E_t, modified by a factor of θ that represents the degree of expected diversity, as it is

shown as follows.

$$Div_t = \theta \times ||N_t - E_t||_2 \tag{3.1}$$

Then, the measurement is included as a penalization inside an enhanced loss function L_t that consists of the result for that metric, such as Mean Absolute Error (MAE), including the diversity component.

$$L_t = MAE + Div_t \tag{3.2}$$

$$L_t = MAE + \theta \times ||N_t - E_t||_2 \tag{3.3}$$

The diversity component inside the loss function sets the relative relevance for each new basic model concerning the current ensemble. Thus, if a new model is more different, it will weigh distinctly in terms of its loss function than another model that is very similar to the previous ones.

3.2.2 Transfer learning

An additional limitation for ensemble modeling concerns the necessary time for training multiple sequential models. In a simple implementation where a single model takes a training time of H, it is likely that if I models are trained, the total running time will be at least $H \times I$. Thus, despite the benefits derived from using ensembles, their results usually take longer.

One remarkable observation regarding the ensemble members is that they share many parameters since every model has the same setting. For example, in an image classification problem shown in Fig. 3.3, every model should process the pixels and edges first, even when the output could be different. Thus, if the structure for the models is similar it seems unnecessary to learn the parameters associated with those steps because some facts such as the input are not changing.

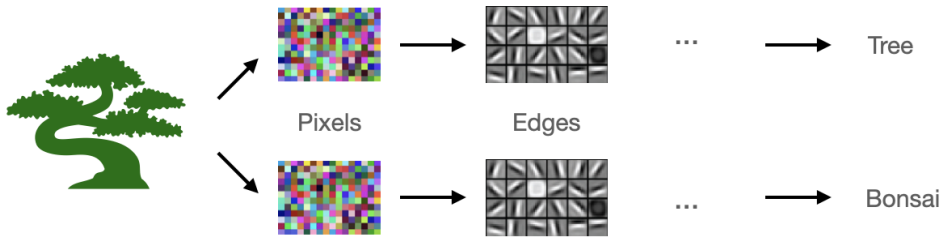

Figure 3.3: Convolutional image classification process for two models and the same input.

Reducing the running time for an ensemble is possible taking advantage that some parameters are already learned in a previous model. So, the knowledge could be transferred to the next model, which will be executed faster, since it only needs to train a small portion of parameters. This process is implemented by selecting a percentage of parameters that will be available in the following basic ensemble model, as is illustrated in Fig. 3.4. The logic is that a completed trained *Model 1* transfers a β portion of the knowledge to *Model 2*, which requires less time to train the remaining parameters. Thus, it is possible to reduce the overall execution time, since every model could transfer parameters to the next one, decreasing the necessary training at each stage.

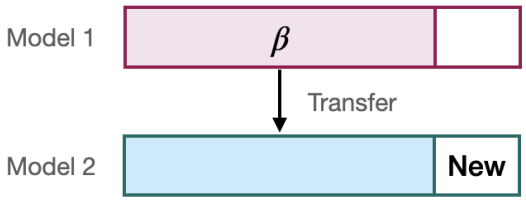

Figure 3.4: Transfer learning example.

3.3 Model Overview

The complete model overview is shown in Fig. 3.5, considering the necessary steps from processing the original data to obtaining the metrics to evaluate and analyze it. The integration into the framework and the execution schema will be review in the following two chapters.

In the first step, the original univariate or multivariate times series are pre-processed before start the training as it will be reviewed in Section 4.4.1. Then, the already processed input data is used for all ensemble members, which are trained sequentially, which allows to transfers of a portion of parameters β to the following cases, reducing the overall running time. Also, they are subjected to the diversity loss function to increase the ensemble diversity, a property that aims to improve the model accuracy.

After training the ensemble, the model is tested considering the reconstruction error, which allows defining the outlier score for each observation. Then, those results are evaluated according to a defined threshold, which could be derived from some domain knowledge or expert criteria. Using that threshold it is possible to classify the points as outliers or normal data.

Finally, using the model results, represented by the observations classified between two categories, the threshold dependent and non-dependent metrics are calculated to evaluate the model. Thus, it is possible to compare its performance

with other models or assess its accuracy with respect to other settings.

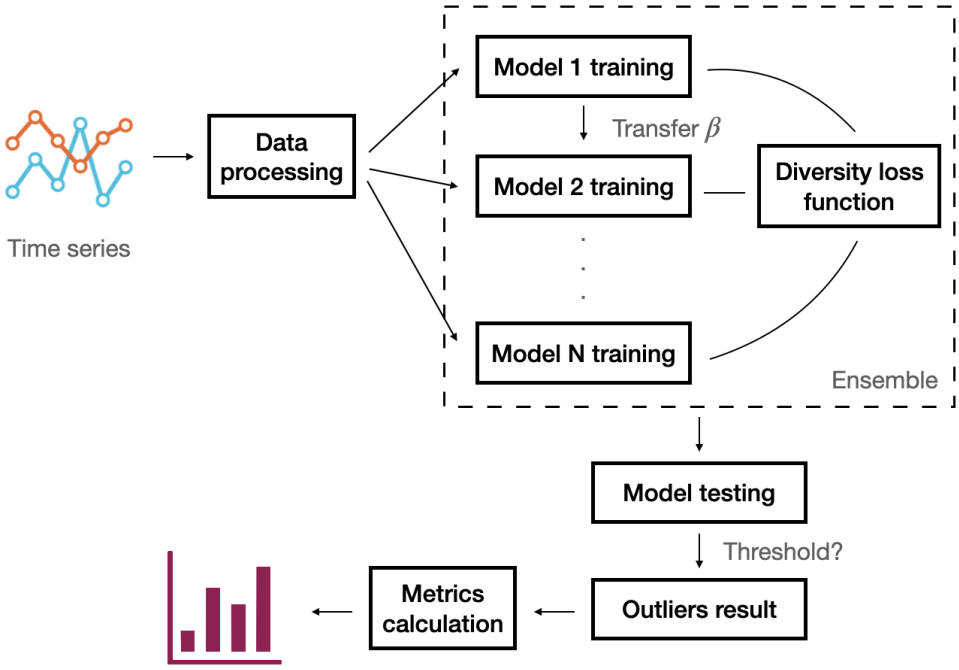

Figure 3.5: Overall overview for `CAE-Ensemble` model.

Chapter 4

Framework Contributions

The model reviewed in the previous chapter was integrated into an outlier detection framework maintained by different developers. It is written in Python language and has several components such as resources management, notification services, and recognized baselines to evaluate models. Additional features are under different stages of development, including an external Graphical User Interface (GUI) and the inclusion of recent models presented in the field [8].

As part of the current work, some components were developed in order to test the reviewed model performance and evaluate it in comparison to other baselines. They are available for all applicable models and could be modified for future changes and improvements. The details for these contributions are shown as follows.

4.1 Non-deep Learning Baselines Data Processing

As part of the `CAE-Ensemble` evaluation, it was found that non-deep Learning models, for example, Isolation Forest (`IS`) [9], shown an inconsistent performance, arguably related to not being design for processing time series. The problem was addressed by including a data processing method before running this kind of model. It consists of partitioning the time series into smaller windows and processing them separately. Then, to emulate some degree of temporal dependency, the windows are overlapped, meaning that each case differs from the previous one only by one step.

The complete process is detailed in Fig. 4.1 for a time series of size five. It generates four cases for a window of size two, which constitutes the number of non-deep learning model executions. Then, each model outputs a score for all the processed observations, which are joined considering only the last observation for each model, given an output equal to the number of executed models.

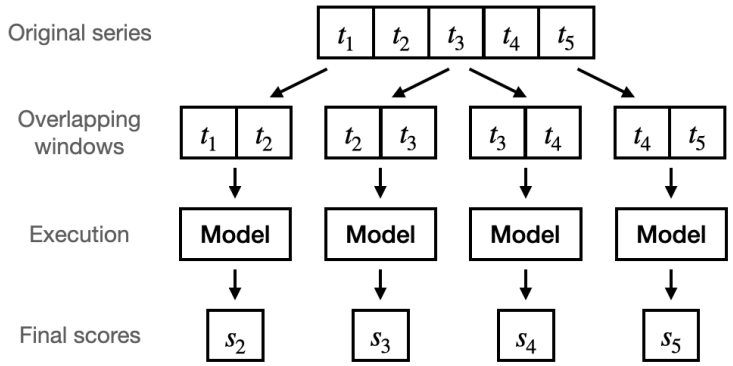

Figure 4.1: Data processing for non-deep learning baselines.

4.2 Data Augmentation

A significant group of data sets for outlier detection comes from sources related to monitoring tasks, such as sensor readings and server usage. One drawback for this type of data involves that usually it does not includes some necessary scenarios for an outlier detection algorithm. For example, a method should consider changes in data that are not necessarily anomalous points, such as the differences in network usage between weekdays and weekends.

Another case involves slow changes over time, for instance, a growing number of website visits during a year. An algorithm must consider that property to avoid false positives like identifying an outlier at the end of the year just because the number is higher than at the beginning.

Addressing the special-case scenarios for detecting outliers and using data that does not necessarily include them, involves augmenting its scope. The goal of this technique is to increase the availability and variability of the data introducing slight and artificial changes without losing generality. For instance, rotating an image helps to train a classification algorithm for managing more cases based on the same picture, so the label is preserved. It is illustrated in Fig. 4.2 where regardless of the position, it is clear the figure retains its label.

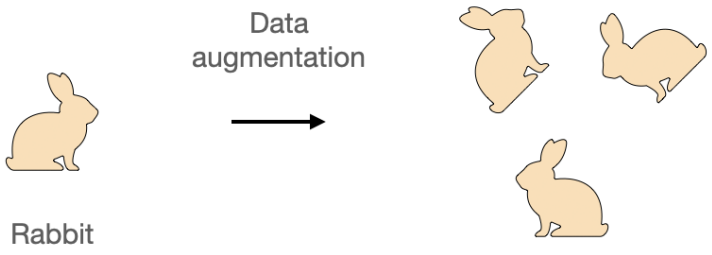

Figure 4.2: Data augmentation for a picture. Extra observations preserve the label.

For time series, the scenarios with trends or regular data changes could be introduced via augmentation techniques. To exemplify that, consider a multivariate time series with no trends or changes, and an identified anomalous point at t_7, as it is detailed in Fig. 4.3.

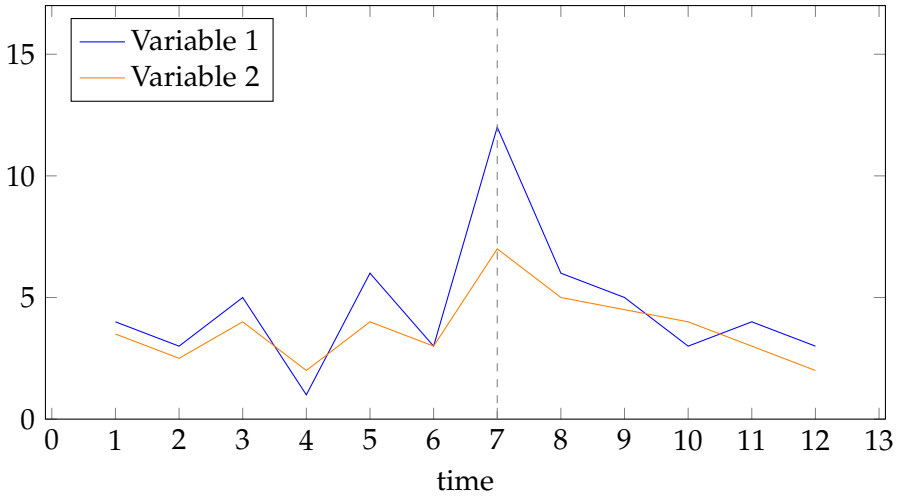

Figure 4.3: An multivariate time series with a labeled outlier at t_7.

Then, it is possible to settle a temporary change in one segment of the series, for example, in the second third of the sequence. Thus, the values on that area are modified homogeneously by a defined shift, such as adding two units, as it is shown in Fig. 4.4.

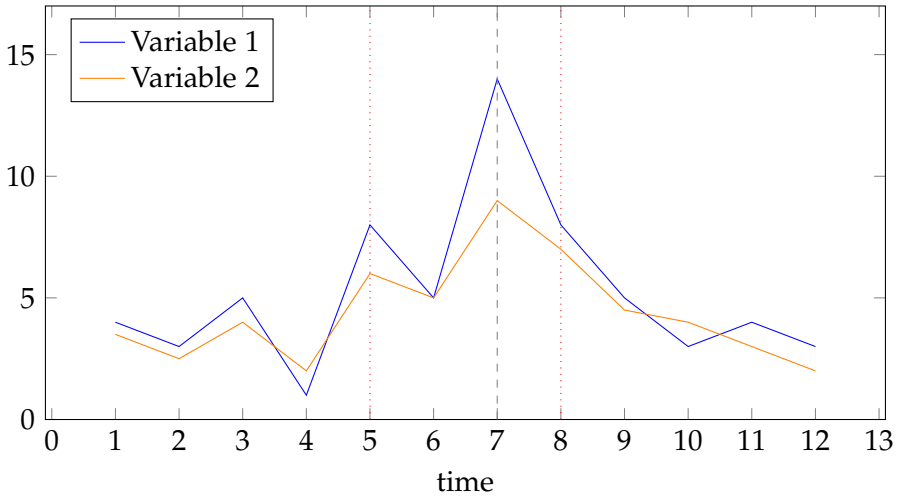

Figure 4.4: An multivariate time series with a shift of two units in the central segment.

The change, depicted between observations t_5 and t_8, still preserves the outlier

tag at t_7 and introduces special scenarios that are valuable to evaluate. For instance, an algorithm could misidentify the boundary points for that range as anomalous if the temporal change is not recognized.

Then, following a similar approach, a trend could be included for the entire sequence modifying the values for all variables by a defined slope. In Fig. 4.5, a slope of 0.6 is introduced to the entire segment, simulating a positive trend. The modification allows evaluating if algorithms can manage a trend without identifying anomalous observations just because they have a high value. For example, values at the end of the sequence are relatively high, but it does not mean they are outliers.

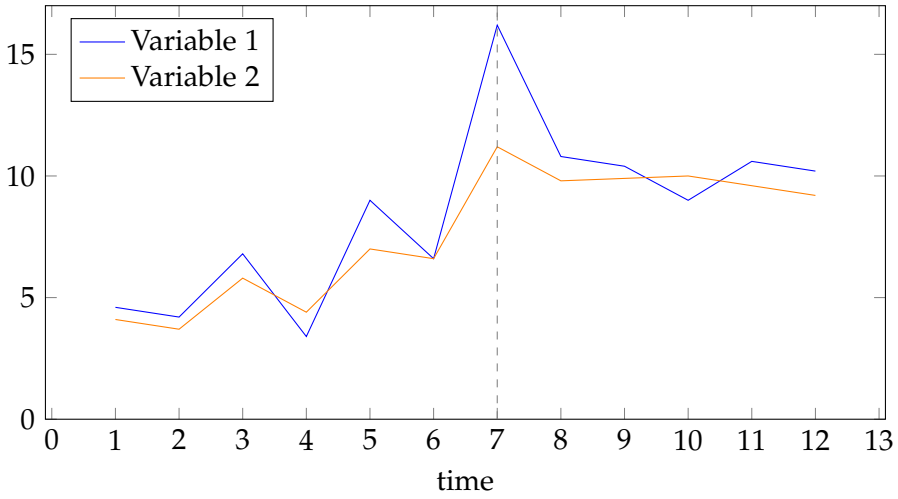

Figure 4.5: An multivariate time series with a 0.6 artificial slope.

Both methods are built into the framework allowing to test any method using the same data sets with settings that reproduce special conditions for time series. They are parameter-driven, so for any case, it is only necessary to specify the type of augmentation (temporal shift or slope) and its specification (portion of affected data and its degree).

4.3 Validation Process for Parameter Tuning

Using autoencoders for outlier detection relies on the reconstruction error (view Section 2.3), which besides its role for scoring data points could be used for evaluating the selection of parameters, as it is proposed by [4]. In that case, the model uses the number of dimensions as an adjustable parameter, so they evaluate their choice with respect to the associated reconstruction error. They do not use that information to make a decision because their results are not conclusive, but the approach is applicable in other scenarios.

As the `CAE-Ensemble` model requires the choice for λ and β parameters, the method for using the reconstruction error for driving this selection is appealing because it reinforces the unsupervised setting approach. Also, as several baselines depend on the autoencoder technique, the methodology is extensible for choosing any parameter defined for each particular case.

The method for tuning parameters considers selecting a portion of the training set for validation purposes, for example, reserving a 30% for adjusting them. Then, the model is trained using different parameter configurations, and each one is evaluated using the validation set to calculate the corresponding reconstruction error.

The procedure is assessed independently for each parameter, meaning that the remaining ones are constant during the evaluation. That technique is used for reducing the training time because if all possible combinations are considered, the validation process will be very slow. For instance, if it is necessary to adjust three parameters with ten scenarios by case, a complete evaluation will take $10^3 = 1000$ training models, while if they are independently evaluated, only $10 \times 3 = 30$ cases are considered. The solution is not necessarily optimal, but it is a good choice considering the positive trade-off in terms of execution time.

After calculating the reconstruction error for each parameter validation, the choice is based on the median values, as it is shown in Fig. 4.6 for five cases, where the value b is chosen for model testing. The criterion for using the median error is based on the fact that the smallest error is usually associated with overfitting, while a very high value reflects a poor model performance. Thus, it is necessary to select a balanced parameter, where a median is a suitable option.

Parameter scenarios	**a**	**b**	**c**	**d**	**e**
Reconstruction error	14	22	34	17	28

Median result

Figure 4.6: Reconstruction error example for choosing a parameter value.

The process is applied in `CAE-Ensemble` for choosing the β and λ parameters and could be applied for other models based on autoencoders. Also, the technique is used for selecting the processing windows size, which is a variable that applies to all baselines.

4.4 Data Management

On a relatively small scale, the evaluation and testing of an analytic model are usually simple because all the results could be managed as part of the execution process. Even so, as it becomes necessary to evaluate more models with different configurations, the task could be complex, requiring a more organized methodology such as a standardized framework.

For the outlier detection case, the systematic infrastructure facilitates how each model works, for example, defining a homogeneous input that is suitable for any method. However, as the system scaled, it was certain that it needed some adjustments since many experiments over several models and data sets were expected. Also, it was possible to use several high-capacity servers, which involved recurrent executions and the requirement for managing the related data accordingly.

Handling the scenario where multiple models run and interact with data across several servers requires the incorporation of a database engine. It is a feature that enhances the framework capabilities bringing scalability and reducing its direct interaction with data sets, for example, reading or writing files. The component works for all baselines and addresses the scenarios where data is involved as it is detailed as follows.

4.4.1 Input data

Data for outlier detection come from different sources without following a standard, since every release depends on the monitored scenario and its design properties. Thus, before executing any model, data should be processed to get a conventional format, including partitions and labels.

The cleansing process, detailed in Fig. 4.7, shows the necessary steps to make the data ready for use in any model into the framework. It includes reading the data from the sources and matching the labels, that usually come from technical reports. Also, it is necessary to apply normalization to have a common scale, and introduce any additional process such as the augmentation reviewed in Section 4.2 or sampling large series.

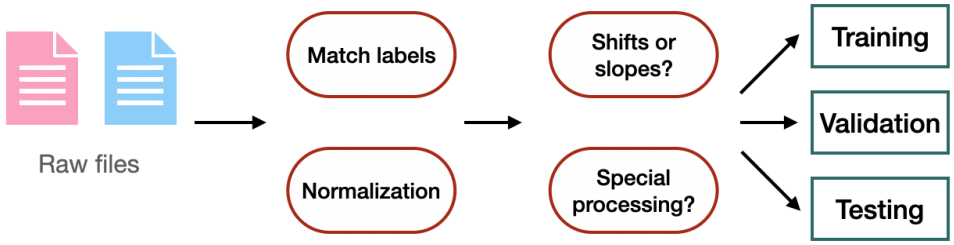

Figure 4.7: Data sets cleansing process.

Even when the detailed process could be run as part of the model execution, it is not efficient, because the tasks are repeated in many cases, for example, comparing models in a given data set. Also, in a scenario with multiple servers, data should be copied in all of them, which could use space unnecessarily. Therefore, the process is segregated into an independent task that stores the processed data into a database, which is accessible on demand for all the models.

The integration of the database engine in this part of the process facilitates how the framework interacts with input data, reducing the execution overhead because training, validation, and testing sets are ready to use. Also, it decreases the space consumption, since only the necessary data is retrieved for executing an experiment. Another advantage is that provisioning new servers is straightforward, the only requirement is having access to the database.

4.4.2 Results management

Considering the framework deployment over different execution servers, a problem regarding how managing the results arouse. In a centralized implementation, all experiments could be saved locally, for example, in a single file or a standalone database. Even so, as many servers run the models, the task for consolidating all the results is time-consuming, because it implies merging the results between implementations, tracking updates, among other tasks. Therefore, the process for getting the results is modified to store them in a centralized database, which manages and controls all data regardless of the source.

Handling the experiment outputs with a database engine provides mainly two benefits. First, it maintains data integrity, preventing duplicates and incorrect insertions, while multiple clients can access it. Second, it eases the consolidation of reports over the experiments even if they run across several servers. For example, getting the current status and progress for a running model.

The implementation of this component, which includes the computation of results and the insertion in the database, runs independently to the model training. Thus, it maximizes the use of resources during the overall execution because it runs in parallel to other tasks. Also, the database engine is connected to terminal spreadsheets with already built reports, so the results are always updated as the experiments are running.

4.5 Derived Work

During the model and framework development, some minor tasks were completed, which are briefly described as follows.

4.5.1 New data sources

New data sets were added to the framework following the structure and process detailed in Fig. 4.7. It helped to understand better the task and migrate altogether to the database approach.

4.5.2 Adjusted points

Following a method used in recent outlier detection works [4, 10], an additional module was developed to adjusted the results for any method according to the ground truth labels. It tries to reproduce the behavior of a human in the task of detecting outliers, who, when find one will mark a complete area as anomalous. The method was tested and it is available for use, but the results seem artificial, and they partially relied on the labels, which make the methods supervised.

4.5.3 Model integration tasks

Coding tasks related to migrating the code from a previous implementation to the framework involved segregating the model completely. For instance, it was necessary to adjust the input to a unified data processing, introduce independent testing tasks, create structures for managing parameters and results, among other tasks.

Chapter 5

Execution Schema

For the framework implementation, including all the components and its distributed architecture, it became comprehensible the need for outlining the complete process in a way that maximizes the resources use and reduces the execution time as much as possible. Thus, as part of this work, a schema was designed to evaluate the `CAE-Ensemble` model and the baselines, addressing most of the issues that arose during the development. The complete overview of the layout is shown in Fig. 5.1, and its components are reviewed as follows.

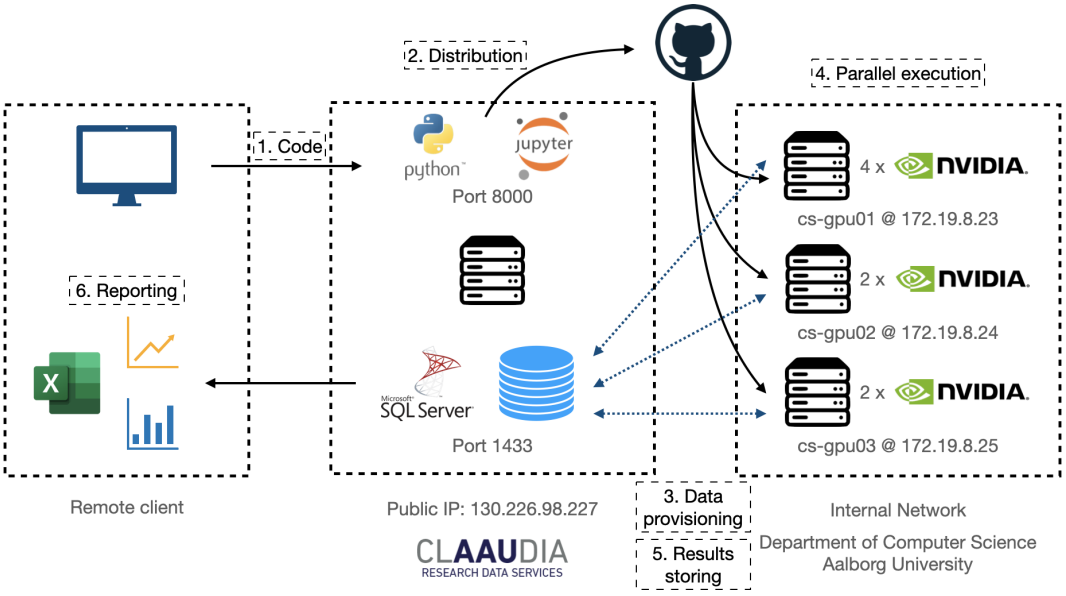

Figure 5.1: Framework deployment. The steps for the flow are numbered.

5.1 Code Distribution

All the code used in this work is written in Python, with some high-level functioning deployed via Bash Scripting. All the functionality is implemented directly into the code and controlled via execution arguments, with no user interface. There are other developments to include Graphical User Interface (GUI) in the framework, but this work does not use them. For running new code, the next steps are followed.

5.1.1 Developing and testing

New functionalities and changes are programmed using a Python editor in a non-GPU server loaded in CLAAUDIA[1], which is a cloud computing service provided by the Aalborg University (AAU) IT department. The deployed Ubuntu instance has 16 vCPUs, 128GB of RAM, and 128GB of storage, and it is remotely accessed via public IP.

The code is tested in the instance using small sub-sets via command-line scripts or the notebook interface Jupyter. The evaluation for the code is mainly focused on checking the correct functionality, since running large models in a CPU-only machine takes a very long time.

5.1.2 Distribution to execution servers

With tested code, the next step involves uploading it to a GitHub repository, a task that provides mainly two benefits. First, it maintains the historical progress for the development, enabling a review for all changes, and second, it consolidates the most updated sources for their execution.

The private repository is shared among the developing instance, the GPU-enabled execution servers, as well with other developers. So, as the new code is updated into the repository, the servers are able to get the new sources, maintaining the latest version across all of them. The released version always prevailed, so local code changes are dismissed. Also, scaling the system with extra machines only requires cloning the repository to get the last code.

5.2 Data Provisioning

As it was reviewed in Section 4.4.1, the database engine manages the already cleansing data input. So, when the execution servers run an experiment, the necessary data is requested directly to the database, reducing the storage requirements for running the models.

[1] `https://www.claaudia.aau.dk/platforms-tools/compute/compute-cloud/`

The Database Management System (DBMS) is based on SQL Server Express, running in the same CLAAUDIA instance used for coding. The selection is based on its integration with reporting tools and the relatively small scale of the system, which is an application aligned with its license terms.

5.3 Parallel Execution

To maximize the use of resources, distributed across three execution servers with eight GPU processors overall, the following tasks were conducted to reduce the execution time considering other processes that could be running in there.

5.3.1 Server distribution

First, each server was checked via the GPU manager to know the current load and define which is a better case to run a specific experiment. Fig. 5.2 shows an example, where the load is relatively high in terms of memory consumption, so running a process there could represent an issue if the memory requirements are higher than the available in both cores.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.45.01    Driver Version: 455.45.01    CUDA Version: 11.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  TITAN RTX           On   | 00000000:65:00.0 Off |                  N/A |
| 76%   87C    P2   224W / 280W |  19493MiB / 24220MiB |     97%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  TITAN RTX           On   | 00000000:B3:00.0 Off |                  N/A |
| 66%   84C    P2   225W / 280W |  18953MiB / 24220MiB |    100%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Figure 5.2: NVIDIA System Management Interface example.

Future developments would automatize this task using a controller server, which could manage the assignation according to the current system load and the memory requirement for each experiment.

5.3.2 GPU cores assignation

When a process is executed in a given server, the framework already has a task that assigns it to the GPU core with the reduced load, balancing the execution. Then, considering this property, a bash script maintains a batch with the next experiments, which will be executed as the previous ones finished and the load in

the server is balanced. The goal is not to overload the server while other processes are running, but using all resources if they are free.

5.3.3 GPU and CPU tasks detachment

During the experiment execution, the deep learning models run in GPU cores, while other calculations, such as computing metrics and writing data relied on the CPU. Thus, for maximizing the GPU use, the CPU tasks run independently in background threads, preventing that a GPU process enters an idle mode without freeing memory while waits for the CPU task to finish.

An example is shown in Fig. 5.3 illustrating how the GPU processes only models, while the remaining tasks are delegated to CPU threads, reducing the overhead of the system and maximizing server resources.

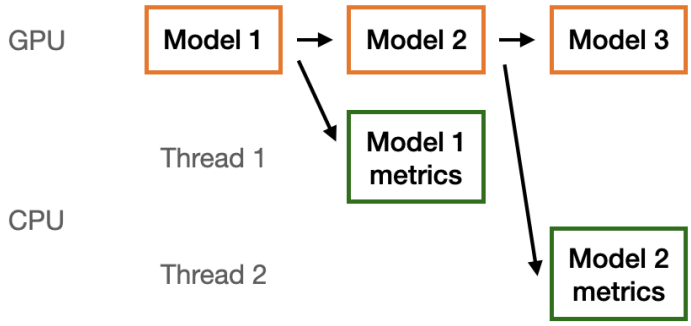

Figure 5.3: GPU and CPU threads management example.

5.4 Results Storage

A task executed by CPU threads concerns the insertion of the results in a centralized database, as it was reviewed in Section 4.4.2. It is managed by the same engine used for data provisioning, relying on SQL Server at a CLAAUDIA instance. One reason for using this platform is because the results could be accessed directly by reporting tools such as Microsoft Excel and Power BI. Other DMBS usually require specific Open Database Connectivity (ODBC) drivers to do that, so a simpler choice was adopting the SQL Server system.

5.5 Reporting

All experiments executed in any server with different configurations store the results in a centralized database. Then, to evaluate and compare between settings,

models, and data sets, it was necessary to introduce reporting tools such as Microsoft Excel. It allows efficiently analyzing the results using tables and charts.

The software has a direct connection with the database, so the results are updated as soon as the experiments were completed. Therefore, already made reports are continuously growing as new experiments are including, which prevents redesigning them.

An example for a reporting table is shown in Fig. 5.4, which evaluates a single data set (*ECG*) and twelve models considering the five metrics represented by the columns. The colors highlight the highest value for each case, given a complete overview of the performance.

Row Labels	Average of precision	Average of recall	Average of fbeta	Average of pr_auc	Average of roc_auc
ECG					
ISF	0.158730159	0.001658992	0.007791877	0.050126232	0.506154246
LOF	0.037372933	0.030789707	0.0350831	0.049968171	0.491165869
MAS	0.051244871	0.026714209	0.042038919	0.057770686	0.534222465
OCSVM	0.055809506	0.042530837	0.052053335	0.05882683	0.534249829
ANNAE	0.238898845	0.132636061	0.194504391	0.151075066	0.569158157
MSCRED	0.053571429	0.068027211	0.055949273	0.105964596	0.516598925
OMNIANOMALY	0.222590637	0.097178689	0.163427386	0.140957973	0.558382283
RNNVAE	0.120165146	0.058630759	0.097665906	0.089545589	0.550042377
RAE	0.153112239	0.077141837	0.126185583	0.093638218	0.592163458
RAE_ENSEMBLE	0.166337217	0.083007817	0.136171153	0.109208836	0.566887393
CAE	0.277236213	0.091423814	0.166335141	0.129732683	0.563270316
CAE_ENSEMBLE	0.207750856	0.135823108	0.180050996	0.168184991	0.576074357

Figure 5.4: Report table example in Microsoft Excel.

Then, a chart in Fig. 5.5 illustrates the evolution for three metrics in different threshold configurations, giving insights about a possible trend in the scenario.

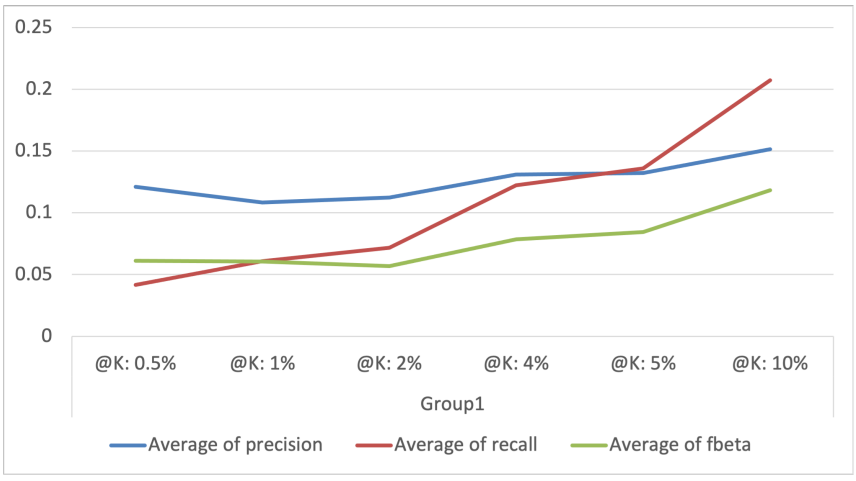

Figure 5.5: Report chart example in Microsoft Excel.

The reports were made using pivot tables, a tool that shows the data in arrangements of rows and columns, and provides control over several variables dynamically. For instance, Fig. 5.6 shows a controller including some scenarios to evaluate, the model configuration that is tested, the use of adjusted points (Section 4.5.2), and the data set.

Additionally, the figure displays an unavailable data set (*SWaT*) which illustrates how the spreadsheet is connected to the database. At that moment, there was no data for the scenario of *SWaT* meaning that the experiment was under execution, so the results will be eventually available.

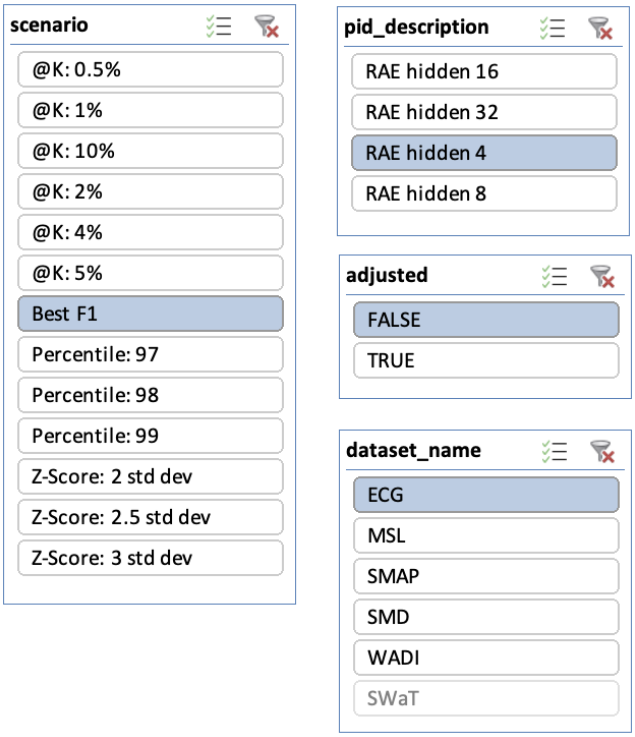

Figure 5.6: Dashboard manager for pivot tables.

Chapter 6

Experiments

The reviewed model is tested in terms of accuracy and execution time using real-world data sets and compared to several baselines. Therefore, the experiments show the applicability and effectiveness of the model for addressing the outlier detection problem. The implementation details are outlined as follows.

6.1 Data Sets

- *ECG*[1] is a two-dimensional time series related to electrocardiogram readings for seven patients. Each time series contains around 3,700 and 5,400 observations. The outlier ratio is 4.88%.
- *SMD*[2] is a public data set for server metrics released by [10]. It has 28 time series, where each one has 38 dimensions, containing 708,405 observations for training and 708,420 for testing. The outlier ratio is 4.16%.
- *MSL*[3] consist in telemetry data from the Curiosity Rover on Mars. It comprises 36 time series with 55 dimensions, containing 58,317 training observations and 73,729 for testing purposes. The outlier ratio is 9.17%.
- *SMAP*[3] is a data set from a Soil Moisture satellite consisting in 69 time series with 25 different variables. In total, they consist of 138,004 training timestamps and 435,826 testing observations. The outlier ratio is 12.27%.
- *SWaT*[4] is a public data set from sensors and actuators for a water treatment plant. It is a single time series with 51 dimensions under normal operation (495,000 training observations) and during a period of intrusion attacks

[1] https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
[2] https://github.com/NetManAIOps/OmniAnomaly/
[3] https://github.com/khundman/telemanom
[4] https://itrust.sutd.edu.sg/itrust-labs_datasets/

(449,919 testing timestamps). The percentage of outliers during the attacks is 12.14%.

- *WADI*[4] consists in two time series with 127 dimensions representing a water distribution system under normal operation (1,994,172 observations) and during intrusion attacks (345,604 testing timestamps). The outlier ratio is 5.76% and it is sampled every ten timestamps, given its extensive size.

- *Yahoo*[5] is a dataset with real and synthetic time series for metrics of different Yahoo services. It includes 162 time series with around 1,400 observations for each time series. The outlier ratio is less than 1%.

6.2 Baselines

The model `CAE-Ensemble` is compared to the following deep and non-deep learning methods in order to demonstrate its effectiveness.

- Isolation Forest (`IS`) [9]: an ensemble of randomized clustering trees, where the goal is isolating outliers in sparse clusters. It works by generating partitions over the data, where the points with a relatively small number of partitions are likely to be anomalous. It is a non-deep learning method not designed for processing time series, so it could miss properties as a temporal dependency.

- Local Outlier Factor (`LOF`) [11]: it is a clustering-based method, which detects outliers according to their local deviations with respect to their neighbors. It assigns a density score for each point given by its distance to a group of k neighbors, where the points with a lower score are likely anomalous. It is also a non-deep learning method not specifically designed for time series.

- One-Class SVM (`SVM`) [12]: it corresponds to a one-class classification method, which employs Support Vector Machines to learn the boundary that covers normal data. Using that information classifies observations between positive and negative regions. Thus, it is able to distinguish between normal data points and outliers.

- Moving Average Smoothing (`MAS`): a simple method used for time series, where the values that are more deviated from a moving average window are likely to be considered as outliers.

- Autoencoder Ensemble (`AE-Ensemble`) [1]: an ensemble constituted by feed-forward autoencoders that aim to reconstruct the original input, a process

[5] `https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70`

that could miss some information and allows to distinguish anomalous points. The method is the base for several deep-learning methodologies since the autoencoder constitutes a valuable unsupervised technique.

- Recurrent Autoencoder (`RAE`) [13]: it extends the autoencoder modeling using Long Short-Term Memory `LSTM` units to reconstruct time series via a sequence-to-sequence architecture.
- Convolutional Autoencoder (`CAE`): the simplified case for the presented convolutional sequence-to-sequence autoencoder, which dismisses the use of an ensemble, only considering a single base model.
- Correlation Matrices Recurrent Autoencoder (`MSCRED`) [14]: it corresponds to a state-of-the-art method for multivariate time series outlier detection, which uses an autoencoder to reconstruct correlation matrices instead of using the time series directly.
- Variational Recurrent Autoencoder (`RNNVAE`) [15]: another extension to the autoencoder model that introduces a stochastic latent component for learning a probabilistic distribution to improve the reconstruction output.
- Temporal Variational Autoencoder (`OMNIANOMALY`) [10]: an elaboration over the previous variational modeling including an additional component to capture the temporal dependency in a context of stochastic variables.
- Recurrent Autoencoder Ensembles (`RAE-Ensemble`) [16]: a state-of-the-art recurrent autoencoder ensemble, which is created using recurrent neural networks with random architectures, which captures better the temporal dependencies than other methods.

6.3 Evaluation Metrics

The model and the baselines are evaluated considering two types of metrics, depending on how the threshold is used, as detailed as follows.

6.3.1 Threshold-dependent metrics

They are calculated after defining the criterion for separating outliers from normal data. For instance, considering a defined number of observations with the highest outlier score, or selecting values with a great deviation from the mean. In the current case, a threshold is fixed where 99% of data is considered normal points, and the remaining outliers. The metrics calculated using a threshold are shown next, where for each case, a higher value represents a better performance:

- *Precision*: it is the proportion between the true positive (tp) observations and the total number of positive instances: true (tp) and false (fp).

$$Precision = \frac{tp}{tp + fp} \tag{6.1}$$

- *Recall*: it corresponds to the detected instances (true positives) divided by the real number of outliers: true positives (tp) and false negatives (fn).

$$Recall = \frac{tp}{tp + fn} \tag{6.2}$$

- *F1*: it is the harmonic mean between *Precision* and *Recall*, showing a balance over those metrics, and facilitating their comparison.

$$F1 = \frac{2}{Precision^{-1} + Recall^{-1}} \tag{6.3}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6.4}$$

6.3.2 All-threshold metrics

They consider all possible thresholds for the calculation, which is useful when the threshold definition is not clear, as it arises in unsupervised models. The two metrics are detailed next, where higher results reflect better model accuracy.

- *PR-AUC*: it means Precision-Recall Area Under Curve and it is used to represent the average *Precision* scores for each *Recall*, computed over all possible thresholds ϵ. It is represented by the overlap for the following functions.

$$Precision(\epsilon) = \int_{\epsilon}^{\infty} Precision(x)dx \tag{6.5}$$

$$Recall(\epsilon) = \int_{\epsilon}^{\infty} Recall(x)dx \tag{6.6}$$

- *ROC-AUC*: it means Receiver Operating Characteristic Area Under Curve, and it represents the probability that a random positive value has a higher outlier score than a random negative instance. It represents the overlap between the following two curves for the true positive (TPR) and false positive (FPR) rates, with a varying threshold ϵ. f_1 represents the true positive rate

given a threshold, while f_0 is the false positive one. tn stands for true negative results.

$$TPR(\epsilon) = \int_{\epsilon}^{\infty} f_1(x)dx \tag{6.7}$$

$$f_1 = \frac{tp}{tp + fn} \tag{6.8}$$

$$FPR(\epsilon) = \int_{\epsilon}^{\infty} f_0(x)dx \tag{6.9}$$

$$f_0 = \frac{fp}{fp + tn} \tag{6.10}$$

6.4 Experiment Results

The accuracy results for all data sets are shown as follows. Table 6.1 presents the results for *ECG* where `CAE-Ensemble` outperforms all baselines in almost every metric. For *ROC*, the model is the second-best, even when the overall performance of `RAE` is deficient.

Table 6.1: *ECG* accuracy results.

Model	*Precision*	*Recall*	*F1*	*PR*	*ROC*
`ISF`	0.0347	0.0076	0.0198	0.0501	0.5062
`LOF`	0.0713	0.0174	0.0436	0.0500	0.4912
`MAS`	0.0483	0.0100	0.0266	0.0578	0.5342
`OCSVM`	0.0426	0.0094	0.0243	0.0588	0.5342
`MSCRED`	0.0429	0.0113	0.0275	0.1046	0.5062
`OMNIANOMALY`	0.2364	0.0419	0.1198	0.1410	0.5584
`RNNVAE`	0.1583	0.0283	0.0808	0.0895	0.5500
`AE-Ensemble`	0.2684	0.0529	0.1445	0.1511	0.5692
`RAE`	0.1249	0.0222	0.0634	0.0936	**0.5922**
`RAE-Ensemble`	0.1952	0.0378	0.1042	0.1092	0.5669
`CAE`	0.2324	0.0494	0.1309	0.1297	0.5633
`CAE-Ensemble`	**0.2813**	**0.0635**	**0.1634**	**0.1682**	0.5761

The results for *SMD* shown in Table 6.2 are similar to the previous case, with the *ROC* being better in non-deep method `MAS`, which in this case is a good contender. Even so, as before, `CAE-Ensemble` achieves the second-best accuracy in this metric, which reflects its very good performance.

Table 6.2: *SMD* accuracy results.

Model	*Precision*	*Recall*	*F1*	*PR*	*ROC*
ISF	0.1288	0.0277	0.0669	0.0635	0.5094
LOF	0.2531	0.0945	0.1526	0.1208	0.5695
MAS	0.4813	0.2101	0.3165	0.3261	**0.7520**
OCSVM	0.4021	0.1462	0.2486	0.1934	0.5783
MSCRED	0.0233	0.0036	0.0097	0.0472	0.4956
OMNIANOMALY	0.2497	0.1165	0.1699	0.1507	0.6148
RNNVAE	0.4340	0.1667	0.2723	0.2408	0.6917
AE-Ensemble	0.3782	0.1481	0.2372	0.2481	0.7137
RAE	0.4794	0.1996	0.3073	0.2651	0.6998
RAE-Ensemble	0.4781	0.1901	0.3013	0.2646	0.7139
CAE	0.5281	0.2267	0.3454	0.3351	0.7435
CAE-Ensemble	**0.5316**	**0.2269**	**0.3479**	**0.3368**	0.7441

For *MSL*, detailed in Table 6.3, the CAE-Ensemble model outperforms the baselines only in the all-threshold metrics, showing the importance for choosing a good threshold. For that case, a single threshold shows a worse performance for the model, even when considering all *Precision* and *Recall* scenarios, represented by *PR*, the performance is better.

Table 6.3: *MSL* accuracy results.

Model	*Precision*	*Recall*	*F1*	*PR*	*ROC*
ISF	0.1154	0.0183	0.0380	0.1085	0.5036
LOF	0.2101	0.0456	0.0758	0.1433	0.5268
MAS	0.2477	**0.0680**	**0.1058**	0.1596	0.5469
OCSVM	0.2286	0.0528	0.1019	0.1583	0.5629
MSCRED	0.2016	0.0057	0.0251	0.1456	0.5157
OMNIANOMALY	0.1877	0.0245	0.0695	0.1619	0.5429
RNNVAE	0.1699	0.0291	0.0608	0.1380	0.5335
AE-Ensemble	**0.2590**	0.0192	0.0594	0.1424	0.5456
RAE	0.1766	0.0416	0.0887	0.1573	0.5714
RAE-Ensemble	0.1935	0.0500	0.1003	0.1588	0.5720
CAE	0.1980	0.0386	0.0731	0.1604	0.5760
CAE-Ensemble	0.2128	0.0415	0.0823	**0.1643**	**0.5843**

The accuracy for *SMAP*, shown in Table 6.4, is the best for the variants of the developed model, with and without an ensemble, in almost every case. The *Precision* for RAE and its ensemble are relatively high, representing that they are very selective, dismissing several positive instances since the *Recall* is low. The results

present that this data set could be not benefited from the ensemble approach, because most of its metrics are better in single models. Even so, it could be tested further, since it is a relatively unexpected condition.

Table 6.4: *SMAP* accuracy results.

Model	*Precision*	*Recall*	*F1*	*PR*	*ROC*
ISF	0.0904	0.0115	0.0239	0.1338	0.4976
LOF	0.1259	0.0197	0.0370	0.1316	0.5035
MAS	0.1285	0.0486	0.0675	0.1651	0.5251
OCSVM	0.1497	0.0362	0.0526	0.1474	0.4902
MSCRED	0.0431	0.0021	0.0075	0.0934	0.4343
OMNIANOMALY	0.0883	0.0229	0.0308	0.1558	0.5402
RNNVAE	0.0764	0.0217	0.0250	0.1192	0.5119
AE-Ensemble	0.1040	0.0357	0.0479	0.1987	0.5773
RAE	**0.1528**	0.0414	0.0542	0.1660	0.5641
RAE-Ensemble	0.1460	0.0420	0.0492	0.1628	0.5601
CAE	0.1302	**0.0659**	**0.0733**	**0.2123**	0.5905
CAE-Ensemble	0.1208	0.0609	0.0709	0.2118	**0.5959**

For *SWaT* in Table 6.5, the best results are achieved in a simple autoencoder scenario, so possibly the components included in other autoencoder-based methods do not fit in this case. Also, for MSCRED, the results for the threshold-dependent metrics are zero, meaning the current setting is very restricted for the method.

Table 6.5: *SWaT* accuracy results.

Model	*Precision*	*Recall*	*F1*	*PR*	*ROC*
ISF	0.0956	0.0079	0.0296	0.1268	0.5005
LOF	0.0889	0.0073	0.0275	0.1144	0.4838
MAS	0.1067	0.0088	0.0330	0.0786	0.2272
OCSVM	0.0889	0.0073	0.0275	0.0901	0.3457
MSCRED	0.0000	0.0000	0.0000	0.0727	0.5666
OMNIANOMALY	0.9822	0.0807	0.3036	0.7041	0.8123
RNNVAE	0.2644	0.0218	0.0819	0.1629	0.6311
AE-Ensemble	**0.9867**	**0.0812**	**0.3054**	**0.7567**	**0.8490**
RAE	0.9687	0.0802	0.3045	0.7494	0.8420
RAE-Ensemble	0.9844	0.0810	0.3048	0.7536	0.8488
CAE	0.4156	0.0342	0.1286	0.3849	0.7634
CAE-Ensemble	0.8778	0.0722	0.2717	0.6521	0.8358

The *WADI* data set, detailed in Table 6.6, has excellent results in most metrics, although ONMIANOMALY shows a better *ROC* even when its remaining results are

very poor in comparison to the baselines.

Table 6.6: *WADI* accuracy results.

Model	*Precision*	*Recall*	*F1*	*PR*	*ROC*
ISF	0.0520	0.0090	0.0266	0.0610	0.5248
LOF	0.1590	0.0275	0.0813	0.0703	0.5284
MAS	0.4538	0.0785	0.2320	0.1500	0.5788
OCSVM	0.3815	0.0660	0.1950	0.1193	0.5754
MSCRED	0.0000	0.0000	0.0000	0.0993	0.6730
OMNIANOMALY	0.0872	0.0150	0.0444	0.1724	**0.7261**
RNNVAE	0.4682	0.0810	0.2393	0.1737	0.5739
AE-Ensemble	0.1647	0.0285	0.0842	0.1356	0.6261
RAE	0.2630	0.0455	0.1344	0.1580	0.6516
RAE-Ensemble	0.0983	0.0170	0.0503	0.1263	0.6527
CAE	0.3353	0.0579	0.1713	0.1243	0.5994
CAE-Ensemble	**0.5462**	**0.0945**	**0.2793**	**0.1912**	0.6023

In the *Yahoo* data set, shown in Table 6.7, the best results are achieved in non-deep learning methods. Possibly, the synthetic time series available in that data set benefit non-deep learning methods since the anomalous point are not difficult to distinguish as they are introduced artificially.

Table 6.7: *Yahoo* accuracy results.

Model	*Precision*	*Recall*	*F1*	*PR*	*ROC*
ISF	0.1251	0.4457	0.1335	0.4065	0.8580
LOF	0.2629	0.6891	0.2748	0.6597	0.9053
MAS	0.3177	**0.7901**	0.3279	0.6792	0.9335
OCSVM	**0.3290**	0.7813	**0.3390**	**0.7779**	0.9115
OMNIANOMALY	0.3116	0.6794	0.3145	0.6523	0.9212
RNNVAE	0.2248	0.4382	0.2191	0.3754	0.8345
AE-Ensemble	0.3087	0.6969	0.3155	0.7144	0.9188
RAE	0.2893	0.6649	0.2938	0.6512	0.9187
RAE-Ensemble	0.3170	0.7260	0.3223	0.7298	**0.9445**
CAE	0.2751	0.6381	0.2769	0.4700	0.9144
CAE-Ensemble	0.2648	0.6202	0.2647	0.5191	0.9186

6.5 Execution Time

In order to evaluate the execution time for the `CAE-Ensemble`, it is compared with `RAE-Ensemble` because both have a similar architecture. Also, as the former is based on convolutional networks instead of recurrent ones, it is relevant to overview how this change reduces the running time.

Table 6.8 shows the execution time for `RAE` and `CAE` with and without ensemble. First, in all data sets, the results for `CAE-Ensemble` are very low in comparison to the recurrent model, demonstrating its effectiveness in terms of reducing the execution time. Then, comparing the ratio between every single model and its ensemble, the results are better for `CAE`, showing the effect of using transfer learning where each new single model takes less time to be trained.

Table 6.8: Training Time Comparison (in minutes).

Model	*ECG*	*SMD*	*MSL*	*SMAP*	*SWaT*	*WADI*	*Yahoo*
`RAE`	7.84	246.43	16.63	32.19	12.88	72.32	59.13
`RAE-Ensemble`	59.66	1959.13	129.99	254.83	102.44	566.89	470.80
`Ratio`	7.60	7.95	7.82	7.92	7.95	7.84	7.96
`CAE`	4.16	74.34	7.65	20.36	4.45	22.37	31.05
`CAE-Ensemble`	24.05	452.06	45.45	122.13	25.92	129.58	188.98
`Ratio`	5.78	6.08	5.94	6.00	5.82	5.79	6.09

Chapter 7

Discussion

The current work consolidates the development for an outlier detection model in different stages over a year, which are detailed next. Also, as part of this process, a paper is likely to be published in a high-ranked conference.

1. Development of model components in Spring 2020.
2. Paper sent to International Conference on Data Engineering (ICDE) in Autumn 2020.
3. Paper rejected by Winter 2020 with observations about some inconsistencies in the results.
4. Overall model review, incorporation into a framework, new baselines, and tests during Spring 2021. The current work.
5. The paper is under review by International Conference on Very Large Data Bases (VLDB).

After overseeing the context of the current project, its evaluation considers the features that contribute to improving the model and its evaluation process, not exclusively its comparable performance, as it is detailed as follows.

7.1 Model Performance

The setting shown in this work, based on the premise of 99% of observations are normal data, shows very good accuracy for `CAE-Ensemble` in five of seven data sets. It is a remarkable achievement since this threshold selection is a rigorous choice and the domain for the time series is very different. For instance, there are web server metrics, health readings, satellite measurements, among others, and the results are consistently correct for all of them.

Then, for almost every case, the performance is better for deep learning methods showing how they were improving the state-of-the-art techniques for this problem in recent years. Also, for threshold-dependent metrics, most non-deep learning methods have a performance one degree of magnitude below the deep learning cases, reflecting that the selected threshold declines significantly their performance.

In an unsupervised setting, choosing a good threshold is a difficult decision because there is no domain information available. For instance, knowing in advance the outlier ratio (see Section 6.1) allows choosing the threshold that fits better for each data set, but this is only possible in supervised scenarios. Thus, the 99% of normal data was a random selection among all the evaluated cases, aiming to evaluate all methods fairly. Other scenarios, such as [6], reported the best possible metrics derived from *PR*, but for the current case, that choice could be not illustrative enough. It is relevant to show how the threshold affects some results, something that is not clear when only the best cases are outlined.

In terms of execution time, the direct comparison between `CAE-Ensemble` and `RAE-Ensemble` shows a significant improvement for the convolutional model even in the single configuration, a result that was expected. Therefore, this setting enhances the previous recurrent method satisfactorily, getting more accurate results in less time.

7.2 Framework

The migration of `CAE-Ensemble` into an outlier detection framework introduced several advantages for its evaluation. First, it allowed a homogeneous comparison between all models, since the input and output are processing equally for every case. Usually, when the original implementation for models is executed directly some differences could arise between them. For instance, data processed with no standard scale, or dissimilar parameters would point to unmatched results. Using a framework helps to control these scenarios providing fair measurements.

Moreover, the infrastructure enables to make common tasks independent for any model, data set, and experiment. Thus, introducing changes for these functions is transparent over the system, reducing programming work and possible mistakes. For instance, new results calculations and database management is unconstrained to each scenario, facilitating its use and future scalability of the system with new models and functions.

Using a framework also increases the number of experiments that were performed over all the models and data sets. For the evaluation of `CAE-Ensemble` and the baselines at least fifty types of settings were tested, a situation that would be unmanageable if each model was evaluated independently.

7.3 Scalability

The progress achieved by the inclusion of parallel features and distributed execution for the experiments reduces significantly the running time allowing to make more testing and evaluation. During the development, it becomes necessary to manage better the workload for a single server, so incorporating a database engine potentializes its distribution across several machines with a unified point for provisioning data and results.

Also, as the tasks for calculating metrics worked independently in CPU threads, it was possible to build a set of metrics without affecting the model training time. It allowed evaluating even more scenarios for a single setting, such as using different thresholds to illustrate their evolution. Thus, every model has metrics for different threshold configurations, for example, using percentiles, standard deviations, best possible cases, among others.

Chapter 8

Related Work

The outlier detection problem had been studied for a long time since it is closely related to critical circumstances. For example, it is important to keep systems working and monitor any issue with them. Also, it is desirable the early detection of problems in health, environment, and many other scenarios, to execute preventing actions accordingly.

The existent methods for addressing the outlier detection problem could be classified in at least five groups, as it is detailed in Fig. 8.1. The categories lead to define some differences between the models, so they are not unique, and there are methods that be could be merged in between.

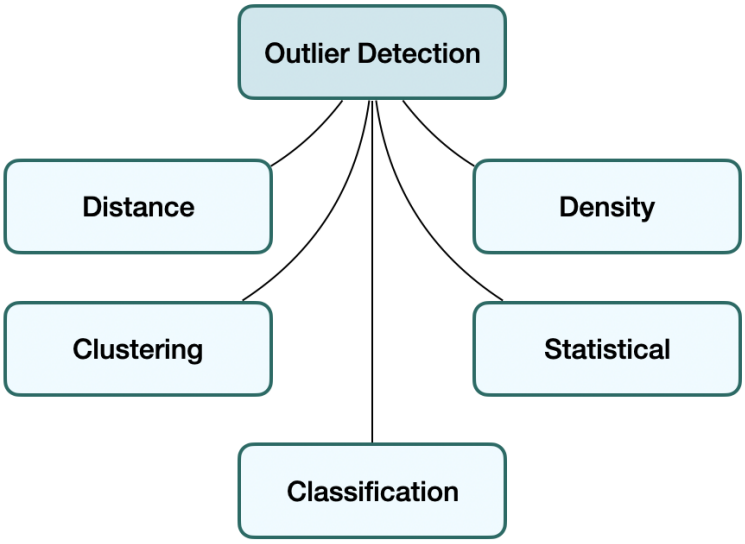

Figure 8.1: Outlier detection methods classification.

8.1 Distance-based Methods

The first category, related to distance-based methods, considers the distance calculation between data points and then finds those that comply with some rules, like the highest-ranked. A classic method is k-Nearest Neighbors (*k-NN*) algorithm [17], while there are recent developments with refined distance calculations and different rules [18, 19].

8.2 Density-based Methods

Derived from the distance case, the density-based methods calculate a local density measurement for each point, defining principles to classify them using the metric. Local Outlier Factor (*LOF*) [11] is the main exponent for this methodology, while other works introduced improvements with probability distributions and parametrizations [20, 21].

8.3 Clustering-based Methods

Clustering-based methods create groups of similar data points and measure the distances between them, usually selecting the highest-ranked as outliers. Some works developed adaptations from (*LOF*) [22], or use a different criteria such as *DBSCAN* [23], Gaussian Mixture Models (*GMM*) [24] and *k-means* [25].

8.4 Statistical Methods

Statistical methods for outlier detection usually relied on some parametric features, for example, assuming a data distribution and considering the points outside it as outliers. In a normal distribution, the *Z-Score* rates the points according to mean and variance. Following a similar approach, an extensively used method corresponds to Principal Component Analysis (*PCA*), while other methods depend on graphical representations, such as histograms and scatter plots.

8.5 Classification Methods

Classification methodologies define a criterion to separate groups into two or more classes. For outlier detection, there are two categories, normal data, and anomalous points, so the One-Class Support Vector Machines [12] is a recognized method that uses a hyperplane to separate data between positive and negative instances. Then,

most deep learning methods use this principle to address the problem, incorporating better techniques to classify the observations. In most cases, they follow two types of strategy: discriminative and generative processes.

8.5.1 Discriminative Strategy

A discriminative model categorizes data instances between several classes, which is basically the task of assigning labels. For outlier detection, it involves building a network that outputs a binary categorization for each observation, which is developed in works such as [26].

8.5.2 Generative Strategy

The generative strategy aims to characterize and identify valuable data over the observations. For instance, it provides details over a picture or creating a new one based on that information. For outlier detection, the autoencoder model is a usual example because it generates a reconstructed input to identify anomalous points [1, 4, 10], as it was developed in this work. Other techniques usually involve probabilistic distributions, such as the Random Sum-Product Networks introduced by [27].

Training deep learning models is constrained by the availability of labels, which depends on a data recollection and tagging process completed by a domain expert. Thus, it is very likely that many data sets have limited or not labels, so the models are developed considering this drawback. When there is plenty of labels, a model could be trained using them, which is a supervised setting including works as [28]. Then, if there are some labels, the training would use them barely in a semi-supervised scenario, which is followed by [29, 30]. The unsupervised case does not use labels, as this work, which is an extended used technique since does not need any human intervention [10, 8]. Also, recent works [31] are leveraging the labeling stage to the model itself, which is called a self-supervised setting.

8.6 Time Series Application

Most outlier detection methods are applicable for time series processing because an anomalous point in an ordered series usually does not differ very much from an outlier observation in other settings. Even so, as recent deep learning methods surge the performance for ordered data such as sequences [32], it becomes coherent to apply it for time series. Thus, current developments take advantage of temporal properties to specialize the techniques for its use in time series [4, 16, 33]. This work is part of this process, including contributions for modeling and supporting infrastructure for evaluating them fairly.

Chapter 9

Conclusion

The current work integrates the outlier detection model `CAE-Ensemble`, designed and improved as part of previous projects, into an extensive under-developing framework for managing and maintaining models related to that domain. The process supports the future development of the algorithm and its possible derivations, including further evaluation with updated techniques.

Then, the work introduces new features to the framework allowing more exhaustive evaluations for the models, considering advanced scenarios and improved optimization techniques. Also, the incorporation of data management tools alleviates the system workload and establishes a milestone that assures stable scalability across different machines and environments.

Finally, the integrated process was delineated over a complete schema that maximizes the available resources, providing efficient mechanisms to run, evaluate and analyze different outlier detection models. `CAE-Ensemble` was tested in comparison to several baselines to show its good performance and prove the system efficacy.

As future work, the paper publication [6] is under review, with an expectation of getting an acceptance in the following months. Also, further automatization in the framework is under consideration, mainly to control the model execution across current and future instances via a load-balancing mechanism.

Bibliography

[1] J. Chen, S. Sathe, C. C. Aggarwal, and D. S. Turaga, "Outlier detection with autoencoder ensembles," in *SDM*, 2017, pp. 90–98.

[2] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *ICML*, D. Precup and Y. W. Teh, Eds., 2017, pp. 1243–1252.

[3] W. Zhang, J. Jiang, Y. Shao, and B. Cui, "Efficient diversity-driven ensemble for deep neural networks," in *ICDE*, 2020, pp. 73–84.

[4] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao, "Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications," in *WWW*, 2018, pp. 187–196.

[5] D. Chaves, *Time Series Outlier Detection with Diversity-Driven Sequence-to-Sequence Convolutional Ensembles*. Aalborg University, 2020.

[6] D. Chaves, T. Kieu, C. Guo, F. Huang, K. Zheng, and B. Yang, "Time series outlier detection with diversity-driven convolutional ensembles," *Unpublished*, 2021.

[7] C. C. Aggarwal and S. Sathe, *Outlier Ensembles - An Introduction*. Springer, 2017.

[8] T. Kieu, B. Yang, C. Guo, Y. Song, and C. S. Jensen, "Robust and explainable autoencoders for time series outlier detection," in *Unpublished*, 2021.

[9] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *ICDM*, 2008, pp. 413–422.

[10] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *SIGKDD*, 2019, pp. 2828–2837.

[11] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *SIGMOD*, 2000, pp. 93–104.

[12] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection," in *NIPS*, 1999, pp. 582–588.

[13] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. M. Shroff, "LSTM-based encoder-decoder for multi-sensor anomaly detection," in *ICML Anomaly Detection Workshop*, 2016, p. 5.

[14] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *AAAI*, 2019, pp. 1409–1416.

[15] M. Soelch, J. Bayer, M. Ludersdorfer, and P. van der Smagt, "Variational inference for on-line anomaly detection in high-dimensional time series," in *ICML*, 2016.

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

[17] E. Fix and J. Hodges, "Discriminatory analysis; non-parametric discrimination: Consistency properties," *USAF School of Aviation Medicine*, 1951.

[18] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *NIPS*, 2018, pp. 7167–7177.

[19] W. Huo, W. Wang, and W. Li, "Anomalydetect: An online distance-based anomaly detection algorithm," in *Web Services - ICWS 2019*, ser. Lecture Notes in Computer Science, J. A. Miller, E. Stroulia, K. Lee, and L. Zhang, Eds., vol. 11512. Springer, 2019, pp. 63–79.

[20] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos, "LOCI: fast outlier detection using the local correlation integral," in *ICDE*. IEEE Computer Society, 2003, pp. 315–326.

[21] H. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Loop: local outlier probabilities," in *CIKM*. ACM, 2009, pp. 1649–1652.

[22] Z. Gao, "Application of cluster-based local outlier factor algorithm in anti-money laundering," in *International Conference on Management and Service Science*, 2009, pp. 1–4.

[23] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*. AAAI Press, 1996, pp. 226–231.

[24] A. Reddy, M. Ordway-West, M. Lee, M. Dugan, J. Whitney, R. Kahana, B. Ford, J. Muedsam, A. Henslee, and M. Rao, "Using gaussian mixture models to detect outliers in seasonal univariate network traffic," in *IEEE Security and Privacy Workshops (SPW)*, 2017, pp. 229–234.

[25] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[26] S. Wang, Y. Zeng, X. Liu, E. Zhu, J. Yin, C. Xu, and M. Kloft, "Effective end-to-end unsupervised outlier detection via inlier priority of discriminative network," in *NIPS*, 2019, pp. 5960–5973.

[27] R. Peharz, A. Vergari, K. Stelzner, A. Molina, M. Trapp, X. Shao, K. Kersting, and Z. Ghahramani, "Random sum-product networks: A simple and effective approach to probabilistic deep learning," in *AUAI*, ser. Proceedings of Machine Learning Research, vol. 115. AUAI Press, 2019, pp. 334–344.

[28] V. Jumutc and J. A. K. Suykens, "Multi-class supervised novelty detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 12, pp. 2510–2523, 2014.

[29] N. Shi, X. Yuan, and W. Nick, "Semi-supervised random forest for intrusion detection network," in *MAICS*, ser. CEUR Workshop Proceedings, vol. 1964, 2017, pp. 181–185.

[30] H. Wu and S. Prasad, "Semi-supervised deep learning using pseudo labels for hyperspectral image classification," *IEEE Trans. Image Process.*, vol. 27, no. 3, pp. 1259–1270, 2018.

[31] H. M. Song and H. K. Kim, "Self-supervised anomaly detection for in-vehicle network using noised pseudo normal data," *IEEE Trans. Veh. Technol.*, vol. 70, no. 2, pp. 1098–1108, 2021.

[32] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.

[33] K. Le and P. Papotti, "User-driven error detection for time series with events," in *ICDE*, 2020, pp. 745–757.