UNIVERSITY OF DEUSTO

LARGE-SCALE ANOMALY DETECTION AND DIAGNOSIS ON INDUSTRIAL HETEROGENEOUS MULTI-SENSOR SYSTEMS USING DEEP LEARNING

Doctoral thesis by
MIKEL CAÑIZO ZUBIZARRETA
within the program
ENGINEERING FOR THE INFORMATION SOCIETY
AND SUSTAINABLE DEVELOPMENT
Supervised by
ENRIQUE ONIEVA CARACUEL
ÁNGEL CONDE MANJÓN

UNIVERSITY OF DEUSTO

LARGE-SCALE ANOMALY DETECTION AND DIAGNOSIS ON INDUSTRIAL HETEROGENEOUS MULTI-SENSOR SYSTEMS USING DEEP LEARNING

Doctoral thesis by
MIKEL CAÑIZO ZUBIZARRETA
within the program
ENGINEERING FOR THE INFORMATION SOCIETY
AND SUSTAINABLE DEVELOPMENT

Supervised by
ENRIQUE ONIEVA CARACUEL
ÁNGEL CONDE MANJÓN

PhD Candidate Advisor Advisor

Bilbao, September 2019

Large-Scale Anomaly Detection and Diagnosis on Industrial Heterogeneous Multi-Sensor Systems Using Deep Learning

Author: Mikel Cañizo Zubizarreta
Advisor: Enrique Onieva Caracuel
Co-advisor: Ángel Conde Manjón

Text printed in Bilbao
First edition, September 2019

It's not the load that breaks you down,

it's the way you carry it.

Abstract

Industry 4.0 has played an important role in the digitalization of modern industrial processes, where multiple sensors are typically used to monitor the industrial systems. This means that large volumes of data regarding the performance of the systems are now available, which are then used for various industrial applications that range from process control and optimization to maintenance operations. In this context, processing these volumes of data to detect and diagnose anomalies has become a necessity as industrial systems are prone to faults and an undetected fault can lead to critical damage, besides reducing the productivity and increasing the maintenance operation costs.

This thesis presents a large-scale industrial monitoring system to detect and diagnose anomalies in multi-sensor systems. The monitoring system is composed of three main modules. First, a supervised Deep Learning (DL) based anomaly detection system that identifies anomalies within multi-sensor systems. Second, an anomaly diagnosis system that relies on interpretability methods to explain how the DL based anomaly detection model reaches a decision, thus identifying in which sensors and time span the anomalies occur. Third, a large-scale monitoring system based on Big Data and cloud computing technologies to process the large volumes of data in a fast, scalable, and fault-tolerant way. Therefore, the anomaly detection and diagnosis systems can be deployed in the cloud to support the monitoring of multiple industrial systems. To validate the proposal, each module has been validated in an industrial case study.

The algorithms and methods resulting from this thesis have demonstrated to be suitable for detecting and diagnosing anomalies in industrial multi-sensor systems since they improve state of the art results.

Resumen

La Industria 4.0 ha jugado un papel importante en la digitalización de los procesos industriales modernos, en los que normalmente se utilizan múltiples sensores para monitorizar los sistemas industriales. Esto significa que en la actualidad se dispone de grandes volúmenes de datos referente al rendimiento de los sistemas, que luego se utilizan para diversas aplicaciones industriales que comprenden desde el control y la optimización de procesos hasta las operaciones de mantenimiento. En este contexto, el procesamiento de estos volúmenes de datos para detectar y diagnosticar anomalías se ha convertido en una necesidad, debido a que los sistemas industriales son propensos a las averías y no detectar una de estas averías puede provocar daños críticos, además de reducir la productividad y aumentar los costes de mantenimiento.

Esta tesis presenta un sistema de monitorización industrial a gran escala para detectar y diagnosticar anomalías en sistemas multi-sensor. El sistema de monitorización se compone de tres módulos principales. En primer lugar, un sistema de detección de anomalías basado en técnicas de Deep Learning (DL) que identifica anomalías dentro de sistemas multi-sensor. En segundo lugar, un sistema de diagnóstico de anomalías que se basa en métodos de interpretación para explicar cómo el modelo de detección de anomalías llega a una decisión, identificando así en qué sensores y períodos de tiempo se producen las anomalías. En tercer lugar, un sistema de monitorización a gran escala basado en tecnologías Big Data y de computación en la nube para procesar grandes volúmenes de datos de forma rápida, escalable y tolerante a fallos. Por lo tanto, los sistemas de detección y diagnóstico de anomalías pueden desplegarse en la nube para facilitar la monitorización de múltiples sistemas

industriales. Para validar el sistema de monitorización propuesto, cada módulo ha sido validado en un caso de uso industrial.

Los algoritmos y métodos resultantes de esta tesis han demostrado ser adecuados para detectar y diagnosticar anomalías en sistemas industriales multi-sensor, ya que mejoran los resultados del estado del arte.

Acknowledgments

After three years of hard work, I waited until the end to thank all the people who made this work possible.

First of all, I would want to thank my supervisors, Enrique and Angel, for the confidence they had in me and for the help and advice they gave me over these years.

I would also thank Ikerlan and the University of Deusto for giving me the opportunity to carry out my thesis in an institution in which I have been able to develop myself both professionally and personally.

Thanks to all my colleagues for the good atmosphere and the help given.

I want to thank the Optimisation and Learning Lab of the University of Nottingham for welcoming me and making my stay pleasant, particularly, Isaac and Salva for your help and valuable comments.

On the personal side, I would like to thank all my friends for their support.

I want to thank my parents and my brother for always being there and encouraging and supporting me unconditionally.

Finally, I want to thank Ianire for putting up with me all this time and for encouraging me when I needed it most.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Like what you do, and then
you will do your best.

Katherine Johnson

1

Introduction

This chapter contains the motivation and challenges behind this thesis. It highlights the hardness of detecting anomalies in industrial heterogeneous multi-sensor systems and how Deep Learning (DL) can help to address these challenges. Then, the contributions of this thesis and the research methodology followed to accomplish them are introduced. Finally, the research activities and the outline of this work are presented.

1.1 Motivation and Scope

Time series anomaly detection refers to the technique of finding unusual patterns in sequential data that do not conform to expected behavior (Chandola et al. 2009). It has many applications for companies, from stock market anomaly detection to fraud detection in banking transactions. Time series anomaly detection has become particularly important in industrial domains such as automotive (Theissler 2017), manufacturing (Scime and Beuth 2018), energy (Yang et al. 2018), or industrial sensor networks (Cenedese et al. 2017) since industrial machines are prone to faults and an undetected fault can lead to critical consequences, such as the damage to machinery, or the unexpected stop of the production. In these scenarios, effective

anomaly detection can improve the availability and the reliability of the systems, which will then directly affect the productivity, and the operation and maintenance costs (Hashemian and Bean 2011).

Recent advances in technology have made the industry evolve towards the Industry 4.0. This term refers to the fourth industrial revolution that briefly stated, it consists of connecting the industrial machines to the Internet to monitor their performance and visualize the entire production line to support the decision-making. Therefore, industrial companies have experienced a technological transformation by digitalizing their production processes by adopting Cyber-Physical System (CPS)s rather than traditional embedded systems (Cheng et al. 2016). In short, CPSs are machines composed of a physical and a cybernetic part that in collaboration facilitate gathering data from the industrial machines. CPSs enable new advanced strategies to be implemented to improve and optimize the processes in the entire lifecycle of industrial system (Tao and Qi 2019). This has encouraged the European Monitoring and Control (M&C) market to invest €143 billion in this area by 2020, making a total of €500 billion invested by the worldwide M&C market (Colombo et al. 2014). CPSs has also aided to make advances in the anomaly detection field since companies now own more efficient and reliable monitoring systems, which facilitate data collection (Zhong et al. 2017). Industrial machines are complex systems and thus, they are typically monitored by multiple sensors, which form multi-sensor systems (Gołuch et al. 2018), to control the performance of all their components. Consequently, more data is now available in greater quantity and quality.

Although multi-sensor systems provide more data related to machines' condition, they also entail a number of challenges that must be addressed. The main challenge arises from the use of heterogeneous sensors, which means that they might be of a different nature, measure differently scaled real value data, or collect data at different frequencies (Cauteruccio et al. 2019). This fact often implies a previous data processing such as cleaning the data, extracting meaningful features, or reducing the dimensionality to convert this data into smart/usable data (Triguero et al. 2018). Overall, this is a very time-consuming task and it may require domain knowledge, that is, to have knowledge about the characteristics of the data or about what an anomaly looks like. In industrial multi-sensor systems, variability (Iglesias et al. 2017) is another key issue as it is common that the sensor configuration changes

depending on the current requirements, which includes installing, removing or modifying sensors. These changes imply that the model used to detect anomalies must be continuously adapted to the current configuration. In most cases, this involves retraining the model from scratch, which can be expensive in terms of time and computation. As inherent issues within the anomaly detection field, imbalance of the data is a particularly challenging problem due to the fact that, usually, there are very few anomalous observations available in comparison to normal observations. Furthermore, the boundary between normal and anomalous instances is often very thin, and data might also contain noise due to sensor malfunctioning or wrong measurements that may look similar to an anomaly.

In recent years, DL techniques have become the "de facto" approach for time series modeling and classification. In fact, they overcome most of the traditional Machine Learning (ML) algorithms as they obtain state of the art results in several domains (LeCun et al. 2015). Furthermore, it has been demonstrated that these algorithms are capable of modeling and classifying time series based on raw data without the need for expert knowledge in the domain. Although extensive research has been done to date in this field, the use of DL algorithms for multi-time series anomaly detection is not extended. Hence, the motivation of this thesis is to investigate new DL techniques to detect anomalies in industrial multi-sensor systems.

In the industrial domain, it is often equally important to detect anomalies as to identify the conditions that caused them. Having an insight into what causes a fault might be the key to fix it and to improve maintenance operations (Ren et al. 2017). However, identifying the error within multiple sensor data might be a hard task as long as an error might be produced as a result of a combination of events coming from multiple sensors. Moreover, DL algorithms are considered "black-boxes", meaning that it is yet not clear how they internally work. This fact hinders the ability to identify the reasons why the network has made one decision or another. Therefore, the presented DL anomaly detection technique, besides effectively identifying anomalies, it must also diagnose the anomalies o point out the underlying reasons why they occurred.

Furthermore, a reliable anomaly detection technique is not useful without an effective deployment infrastructure that allows performing a real-time anomaly detection of all CPSs of a company. In many industrial scenarios, systems often

work 24/7 and cannot be interrupted since this would cause a drop in production and a loss of money. In the era of the Industry 4.0 and the Internet of Things (IoT), these systems generate large volumes of data considering that they are continuously sending data involving their performance and that many of them might be monitored at the same time. Therefore, being able to process and manage these volumes of data becomes critical for the anomaly detection task since obtaining the correct information at the correct moment is a key issue for decision-making (Faiz and Edirisinghe 2009). In this context, Big Data frameworks and cloud computing are particularly important since they provide fast, scalable, and fault-tolerant data processing capabilities. Consequently, a Big Data infrastructure is required to gather data from industrial systems and process it in real-time to feed the anomaly detection model to detect anomalies as soon as they occur.

This thesis was originated by four main motivations derived from industrial challenges and the deficiencies of current methods to address them. The challenges are summarized below:

1. **Anomaly detection in heterogeneous multi-sensor systems:** considering that DL algorithms achieve state of the art results in time series classification but they are not extensively used for anomaly detection tasks, the main motivation is to research in new DL algorithms to detect anomalies in industrial multi-sensor systems.

2. **Adaptability to changing scenarios:** due to the variability on the configuration of multi-sensor systems as sensors are installed, removed, or modified, the aim is to design a DL anomaly detection technique capable of adapting to changing scenarios.

3. **Interpretability of DL algorithms for anomaly diagnosis:** considering that in the industrial domain identifying the reasons why an anomaly occurred might be as important as detecting it, that an anomaly can be hard to diagnose since it might be produced by data coming from multiple sensors, and that DL algorithms are "black-boxes", the goal is to investigate how to improve the interpretability of the DL algorithm to explain why it reaches a decision. Thus, it would be possible to point out what causes the anomaly to help the maintenance operator to fix it.

4. **Model deployment:** considering the large volumes of data generated by industrial systems that must be processed, the motivation is to research in Big Data approaches to deploy the anomaly detection model in a fast, scalable, and fault-tolerant infrastructure to detect anomalies in real-time.

Given these challenges, this thesis presents a DL based approach to address the outlined issues and thus, detect and diagnose anomalies in industrial heterogeneous multi-sensor systems.

1.2 Research Methodology

The research field of this thesis is moving fast due to technological advances and the continuous generation of new contributions in ML. Consequently, an iterative research methodology was followed. The main idea of this cyclical process is that the knowledge acquired in its initial phase helps to design an increasingly promising technique, capable of competing face-to-face against current methods, either in terms of results, or in terms of concept, offering originality and remarkable contributions. Figure 1.1 shows the different phases of this research methodology. These phases are briefly described below:

1. **Review of the current state of the art:** the main objective of this phase is to investigate the state of the art related to the field under consideration to identify problems and/or challenges. To achieve this, the related bibliography is used, reviewing publications from the scientific community published in journals, and proceedings of national and international congresses. The knowledge acquired in this phase should lead to a proposal to address the identified challenges.

2. **Design and development:** in this phase, a novel proposal to solve the identified challenges is designed (or modified, as long as it is not in the first cycle) and developed. To this end, previously acquired or updated knowledge is used to ensure that the solution is always up-to-date with the current state of the art.

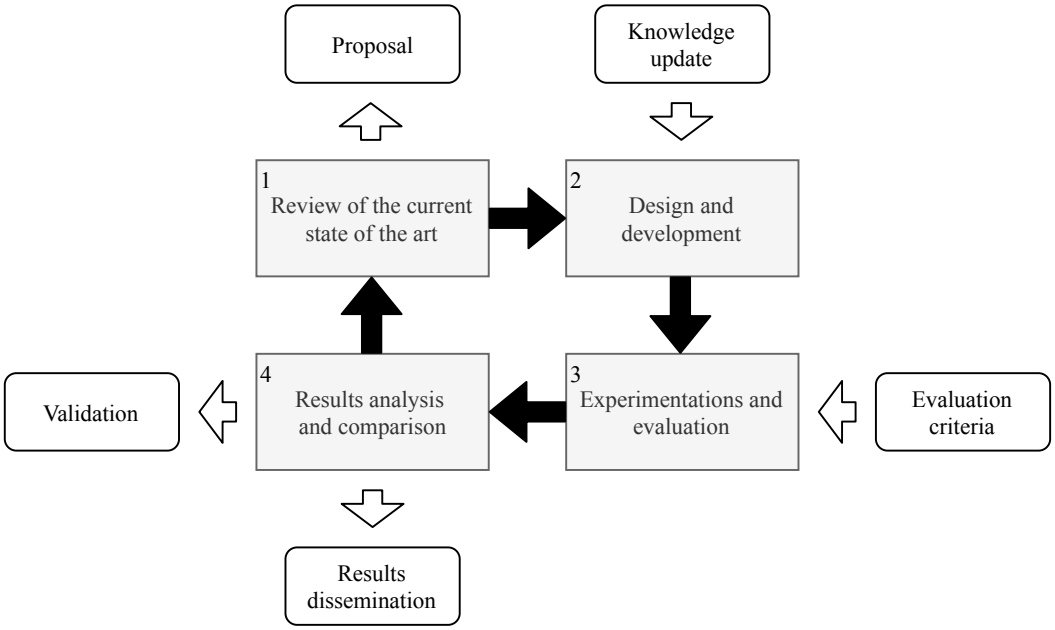


Figure 1.1: Followed research methodology.

3. **Experimentation and evaluation:** the goal of this phase is to subject the solution resulting from the previous step to a process of experimentation and evaluation. To carry out this procedure, it is crucial to provide some criteria and evaluation methods, which include the parameters and evaluation metrics, or the techniques with which the results will be compared in the subsequent phase. All these criteria and methods must be built using the knowledge acquired in the first stage of this methodology.

4. **Results analysis and comparison:** after conducting the pertinent experimentation, obtained results must be analyzed and contrasted with those obtained by other leading techniques existing in the literature. At this point, it is advisable to check whether the identified challenges are addressed. In such a case, the development of the thesis could be considered finished, always taking into account potential updates in the state of the art. Otherwise, the cycle begins again at its starting point. To this end, conclusions would have to be drawn to identify why the results have not yielded satisfactory results, thus gaining knowledge about how to improve the results in the next iteration. At the same time, and being aware that it is one of the most important aspects

of the development of the thesis, this stage must result in the dissemination of results, materialized in scientific production, either in congresses, as in national or international journals.

These four phases have been followed attractively to address the proposed challenges and generate the solutions and contributions presented in this thesis.

1.3 Research Hypotheses

In this thesis, three hypotheses are considered to be validated, which are presented below:

- **Hypothesis 1:** the use of DL methods in which the sensors are processed individually can improve the heterogeneous multi-time series anomaly detection task, besides dealing with changing scenarios. This hypothesis corresponds to challenges 1 and 2 (*Anomaly detection in heterogeneous multi-sensor systems* and *Adaptability to changing scenarios*).
- **Hypothesis 2:** the use of interpretability models can help to understand why DL algorithms reach a decision to diagnose an anomaly. This hypothesis corresponds to challenge 3 (*Interpretability of DL algorithms for anomaly diagnosis*).
- **Hypothesis 3:** the use of Big Data and cloud computing technologies can help real-time monitoring CPSs at large-scale. This hypothesis corresponds to challenge 4 (*Model deployment*).

To validate these hypotheses, three contributions are developed throughout this thesis, which have meant the definition of a working and experimental environment to obtain significant results.

1.4 Contributions

The research carried out during this thesis aims to contribute to the scientific community by addressing the identified challenges following the described methodology. Consequently, a novel approach for detecting and diagnosing anomalies in

heterogeneous multi-sensor systems is presented. Below, the main contributions of this thesis are briefly discussed:

- **Anomaly detection in multi-sensor systems:** a supervised anomaly detection technique based on DL is proposed in which a combination of Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) is used. This architecture uses an independent CNN to process each sensor data, which are defined as convolutional heads. Hence, it is referred to as Multi-head CNN-RNN. This architecture is valid to detect anomalies in multi-sensor systems and it allows to 1) work directly over raw data, 2) adapt each of the convolutional heads to the specific requirements of each sensor, and 3) adapt to changing configurations by adding or removing convolutional heads to the architecture.

- **Diagnosis of anomalies:** a system capable of diagnosing anomalies by identifying the reasons why the proposed Multi-head CNN-RNN architecture classifies an observation as anomalous is designed and implemented. This is achieved by attaching to the developed anomaly detection model an attention mechanism and the SHapley Additive exPlanations (SHAP) interpretability model, which in combination are able to identify the reasons why the model reaches a decision. The former identifies the most relevant time steps of the time series whereas the latter identifies the most relevant sensors, thus obtaining an accurate view of the instant and place where the anomaly occurred.

- **Large-scale CPS monitoring and model deployment:** a large-scale architecture for CPS real-time monitoring is proposed. This architecture also allows deploying in production the developed model to detect and diagnose anomalies in real-time. The proposed architecture enables gathering data from CPSs and processing it in a fast, scalable, and fault-tolerant manner to perform the anomaly detection by adopting Big Data and cloud computing technologies.

As an additional contribution, a large-scale Distributed Simulation (DS) framework is designed and implemented. This framework uses cloud computing technol-

ogy to enable running in parallel multiple conditions of a single physical model in the cloud in a fast, scalable, and fault-tolerant way.

1.5 Publications

During the research activities of this thesis, several international peer-reviewed journal and conference articles were published to disseminate obtained results. The publications can be found in Table 1.1.

TITLE Multi-Head CNN-RNN for Multi-Time Series Anomaly Detection: An industrial case study ***AUTHORS*** M. Canizo, I. Triguero, A. Conde, E. Onieva ***JOURNAL*** Neurocomputing (Impact Factor = 4.072 → Q1) ***STATUS*** Published. Vol. 363, pp. 246-260, 2019
TITLE Explaining a Deep Neural Network for Multi-Sensor Systems Fault Diagnosis ***AUTHORS*** M. Canizo, A. Conde, J. Labaien, E. Onieva ***JOURNAL*** IEEE Transactions on Industrial Informatics (Impact Factor = 7.377 → Q1) ***STATUS*** Submitted
TITLE Implementation of a Large-Scale Platform for Cyber-Physical System Real-Time Monitoring ***AUTHORS*** M. Canizo, A. Conde, S. Charramendieta, R. Miñón, R. G. Cid-Fuentes, E. Onieva ***JOURNAL*** IEEE Access (Impact Factor = 4.098 → Q1) ***STATUS*** Published. Vol. 7, pp. 52455-52466, 2019
TITLE CloudFMI Distributed Simulation Framework: speeding up experimentations ***AUTHORS*** M. Canizo, M. Gonzalez, A. Conde, O. Salgado, E. Onieva ***JOURNAL*** Simulation Modelling Practice and Theory (Impact Factor = 2.426 → Q1) ***STATUS*** Submitted
TITLE Real-time predictive maintenance for wind turbines using Big Data frameworks ***AUTHORS*** M. Canizo, I. Triguero, A. Conde, E. Onieva ***CONGRESS*** IEEE International Conference on Prognostics and Health Management, 2017
TITLE The use of Relational and NoSQL Databases in Industrial Asset Management ***AUTHORS*** J. Campos, P. Sharma, M. Canizo, E. Jantunen, D. Baglee, S. Charramendieta, A. Conde ***CONGRESS*** 13Th World Congress on Engineering Asset Management, 2018

Table 1.1: Publications in journals and conferences conducted during this thesis.

1.6 Research Context

In this section, some information about the institutions that have supported this research as well as about the research stay is provided.

1.6.1 Research Support

Below, a brief description of the institutions that participated in the development of this thesis is presented: Ikerlan, University of Deusto, and University of Nottingham.

Ikerlan

Ikerlan[1] (Figure 1.2) is a leading knowledge transfer technological centre providing competitive value to companies. It seeks for excellence in R&D&i, thanks to the continuous adaptation to the needs of its customers and the proximity with the business reality. Faithful to its mission, it has been working daily since 1974 to develop solutions that allow customers to become more and more competitive. Ikerlan is a cooperative member of the MONDRAGON Company.

Thanks to a unique cooperation model, which combines technology transfer activities, internal research, and training of highly qualified personnel, Ikerlan is currently the trusted technological partner of major companies in the country. Its research is structured in three technological specialization units: 1) electronics, information, and communication technologies, 2) energy and power electronics, and 3) advanced manufacturing.

University of Deusto

The University of Deusto[2] (Figure 1.3) was inaugurated in 1886. The concerns and cultural interest of the Basque Country in having their own university, as well as the interest of the Jesuits in establishing higher studies in some part of the Spanish State, coincided in its conception. Bilbao, a seaport and commercial city which was undergoing considerable industrial growth during that era, was chosen as the ideal

[1] https://www.ikerlan.es/en/

[2] https://www.deusto.es/cs/Satellite/deusto/en/university-deusto?cambioidioma=si

Figure 1.2: Ikerlan Technology Research Centre.

location. Currently, it also has faculties located in Donostia, Vitoria-Gasteiz, and Madrid.

As a university, its guidelines are the love of wisdom, desire for knowledge and rigor in scientific research and methodologies. Therefore, its main focus is on achieving excellence in research and education. Another objective is to provide the background for free persons, who are responsible citizens and competent professionals, equipped with the knowledge, values, and skills needed to take on the commitment to foster learning and transform society.

University of Nottingham

The University of Nottingham[3] (Figure 1.4) is consistently ranked among the world's top 100 universities (QS World University Rankings), and ranked #11 overall in the UK by 2019. It has over 46,000 students from 150 countries, overseas campuses in China and Malaysia, and strong links with universities and companies around the world, creating an inspiring place to study.

The university is committed to excellence, enterprise, and social responsibility. Consequently, its goal is to strengthen and enrich its core activities of Education and

[3]https://www.nottingham.ac.uk/

Figure 1.3: University of Deusto.

Research to develop skilled, reflective global citizens and leaders by undertaking fundamental and transformative discovery. The University of Nottingham offers an outstanding, broad-based, international education to talented students and is engaged internationally to enhance industry, health and well-being, policy formation, culture, and purposeful citizenship.

Figure 1.4: University of Nottingham.

1.6.2 Research Stay

During the second year of the Ph.D, an international research stay was made as part of the research activities. The research stay was carried out in Nottingham, concretely, in the University of Nottingham within the Automated Scheduling Optimisation and Planning (ASAP) Research Group.[4] Overall, the stay lasted three months, from February to May 2018.

The objective of the research stay was to collaborate with international experts in the ML field to share knowledge and get feedback from them. Therefore, it was possible to deepen into ML techniques for time series anomaly detection tasks. Consequently, a novel time series anomaly detection technique was designed and developed in collaboration, which was then published as a journal paper (Canizo et al. 2019b).

1.7 Outline

This dissertation is divided into 7 chapters. Next, a summary of each chapter is presented:

- **Chapter 2:** describes the definitions and the theory of the terms and algorithms used in this thesis. It also presents the industrial case study used to validate this work.
- **Chapter 3:** presents a review of the literature describing the trends, techniques, and resources used for multi-time series anomaly detection, DL interpretability methods for anomaly diagnosis, and large-scale CPS monitoring.
- **Chapter 4:** presents the proposed DL algorithm for multi-time series anomaly detection.
- **Chapter 5:** presents the methodology followed to interpret the output of the proposed anomaly detection model, providing information about why the algorithm reaches a decision and thus, diagnosing the anomalies.

[4] https://www.nottingham.ac.uk/research/groups/asap/

- **Chapter 6:** presents the Big Data architecture for CPS monitoring. This will be the module responsible to deploy the proposed anomaly detection and diagnosis models into an industrial production scenario.
- **Chapter 7:** concludes the work presented and discusses the reflections drawn throughout the work, as well as details future lines of research.

As a technologist, I see how AI and the fourth industrial revolution will impact every aspect of people's lives.

Fei-Fei Li

2

Background

This thesis focus on detecting and diagnosing anomalies on multi-time series using DL within the Industry 4.0 domain. This chapter presents a brief background and theory of some of the concepts used throughout this thesis. First, the Industry 4.0 is introduced, where the key enabling technologies are described and the necessity for efficient monitoring that allows detecting anomalies within the industrial machines is highlighted. Afterward, the basis of time series anomaly detection (Section 2.2), and how DL can yield this task (Section 2.3) are detailed. Next, what interpretability means within the ML field and why it is necessary for the industrial domain is explained (Section 2.4). Following, a brief description is provided about what Big Data and cloud computing are, besides explaining their role within the Industry 4.0 (Section 2.5). Finally, the industrial case study used to evaluate the developed contributions (Section 2.6), and the execution environment used during the experimentations (Section 2.7) are detailed.

2.1 Industry 4.0

The industry has evolved throughout the history driven by technological advances. In this way, the industry has experienced four revolutions during this time,

mainly determined by the implementation of the current cutting-edge technology to the industry. Figure 2.1[5] depicts the storyline of these revolutions. The first revolution changed the concept of manufacturing by implementing mechanization and steam power. The second revolution occurred when the mass-production assembly lines and electrical energy took place, thus enabling a giant step in production efficiency. The third revolution emerged as a consequence of automating production by means of computers and robots. The fourth revolution, in which we are now involved in, refers to the digitalization of the industrial processes involved in the previous revolution. Therefore, the production is not only automated, but all the data related to it is now also collected by sensors that measure the performance of the industrial equipment that facilitates the automation. However, the Industry 4.0 is not only about gathering data, but it is also about systems' connectivity and the use of the collected data for monitoring the performance of the industrial systems. Therefore, the Industry 4.0 cannot be understood without CPSs, IoT, and cloud computing, which are the key enabling technologies in this field.

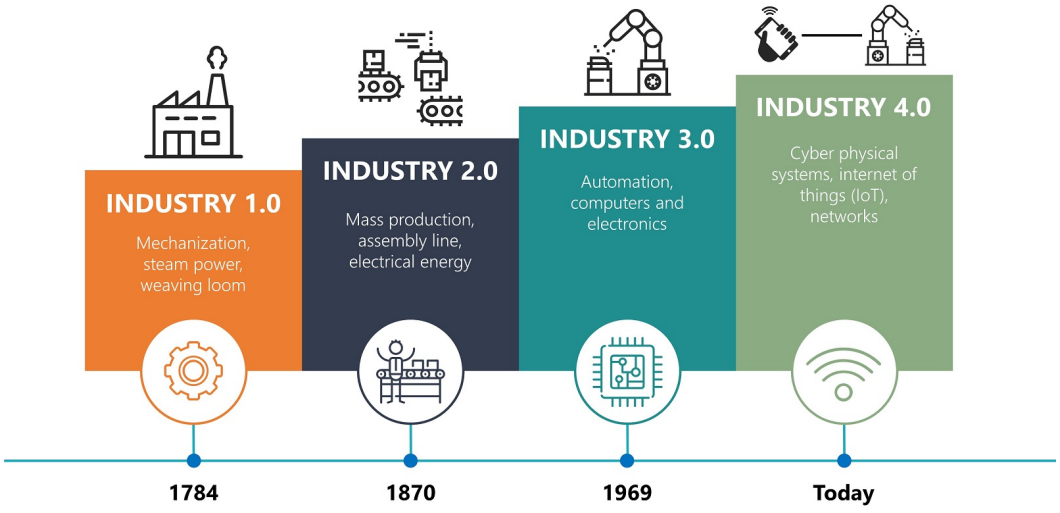


Figure 2.1: Visual representation of the industrial revolutions throughout the history.

The term Cyber-Physical System was first coined in 2006 by Helen Gill at the National Science Foundation. Briefly, it can be referred to as a new generation of

[5]http://trilliummfg.ca/the-rise-of-big-data-and-industry-4-0/

systems with integrated computational and physical capabilities that can interact with humans through many new modalities (Baheti and Gill 2011). In fact, CPSs can be defined as "*physical, biological and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core*" (Antsaklis et al. 2012), although more definitions can be found in the literature (Gerostathopoulos et al. 2016; Harrison et al. 2016; Leitão et al. 2016; Monostori et al. 2016). CPSs are composed of:

1. A device which refers to the physical component (hardware) that provides useful functionality and information when it is connected to a computer.

2. A software interface that is included within the hardware to send or receive data.

3. A communication interface to enable externalizing the gathered data.

Basically, CPSs are responsible for transforming industrial processes into digital data (digitalization) by means of sensors that measure the performance of the physical components. CPSs might be composed of multiple physical and sensor devices of different nature, where a single component can have several sensors installed on it. Moreover, due to the rapid growth of technology and the changing requirements of industrial systems, the CPSs are evolving systems since the physical components and/or sensors might vary due to addition, removal, or replacement of already installed devices (Z. Chen et al. 2017). As CPSs are equipped with multiple sensors, they form a multi-sensor system which, in addition to the different nature of the components and the fact that industrial machines usually operate 24/7, CPSs can continuously generate large amounts of heterogeneous data which must be then sent to the cloud to process it (J. Lee et al. 2015; J. Lee et al. 2014).

At this point is when the IoT takes place. The IoT is a paradigm that can be briefly defined as connecting any device to the Internet, although it can also refer to the connection of the devices to each other. In the broadest sense, the IoT is made up of devices such as sensors, smartphones, or wearables connected together. Within the industrial domain, this entails connecting the industrial machines to the Internet. Therefore, the IoT is responsible for pushing the data gathered by the CPSs to the Internet. To this end, the IoT is divided into four stages, which cover the entire

process that gathered data follows, from sensors to the Internet (Atzori et al. 2010). These stages refer to:

1. Sensors and actuators that gather data from the physical industrial system. In this stage, sensors are limited to gather data, whereas actuators can also interact with the hardware (i.e., switch on/off a physical component).

2. Internet getaways and data acquisition systems that perform data aggregations and control which data should be sent to the cloud since not all gathered data is useful. Besides, they also control when to push the data as it can be made continuously, periodically, or following a given strategy.

3. The edge, which is in charge of performing enhanced analytics and pre-processing. As many devices might send data to the cloud continuously, the goal of this stage is to reduce the volume of data sent to the cloud, avoiding network overheads. The underlying idea is to avoid processing the data coming from all the devices in the cloud, leaving to each device or group of devices processing the collected data and sending it ready to use. This is referred to as edge computing. Notice that this process is conducted close to the physical element (local) and thus, the available computational capacity is typically reduced. It should also be noted that this stage is optional as the raw data can be directly sent to the cloud without pre-processing it.

4. The cloud, which is responsible for processing all the data gathered in the previous stages.

The cloud (check Section 2.5 for more details) refers to a pool of virtualized computational resources that provides on-demand computing services including servers, storage, databases, networking, software, analytics, and intelligence over the cloud (Boss et al. 2007). Thus, it is called cloud computing to all the computation processes that are conducted in the cloud. Typically, the computational capacity of the cloud is larger than a common computer, which makes it suitable to process large volumes of data. Therefore, the cloud is responsible for providing computational power to process all the data generated by the industrial systems. Within the Industry 4.0, the cloud is used to centralize all the data gathered from a wide range

of industrial systems in order to perform further actions such as real-time monitoring, detect anomalies, or advanced analytics. Having all the data centralized makes it possible to have a general view of all the industrial systems, thus improving the decision-making. Furthermore, the cloud is responsible for hosting web interfaces and applications that are used to visualize all the data related to the condition of the CPSs.

Figure 2.2 shows the CPS, the IoT, and cloud computing technologies integrated within the Industry 4.0. As it can be shown, the last two components of the CPS and the first two stages of the IoT seems to be similar. Actually, the sensors and actuators equipped within the CPSs are IoT devices that allow sending data to the cloud. Similarly, cloud computing is also within the IoT (fourth stage). Therefore, the lines that determine when the CPS, IoT, and cloud computing start and finalize are often blurred. In this thesis, we consider that the CPS ends when the data is sent to the cloud through the gateways (second IoT stage) and the IoT finalizes when the data reaches the cloud. Hence, we consider that the cloud is not within the scope of the IoT and that it is a distinguished component.

Considering all this, it can be said that the Industry 4.0 is the combination of CPSs, IoT and cloud computing technologies, which form a complete technological stack to gather all the data related to the performance of industrial systems and to monitor their operational status, thus leading to a smart industry or smart manufacturing (Kang et al. 2016). One of the main goals of the Industry 4.0 is to monitor the condition of the industrial systems to avoid down-times due to unexpected faults (Kagerman and Wahlster 2013). Therefore, an efficient anomaly detection system is required, which must be capable of detecting anomalous behaviors within the large amount of heterogeneous data generated by the CPSs. Furthermore, a monitoring system capable of efficiently managing all this data is required in order to shorten the time between the system failure and its detection (Kagerman and Wahlster 2013).

2.2 Time Series Anomaly Detection

The data gathered from the sensors installed on the industrial systems is usually regarded as a chronologically ordered sequence of measurements, referred to as time series (Das 1994). Thus, the anomaly detection system required for monitoring

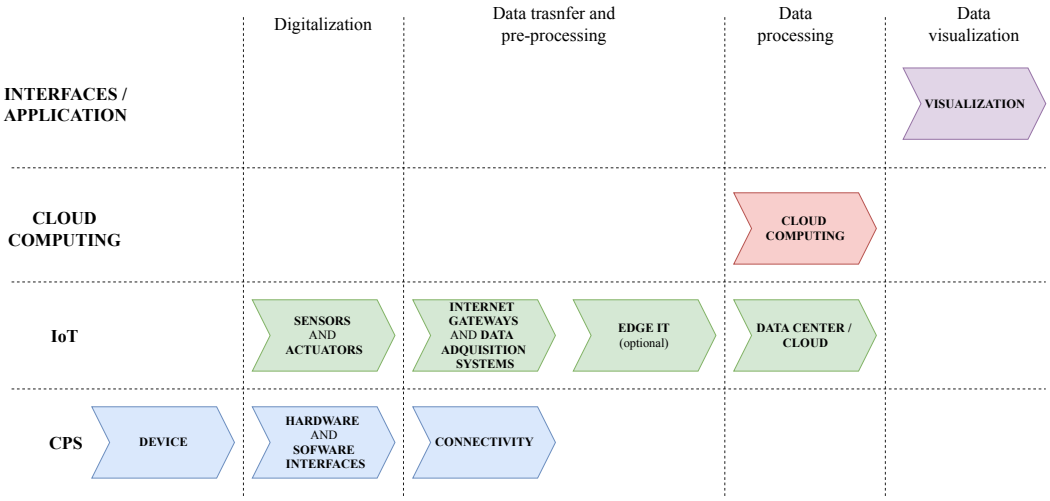


Figure 2.2: Diagram of the stages and elements involved in the development of the Industry 4.0.

the operating status of industrial systems must deal with time series data. This section provides a brief overview of the field of anomaly detection in time series. Specifically, it discusses the concept of anomaly (Section 2.2.1), the differences between univariate and multivariate time series (Section 2.2.2), the different types of anomalies covered in this thesis (Section 2.2.3), and the existing types of anomaly detection techniques (Section 2.2.4).

2.2.1 Anomaly Detection Definition

An anomaly can be defined as a pattern in data that do not conform to expected behavior (Chandola et al. 2009). However, several definitions of what an anomaly is can be found in the literature (Fu 2011). These definitions might vary depending on the field or task under consideration. Similarly, other words such as novelty or outlier are also used in reference to an anomaly (Pimentel et al. 2014).

Although these terms can be used interchangeably, there is not any definition that is universally accepted. For instance, a novelty can be defined as something new that does not resemble anything previously seen (Markou and Singh 2003). Thus, a novelty does not necessarily have to be a fault. On the other hand, an outlier can be defined as an observation that deviates significantly or that appears inconsistent

with respect to other observations within the data (Barnett and Lewis 1994). An outlier can also be shown as an observation or set of observations that are far from most of the normal observations in the feature space (Markou and Singh 2006).

Unlike traditional anomalies, time series anomalies are time-dependent. The main difference is that the input attributes are temporally ordered so that this relationship must also be considered. This means that an anomaly is not only given by a novelty or outlier but also by the time in which it occurs. Therefore, an observation within a time series might be normal at a given period but abnormal on another. Since the values of a time series are dependent on previous events, Chandola's anomaly definition seems more appropriate as a novelty or outlier can be considered in a time series as an unexpected behavior.

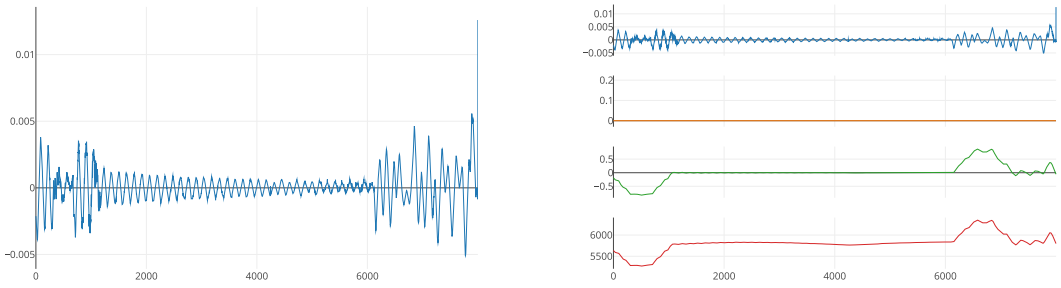
In the industrial context, an anomaly is usually associated to a fault or failure (interchangeable terms) in industrial systems, where a fault can be defined as an unpermitted deviation of at least one feature of the system from the acceptable, usual, or standard condition (Isermann 2006). Hence, the detection of a fault can be considered as finding the existence of failures in monitored systems by using dependencies between different measurable signals (time series) (Biljanovicc 2011).

As far as this thesis focus on detecting anomalies in industrial systems, we define anomaly detection as a combination of Chandola's anomaly detection definition and fault detection definition. In this way, we define time series anomaly detection as the technique of finding patterns in data that do not conform to the expected behavior and that lead to a potential failure in the industrial system.

2.2.2 Univariate vs Multivariate Time Series Anomaly Detection

A time series is a sequence of data consisting of time-stamped entries ordered in time, which can have one to many measurements associated with each time-stamp. Thus, time series with a single observation measured at each time interval are considered as univariate time series, whereas time series with multiple variables at each time-stamp are considered as multivariate time series. Figure 2.3 shows both types of time series. Note that a multivariate time series can also be seen as multiple univariate time series. As we focus on anomaly detection based on signals coming

from multiple sensors, we treat each sensor data as a univariate signal. Consequently, we define them as multi-time series.



(a) Univariate time series **(b)** Multivariate time series

Figure 2.3: Type of time series depending on the number of variables.

Detecting anomalies in univariate time series is a well-known research field (Gupta et al. 2014). However, real word problems are typically complex and thus, it is not possible to fit them with only one variable. In this scenarios, multiple variables are used to describe all aspect of the scenario under consideration, leading to multi-time series. This fact hinders the task of detecting anomalies due to the fact that each variable does not only depend on its past values but also on other time-dependent variables. Consequently, it is necessary to consider the relationship between time series, since an anomaly might occur when a certain set of time series have a specific joint behavior at the same time (J. Wu et al. 2018). Similarly, two or more weakly relevant features might become significantly useful when working together. On the contrary, two relevant features might become redundant when working with each other because they provide correlated information.

Some of the multivariate time series anomaly detection techniques existing in the literature does not typically perform well if they are directly applied over raw time series. Furthermore, in scenarios such as the industrial in which systems are continuously monitored, time series might be large due to continuous streams of data. Therefore, anomaly detection could be more challenging because of the huge amount of data to be processed and analyzed, and the complexity of setting a boundary between normal and abnormal observations. In fact, some traditional algorithms such as Random Forest (RF), Support Vector Machines (SVM), or k-nearest neighbor suffer from high dimensionality and/or processing of large volumes

of data. Overall, they require analyzing the time series to find correlation between them (Bankó and Abonyi 2012), cleaning the data, extracting meaningful features (Toshniwal 2009), or reducing the dimensionality (Bianchi et al. 2015; Krawczak and Szkatuła 2014) to convert this data into smart/usable data (Triguero et al. 2018), as shown in (J. Li et al. 2017; Serdio et al. 2014; Theissler 2017). This is typically a very time-consuming task and it may require expert knowledge in the domain and about what an anomaly looks like.

2.2.3 Time Series Anomaly Types

According to (Chandola et al. 2009), time series anomalies can be categorized into three main types: point, context-specific, and collective. A brief description of these anomalies is presented below:

- **Point anomaly:** is the simplest form of an anomaly since it refers to a single point that deviates from the expected behavior. Figure 2.4 shows an example of this. Note that in the case of multi-time series, a point anomaly can be shown in some of them at the same time interval since they might be correlated.

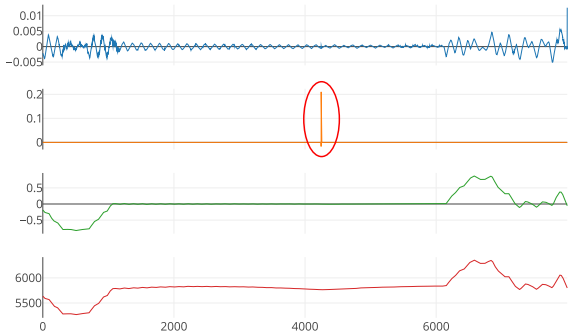


Figure 2.4: Point anomaly example. The red circle points out the anomaly.

- **Context-specific anomaly:** refers to a point or sequence of points that are considered as anomalous with respect to the time interval in which they occur, that is, they can be considered as normal in a given condition and anomalous in another. Figure 2.5 shows an example of this type of anomaly, in which the same example as in the above figure is followed. It can be observed how the

expected behavior in the interval marked by the red circle is ascending, while in this case, it is descending.

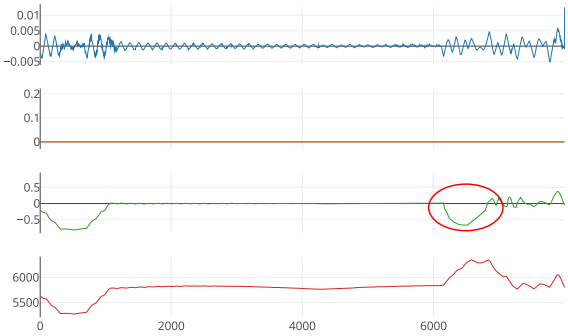


Figure 2.5: Context-specific anomaly example. The red circle points out the anomaly.

- **Collective anomaly:** refers to a sequence of points which, if treated as individual points, do not form an anomaly, but they can show an unexpected behavior if treated as a group or set of points. Figure 2.6 depicts this type of anomaly, in which the previous example is followed. It must be noted that in multi-time series, a collective anomaly can be considered as a group of points of different time series.

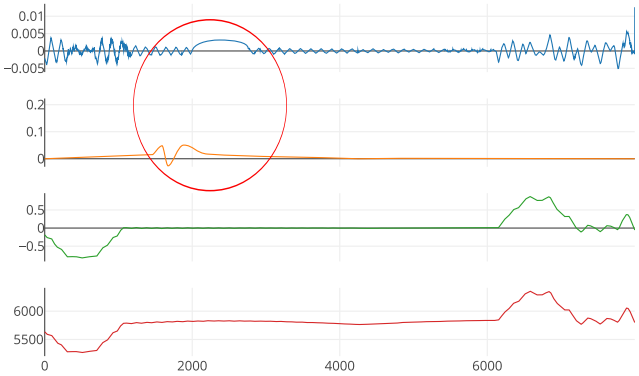


Figure 2.6: Collective anomaly example. The red circles point out the anomalies.

2.2.4 Machine Learning Methods in Time Series Anomaly Detection

Within the ML field, anomaly detection techniques can be classified into three main categories (Chandola et al. 2009): supervised, semi-supervised, and unsupervised. Techniques regarding these categories differ in the type of data they use, which may be labeled or not. The fact that the data is labeled means that each observation of the dataset has an associated label that identifies it as normal or anomalous. In contrast, if no label is available, it is not possible to know the nature of a given observation. Next, a brief description of these categories is presented:

- **Supervised techniques:** are used when the entire training dataset is labeled. In the training phase, there are usually input variables (x) and an associated output variable (y). Then, an algorithm is used to learn the mapping function from the input to the output ($y = f(x)$). Therefore, they attempt to uncover insights, patterns, and relationships within the training dataset to relate them to the target class (y). The goal is to approximate the mapping function in order to correctly predict the output variable (y) given a new input data (x). Supervised techniques can also be categorized into classification and regression techniques. The former is used when the output variable is a categorical variable (i.e., normal or anomalous). Thus, the output of the algorithm indicates the probability that an observation belongs to one class or to another. The latter is used when the output variable is continuous, meaning that the output is numerical.

- **Semi-supervised techniques:** are appropriate in environments where labeled data exist but only represent normal behavior. They are also used in scenarios where although anomalous data is available, there is not enough data to learn it in a supervised way. At training phase, these techniques typically use only observations regarding the normal class in order to model the normal behavior. Afterward, they classify as anomalous all observations that do not match the normal distribution.

- **Unsupervised techniques:** are used when no labeled data is available. These types of algorithms assume that normal data or sequences are much more fre-

quent than anomalous ones. Thus, they classify as normal those observations that follow the most frequent patterns, whereas they classify as anomalous those observations that deviate from those patterns. As the data is not labeled, these techniques do not require prior knowledge since they evaluate data based exclusively on the intrinsic properties of the dataset. Unsupervised techniques can be further classified into clustering and association problems. The former, attempt to group similar observations together by identifying similar patterns. Therefore, each group would correspond to a different class. The latter, try to analyze data for patterns, identifying frequent if-then associations, which are called association rules. Hence, these methods associate patterns that lead to an anomaly.

2.3 Time Series Modeling with Deep Learning

In recent years, DL has obtained promising results modeling temporal data (LeCun et al. 2015). In this way, DL algorithms might help at detecting anomalies within time series due to its powerful learning process. However, DL algorithms may be complex to comprehend and/or to implement. Therefore, this section provides a detailed description of the use of DL for time series modeling. First, a general description of DL is presented (Section 2.3.1). Afterward, how time series are processed using both CNNs (Section 2.3.2) and RNNs (Section 2.3.3) is explained, which are currently the most used DL algorithms for processing time series. Hence, these are the base algorithms used in this thesis.

2.3.1 Deep Learning Definition

DL is a subfield of ML that is based on Artificial Neural Networks (ANN)s. ANNs were designed to simulate the human brain in an effort to provide computers with the capabilities of humans to learn. The very beginning of modern ANNs is the perceptron (Rosenblatt 1958), which is a binary linear classifier that attempts to simulate a biological neuron, and represents the simplest form of neural network. Figure 2.7 shows its internal structure. As it can be observed, a perceptron takes one to n variables as input and it then performs a weighted sum. To this end, each input

variable has an associated weight that measures how relevant each variable is. In this way, some variables will have more influence than others in the final decision. Finally, a linear activation function is applied over the resulting weighted sum to get a binary output.

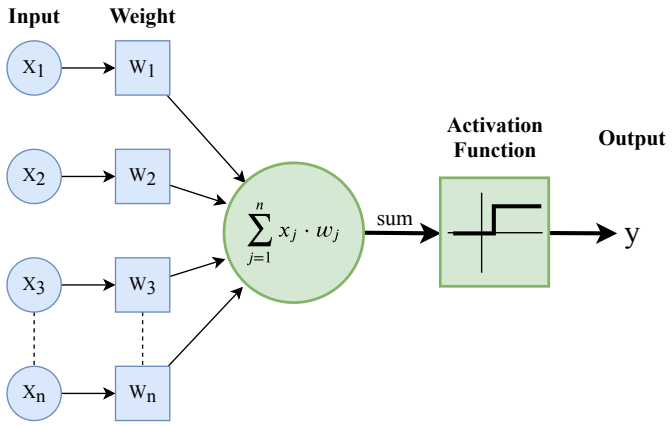


Figure 2.7: Structure of a perceptron.

Based on the perceptron, the Multilayer Perceptron (MLP) (Hornik et al. 1989) was one of the first most popular ANN. The MLP is a supervised feed-forward neural network which is composed of stacked layers of perceptrons (neurons), thus simulating a human brain where multiple layers of neurons are connected to each other. The MLP is composed of at least three layers: input, hidden, and output. Figure 2.8 shows an example of this. In this network, the data flows through the network from the first to the last layer. Here, the neurons represent a computational node, whereas the edges represent the corresponding weights. In the MLP, the input layer refers to the layer responsible for feeding input data into the network. This layer contains as many neurons as input variables are. Hidden layers are considered those all layers located between the input and the output layer, and there can be one to l layers. Opposite to the original perceptron, neurons within these layers apply non-linear activation functions due to the fact that an output (y) that varies non-linearly with its explanatory variables is required. Note that a simple linear function cannot approximate every function ($f(X)$) and that if a linear function is used, the whole network would behave like a perceptron since the sum of all the layers would result in another linear function. Finally, the output layer is responsible

for giving the final result (y). For classifications tasks, this layer typically contains as many neurons as target classes (labels). It is worth mentioning that all neurons of a given layer are fully connected to the rest of the neurons of the next layer, but not between neurons in the same layer.

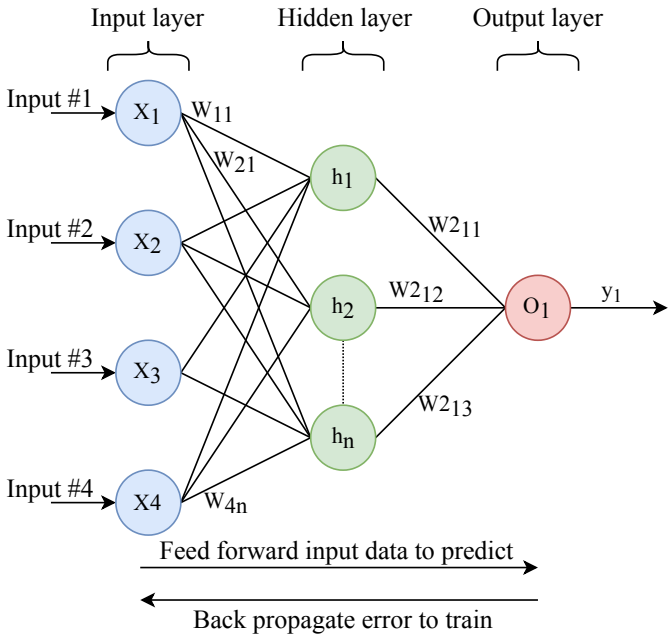


Figure 2.8: Multi Layer Perceptron.

At the training phase, the input data is feed-forwarded into the network until obtaining the final prediction. At this stage, all the neurons of each layer process the corresponding received data (equal to how the perceptron does but with non-linear activation function), which may come from the input data or from other neurons within the previous layer. As this is a supervised method, the output values are compared against the ground truth to compute the error between them. Next, the error is propagated from the last to the first layers of the network using the back-propagation method (Rumelhart et al. 1986). Hence, all the weights are adjusted consequently in order to approximate the target function.

As mentioned, the number of hidden layers can be increased as much as required, where a more abstract representation of the input data can be obtained as this number increases, thus allowing to solve more complex problems. Since the number of hidden layers determines the depth of ANNs, DL can also be defined according to

the depth of the network (how large the network is), considering as Deep Neural Network (DNN)s those networks that contain a larger number of hidden layers. However, there is not a concrete number of layers at which a network is considered as "deep" (Schmidhuber 2015). Therefore, the following definition seems more appropriate as it matches the previous statement:

> "*Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. [...] The key aspect of deep learning is that these layers of features are not designed by human engineers: they are learned from data using a general-purpose learning procedure* (Goodfellow et al. 2016)."

In this way, DL algorithms are able to build hierarchical concepts by stacking lower to higher feature representations to learn complex functions, mapping the input to the output directly from raw data, without requiring human-based features (Bengio 2009; Bengio 2012).

As a downside, increasing the number of hidden layers will also increase the required computational power to train the network, since computing the calculations of each neuron and then adjusting their weights is a computationally expensive process. Furthermore, ANNs might suffer from vanishing/exploding gradient problem (Pascanu et al. 2012) as very little values will prevent weights from being changed and could stop the network to train, whereas large values will result in large updates to the network weights and, in turn, leading to an unstable network. These were the main reasons for the poor success of ANNs until a decade ago (approximately), when technology advances provided larger computational power to apply large DL within reasonable time intervals. Moreover, new DL architectures such as CNNs and RNNs were designed not to connect all the neurons to each other, thus reducing the required number of operations, in addition to better managing the learning process to avoid the vanishing/exploding gradients problem. As a consequence, large neural networks can currently be trained with large volumes of data to solve more complex

problems, outperforming more traditional algorithms in a wide range of domains. As Andrew Ng stated:

> "*For most flavors of the old generations of learning algorithms [...] performance will plateau. [...] deep learning is the first class of algorithms that is scalable. [...] performance just keeps getting better as you feed them more data* (Goodfellow et al. 2016)."

2.3.2 Time Series Processing with Convolutional Neural Networks

A CNN is a type of DL algorithm that was originally designed for processing images although it was used mainly for character recognition tasks (Lecun et al. 1998). Currently, they are also used for tasks such as signal (P. Wang et al. 2016) or text (Salamon and Bello 2017) processing. Its success lies in its ability to extract the most relevant features from input data, which is then used to make the final decision.

As motioned before, connecting all the neurons to each other is impractical and thus CNNs connect each neuron to a local region of the input data. This local connectivity is called filter or kernel (these terms are used interchangeably) and it is slid over the input data to extract features from it. A filter is a set of weights which are learned during the training phase. In this context, a convolution is defined as the dot product between the filter and the segment of the input data where the filter is applied. As a result of the convolution, a feature map containing the features of a given segment is obtained. Thus, filters act as feature detectors. At the time of applying a convolution, multiple filters can be used so that different features of the same data can be detected. Furthermore, convolutional layers are composable, meaning that the output of a convolutional layer can be fed into another. Consequently, several convolutional layers can be stacked to form a deeper CNN so that the network can detect higher level, more abstract features.

Regarding time series processing, CNNs use a one-dimensional (1D) kernel as far as time series work in the time domain (one dimension). 1D CNNs are suitable for processing sequential data such as voice records, sensors signals, or text data, among others. The benefit of using 1D CNNs for sequence processing is that they can learn from raw time series data directly and thus, they do not require domain expertise to manually perform feature engineering. Therefore, 1D CNNs can learn

the internal representation of a time series and might obtain comparable performance to other algorithms that fit on a version of the dataset with engineered features.

Figure 2.9 shows an example of a 1D CNN architecture, which is composed by the following layers:

- **Input layer:** holds raw time series in a matrix (W_I, H_I), where W_I refers to the length of the time series, and H_I to the number of time series or variables. In this case, the input data corresponds to five time series of length 200 (200×5).
- **1D convolutional layer:** computes the output of neurons that are connected to local regions in the input, each computing a dot product between the filters (F) and the segment of data where they are applied, plus a bias (b). As a result of the convolution, a matrix of (200×100) is obtained as 100 filters are used. To slide the filters along the width of the time series, a stride (S) must be set to indicate how much they have to move. Moreover, often the input data is padded with zeros around the border. Both the stride and the number of padded zeros (P) are hyperparameters that are used to control the spatial size of the output.
- **Activation layer:** introduces a non-linearity into the output of a neuron. It decides whether a neuron should be activated or not by calculating a weighted sum and further adding bias with it. Rectified Linear Units (ReLU) is the most used layer. Note that this layer does not change the output shape.
- **Pooling layer:** performs a downsampling operation in the temporal dimension to reduce the dimensionality and thus the number of parameters. *Max pooling* is the most used. The result of applying a pooling layer of size 2, is to halve the length of the time series. Thus, the output shape will refer to (100×100).
- **Flatten layer:** flattens the data introduced in this layer to format it for the next layer. This results in an output shape of (1×10.000).
- **Fully Connected layer:** computes the class score, which indicates what probability the input data has of being assigned to each class. In this case, the output shape will be (1×4) as there are four classes.

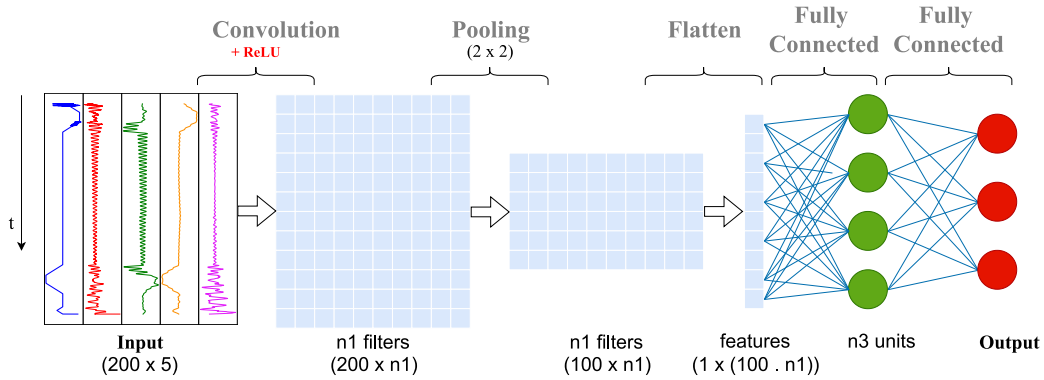


Figure 2.9: Example of a 1D CNN architecture for time series processing.

To deepen in how the convolution is carried out, Figure 2.10 depicts this process, following the previous example. Note that $F = 3$, $S = 1$, and $P = 1$. As mentioned before, the convolution is a dot product between filters and the region of data where these filters are applied, which is of size $(F \times W_I)$. Therefore, this region contains a small segment of all the time series. The output (O) of the convolution is a matrix $(W_O \times H_O)$, which size is computed by Equations 2.1 and 2.2. Each of the elements of the output matrix (O_{ij}) is computed by Equation 2.3, where "*" refers to an element-wise multiplication.

$$W_O = (W_I - F + 2P)/S + 1 \tag{2.1}$$

$$H_O = K \tag{2.2}$$

$$O_{ij} = sum(X[i : (i + F), :] * F_j) + b_j \tag{2.3}$$

It is worth highlighting that the convolution is applied to all the time series as a whole and therefore, the values of the output matrix are features of all the time series mixed all together.

2.3.3 Time Series Processing with Recurrent Neural Networks

RNNs are DL architectures that are able to remember past events, which makes them suitable for time series processing. In RNNs, the data cycles through a loop, unlike in traditional neural networks where the information only flows in one direction. Thus, the network uses what it has learned before in addition to the

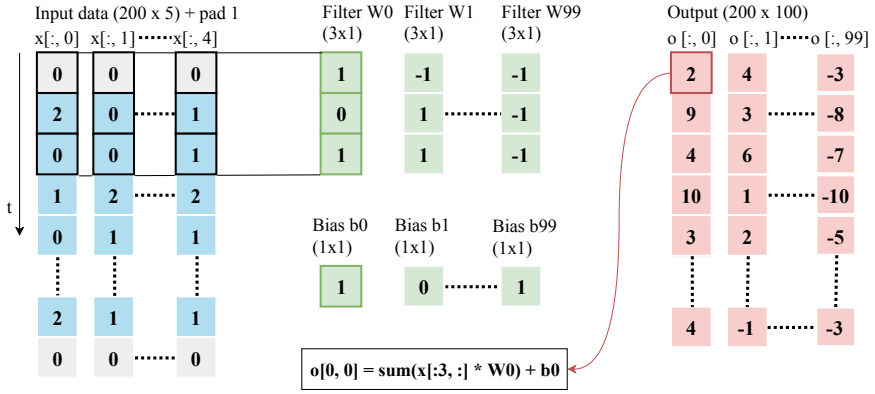


Figure 2.10: 1D convolution example. Note that $F = 3$, $S = 1$, and $P = 1$.

current input to make a decision. To this end, the neurons of the recurrent layers, called units, have two inputs instead of one: current and recent past data. As a consequence, an internal memory is formed with which temporary behaviors can be captured throughout a sequence. This fact can be shown in Figure 2.11, where an unrolled RNN layer is depicted. As it can be observed, the knowledge acquired in the previous time-steps is passed through the RNN blocks (A) until the last time-steps. The input (X_t) of RNNs is of three dimensions ($batch\_size$, $time_steps$, $features$), where the first refers to the number of samples within a batch, the second to the lagged observations within the time series, and the third to the number of features of the time series.

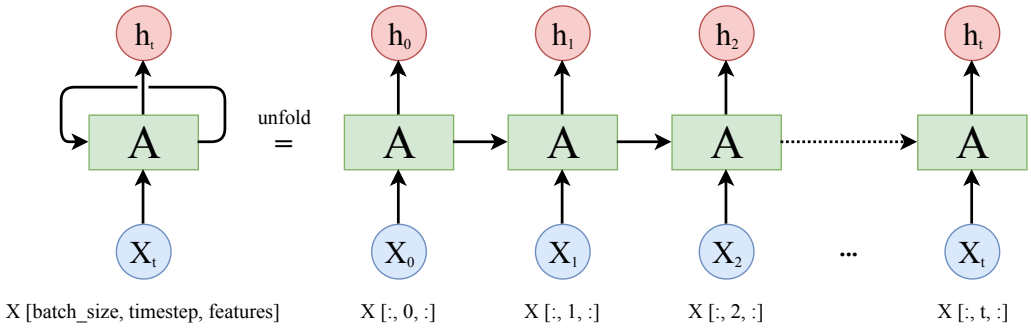


Figure 2.11: An unrolled recurrent neural network.

In this way, RNNs can process any sequence of vectors, which can operate in the input, the output, or in both cases. This fact depends on the input data and the required output shape. Considering this fact, there are typically three scenarios,

which are depicted in Figure 2.12. The first scenario is referred to as "*one to many*", where a single data is introduced as input to the model and a sequence is obtained as a result. This is the case of image captioning since an image is taken as input and a sequence of words as output. Conversely, the second scenario corresponds to the "*many to one*" category, where a sequence is used as input and a single output is obtained as a result. Sequence classification is the most common in this category since a sequence is introduced as input and a single classification score as output. The third scenario is referred to as "many to many", where a sequence is used in both input and output. Within this category, machine translation is a common task where the model reads a sentence in one language and then outputs a sentence in another one. In this work, we focus on the "*many to one*" as we are detecting anomalies in time series and thus, we have sequential data as input and a classification score as output.

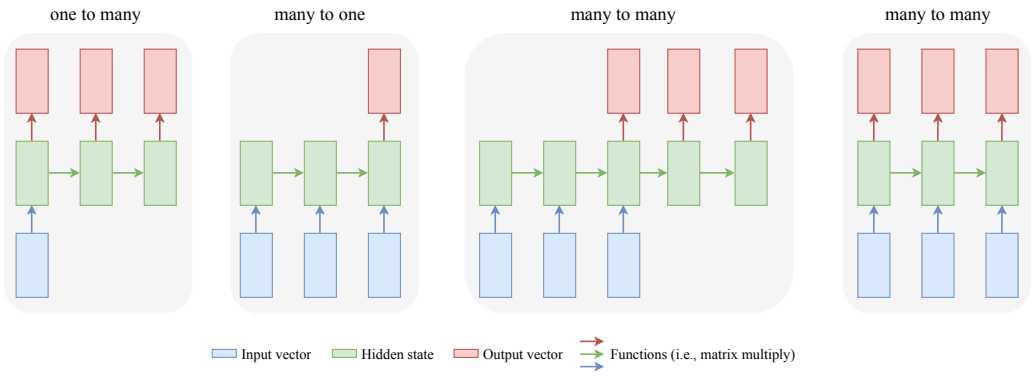


Figure 2.12: Input-output relationship modes for RNNs.

Although RNNs were designed to remember long-term dependencies, the fact is that they struggle to learn them. This is due to how the output (h_t) of each recurrent unit is processed internally, as a single *tanh* layer is only used. Consequently, RNNs are limited to learn short-term dependencies (Bengio et al. 1994). Furthermore, this fact makes this type of network to be vulnerable to the exploding/vanishing gradients problem (Hanin 2018) at training time. As a consequence, Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) emerged in an effort to avoid these problems.

2.3.3.1 Long-Short Term Memory (LSTM)

LSTMs are a type of RNN that were designed by Hochreiter & Schmidhuber (Hochreiter and Schmidhuber 1997) to learn long-term dependencies. LSTMs differ from RNNs in the way in which they internally process the data and in how they transfer their knowledge through time.

Figure 2.13 shows the internal structure of a LSTM unit. Unlike traditional RNNs, LSTMs have a cell state (C_t) that runs straight down the entire chain of units. This cell is used to flow the information along the chain with only some minor linear interactions, which facilitates learning long-term dependencies. Each LSTM unit has also a hidden state (h) to handle its internal state. The LSTM has the ability to add or remove information from the cell state by means of structures, called gates, that manage the information flowing through the cell state. There are three gates: forget (f_t), input (i_t), and output (o_t) gates.

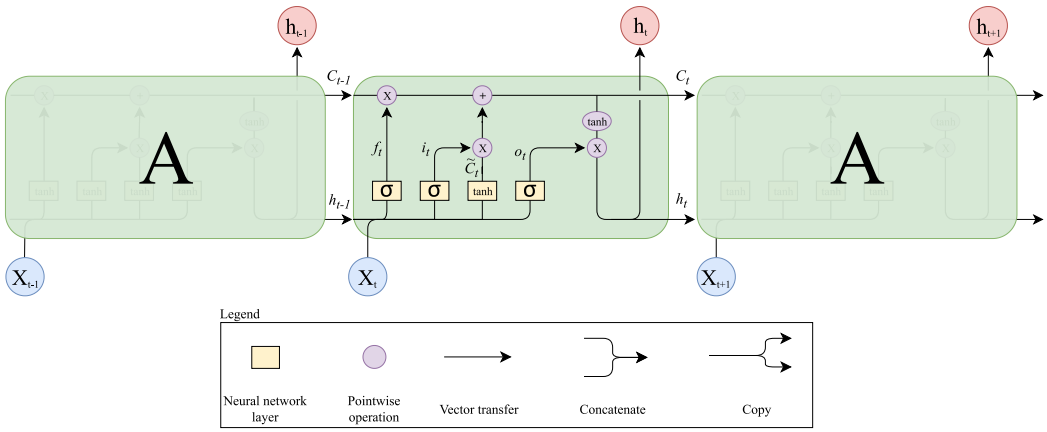


Figure 2.13: LSTMs' internal structure.

First, the information that is not relevant is removed from the cell state by using the forget gate, which is computed by Equation 2.4. In the following equations, W refers to the weights, σ refers to a *sigmoid* function, and the b refers to the bias. It is also worth to point out that "$\cdot$" denotes a normal multiplication, whereas "$*$" denotes a point-wise multiplication.

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

Afterward, new information is added to the cell state through the input gate. This gate is composed of a *sigmoid* layer that decides which values will be updated (Equation 2.5), and a *tanh* layer that creates a vector of new candidate values ($\tilde{C}_t$) that could be added to the cell state (Equation 2.6). Then, the result of both layers is combined to obtain the values to create or update the cell state.

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

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

Once that the information to be removed and added from the cell state is already decided, it must be updated correspondingly. To do so, the old state (C_{t-1}) is first multiplied by f_t to forget not relevant features, and then the new candidate values are added to the cell state. This is computed by Equation 2.7.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.7}$$

Finally, it must be decided the information that is forwarded to the next time-step, as the output of the LSTM block. This is performed by applying a *sigmoid* layer to the input data, and a *tanh* layer to the cell state (Equation 2.8). Then, both results are multiplied to select what information to send to the next LSTM unit (Equation 2.9) and to the next layer.

$$o_t = \sigma \cdot (W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.8}$$

$$h_t = o_t * \tanh(C_t) \tag{2.9}$$

2.3.3.2 Gated Recurrent Unit (GRU)

A GRU (K. Cho et al. 2014) is another type of RNN that surged from the LSTM in an effort to facilitate the training of these types of neural networks. Although LSTMs have successfully been applied in a wide range of domains, it can be challenging to train them due to their complexity. The GRU can be seen as a simplified version of LSTMs, since it only has two gates: update (z_t) and reset (r_t) gates. It unifies the forget and input gates into a single update gate, and the forget

gate is changed by the reset gate. Moreover, it merges the cell and hidden states. Figure 2.14 shows the internal structure of the GRU.

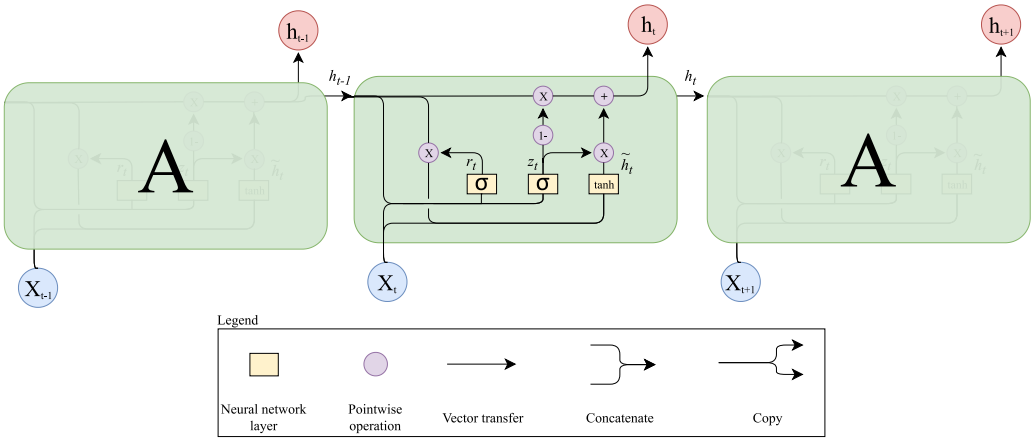


Figure 2.14: GRU internal structure.

In this case, the first step is to update the values to determine how much of the past information must be forwarded along the chain. To this extent, the input data and the previous hidden state are multiplied by their corresponding weights (W^z and U^z, respectively). Afterward, both results are added together and a *sigmoid* layer is then applied to obtain a score between 0 and 1. This is computed by Equation 2.10.

$$z_t = \sigma \cdot (W^{(z)} \cdot x_t + U^{(z)} + h_{t-1}) \tag{2.10}$$

The second step is to decide how much of the past information to forget. To do so, the reset gate is used, which is computed by Equation 2.11. As it can be observed, this equation looks similar to the previous one. They differ in the usage of its weights and gates.

$$r_t = \sigma \cdot (W^{(r)} \cdot x_t + U^{(r)} + h_{t-1}) \tag{2.11}$$

After applying both gates, new memory content must be introduced to the current memory ($\tilde{h}_t$) which will use the reset gate to store the relevant information from the past. This is computed by Equation 2.12.

$$\tilde{h}_t = \tanh(W^{(h)} \cdot x_t + (r_t * h_{t-1}) \cdot W^{(h)}) \tag{2.12}$$

Finally, it must be computed which information to push along the chain (h_t). For this matter, the update gate is used to determine what to collect from both the current memory and the previous hidden state. This is computed by Equation 2.13.

$$h_t = (1 - z) * h_{t-1} + z_t * \tilde{h}_t \tag{2.13}$$

Although GRUs are simpler than LSTMs, they are able to achieve similar results in several domains.

2.4 Interpretability in Machine Learning

Within the ML field, the techniques that are typically used for detecting anomalies lack interpretability, which means that it is not possible to understand the reasons why the model has made one decision or another. Therefore, the interpretability of ML models is a critical aspect in several domains such as the industrial. Nevertheless, there is not yet a universally accepted definition of what interpretability is. It can be defined as the degree to which a human can understand the cause of a decision (B. Kim et al. 2016), or to the degree to which a human can consistently predict the model's result (Miller 2019). Therefore, interpretability means explaining the decision made by the model in a human-understandable manner.

Having an insight into the output of a ML model can entail a number of benefits at the time of building or using it. First, it may help to debug a model since understanding how the model reaches a given output can yield the identification of potential errors. It can also improve feature engineering (C. R. Turner et al. 1999) by informing how the model behaves depending on the introduced features. On the other hand, there are cases in which the output of the model is used by a human to make the final decision and thus, explaining the output can be more valuable than the prediction itself. Similarly, explaining the output of the model can help humans to trust the model by showing that the model understands the underlying problem (Doshi-Velez and B. Kim 2017).

It must be noted that some ML algorithms are easier than others to interpret, which will be determined by their internal structure and learning process. Although the nature of the model highly impacts on the difficulty of interpreting a model, the interpretability is not a property of ML models (R. Turner 2016). Typically, complex

algorithms obtain better accuracy and are able to solve more complex problems, but they do not provide clear explanations of why and/or how they reach a decision. These algorithms are considered black-boxes. In these cases, the interpretability is achieved by applying a method that analyzes the model after training it, which is referred to as "*post hoc*" interpretability. In contrast, simpler algorithms provide more understandable predictions although they are weaker in terms of accuracy. In this case, the interpretability is considered as "*intrinsic*", since it is achieved by restricting the complexity of the ML model. Figure 2.15 shows the trade-off between interpretability and accuracy of some well-known algorithms. As it can be observed, linear regression and decision trees are considered weaker but more interpretable models, whereas DNNs are considered more powerful but less interpretable models.

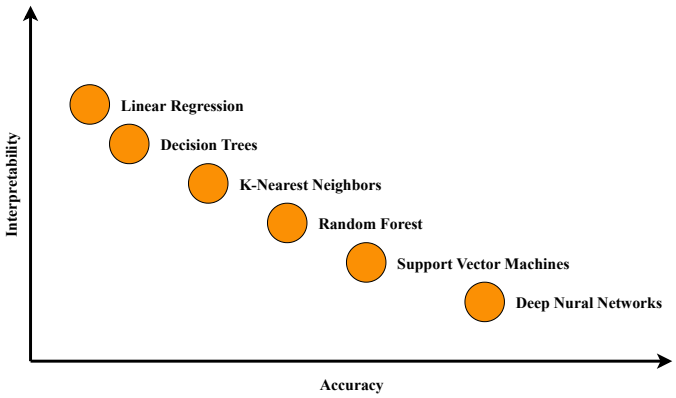


Figure 2.15: Relation between accuracy and interpretability of ML algorithms. Note that this is only a guidance picture and it might not reflect the real scenario precisely.

According to (Molnar 2019), interpretability methods can also be classified based on their results, whether they are model-specific or model-agnostic, and whether the interpretation is global or local.

As the results of these methods are regarded, they can be categorized into five groups:

- **Feature summary statistic:** methods provide statistical results indicating the importance of each input variable or the correlation between pairs of features.

- **Feature summary visualization:** methods are based in the former since they visualize the resulting feature summary. This is due to the fact that in some cases, statistical results are only meaningful if they are visualized.
- **Model internals:** methods consist in interpreting the internal representation of the models such as the weights of linear models and CNNs, or the tree structure of decision trees.
- **Data point:** methods return data points that currently exist or that are created by the method itself to make the model more interpretable.
- **Intrinsically interpretable model:** methods are the most used ones to interpret black-box models, which is achieved by approximating them with an interpretable model. To this end, this model must be interpretable itself (intrinsic) to check its internal parameters or feature summary statistics.

These methods can be model-specific or model-agnostic. The former involves methods which are only valid for a specific type of algorithms such as linear models, decision trees, or neural networks. Intrinsic methods always fall within this category. The latter involves methods that can be used for any ML model regardless of its nature. These methods are usually applied after training the model (post hoc) and interpret them by analyzing the correlation between the input and output variables.

Regarding global and local interpretability methods, they differ in that former methods attempt to interpret the entire model at once, whereas latter methods try to explain a specific prediction (Lipton 2018). To explain the global model output, global methods require the trained model, knowledge of the algorithm, and the training dataset in order to have a global view of all the components of the model. Global methods may help to understand the model's distribution of the target outcome based on the features, thus explaining how the model makes predictions. However, the difficulty of applying these methods increases proportionally with the number of features. This is why some of these methods focus on modular level explanations, that is, they attempt to explain how certain parts of the model affect on the predictions. This is the case of weights in linear models or splits of the trees in decision trees. As local interpretations are regarded, they attempt to answer how the model makes a prediction for a single instance. To this end, these methods assume

that an individual prediction might be simpler than the complex model due to the fact that the prediction may have a simple dependence with respect to some of the features rather than having a complex dependence on them or to the entire model.

As detecting anomalies in the industrial domain is concerned, interpretability methods can be particularly suitable to understand why the model reaches a decision and thus, to explain how anomalies arise. As mentioned, industrial systems are monitored by multiple sensors and detecting an anomaly without knowing its causes might not be sufficient to fix the anomaly. Therefore, interpretability methods might help to diagnose the anomalies by making the outputs of the anomaly detection model understandable for humans.

2.5 Big Data and Cloud Computing

Big Data is a paradigm that is related to managing large volumes of data. Although Big Data has been one of the most trending paradigms of the last two decades, there is not yet any universally accepted definition. Originally, Big Data was defined by leading technological companies as the challenges and opportunities associated with the increase of data *Volume*, *Velocity*, and *Variety* (3V model) (Laney 2001; Meijer 2011; Zikopoulos, Eaton, et al. 2011). In this context, *Volume* refers to massive data volumes that becomes increasingly big; *Velocity* refers to the speed with which the data is generated; *Variety* refers to the existence of several types of data coming from various sources, which might include structured (traditional data), or unstructured (i.e., images, audio, text) data.

However, Big Data is not only about processing large amounts of disparate data in a fast way, but it is also about getting valuable information from it to support decision-making. Therefore, this definition might be more appropriate:

> "*Big Data technologies describe a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling the high-velocity capture, discovery, and/or analysis*" (Gantz and Reinsel 2011).

Consequently, another "*V*" was added to previous ones. This refers to *Value*, which as mentioned, it refers to extracting valuable information from the data.

Furthermore, *Veracity* came to play an important role in Big Data as it refers to the quality or trustworthiness of the data. This is a key aspect to support decision-making. Hence, Big Data can be defined with these five "*V*"s.

On the other hand, the NIST provides a more technological view of what Big Data is:

> "*Big Data shall mean the data of which the data volume, acquisition speed, or data representation limits the capacity of using traditional relational methods to conduct effective analysis or the data which may be effectively processed with important horizontal zoom technologies*" (Chang and Grady 2015).

Nevertheless, these definitions are subjective due to the fact that the terms big and fast are relative. For instance, a given size of data might not be considered as big for a leading company but massive for a small or medium-sized company. Furthermore, these terms also evolve according to technological advances and thus, what is currently considered big and fast might be considered, conversely, small and slow in the short- to medium-term future.

Big Data is closely related to cloud computing as Big Data technologies usually run within the cloud. As mentioned in the previous section, cloud computing refers to a pool of virtualized computers that offer a wide variety of services over the Internet. These services can be categorized into three groups (Rahimi et al. 2014): Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). They are briefly described below:

- **IaaS:** offers computing resources such as servers, networking, storage, and data-center space as a service. Here, only computational resources are offered and thus, the user must install and manage all the required software.

- **PaaS:** provides a cloud-based environment with everything required to support the complete lifecycle of building and delivering web-based (cloud) applications. In this case, an already configured computational environment is provided in addition to computational resources.

- **SaaS:** provides end-users with access to a specific application, meaning that the management of the resources and the computation environment lies in the cloud provider.

The main benefit of using the cloud is that it offers on-demand services so that the users do not have to worry about the management of these services or to their scalability. In this context, scalability refers to the idea of a system in which every application or piece of infrastructure can be expanded to handle increased loads. The cloud is composed of multiple computational nodes (i.e., computers or servers) interconnected with each other, acting as a single computer thus offering a large computational power. Typically, this pool of resources is called a cluster. Therefore, the cloud offers horizontal scaling, which means that more servers can be added to the cluster to spread the load across multiple machines (Furht and Escalante 2010). In this way, services can be scaled by adding more computational resources. Vertical scaling is the opposite of horizontal scaling, in which the scalability is achieved by increasing the capacity of a single machine. However, vertical scaling is limited by the current technology and thus, the required computational resources might exceed the capacity of the most powerful computer available. In this manner, horizontal scaling is a better choice in this context as many machines as required can be added, although this increment also hinders the management of the cluster.

Within cloud computing, fault-tolerance is another key aspect. This refers to the ability of the infrastructure to continue providing service to underlying applications even if a failure occurs on one or more components of the infrastructure (Tanenbaum and Van Steen 2007). As described before, the cloud is composed of multiple interconnected servers and thus, fault-tolerance is achieved by replicating the services throughout several computational nodes so that if some of the nodes fail, the others can still provide the corresponding service. In cloud computing, there are typically active and replication nodes. The former are the ones that are actually providing the services, whereas the latter are waiting (stand by mode) ready to become active when an active node fails.

Big Data frameworks are specially designed to run within the multi-node structure of the cloud as they rely on cloud technology to offer fast, scalable, and fault-tolerant services. Consequently, Big Data provides the design of the frameworks to

run on multi-node structures while the cloud offers computational flexibility and reliability. Therefore, Big Data frameworks can be considered as cloud computing frameworks that adhere to the Big Data paradigm. Nonetheless, we will refer to them as Big Data frameworks.

Due to the described characteristics of Big Data and cloud computing technologies, they have become particularly relevant at the time of monitoring industrial systems within the Industry 4.0. As mentioned, CPSs continuously generate large volumes of streaming data that must be sent to the cloud to be then processed. However, managing these volumes of data entails a number of technological challenges regarded to how to gather, send to the cloud, and process the data efficiently in order to early detect anomalies (Kagerman and Wahlster 2013). In this sense, Big Data and cloud computing might be the key technologies to enable these tasks.

2.6 Industrial Case Study

The contributions of this thesis are validated in a real industrial scenario in which the operating status of a service elevator is monitored. This case study comprises two main problems which are related to the performance of the elevator itself and to how to monitor elevators at large-scale. Therefore, this section describes first the main components of the elevator and the variables used to detect anomalies in it (Section 2.6.1), and then the requirements to monitor multiple elevators to perform a real-time anomaly detection of all of them are detailed (Section 2.6.2).

2.6.1 Elevator Performance

The contributions of this thesis are validated in a real industrial scenario in which the operating status of a service elevator is monitored. Elevators are complex electro-mechanical systems which carry loads vertically. Figure 2.16 shows its components. In this work, the most relevant components correspond to:

1. The cabin that holds people and/or objects for transport and is encased in the elevator shaft.

2. The electric machine that is in charge of generating the power to move the cabin, either up or down.

3. The counterweight that is used to reduce the strain on the motor as it creates constant energy that can be used to propel the elevator car in either directions.

4. The guiding system which is composed of rails that are positioned both in the elevator cabin and on the counterweight to prevent them from swinging, thus creating a smooth movement of the elevator in any direction.

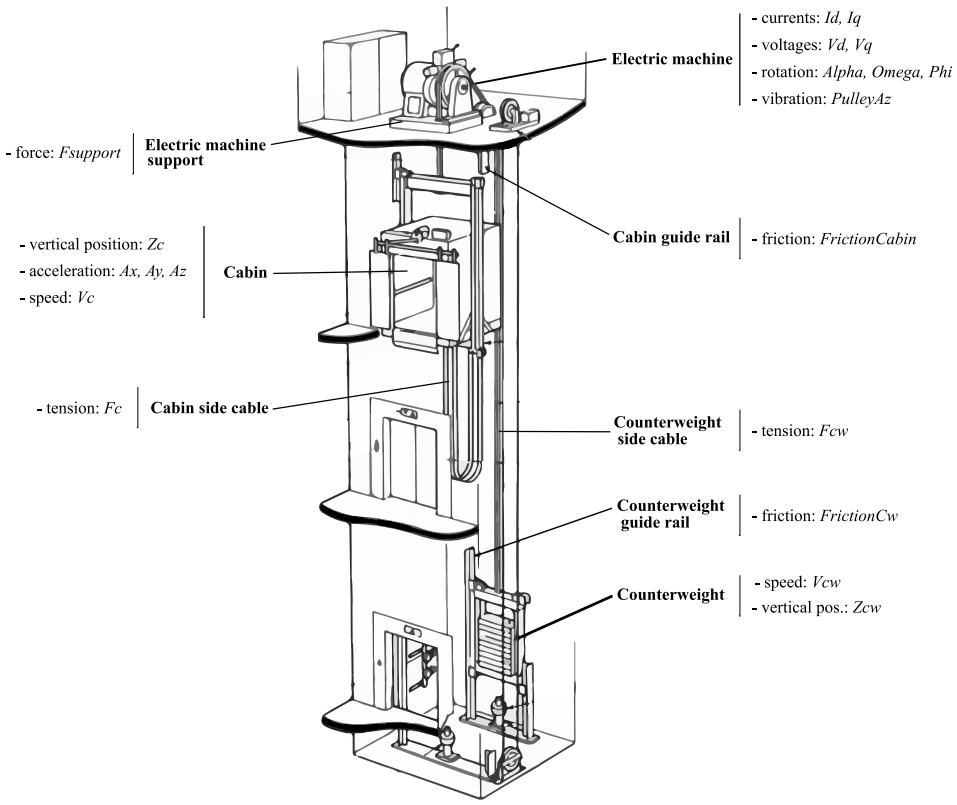


Figure 2.16: Main components of the elevator and location of sensors (check Table 2.2 for sensor description).

Often, the assessment of elevators might be limited due to the large heights, the reduced accessibility, and the reduced number of sensors available in these installations. Thus, it can be hard and time-consuming to gather data from elevators. However, a large amount of data that matches all possible scenarios of normal and anomalous activity is typically required for generating an effective anomaly detection system. Moreover, might be the case that no faults occurred in the real

operating scenario, or that testing some fault conditions is expensive. These facts often hinder the assessment of their different failure modes.

In this context, physics-based modeling is an attractive solution to gain insight into the system's behavior under faulty conditions. Consequently, a physics-based model of the elevator was developed by domain experts, which includes all its relevant components, and where different faults can be introduced as input parameters. With this model, it is possible to simulate normal conditions as well as different fault scenarios and their combination. Namely, faults in the guiding system (reduced lubrication, misalignment), and in the electric machine (de-magnetization, lose of inductance) are studied. These faults are introduced as model parameters that may be changed in each simulation. In addition, other operational parameters such as the cabin's load are also included in the model. Note that each simulation refers to a single elevator journey that goes from floor *a* to floor *b*, either up or down. Table 2.1 describes the main components of the elevator's model, including their degrees of freedom (DoF) and fault parameters. This model is partially based on the models presented in (Gonzalez et al. 2016) and (Esteban et al. 2016).

Although the physical model can be used to simulate every possible condition, each simulation takes an average time of 350 s. The execution time for each simulation depends on the introduced parameters to the model. Considering that thousands of simulations must be executed to simulate all required normal and anomalous conditions, it would be very expensive in terms of time to conduct all the simulations sequentially. Therefore, the CloudFMI Distributed Simulation Framework was designed and implemented. This is a DS framework that allows executing multiple simulations in parallel using cloud computing technology. Consequently, the time required to execute all the simulations was reduced up to 30 times given the available computational resources (see Appendix B for more details). This framework is freely available in `https://github.com/ikerlan2015/CloudFMI-Distributed-Simulation-Framework`.

In this way, 14,000 different normal and anomalous conditions were executed to generate a dataset and validate this work. Each simulation represents an elevator journey of fixed length, both uphill or downhill. Consequently, each simulation is composed of 20 univariate time series corresponding to 20 sensors located all around the elevator that record data at a frequency of 500 Hz, according to the real scenario.

The location of each sensor can be found in Figure 2.16, and a brief description of them is presented in Table 2.2. Each time series has a fixed length of 8,000 data points. All the simulations were labeled by domain experts and they are classified into two types of classes: "*normal*" as the negative, and "*anomaly*" as the positive.

In this use case, an anomaly can be of different nature: point, context-specific, or collective, all of them belonging to the positive class. A point anomaly can be understood as an isolated friction or as an impact in the cabin that is reflected as a peak in a sensor. A context-specific anomaly can occur when the elevator starts a journey. A particular behavior may be normal when the journey is ascending but anomalous when it is descending. A collective anomaly is considered when there is an accumulation of certain factors that lead to a global anomaly. Note that when one of these anomalies is detected, the entire journey is classified as anomalous. As these anomalies are difficult to recognize, they were labeled by domain experts.

2.6.2 Current Maintenance Operations

In the current scenario, the maintenance of elevators is currently performed locally by technicians. This implies that maintenance operators must physically access the elevator every time its status is assessed. The data involving the performance of the elevator is stored in a local hard drive so that operators can connect to it to analyze the status of the elevator. Therefore, anomalies can only be detected at the time of manually accessing historical data or when a user reports a fault. Consequently, maintenance operations are limited to manual corrective and preventive tasks. Moreover, it is worth to point out that a number of travels made by operators to perform maintenance tasks may be unnecessary due to the correct condition of the elevator.

Considering that the company owns thousands of elevators located all over the world, many operators are currently required to provide maintenance support for all of them. Furthermore, elevators have to work 24/7 and thus availability becomes a key issue. In this context, late detection of an anomaly can lead to unplanned down-times and consequently to expensive repair work in terms of availability, user satisfaction, and the cost of the repair itself.

Component	Description	DOF		Operational parameters	
		DOF	Description	Parameter	Description
Cabin	Component carrying the payload	$x,\ y,\ z$	Cabin translation	-	
		α, β, γ	Cabin rotation	-	
Counter-weight	Component balancing the payload	$x,\ y,\ z$	Counter-weight translation	-	
		α, β, γ	Counter-weight rotation	-	
Guiding system	Steel rails constraining the cabin and counterweight's horizontal movement	-		μ	Friction coefficient
		-		δ	Deviations in the guiding system's alignment
Suspension means	Steel ropes which connect cabin and counterweight	-		-	
Driving pulley	Sheave that transmit the traction from the electric machine to the suspension means	φ	Pulley's rotation	-	
Electric machine	Driving machine which moves the system according to the input speed reference.	$i_q,\ i_d$	Machine's quadrature and direct axis currents	λ	Reduction in permanent magnets generated flux
				L_q	Reduction in the electric machine's inductance
Control system	Field Oriented Control consisting of three PI loops, which ensures that the electric machine follows the given speed reference, while maximizing the quadrature current fed to the machine.	x_s, x_q, x_d	Derivatives of the controller's PI loops	-	

Table 2.1: Description of the elevator model's components and fault parameters.

In this way, there is a need for monitoring in real-time the status of all the elevators in a centralized way. This is typically achieved by sending the data generated by the elevators to a remote central server where the data is processed to detect anomalies. This would allow processing in real-time the data corresponding to elevators' performance to detect anomalies as soon as they occur. Therefore, the maintenance operations would be improved by reducing the time between the anomaly and its corresponding repair work, thus increasing the availability. It would

Sensor name	Description
Alpha	Angular acceleration of the pulley
Ax, Ay, Az	Lateral acceleration of cabin on X, Y, and Z axis
Fc, Fcw	Tension on the cabin's and counterweight cable
FrictionCabin, FrictionCw	Cabin and counterweight friction
Fsupport	Force on the support of the machine-pulley
Id, Iq	Direct and quadrature power
Omega	Angular speed of the pulley
Phi	Angular position of the pulley
PulleyAz	Vertical acceleration of the pulley
Vc, Vcw	Cabin and Counterweight speed
Vd, Vq	Direct and quadrature voltage
Zc, Zcw	Cabin and counterweight position

Table 2.2: Name and description of the sensors installed in the elevator. Note that they are grouped by sensor type and that there are a total of 20 sensors.

also be possible to monitor all the elevators from a central maintenance service, and thus to improve the maintenance operation schedule since the operator would only move to the elevators when necessary.

However, this entails a number of challenges. First, the monitoring system must be able to process in real-time the large volumes of data generated by all the elevators. Moreover, it is expected that the number of elevators, as well as the number of sensors installed on them, increases in the future. On the other hand, the volume of data can vary over time as there are time intervals where elevators are used more often. Hence, the monitoring system must be scalable to provide support for peak workloads and future demanding requirements. As the transfer of data to a remote server is concerned, it must be noted that large amounts of data must be sent to the server after each elevator journey, since the data of all the sensors corresponding to the journey is required to detect anomalies in it. However, connectivity might be limited in several scenarios, and the computational power of the elevators are typically reduced, which might lead to high latencies in data

transmission. Consequently, the monitoring system must be fault-tolerant to keep working properly even if network failures arise.

2.7 Execution Environments

This section details the resources used in the experimentations conducted in this thesis. This includes both hardware and software resources. In this thesis, two types of experimentations can be differentiated: the ones involving the implementation of DL architectures to detect and diagnose anomalies, and the ones referring to the real-time data processing by means of Big Data and cloud computing technologies. Hence, both types use different execution environments, which are detailed next.

Experimentations involving DL to detect and diagnose anomalies

All the experiments of this type were executed on a single computer with the following hardware characteristics (Chapers 4 and 5):

- **GPU:** Titan V.
- **Processor:** Intel i7-6850K 3.6 Ghz Box.
- **Motherboard:** ASUS X99-E-10G WS Intel X99 LGA 2011-v3 SSI CEB.
- **RAM:** 32 GB.

Used software details are shown below:

- **Model implementation:** Keras 2.2.0 and Tensorflow 1.6.0.
- **Parallel computing platform:** CUDA 9.2.
- **Operating System:** Ubuntu 16.04.4 LTS.

Experimentations involving Big Data and cloud computing

Regarding the hardware and software resources used in this type of tests, a private cloud hosted in the servers of the company was used. Below, the hardware resources are detailed:

- **Processor:** Intel(R) Xeon(R) CPU E5-2640 v3 2.6 GHz.
- **Cores:** 36.
- **RAM:** 66 GB.

As the software used to manage the resources of the cluster, a distributed operating system such as DC/OS[6] was used.

2.8 Summary

In this section, the main characteristics and challenges of the Industry 4.0 have been presented. Concretely, the process that is carried out since the CPS digitize the industrial processes until the data reaches the cloud by means of the IoT has been detailed. Moreover, the need to detect anomalies over large volumes of heterogeneous streaming data is exposed, providing useful information regarding what time series anomaly detection is, and how DL can help to accomplish this task. This section has also highlighted the importance of the interpretability within the ML field, describing its bases and the impact it can have at the time of diagnosing anomalies using. Finally, the terms Big Data and cloud computing have been presented in addition to explaining how they can facilitate the management of the large volumes of data generated.

This section has also presented the industrial case study used to evaluate the contributions developed in this thesis. The industrial case study involves the three challenges covered in this thesis. First, detecting anomalies based on the data gathered from multiple sensors. Second, the need for identifying the root causes of the anomalies to know when and where the anomaly has occurred. Third, the real-time monitoring of multiple elevators.

Finally, the hardware and software settings used to conduct the experimentations throughout this thesis have been described. In the following chapter, the literature review involving the three contributions of this work are presented.

[6]https://dcos.io/

Research is to see what everybody else has seen, and think what nobody has thought.

Albert Szent-Györgyi

3

State of the Art

This chapter presents a brief literature review regarding the three contributions of this thesis. First, the current state of the multi-time series anomaly detection field is exposed (Section 3.1). Second, the state of the art involving the interpretability of DL based techniques is presented (Section 3.2). Finally, the research and technological trends for monitoring industrial systems are exposed (Section 3.3).

3.1 Time Series Anomaly Detection

Anomaly detection can be approached in multiple ways. The choice of a particular technique heavily depends on the nature of the data and the requirements of the use case under consideration. A different number of approaches can be found in the literature as it can be shown in a survey on anomaly detection that covers different domains (Chandola et al. 2009), or in another that focuses on data mining techniques (Fu 2011).

In the anomaly detection field, as in other ML problem scenarios, the fact that data is labeled or not is a key factor at the time of selecting which technique to use. In this way, data mining techniques for anomaly detection can be categorized into three main groups (Samuelsson 2016): supervised, semi-supervised, and unsupervised.

As time series anomaly detection is regarded, the capability to detect anomalies in complex scenarios is another key factor as there are techniques that only consider univariate time series (Pang et al. 2017). However, in industrial scenarios, machines typically have heterogeneous multi-sensor systems to monitor their performance and therefore, techniques that can consider multiple time series are required. Hence, this section focuses mainly on multivariate anomaly detection techniques. Furthermore, as within the Industry 4.0 large amounts of data are continuously generated and CPSs are prone to modifications regarding sensor configurations, anomaly detection techniques must also support training with large datasets, besides being flexible to adapt to changing sensor configurations. Next, a brief review of unsupervised (Section 3.1.1), semi-supervised (Section 3.1.2), and supervised (Section 3.1.3) methods is presented.

3.1.1 Unsupervised Techniques

Unsupervised techniques are the most extended ones due to the fact that in the industrial domain it is often difficult to obtain labeled data. In this way, a wide variety of techniques can be found in the literature. In this context, the management of multiple time series is typically approached by calculating distances between time series in order to cluster them and then estimate whether an observation is normal or abnormal.

Within clustering methods, several approaches can be found such as the combination of K-means clustering and fuzzy models (Diez-Olivan et al. 2017) or the combination of Hidden Markov Models with Fuzzy C-Means clustering (J. Li et al. 2017), the use of distributed clusters for wireless sensors (Cenedese et al. 2017), or the use of a weighted clustering based on entropy and dynamic time warping (Benkabou et al. 2018). Online clustering methods can also be found as is the case of the use of arbitrary shaped clusters for data streams (Hyde et al. 2017), or the use of a discrete time-sliding window to update continuously the feature space and perform an incremental grid clustering (Dromard et al. 2017). However, calculating distances between time series is expensive in terms of computational resources and time. Moreover, this problem will be increased proportionally as the training dataset increases in volume and dimension. This is why some works can

be found attempting to avoid this problem by using nearest and farthest neighbors approach (Faroughi and Javidan 2018), rather than only using the nearest neighbors since it suffers from high dimensionality. Nevertheless, it still requires reducing the dimensionality by means of a Principal Component Analysis (PCA), such as in (W. Wang et al. 2018).

On the other hand, some other approaches make use of a Gaussian process to extract the most meaningful features, thus reducing the number of distances to compute (Aljawarneh and Vangipuram 2018; Fei and Sun 2019). In addition to clustering methods, ANN based approaches have also arisen recently in an effort to avoid the problem of training with large datasets. This is the case of using a Restricted Boltzmann Machine approach in which the ANN captures the system-wide patterns that lead to an anomaly in an unsupervised manner (Yang et al. 2018).

Although unsupervised techniques are more flexible than the others (Goldstein and Uchida 2016), most of them rely on the assumption that normal instances are far more frequent than abnormal ones (Samuelsson 2016). However, this assumption does not always hold true. Moreover, in case the sensor configuration of the CPS changes, it would imply computing all the distances again for clustering methods. Furthermore, since there are no labels in unsupervised learning, it is near impossible to get a reasonably objective measure of how accurate the algorithm is. This fact might hinder the implementation of unsupervised techniques in the industrial domain since being able to evaluate the performance of the used anomaly detection technique is critical to trust the system.

3.1.2 Semi-Supervised Techniques

Within semi-supervised techniques, one-class methods are the most used ones as they only require having data regarding the normal class. This is a frequent use case within the industrial domain as often no anomalous data is available or there exist too few anomalous observations to train a supervised classification algorithm. There are two main semi-supervised approaches, being both methods related to modeling the normal behavior. The first one refers to delimiting a boundary in the feature space in which all normal observations fall into it. Then, all new observations that fall far from that region will be considered as anomalous.

In this context, one-class SVMs are typically used (W. D. Fisher et al. 2017; M. Hu et al. 2018; Tang et al. 2018). Similarly, isolation forest approaches build multiple decision trees such that the trees isolate the observations in their leaves, and that they assume that anomalous observations will fall into shorter trees (Puggini and McLoone 2018; Stripling et al. 2018). However, modeling a normal region that captures all normal behavior is extremely difficult and the boundary between normal an abnormal is often blurred (Samuelsson 2016). Furthermore, these approaches also suffer from training with large datasets and thus, they require a previous data processing to extract the most relevant features or to reduce the dimensionality. This fact hinders detecting anomalies in multi-time series. This is why some researchers make use of hybrid models in which they combine CNNs and one-class SVMs in an effort to automatically extract features and reduce dimensionality (Erfani et al. 2016).

The second semi-supervised approach consists in modeling the normal behavior of a sequence to create a predictive model and then calculate the anomaly score of an observation as the deviation from the predicted value. Hence, these methods are first trained in a supervised way (regression) and then define an anomaly threshold in a semi-supervised way. In this context, ANN based approaches are the most used ones as is the case of Hierarchical Temporal Memory (Ahmad et al. 2017). Nonetheless, DL algorithms are the state of the art in time series modeling (LeCun et al. 2015), concretely, the LSTM based algorithms. Thus, the LSTM is more frequently used as it can be shown in (Bontemps et al. 2016; Chauhan and Vig 2015; Filonov 2016; Malhotra et al. 2015; Nguyen Thi et al. 2017).

Similarly, Auto-Encoder based approaches can also be found in the literature (Malhotra 2016). These methods attempt to encode the input data into a vector (encoded vector) and then reconstruct it based on that vector. Here, the model is only trained with normal data so that it is assumed that the reconstruction error of normal observations will be low, whereas the error will be high at the time of reconstructing anomalous data (unseen data). However, all of these works require defining an error threshold to determine if the error is large enough to be considered as anomalous. This can be challenging and an incorrect definition of the threshold can lead to a high rate of false positives or false negatives. In fact, some works claim that it is more important the strategy defined to determine an anomaly based

on the prediction error rather than the algorithm used to model the time series itself (Shipmon 2017). Moreover, predicting the value of multiple time series at the same time can be challenging, especially as the number of time series increases.

3.1.3 Supervised Techniques

In the anomaly detection field, it is not common to have well-defined anomalies. Hence, although much research has been done on time series classification, little research has been done on supervised time series anomaly detection. Actually, the difference between time series classification and time series anomaly detection lies into the imbalanced ratio (Saito and Rehmsmeier 2015) of the dataset under consideration. Nevertheless, some approaches can be found such as the use of SVMs (B. Zhang et al. 2015; J. Zheng et al. 2017), ensemble methods (Chattopadhyay et al. 2018), or the use of DNNs (Bao et al. 2018; Cha et al. 2017; Paragliola and Coronato 2018). The downside of these classification-based algorithms is that they suffer from the imbalanced data problem since in the anomaly detection field there is much more data related to normal behavior than to the anomalous. Hence, over-sampling (Abdi and Hashemi 2016) and under-sampling (Triguero et al. 2017) techniques are usually used to balance the number of instances of both classes. However, applying these techniques to time series is challenging, particularly in cases where new time series must be artificially generated (Cao et al. 2013; Moniz et al. 2017). The difficulty increases in the multi-time series domain as the values of a given time series might be affected by the others.

On the other hand, most of the multivariate time series anomaly detection techniques existing in the literature (including unsupervised and semi-supervised) do not perform well if they are directly applied over raw time series. Furthermore, in scenarios such as the industrial in which systems are continuously monitored, time series might be large due to continuous streams of data. Therefore, anomaly detection could be more challenging because of the great amount of data to be processed and analyzed, and due to the little separation that might exist between normal and abnormal observations in the feature space. Overall, most of these techniques require analyzing the time series to find correlation between them (Bankó and Abonyi 2012), cleaning the data, extracting meaningful features (Toshniwal

2009), or reducing the dimensionality (Bianchi et al. 2015; Krawczak and Szkatuła 2014) to convert this data into smart/usable data (Triguero et al. 2018), as shown in (J. Li et al. 2017; Serdio et al. 2014; Theissler 2017). This is often a very time-consuming task and it may require expert knowledge in the domain and about what an anomaly looks like. Typically, all the time series are processed all together which, in the case of heterogeneous multi-time series, it might hinder this process, especially for the feature extraction. This is due to the fact that time series might be of very different nature or might be measured at distinct frequencies. Therefore, extracting the features from these time series all together might result in not capturing properly the most meaningful features.

At this point, DL techniques, concretely CNNs and RNNs, can help to overcome some of these challenges. On the one hand, CNNs achieve state of the art results at automatically extracting the most meaningful features from input data, thus eliminating the need for extracting features manually or using other techniques such as PCA (Cha et al. 2017). However, CNNs do not usually take into account the temporality of the data, which is a critical aspect in time series processing since the chronology of the events might be the key to detect an anomaly. On the other hand, RNNs achieve state of the art results at identifying temporal patterns within sequential data due to the fact that they are able to memorize long and short-term events in order to make an accurate prediction. Although RNNs have demonstrated achieving good results at classifying multi-time series data, they might also require a previous data processing, like some other traditional algorithms (Paragliola and Coronato 2018).

In recent years, the combination of these two DL algorithms has attracted the attention of many researchers as it takes the strengths of both of them, the ability to automatically extracting features of CNNs and the capability of RNNs to identify temporal patterns. This DL architecture is referred to as CNN-RNN since it first applies the CNN to extract features to then find temporal patterns over them by means of the RNN. This architecture has been successfully applied in several domains such as speech recognition (Qian et al. 2016), gesture recognition (Tsironi et al. 2017), weather recognition (Bin Zhao et al. 2018), or emotion detection (Kanjo et al. 2019). Furthermore, it showed promising results at classifying multi-time series working directly over raw sensor data (Ordóñez and Roggen 2016).

Hence, the CNN-RNN architecture would be suitable for detecting anomalies within sensor data. However, there are still a number of challenges to be addressed. First, the performance of this architecture under imbalanced data scenarios is not yet been proven. Second, the feature extraction in heterogeneous multi-time series could require further actions as processing data of different nature is challenging. Third, the architecture must be flexible to adapt to changing sensor configurations without requiring training it from scratch. Finally, the architecture must also be capable of identifying the root causes of the anomaly although the fact that DL models are black-boxes hinders this task.

3.2 Deep Learning Model's Interpretability

It has been demonstrated that DL algorithms achieve state of the art results in time series modeling and that they can yield the supervised time series anomaly detection field. However, DL algorithms are considered black-boxes, meaning that they lack interpretability. In the industrial domain, interpretability is a critical aspect as often explaining the reasons why the model has made a given prediction is equally important (or more) as the prediction itself (Lipton 2018; Miotto et al. 2017; Preece 2018). This is the case of detecting anomalies in industrial multi-sensor systems as detecting an anomaly might not be sufficient since it is also necessary to specify the causes of the anomaly, or which sensor(s) caused the anomaly, in order to help the maintenance operator to solve it. However, researchers have focused their efforts on designing even deeper and more complex DL algorithms to solve challenging real-world problems, leaving interpretability rather aside. In recent years, the implementation of DL approaches in several domains in which the interpretability of the model matters (i.e., health care (Gonzalez-Diaz 2019; Kermany et al. 2018)) has made researchers focus on the interpretability of DL algorithms. Although recently some promising works emerged, this is still a challenging and ongoing research field (S.-C. Chen 2019; Xiao et al. 2018).

To interpret DL algorithms, global methods are not useful as state of the art DL models typically have a large number of hidden layers and neurons, resulting in millions of trainable parameters. Hence, it would be near to impossible to interpret the entire network. Moreover, interpreting the entire model is not useful

at the time of explaining why a given anomaly has occurred due to the fact that a single prediction does not have to follow the same criteria as the majority of the predictions. Consequently, agnostic local methods could be used to explain single predictions. Here, one of the most promising approaches is LIME[7] (Ribeiro et al. 2016), which is a surrogate model that attempts to explain individual predictions by generating synthetic variations of the input data corresponding to the given prediction. Afterward, it trains an interpretable model using the newly generated data, and then uses the proximity of these instances to the current real sample to interpret it. Nevertheless, explaining a single prediction in a large network would also imply understanding millions of weights, meaning that following the exact mapping from input data to the prediction might be difficult. Therefore, DL algorithms require model-specific approaches that can deal with the specific internal learning structure of neural networks (Molnar 2019). As model-specific methods for interpreting DL models, feature visualization (Section 3.2.1), backpropagation-based (Section 3.2.2), and attention-based (Section 3.2.3) methods are the most used ones.

3.2.1 Feature Visualization Methods

Feature visualization methods are the most commonly used (Q.-s. Zhang and Zhu 2018). These methods are mainly applied to CNNs in an effort to explain what the network has learned by visualizing the weights of the filters, which are responsible for capturing the meaningful features of the input data. Since CNNs are widely used for image processing, most of the approaches mainly focus on visualizing features to interpret images (Kermany et al. 2018; Kondo et al. 2017; Nguyen et al. 2016; B. Zhou et al. 2018), being some of them limited to a concrete problem (Gonzalez-Diaz 2019).

As time series are regarded, although it has been demonstrated that the cells of the RNNs can also be interpreted to track temporal dependencies (Karpathy et al. 2015), little research has been made to date in visualizing the features of recurrent units, since some of the approaches working with sequential data use 1D CNNs rather than RNNs (Schirrmeister et al. 2017; Sturm et al. 2016). Nonetheless, visualizing

[7] https://github.com/marcotcr/lime

the features is not always the best option as there might be too many units/filters to visualize, leading to an unaffordable number of elements to be interpreted (Olah et al. 2017). This fact might imply losing interpretability. Furthermore, often the visualized features are not representative or might be blurred. To avoid this issue, some researchers use regularization and optimization methods for improving the interpretability of the visualizations, which additionally seem to improve the performance of the networks (Lin and Runger 2018; C. Wu et al. 2018). However, visualizing the features can lead to a false illusion of knowing what the network is actually doing, since taking a snapshot of one or more layers might not be representative of the entire learning process of the network (Olah et al. 2018).

Consequently, although visualizing features could be beneficial in some specific use cases (i.e., interpreting CNNs for image processing), it is not the best method to interpret RNNs and thus, other methods are required when working with time series.

3.2.2 Backpropagation-Based Methods

As an alternative approach, backpropagation-based methods provide useful mechanisms to interpret the output of neural networks by propagating the explanation backward from the output to input layers (Montavon et al. 2017; Simonyan et al. 2013). These methods are often closely related to feature importance methods (A. Fisher et al. 2018), which rely on measuring the relevance that each input variable has for a given prediction. In this context, positive importance scores will push the explanation in one way, whereas negative scores will behave conversely. Hence, feature importance methods are more friendly for interpreting sequential data as having an insight into which variable is more relevant may be the key to understand the output of the network. This is the case of multi-time series anomaly detection, in which the most relevant input variables (sensors) may indicate where the anomaly has originated. Hereinafter, we will assume that backpropagation-based methods make use of the feature importance approach.

In this way, backpropagation-based methods typically backpropagate the importance of each input variable through the network. Moreover, these methods are often more computationally efficient than other methods. Within these kinds of

approaches, a guided backpropagation method (Springenberg et al. 2014) showed good interpretability results although it suffers when the importance score is negative due to the ReLU layer. Similarly, another method was proposed which used a different importance score measurement called Layerwise Relevance Propagation method (Bach et al. 2015). As an enhanced version of these latter methods, DeepLift[8] (Shrikumar et al. 2017) appeared as one of the most relevant works within backpropagation-based methods. This method consists in back-propagating the contributions of all neurons in the network to every feature of the input to then compute a score of importance by comparing the activations of the network with the reference activations. Although it has been demonstrated that this method can work with images (Jansen et al. 2018), it does not currently support RNNs.

Nevertheless, SHAP (Lundberg and S.-I. Lee 2017) recently emerged as an approach that connects game theory with local explanations, uniting several previous methods (Bach et al. 2015; Datta et al. 2016; Lipovetsky and Conklin 2001; Ribeiro et al. 2016; Saabas 2014; Shrikumar et al. 2017; Štrumbelj and Kononenko 2014) to explain the output of any DL model, including RNNs. Briefly stated, SHAP measures how important each input variable is for each model's output. To this end, it uses the Shapley value (Lipovetsky and Conklin 2001; Štrumbelj and Kononenko 2014) as a measure of importance. As said, this method is based on Lime and DeepLift, among others, grouping the advantages of all of them in a single solution, showing promising results. The Shapley value is a method taken from game theory and thus, the SHAP considers each input variable of a given sample as a player in a game in which the prediction is the payout (Molnar 2019).

One of the benefits of the SHAP is that it offers several interpreting methods in a single solution. It provides a model-agnostic approach although it also provides model-specific methods for decision trees (Lundberg et al. 2018), linear models, gradient-based models, or DL models. These approaches are faster due to the fact that the model-agnostic version does not make assumptions about the model type.

As interpreting approaches for DL algorithms are regarded, SHAP provides two different methods: "*GradientExplainer*", and "*DeepExplainer*". Both use different approximations to compute Shapley values. Hereinafter, we will refer to them as SHAP values. The former combines ideas from Integrated Gradients

[8] https://github.com/kundajelab/deeplift

(Sundararajan et al. 2017), SHAP, and SmoothGrad (Smilkov et al. 2017) into a single expected value equation. This combination allows an entire dataset to be used as the background distribution and allows local smoothing. Thus, if the model is approximated with a linear function between each background data sample and the current input to be explained, and it is assumed that input features are independent, then expected gradients will compute approximated SHAP values. In contrast, the *DeepExplainer* is a high-speed approximation algorithm for computing SHAP values in DL models that is based on DeepLIFT (Shrikumar et al. 2017). However, this implementation uses a distribution of background samples instead of a single reference value. Furthermore, it uses Shapley equations (Štrumbelj and Kononenko 2014) to linearize components such as max, softmax, products. Hereinafter, we will refer to them as explainers.

Figure 3.1 shows an overview of how SHAP works. First, the corresponding model is trained using a training dataset. Next, the explainer is trained taking the previously trained model and a portion of the dataset. Typically, only a reduced fragment of the original dataset is used due to the fact that training the SHAP is computationally expensive and often it is sufficient to obtain accurate results. Training the SHAP with the entire dataset will provide very accurate results although it will also increase the training time since the complexity of this method scales linearly with the number of data points. Finally, at the time of predicting a new observation, it is introduced into both models as an input parameter. Hence, the user's model will output the obtained score, whereas the SHAP model will output a vector containing the importance of each feature.

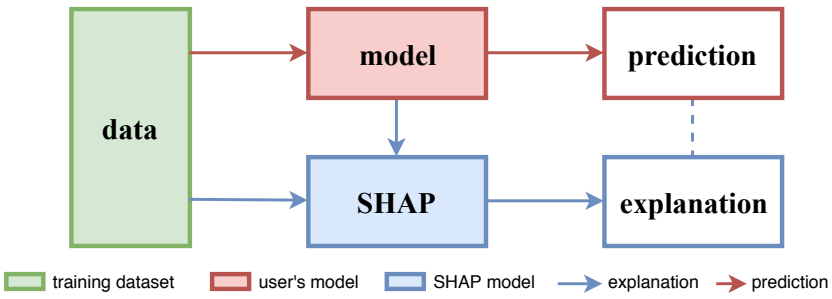


Figure 3.1: General working process of the SHAP.

3.2.3 Attention Mechanism

The attention mechanism was not originally designed for interpreting DL models. In fact, this mechanism was designed with the objective of improving the language translation task by paying attention to the most relevant words of the sentence, while blurring the less important ones (Bahdanau et al. 2014). It was then also used to improve the performance of RNNs in several domains (Cinar et al. 2018; Y. Hu et al. 2018; Karim et al. 2018; H. Zhang et al. 2017) due to the fact that although theoretically LSTMs and GRUs are designed to learn long-term dependencies, in the practice they might also suffer learning such dependencies. This is due to the way in which they manage the information of the hidden states. As mentioned before (Section 2.3), both recurrent networks have a hidden state that flows from the initial to the last time-step with some minor modifications that add, update, or remove information from it. Consequently, the last hidden state is expected to contain the information of all previous time-steps. However, the last hidden state might not be capable of representing all the information provided by all previous hidden states. As far as in sequence classification tasks (*many to one*), the last hidden state is the one used for the final decision-making, this fact can make the RNNs to fail at identifying the most relevant temporal patterns, thus reducing their performance.

To solve this issue, the attention mechanism is used to pay attention to the most important time-steps within the sequential data. This is achieved by creating a context vector that contains information regarding the importance of each time-step. Unlike in traditional RNNs, when the attention layer is used after the last recurrent layer, the hidden states do not only flow through the next time-steps but also flow to the attention layer. This fact can be seen in Figure 3.2. Therefore, the attention mechanism computes the relevance of each hidden state as a probability between 0 and 1, and then it multiplies the weights of each hidden state by its assigned relevance. In this way, time-steps that are more relevant will have more influence in the final decision, whereas time-steps which are less relevant will have little impact on the final decision.

Described mathematically, let's suppose that $X = (x_1, x_2, \ldots, x_t) \in \mathbb{R}^{txd}$ is the input of the RNN, where for each time-step $i \in \{1, \ldots, t\}, x_i \in \mathbb{R}^d$ are feature vectors, and that $H = (h_1, h_2 \ldots, h_t) \in \mathbb{R}^{txs}$ are the hidden states of the RNN.

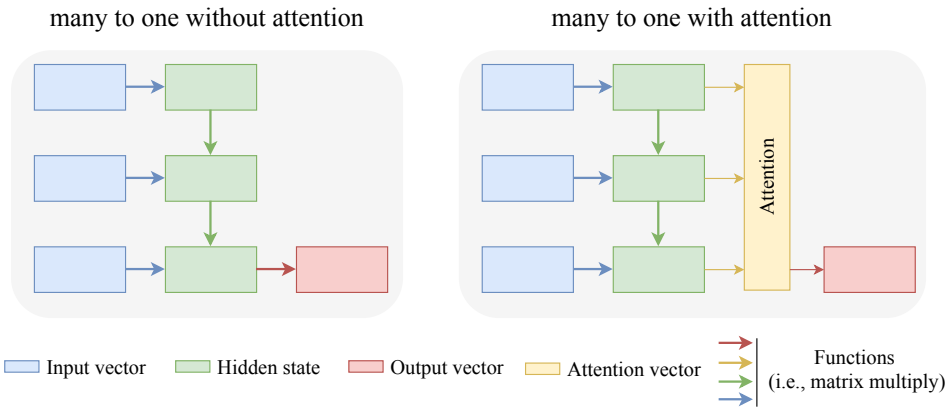


Figure 3.2: Comparison between RNN with and without the attention mechanism.

Therefore, the attention vector (a_t) is computed by Equation 3.1, using Bahdanau's approach (Bahdanau et al. 2014).

$$\alpha_{ts} = softmax(h_t^T \cdot W_\alpha \cdot H) \tag{3.1}$$

$$c_t = \alpha_{ts} \cdot H \tag{3.2}$$

$$a_t = f(c_t, h_t) = \tanh(W_c[c_t; h_t]) \tag{3.3}$$

where $W_\alpha \in \mathbb{R}^{sxs}$, and $W_c \in \mathbb{R}^{sx2s}$ are the weights matrices learned during the training phase. This allows to use the information of all the hidden states instead of using only the last hidden state.

In this way, the attention mechanism has been recently used not only for improving the performance of the network but also for interpreting why it reaches a certain decision (Kaji et al. 2019). To this end, the probabilities within the attention vector indicating the relevance of each time-step can be obtained every time a prediction is made. Hence, it can be determined to which segment of the time series the network focuses most. This fact could be particularly important for diagnosing an anomaly since if the network pays more attention to a particular time-step it could indicate at which point the anomaly occurred. Therefore, the attention mechanism has the potential to point out when the anomaly occurred in a multi-time series problem.

3.3 Large-Scale Monitoring

One of the main goals of the Industry 4.0 is to digitalize the industrial processes in order to reduce down-times in the industrial systems, thus increasing their availability and productivity (P. Zheng et al. 2018). Exploding the data gathered from the CPSs can also improve the maintenance service, besides providing support for the decision-making (Serpanos 2018). In this way, the communication between physical and digital elements has come to play an important role in various industrial domains (Eidson et al. 2012; Leitão et al. 2016; Yue et al. 2015). To this end, the data gathered must go through all the stages of the Industry 4.0 architecture, that is, from CPSs to the cloud in order to process the data and transform it into relevant information for the decision-making (Z. Chen et al. 2017; Dean and Ghemawat 2008; Ly et al. 2015; Wilber 2012).

However, managing the large volumes of data generated by the CPSs entails a number of technological challenges involving all the stages of the architecture:

- First, data must be efficiently gathered and sent to the cloud.
- Second, the cloud must be able to receive all the data sent by a wide range of CPSs without forming a bottleneck.
- Third, the cloud must also be able to process all the received data in real-time in order to early detect anomalies. Any delay during this process might cause critical damage in the industrial systems (Babiceanu and Seker 2016).
- Finally, all the information obtained from the data processing must be accurately filtered and served to the user to support the decision-making.

Hence, a monitoring system that covers all these needs is required. To this extent, these systems should be based on process control algorithms, architectures, and platforms that are scalable, fault-tolerant, modular (plug and play), and which are applicable across several sectors (Colombo et al. 2014). Within the process of monitoring the industrial systems, there are two distinguished phases corresponding to data gathering and sending to the cloud (Section 3.3.1), and to data processing in the cloud (Section 3.3.2).

3.3.1 Gathering and Sending Data to the Cloud

Initially, ubiquitous computing (Sakamura and Koshizuka 2005) was used for monitoring industrial systems, as is the case of a manufacturing architecture that uses a cloud platform to visualize in a dashboard the state of the systems (Ferreira et al. 2013), or the application of ubiquitous methods for manufacturing assembly cells (Kiirikki and Haag 2013) or to support the decision-making (K. C. Lee et al. 2015). However, the communication between the industrial systems and the cloud was limited due to connectivity restrictions, thus hindering the monitoring of the systems. Similarly, ubiquitous computing presents a number of challenges regarding standardization, energy efficiency, network integration, and communications, among others (Babiceanu and Seker 2016).

At this point, the IoT emerged as the key enabling technology to make the communications more efficient (Y. Zhang et al. 2015). Unlike in ubiquitous computing, each IoT device (sensor/actuator) can gather, send, and receive data enabled by communication technologies over wireless and cloud platforms (Cutler 2014). As the IoT facilitates pushing data to the cloud, there was a proliferation of works using this technology to monitor CPSs. In this way, some works exposed the trends and challenges for implementing the IoT within the industrial domain (Lihui Wang et al. 2015), while others highlighted the energy savings resulting from the use of the IoT (Pop and Bessis 2013). As the implementation of IoT is regarded, it has been used to aided the monitoring of robotics applications (Grieco et al. 2014), or to enhance the effectiveness and efficiency of sharing physical assets and services (Qiu et al. 2015). Despite the IoT facilitated data gathering and sending to the cloud, it also signified a large increase in the amount of data sent to the cloud that had to be then processed to extract valuable information.

3.3.2 Data Management in the Cloud

As a consequence of the large volumes of data generated by CPSs, the application of Big Data and cloud computing technologies has attracted the interest of several researchers due to their potential to provide fast, scalable and fault-tolerant data processing (Babiceanu and Seker 2016), which make them suitable for managing all the data coming from the CPSs. Within the cloud, there is a wide range of

tasks/processes that must be carried out to analyze the received data and to perform effective monitoring, which entail several challenges. The literature has identified these challenges and they agree on the need for a monitoring system with the following requirements (Babiceanu and Seker 2016; Battaïa et al. 2018; Serpanos 2018; P. Zheng et al. 2018):

- Cloud-based distributed file systems for ubiquitous access to data.
- Open-source programming frameworks to process and analyze Big Data.
- Large-scale, fast and fault-tolerant data acquisition and processing.
- Real-time data collection from CPSs and storage in the cloud.
- Remote monitoring and control capabilities.
- SaaS, PaaS, IaaS.
- Advanced analytics.
- An intelligent search engine to answer queries.

To address these challenges, Big Data frameworks play an important role as they provide distributed, fast, scalable, and fault-tolerant data acquisition, data processing, and data storage. In fact, these tasks are the most data-intensive besides being the most critical when monitoring the conditions of the CPSs, since any delay during these processes may also delay the decision-making (Faiz and Edirisinghe 2009). Next, the current trends and used Big Data frameworks in terms of data acquisition (Section 3.3.2.1), data processing (Section 3.3.2.2), and data storage (Section 3.3.2.3) are presented.

3.3.2.1 **Data Acquisition**

The data acquisition can be considered as the gateway or the entry point of the data in the cloud. In this context, distributed messaging systems are typically used in which messages are appended in a list as new messages arrive (Magnoni 2015). There are two main types of messaging systems: message queuing and publish-subscribe (pub-sub) systems. Both of them are similar in that there is one

part which sends the data and another which consumes it, although they differ in the way they handle the received messages. The former receives incoming messages asynchronously and ensures that each message for a given topic is delivered to and processed by exactly one consumer (Choi et al. 2017). Conversely, the latter receives incoming messages also asynchronously although, unlike in message queuing, the system ensures that the messages are consumed in the exact order in which they were received. Furthermore, the consumption of messages is not limited to a single consumer, but multiple consumers can consume the messages of a given topic (Happ et al. 2017). Due to this fact, pub-sub messaging systems are more appropriate for the Industry 4.0 domain since multiple agents might require consuming the same data for performing different tasks.

As these messaging systems run in a distributed manner (run on different computational nodes), they provide scalable and fault-tolerant message managing as they are able to recover themselves from unexpected errors, limiting data loss (Magnoni 2015). Moreover, they are able to balance the message traffic between several nodes to spread the load across them and alleviate system overheads (Ghomi et al. 2017). In recent years, there have been several pub-sub messaging systems such as MQTT[9] (Masdani and Darlis 2018), or RabbitMQ.[10] Nevertheless, Apache Kafka[11] is currently the most powerful and scalable engine (Dobbelaere and Esmaili 2017).

Kafka is a distributed messaging system that uses the publisher/subscriber communication pattern (Kreps et al. 2011). Figure 3.3 shows a general overview of how Kafka works. As it can be observed, it uses different topics to manage received data and separate them into groups (i.e., one topic per machine). Internally, each topic contains a queue in which new messages are appended to the old ones. In this manner, producers write messages into the topics while consumers read them in chronological order. Note that Kafka can scale up its throughput by adding partitions to each topic so that more data can be written/read at the same time. Kafka minimizes the loss of messages as it persists all published records, whether they have been consumed. Hence, Kafka meets all the requirements in terms of fast, scalable, and fault-tolerant data acquisition. In addition, Kafka is a mature framework and

[9] http://mqtt.org/
[10] https://www.rabbitmq.com/
[11] https://kafka.apache.org/

has also been adopted in similar use cases (Carcillo et al. 2018; Du et al. 2018; Fernández-Rodríguez et al. 2017; Park and Chi 2016; Yoon et al. 2017; S. Zhao et al. 2015). In this way, Kafka can help to manage all the data coming from the CPSs and thus, it is the data acquisition framework used in this thesis.

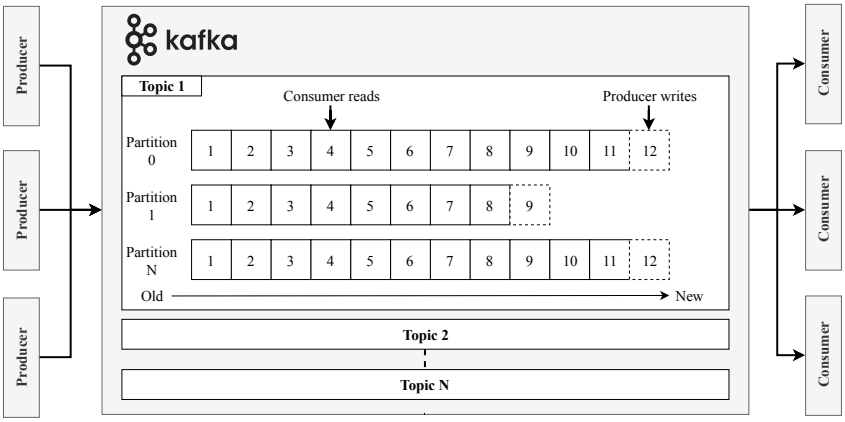


Figure 3.3: General representation of how Apache Kafka works.

3.3.2.2 Data Processing

Processing the data gathered from the industrial machines is a key aspect for assessing their condition in real-time. However, an efficient data processing engine is required to manage the large volumes of data received, besides being scalable to support future demanding workload increments. In addition to Big Data frameworks, the data processing phase must be automated to analyze the data and draw conclusions (J. Lee et al. 2015). This is because humans are unable to draw conclusions in a fast and efficient way when a huge amount of data is involved, and systems managed by humans are hard to maintain, besides being incomplete (Niggemann et al. 2015). Currently, data-driven models (ML models) are typically used to evaluate the condition of the CPSs to detect or predict anomalies. Therefore, there is also a need for engaging ML algorithms to the data processing engine (X. Wu et al. 2013). Consequently, data analysis can be more efficient than analyzing it manually.

In this context, Hadoop MapReduce[12] (Shvachko et al. 2010) was originally the most popular data processing engine and it has been adopted in several domains

[12]https://hadoop.apache.org/

(Boyd 2010; F. Li et al. 2014; Triguero et al. 2015). Hadoop is a distributed batch processing framework that uses the map-reduce data processing paradigm consisting in three phases: map, shuffle, and reduce. The former applies a map function to the local data (within each node) and then writes the output to a temporary storage. Then, the output data is shuffled across the nodes based on a key produced in the previous phase, thus obtaining key/value paired groups of data. Finally, the reduce phase processes the groups of data in parallel. Despite Hadoop was the first framework that provided a fast, scalable, and fault-tolerant data processing, all the operations are limited to these three phases. Hence, migrating the entire logic of the applications of a company might be challenging due to the fact that all the data processing must match the map-reduce paradigm (Kalavri and Vlassov 2013). Furthermore, there was a need for processing streaming data rather than processing data in batch (Shahrivari 2014). Hence, there was a need for a more flexible and faster framework.

In recent years, some promising distributed data processing frameworks such as Apache Spark,[13] Apache Storm,[14] and Apache Flink[15] emerged as alternatives to Hadoop. The main difference between Hadoop and the new frameworks is that the latter ones use in-memory data processing, instead of having to load the data from disk. This fact makes them much faster than Hadoop. Moreover, they are more suited for streaming data. Of these frameworks, Apache Spark is nowadays the most mature and used for processing data in real-time as it can be shown in (Alsheikh et al. 2016; Maillo et al. 2017; Suthakar et al. 2019; D. Zhou et al. 2016).

Apache Spark provides real-time data processing throughout its streaming version Spark Streaming (SS),[16] which is a scalable, high-throughput, fault-tolerant stream processing for live data streams. In general terms, SS receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches. Figure 3.4 shows this process. Internally, it uses a Discretized Stream (DStream) (Zaharia et al. 2012) to represent a continuous stream of data and it is responsible for processing the

[13] https://spark.apache.org/
[14] https://storm.apache.org/
[15] https://flink.apache.org/
[16] https://spark.apache.org/streaming/

incoming data. To process large volumes of data, multiple DStreams can be used in parallel to scale-up the computational capability of the framework and thus, to process more data within each batch. It must be noted that as far as each DStream is an independent process, at least one dedicated CPU is required for each of them. The time interval between batches can vary from very short intervals (micro-batch) to long ones. Furthermore, Spark provides pipelines to connect the data processing engine to ML frameworks such as Tensorflow or Keras to feed processed data into the developed anomaly detection model.

In this way, SS meets all the requirements in terms of fast, scalable, and fault-tolerant data processing. In addition, it is supported by a huge developer community and is powered by companies such as IBM, Hortonworks or Cloudera, which means that the framework is mature and resilient over time.

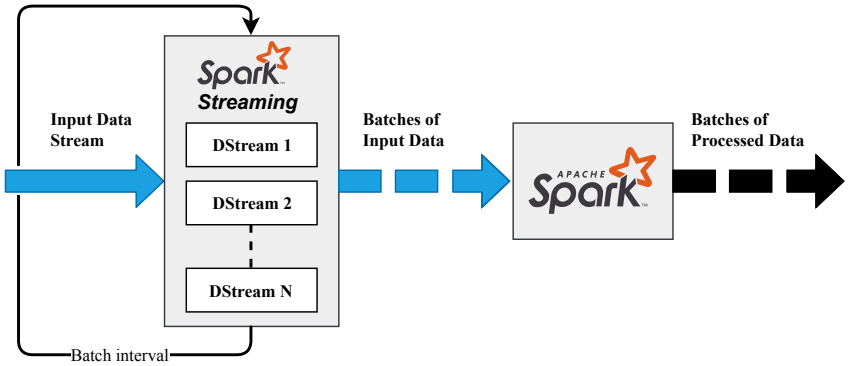


Figure 3.4: General representation of how Apache Spark Streaming works.

3.3.2.3 Data Persistence

Data storage is a key phase in the monitoring of industrial systems to persist the historical data to perform advanced analytics and generate ML models in addition to register all the events occurred during the monitoring. Initially, Relational Database Management Systems (RDBMS) were used to store the data. The strength of these databases lies in the relationship that they maintain between the items of the database and its standardized Structured Query Language (SQL), which is used for querying the database to extract information from one or multiple tables (Chatham 2012). Another key feature of relational databases is that they adhere to ACID properties,

which ensure that database transactions are performed reliably (Mohamed et al. 2014). In this way, RDBMS are efficient at storing and querying data when storing a number of thousands of rows, although they suffer when this number increases to hundreds of thousands or millions of rows (Agrawal et al. 2011).

The factors that make RDBMS efficient (relationship and ACID) also hinder scalability since maintaining the relations makes hard to distribute relational data across several servers and can lead to performance issues both when reading and writing data (Bazar, Iosif, et al. 2014). RDBMS are also CAP theorem compliant, which exposes the fundamental trade-off between consistency (C), availability (A), and partition tolerance (P). Consistency means that all the clients have the same view of the data. Availability refers to the ability of the database to ensure that the clients can always read and write. Partition tolerance enables the database to work adequately despite network and machine failures (Bazar, Iosif, et al. 2014). Partition tolerance is only applicable to databases that run over multiple nodes and thus, RDBMS are CA. In this way, storing and handling large amounts of data is one big obstacle of the current RDBMS (Pereira et al. 2018). Therefore, NoSQL databases emerged because of the need to tackle this issue.

NoSQL databases are designed to run in a distributed manner across several nodes, meaning that they are able to scale horizontally to support future growths of data (Abbasi et al. 2018). Unlike in RDBMS, in NoSQL databases, each record contains a complete set of information without external references, which in addition to the non-adherence to ACID properties facilitate the process of spreading the data across multiple nodes (Bazar, Iosif, et al. 2014). As theoretically it is not possible to achieve the three properties of the CAP theorem, most of NoSQL databases lose on consistency to gain availability and partition tolerance. Furthermore, NoSQL databases adhere to BASE in which an application works all the time (basically available) (Chandra 2015). On the other hand, NoSQL databases have typically more write throughput than RDBMS, meaning that they are faster and more efficient at the time of inserting data into the database (Gessert et al. 2017). This fact is a key issue when monitoring industrial systems as large volumes of streaming data must be stored efficiently not to form bottlenecks.

As a downside, NoSQL databases are not as powerful as RDBMS at the time of querying the database as they do not make use of the SQL. In this way, there

are many NoSQL databases available that meet all these requirements such as Elasticsearch,[17] InfluxDB,[18] or MongoDB.[19]

3.4 Conclusions

This chapter has presented the current trends and technologies in terms of time series anomaly detection, DL algorithms interpretability, and large-scale monitoring of industrial systems. Concretely, it has detailed:

- How the CNN-RNN architecture can improve the multi-time series anomaly detection field.
- How the SHAP and attention mechanism can help to interpret the decisions made by the DL model in order to point out the causes of the anomaly.
- How Big Data and cloud computing technologies can be used for managing the large volumes of data generated by the CPSs.

In the following chapters, the three contributions regarding these challenges will be presented.

[17] https://www.elastic.co/
[18] https://www.influxdata.com/
[19] https://www.mongodb.com/

Real learning, attentive, real learning, deep learning, is playful and frustrating and joyful and discouraging and exciting and sociable and private all the time, which is what makes it great.

Eleanor Duckworth

4

Anomaly Detection in Heterogeneous Multi-Sensor Systems

This chapter presents the proposed Multi-head CNN-RNN architecture for multi-time series anomaly detection. First, the motivation behind this contribution is exposed (Section 4.1). Then, the architecture of the Multi-head CNN-RNN is described (Section 4.2), providing detailed information about both the Multi-head CNN and the RNN sides of the architecture. Furthermore, the benefits of using the proposed architecture to adjust its configuration to each sensor, and to adapt to changing sensors' configuration are described. Afterward, the configuration of the experimentation carried out to validate this architecture (Section 4.3) and the obtained results (Section 4.4) are presented. Finally, some conclusions and future works are exposed (Section 4.5). The content of this chapter is partially published in (Canizo et al. 2019b).

4.1 Motivation

Detecting anomalies in industrial systems is a key issue to avoid unexpected down-times and improve their availability and reliability. As mentioned, industrial systems (CPSs) are equipped with heterogeneous multi-sensor systems to monitor the performance of all their components, which might vary as the CPSs evolve due to the addition, modification, or removal of already installed components/sensors.

Traditionally, dealing with sensor data implies a previous data processing to clean the data, extract meaningful features or reduce the dimensionality to convert this data into smart/usable data (Triguero et al. 2018). Moreover, treating heterogeneous data hinders this task as each sensor data might have its properties, which will require further data processing (Garcia-Ceja et al. 2018). This is because traditional ML algorithms such as RF or SVMs do not typically obtain good results when applied over raw data (Feurer et al. 2015). This data processing usually supposes most of the time when generating an anomaly detection model (or any other ML model), in addition to the fact that it may require expert knowledge in the domain to know how an anomaly looks like. In fact, data pre-processing is the most critical phase as not defining the correct features can lead to a low performance regardless of the used ML algorithm (Ramírez-Gallego et al. 2017).

On the other hand, the variability of the sensor configuration of a CPS makes the developed anomaly detection model to be outdated every time a modification is made. This fact implies, in the best scenario, retraining the model from scratch. Otherwise, it might involve conducting the data pre-processing again since the feature relevance could have also been changed as a consequence. Therefore, a ML model that facilitates adapting to changing scenarios is required.

As stated in Section 3.1.3, DL algorithms achieve state of the art results in multi-time series modeling, where the CNN-RNN architecture has particularly exhibited promising results at classifying multi-time series over raw sensor data. This architecture can automatically extract the most meaningful features from raw data, thus removing the necessity for pre-processing it. However, there are still some challenges to be addressed. First, all sensor data is treated the same way regardless of their properties, which can lead to lower performance. Second, its suitability for detecting anomalies over multi-sensor data is not yet proven and thus, it may suffer

in imbalanced data scenarios as it is a supervised technique. Third, the CNN-RNN architecture is not adaptable to changing sensor configurations since a variation in the configuration might imply training the model from scratch.

Hence, the motivation of this work is to apply and modify the CNN-RNN architecture to address the described challenges, thus adapting this architecture to detect anomalies in changing heterogeneous multi-sensor systems. To this end, we propose a variant of the previous architecture, referred to as the Multi-head CNN-RNN.

4.2 A Multi-Head CNN-RNN Architecture for Multi-Sensor Systems

This section details the proposed Multi-head CNN-RNN architecture. First, the proposed architecture is introduced (Section 4.2.1). Afterward, a detailed explanation of both the CNN (Section 4.2.2) and the RNN (Section 4.2.3) sides of the architecture is provided. Finally, how the proposed architecture can adjust its configuration to each sensor data (Section 4.2.4), and how it adapts to changing sensor configurations are explained (Section 4.2.5).

4.2.1 General Overview of the Multi-Head CNN-RNN

The proposed architecture is a variant of the CNN-RNN architecture that differs in the way it manages the sensor data. Unlike the base CNN-RNN model used in the literature (Liu et al. 2017; Mohammadian Rad et al. 2018; Bendong Zhao et al. 2017), we utilize an independent CNNs to process each sensor data. Throughout this document, we refer to each convolution as a convolutional head, thus forming a Multi-head CNN. Processing each sensor data on independent CNNs entails a number of advantages: 1) the feature extraction is enhanced by focusing only on one particular sensor rather than on all at once, 2) each convolutional head can be adjusted to the specific nature of each sensor data, and 3) it makes the architecture flexible to adapt it to new sensor configurations as the convolutional heads can easily be added, modified or removed.

Figure 4.1 shows the general overview of the Multi-head CNN-RNN architecture. From the left, data coming from sensors are individually processed by independent convolutional heads by means of a window W of length W_L. The window slides over the time series with a step of size W_S. A feature map F_w^n is obtained as a result of applying a CNN to the window w of sensor n. Feature maps corresponding to the same window w are then concatenated. Once all windows of all sensor data are processed, the RNN ($R_{l,\,u}$) of depth l and number of units u yield the classification outcome by processing all the windows chronologically and applying a final dense layer D.

In this way, the Multi-head CNN is responsible for extracting meaningful features from sensor data. In many industrial scenarios, machines have installed multiple heterogeneous sensors that are independent of each other and thus they might not be correlated. Hence, it is reasonable to treat them independently. Another key characteristic of the proposed architecture is that it does process the time series using a sliding window instead of processing the entire sequence at once. Often, multiple behaviors can be represented within a single time series, especially in industrial systems where the action it performs is composed of different phases. In this way, processing the time series in a window-based way makes the network to focus the feature extraction on each phase. Otherwise, the feature extraction would be done according to the entire time series, thus leaving possible key features of each phase uncaptured. Hence, the feature extraction is done window by window for each time series, meaning that it can focus on the different phases existing in a time series. Finally, the features coming from all convolutional heads are processed together window by window by the RNN side to classify the entire event.

The RNN is responsible for finding the hidden temporal patterns from the extracted features. It acts as the classifier of the architecture. To this end, the RNN processes all the extracted features corresponding to each window in chronological order. Finally, it gives a final result according to the temporal behavior that all sensors exhibit throughout a specific event.

This architecture can be implemented with a variety of CNN and RNN layer types. As there is no layer that best suits all scenarios, it is advisable to analyze all alternatives to determine which one performs best for the use case under considera-

tion. In this chapter, all the possibilities to find the architecture that best adapts to our use case are analyzed. See Sections 4.3 and 4.4 for more details.

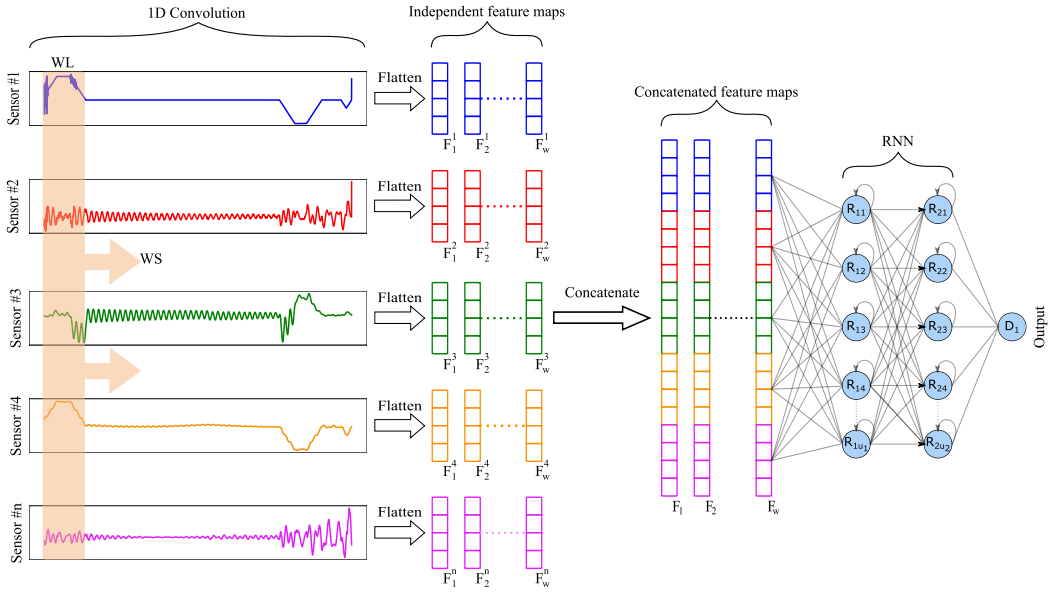


Figure 4.1: General overview of the Multi-head CNN-RNN architecture.

4.2.2 Multi-Head CNN

The Multi-head CNN uses one-dimensional convolutions, where the dimension defines how it processes input data. CNNs are very popular on image processing and they become the state of the art in this field. Since images have two dimensions, two-dimensional convolutions are used. In this way, the kernel moves from left to right and from top to bottom to process the entire image. However, time series can be defined as one-dimensional vectors. Therefore, one-dimensional convolutions are applied to make the kernel move over it in a single dimension, that is, in the time dimension.

To process multiple time series, the Multi-head CNN uses multiple one-dimensional convolutions with a single channel. Traditionally, when dealing with multiple time series, CNNs with multiple channels are used where each channel corresponds to a single time series (Liu et al. 2017; Mohammadian Rad et al. 2018; Bendong Zhao et al. 2017). Hereinafter, we will refer to it as Multi-channel CNN. When a

Multi-channel CNN is used to process multiple time series, a single feature map containing the main features of all the time series is obtained as a result. Although a different set of filters is used for each channel and therefore, the extraction of features is independent for each channel, all of them are mixed together to give a final result. In this way, the specific features of each sensor data might be lost by mixing them all together. In contrast, the Multi-head CNN extracts the features of each time series independently. As a consequence, an independent feature map for each time series is obtained. Conversely, this fact has an impact on the number of parameters of convolutional architectures. The number of parameters of each layer on a traditional Multi-channel CNN is computed using Equation 4.1.

$$p = F_N \cdot K_S \cdot D_{PL} + bias \tag{4.1}$$

where F_N denotes the number of filters, K_S the kernel size, D_{PL} the last dimension of the output vector resulting from the previous layer, and $bias = F_N$. However, for Multi-head CNN layers, the number of parameters must be multiplied by the number of sensors. Hence, it increases linearly according to the number of sensors. As in this architecture the objective on the convolutional network is to extract the main features of each sensor data, processing them on individual convolutions is the only way to preserve their characteristics unaltered.

As described, time series are processed in a window-based way (Ordóñez and Roggen 2016). To divide the time series into smaller segments, all of them are partitioned into the same number of segments. The number of windows is computed as stated in Equation 4.2.

$$W_N = \frac{S_L - W_L}{W_S} + 1 \tag{4.2}$$

where S_L denotes the sequence length or the number of data points within the time series, W_L denotes the window length and W_S denotes the window step, that is, how big is the step to be taken to slide the window over the time series. If $W_S < W_L$, it means that windows overlap with each other. If $W_S = W_L$, it means no overlapping between windows.

Therefore, each convolutional head processes its corresponding time series window by window. As a result, a feature map F_w^n is obtained for each window

and sensor data, where n denotes the sensor number and w the window number. After applying the convolution over all the windows, a sequence of feature maps is obtained for each time series, where each feature map is chronologically sorted within the sequence. As these sequences are independent of each other, they are then concatenated all together. Note that only the feature maps corresponding to the same window number (w) are concatenated with each other. In this way, a sequence of feature maps is obtained where the feature maps of all the time series are concatenated.

Regarding the input data of the Multi-head CNN, each convolutional head requires a four-dimensional input, which is given by Equation 4.3.

$$input\_dim = (n\_samples, W_N, W_L, n\_channels) \tag{4.3}$$

where ***n_samples*** denotes the number of samples within the batch, and ***n_channels*** the number of channels. As the time series are univariate, $n\_channels = 1$.

4.2.3 **RNN**

RNNs are DL architectures with the ability to remember past events. Unlike in traditional neural networks where the information only flows in one direction, in RNNs the data cycles through a loop. Thus, to make a decision an RNN does not only take into consideration the current input but also uses what it has learned before. To this end, the neurons of the recurrent layers, called units, have two inputs instead of one: current and recent past data. As a consequence, an internal memory is formed with which temporary behaviors can be captured throughout a sequence.

In this way, RNNs are suitable for finding temporal patterns throughout the features extracted from each window. The input of the RNN is a three-dimensional vector resulting from the last layer of the Multi-head CNN. The input size is given by Equation 4.4.

$$input\_dim = (n\_samples, W_N, D_{PL}) \tag{4.4}$$

At this point, bear in mind that the feature map of each window is formed by the concatenation of the feature maps resulting from each sensor. In this way, the RNN processes the extracted feature maps in chronological order, that is, from F_1

to F_w. The RNN processes each window to store in its internal memory the relevant information corresponding to the current window. Finally, it makes a decision taking into account what has happened throughout the windows. It is worth to point out that the RNN does not make a decision for each window, instead, it processes the information corresponding to all the windows as a whole to classify the entire event. Therefore, the RNN side of the architecture is not triggered until the Multi-head CNN does not process all the windows. As a consequence, an event is not classified until it is already completed since all sensor data involving the event is required to make the final decision.

4.2.4 Adjusting Convolutional Heads to Each Sensor Data

One of the benefits of the Multi-head CNN-RNN is its capability to adjust each of the convolutional heads to the requirements and needs of each sensor. As mentioned above, heterogeneous sensor systems might measure disparate signals and thus, applying the same convolutional layers to all of them might result in a poor quality feature extraction. It may also be the case that some of the signals of the multi-sensor system are more complex than others. In this case, these sensors might require applying a higher number of filters and/or stacked convolutional layers.

Figure 4.2 shows an example of this. Note that this figure only depicts the convolutional side of the architecture. As it can be observed, this architecture facilitates applying different CNNs to each sensor number without modifying the architecture of the Multi-head CNN-RNN. In this architecture, a convolutional layer within a given convolutional head is defined by $C_{n,\,l}^{K_S,\,F_N}$, where n indicates the sensor number, and l the layer. Recall that K_S refers to the filter size, and F_N to the number of filters. Consequently, the depth of each convolutional head can be modified as well as the used number of filters and their size. It must be noted that the resulting feature map will be affected by the used configuration. In case that only the depth is modified from one convolutional head to another, the output vector size will be equal for all of them. The only difference between them is that some sensor data will experience more transformations due to a higher number of convolutional layers.

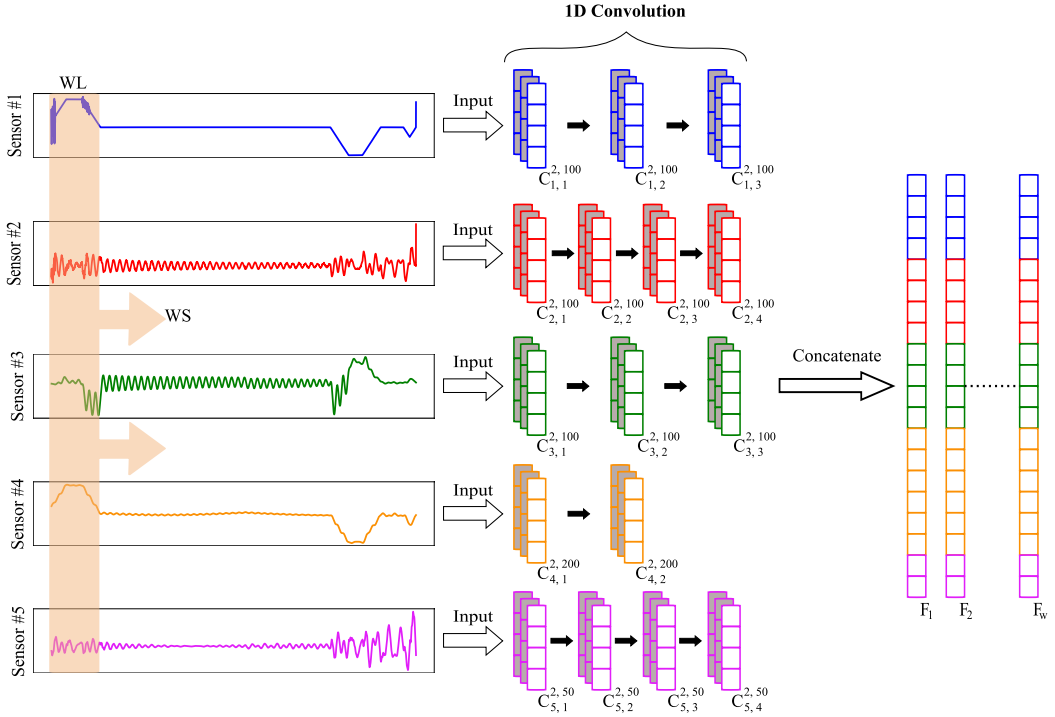


Figure 4.2: Example of different configurations of convolutional heads applied to five sensor data.

In contrast, the size of the output vector will vary if the number of filters, their size, or the stride is modified from one convolutional head to another. A higher number of filters will increase the length of the resulting feature map whereas a lower number will decrease it. On the other hand, larger filters or strides will make the length of the output vector to be decreased, while smaller filters or strides will increase the vector. For instance, Figure 4.2 shows that sensor #4 has a fewer number of convolutional layers than sensor #5. However, sensor #4 has a higher number of filters than the others and sensor #5 has a fewer number of filters than the others. Consequently, the length of the feature map of sensor #4 is larger than the rest of the sensors, whereas the length of the feature map of sensor #5 is smaller than the rest of the sensors.

It is worth to point out that if the length of the feature vector varies from one convolutional head to another, sensors with a larger number of features might have

more influence in the network's decision as they will be represented by more features than the others.

4.2.5 Adapting to Changing Sensor Configurations

On the Industry 4.0, the sensor configuration of industrial machines often varies as new sensors are installed, removed, or modified. Hence, the model has to be adapted each time. In these situations, the previous model becomes old-fashioned and a new model is typically generated from scratch using the current sensor configuration. This is often expensive in terms of time and computational resources. As far as the proposed Multi-head CNN-RNN uses independent convolutional heads to process each sensor data, it facilitates the process of generating a new model suited for the current sensor configuration. This is done by adding, removing, or modifying the corresponding convolutional heads to the old model (the one used until the sensor configuration changed). Therefore, instead of generating a new model from scratch, the knowledge (weights) of the old model can be used to generate the new one by leaving unchanged the parts that are shared by both models and only training the changing part. Consequently, the knowledge of the convolutional heads corresponding to unmodified sensors is transferred directly to the new model. In contrast, the convolutional heads corresponding to added/removed/modified sensors are trained from scratch along with the recurrent side of the architecture. In this manner, less time and computational resources would be required to train the new model.

This is inspired by Transfer Learning (TL) as the underlying idea is to transfer knowledge from one model to another (Weiss et al. 2016). However, this approach differs from TL in that TL typically uses the first layers of the trained network to transfer knowledge from one model to another. Then, the new network is trained by freezing or fine-tuning these layers. This is often used for image classification where a CNN model is trained to classify an image between a given number of classes. Afterward, the knowledge of this model to detect features within images is used to train another model suited to classify images of other classes. Conversely, in this proposal, all the convolutional layers corresponding to unchanged convolutional heads are transferred to the new model, and its weights are then frozen. Therefore,

at the time of training the new model, transferred convolutional heads are never trained. Note that the concatenation layer does not have trainable parameters as it only joins the vectors resulting from the convolutional heads. Hence, no weights are transferred to this layer.

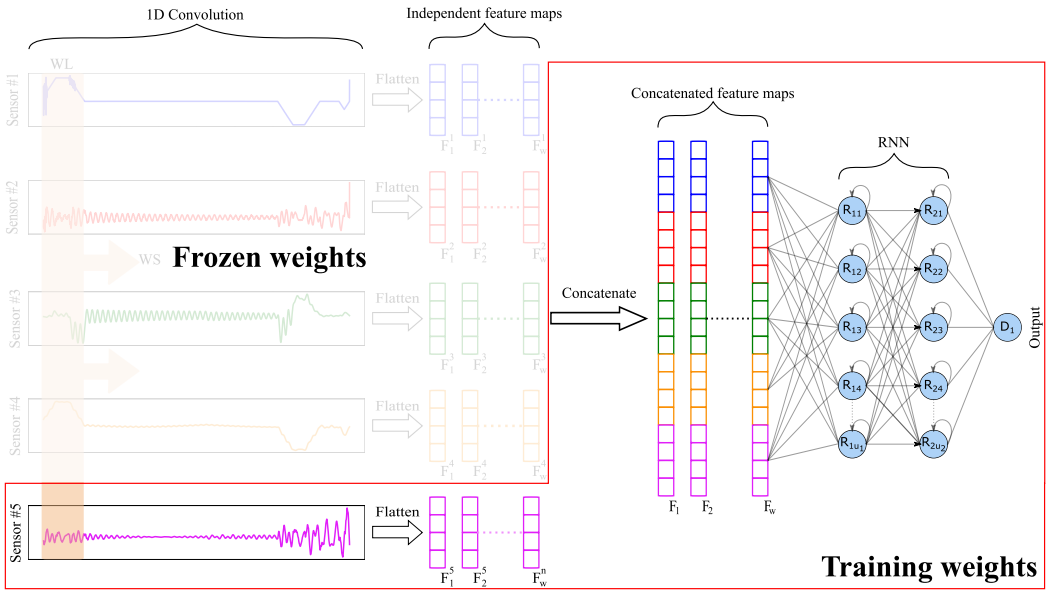


Figure 4.3: Example of training a Multi-head CNN-RNN trained with an additional sensor data. The part of the architecture included within the red box is the part that is trained from scratch. The rest of the architecture remains unaltered.

Figure 4.3 shows an example of how a new model is trained when a new sensor is installed in an industrial machine which was already monitored by four sensors. First, a new architecture is generated by adding a convolutional head to the previous one, meaning that there are now five convolutional heads instead of four. Next, the weights of the four convolutional heads corresponding to the previous sensor data are transferred to the new architecture. The weights of these convolutional heads are frozen as they are already trained. Finally, the new model is trained by training the convolutional head corresponding to the fifth sensor and the recurrent part.

4.3 Experimental Framework

This section describes the configuration and properties related to the experimentation followed in this chapter. In this way, this section presents the measures used to benchmark the performance of the models (Section 4.3.1), the properties of the used validation dataset (Section 4.3.2), the parameters and the base classifiers (Section 4.3.3), and finally, a description of the non-parametric statistical methods used to compare the results obtained (Section 4.3.4).

4.3.1 Performance Measures

In this section, the metrics used to measure the performance of the DL architectures studied in the experimentation are analyzed. As this contribution focuses on the anomaly detection field, datasets are often imbalanced. Thus, selected metrics are widely used in such scenarios to avoid neglecting the minority class (i.e. the anomalies) (Bosman et al. 2017; Hamamoto et al. 2018; T.-Y. Kim and S.-B. Cho 2018).

As standard classification methods, the performance of these classifiers can be measured in terms of precision and recall. Nonetheless, the Precision-Recall Curve (PRC) was used since it is a plot that summarizes the trade-off between the precision and the recall using different probability thresholds. Often, the Receiver Operating Characteristic (ROC) curve is used instead of the PRC (Kanarachos et al. 2017). However, it is demonstrated that the PRC is more informative at the time of evaluating binary classifiers on imbalanced datasets (Saito and Rehmsmeier 2015). The main difference with respect to the ROC curve is that it does not make use of the true negatives as it is only concerned with the correct prediction of the minority class.

In addition, the Average Precision (AP) was used to obtain a specific value with which the classifiers can be compared. The AP summarizes the PRC as the weighted mean of the precision achieved at each threshold, with the increase in recall from the previous threshold used as the weight. It is calculated as in Equation 4.5.

$$AP = \sum_{n} (R_n - R_{n-1}) P_n \tag{4.5}$$

where P_n and R_n are the precision and recall at the n_{th} threshold, respectively. Note that no interpolation was used and thus it is different from calculating the area under the PRC with the trapezoidal rule.

In imbalanced data scenarios, the g_mean (geometric mean) is another extensively used metric (Camps et al. 2018; Hua et al. 2017; Weinberg 2017). It is the geometric mean between precision and recall. It is computed as in Equation 4.6.

$$g\_mean = \sqrt{\frac{TP}{TP + FN} \cdot \frac{TN}{TN + FP}} \tag{4.6}$$

These metrics are calculated from a confusion matrix, which displays the crossing correct and wrong predictions between pairs of categories (classes) (Limin Wang et al. 2017). It represents True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) undertaken by the system.

4.3.2 Validation Dataset

The proposed architecture was validated against the industrial case study described in Section 2.6. As mentioned, the dataset contains a set of 14,000 elevator journeys of fixed length, represented by 20 sensors data of length 8,000. This dataset contains a 20% of anomalous journeys, although four new datasets were generated based on the original one to test the behavior of the architectures under different imbalanced ratios. These new datasets contain 15%, 10%, 5%, and 3% of anomalies, respectively. The selection of anomalies for each dataset was done randomly. Recall that three types of anomalies were considered: point, context-specific, and collective.

Regarding the training of the architectures analyzed in the experimentation, all the datasets were split into train/validation/test sets, with a ratio of 60/20/20%. The considered datasets were partitioned using a Stratified Shuffle Split (SSS) cross-validator (obtained from scikit-learn[20]), which splits the datasets into train/test sets and returns stratified randomized folds. As SSS only splits the dataset into two sets, the datasets were first divided into train/test sets with a ratio of 60/40. Afterward, the test set was divided into two equally sized sets to generate the validation set.

[20]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html

The folds generated by the SSS preserve the percentage of samples for each class. In the experimentation, the SSS was applied 10 times for each model and thus, each of them was trained, validated, and tested with 10 different data distributions.

When a dataset with a reduced amount of anomalies is used, the following method is conducted to randomize the anomalies included on each fold. For each fold, the corresponding percentage of anomalies (15%, 10%, 5%, or 3%) is extracted randomly from the original dataset and the others are removed. Then, the dataset is partitioned into train/validation/test sets. In this way, each fold always contains different anomalies to train, test, and validate the models.

4.3.3 Parameters and Base Classifiers

In this section, the architecture of all the neural networks used in the experiments is described. As mentioned in Section 4.2, the proposed architecture has a 1D convolutional part followed by a recurrent part. Regarding the convolutional side, two types of layers were considered: *Conv1d* (LeCun et al. 2015), and *LC1d* (W. Zhao et al. 2017). The LC1d layer is similar to Conv1d although it uses multiple static kernels instead of using a sliding kernel that moves over the entire vector. Thus, it uses a single kernel for each different patch of the input (W. Zhao et al. 2017). Although this layer is not purely a convolutional layer but behaves similarly to it, the LC1d layer was included within the group of convolutions in this work. In this experimentation, the Conv1d layer was used with both Multi-head and Multi-channel architectures, while the LC1d layer was only used with the Multi-head architecture. Recently, the Multi-channel architecture has been used in several works (Kanjo et al. 2019; Ordóñez and Roggen 2016; Tsironi et al. 2017) and thus it was used as baseline. On the other hand, five types of recurrent layers were examined: *RNN* (LeCun et al. 2015), *LSTM* (Greff et al. 2017), *GRU* (Dey and Salemt 2017), *Bi-LSTM*, and *Bi-GRU*. The last two types of layers refer to the bi-directional version of LSTM and GRU layers (Schuster and Paliwal 1997). These layers process input data in chronological order. However, in bi-directional mode, data is processed in both chronological and reverse order. All possible combinations between convolutional and recurrent layers were analyzed in the experimentation. Thus, there were a total of 15 architectures. Figure 4.4 summarizes the used architectures.

Furthermore, traditional ML algorithms were used for comparison purposes. These algorithms refer to RF, CatBoost, LightGBM, and ensemble methods. Instead of manually configuring (hyper-parameter selection) these classifiers, an automated machine learning (AutoML) library[21] was used. This library is capable of automatically identifying the ML algorithm (from a pre-defined list) that best suits the considered dataset. To this extent, it trains several models with different hyper-parameters and it automatically adjusts them to achieve the best performance. Furthermore, it performs some pre-processing steps to clean the data, extract the most important features, and/or reduce the dimensionality of the dataset. The AutoML will keep testing different models and configurations until a previously defined period of time is reached. For example, if one day is defined, the library will spend an entire day testing several configurations to achieve the best possible performance.

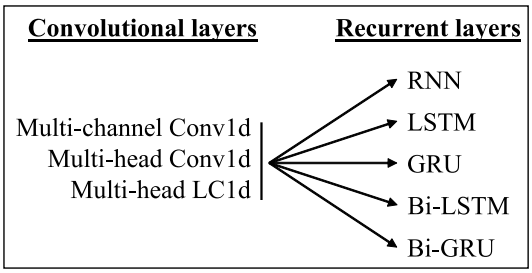


Figure 4.4: Summary of the architectures used in the experimentation. All possible combinations between convolutional and recurrent layers are used, for a total of 15 different architectures.

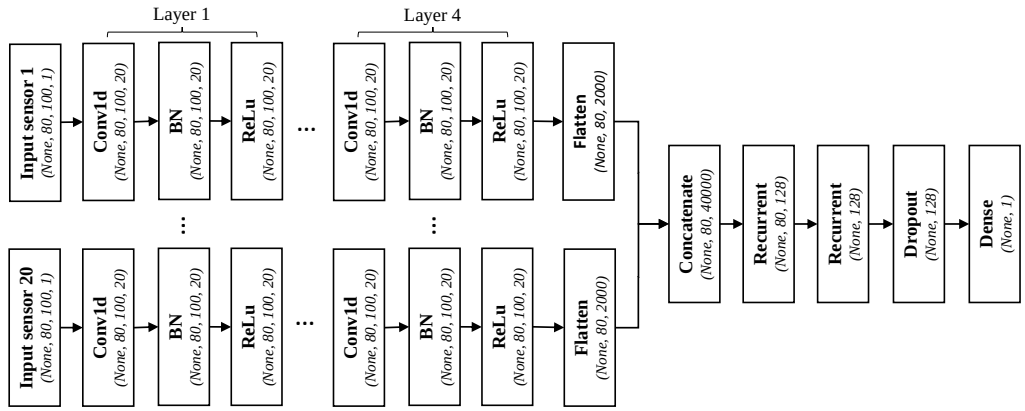
The configuration of the DL architectures was divided into two groups: Multi-head and Multi-channel. The former uses independent single-channel convolutional heads to process each time series separately (Figure 4.5a). The latter uses a single convolutional head with multiple channels to process all the time series (Figure 4.5b). The other difference between both is that on Multi-head architectures, the output of all the convolutional heads is concatenated before reaching the recurrent part. Despite these differences, the layer configuration of both types of architectures remains the same. Note that the Multi-channel architecture served of comparison approach.

[21] https://github.com/mljar/mljar-supervised

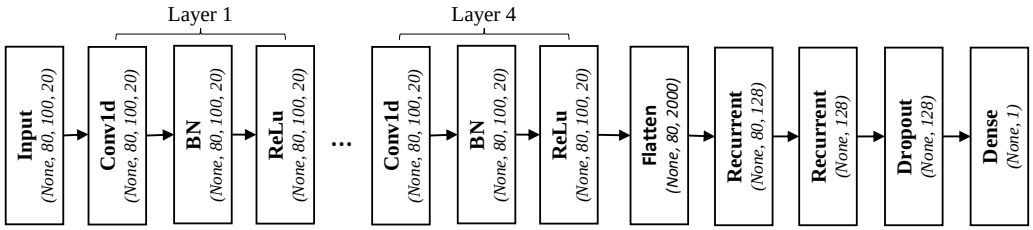
The convolutional part of the architectures is composed of four stacked one-dimensional convolutional layers. Each convolution applies a *Batch Normalization (BN)* layer (Ioffe and Szegedy 2015) followed by a *ReLU* (Krizhevsky et al. 2012) activation layer. The inclusion of the BN layer is particularly important as it reduces the internal covariance shift. This brings a regularization effect between batches and makes training faster (K. Zhang et al. 2017). Unlike other CNN's for time series classification (Ignatov 2018; Pourbabaee et al. 2017; Bendong Zhao et al. 2017), no sub-sampling layer is used after the convolution. In our window-based approach, it is not required as the dimensionality of the input data is already constrained by the windows.

After the convolutions, two stacked recurrent layers are used with ReLU as the activation function. Next, a dropout layer is used to regularize the activations to avoid overfitting. Finally, a dense layer is applied to generate the output of the architecture. For this layer, a sigmoidal activation function is used whose output is a value between 0 and 1. This value refers to the probability that the output is 0 or 1, thus it is then rounded to obtain a binary result.

The training process of these architectures was conducted in a fully-supervised way using back-propagation to adjust gradients from the final dense layer to the initial convolutional layers. As architectures were designed for a binary classification task, Binary cross-entropy (Golik et al. 2013) loss function was used. Regarding the optimizer, Adam (Kingma and Ba 2014) was used since it obtained the most stable results throughout all the tests. To find the best values for hyper-parameters, a grid search was used. Table 4.1 summarizes the parameter specifications used to train the architectures. Note that the grid search was performed for this specific use case. Therefore, this parameter configuration might not be suitable for another use case. In such a case, another grid search would have to be done. To make the training more efficient, batches of 50 instances were used and gradients were computed after each batch. As architectures containing LC1d layers require more computational resources, batches of 10 instances were used to avoid memory problems at the time of training. Furthermore, the maximum number of epochs was set to 30. However, an early stop method was used to stop the training process in case it converges before reaching the defined maximum number of epochs. In this way, the training



(a) Multi-head



(b) Multi-channel

Figure 4.5: Layer configuration of all the architectures. Each box contains the name of the layer and its corresponding output shape. Note that, for input layers, the last dimension of the output shape refers to the number of channels (sensors), while for convolutional layers it refers to the number of filters.

set was used to train the architectures while the validation set was used to tune them. Finally, the test set was used to check their performance against unseen data.

4.3.4 Statistical Test for Performance Comparison

In order to provide statistical support to the results obtained in the experimentation, a hypothesis-testing technique was used (García et al. 2009). In particular, a non-parametric test was used since parametric tests might lose credibility because the initial conditions guaranteeing their reliability may not be satisfied (i.e., independence, normality, and homoscedasticity) (Sheskin 2007).

Parameter	Value
conv. filters	20
recurrent units	128
dropout	0.25
window length	100 (It varies on Section 4.4.4)
window step	100 (No overlapping windows)
learning rate	0.00001
epochs	30 (with early stop)
batch size	50 (10 for LC1d architectures)

Table 4.1: Parameter specification for all the architectures employed in the experimentation.

In this work, these tests were used in cases where multiple datasets were used (Section 4.4.3). For these cases, a Friedman test (García et al. 2010; García et al. 2009) was used to analyze whether there were statistical differences between them. Furthermore, Holm post hoc test (García et al. 2010) was used to identify which of the architectures were distinctive among a 1 x n comparison. In this way, a given hypothesis could be rejected at a specified level of significance of α ($\alpha = 0.05$ was used). Hence, the adjusted p-value (APV) was computed for each comparison which denotes the lowest level of significance of a hypothesis to be rejected. In addition, the architectures were classified employing a ranking that determines how good each of them are in comparison to the others. The positions of the ranking were assigned by computing the average performance obtained by each architecture in all the tested datasets. The architecture obtaining the best APV have the best ranking.

These tests are widely used in the field of ML as can be shown in (Demšar 2006; García et al. 2010; García et al. 2009; García and Herrera 2008), where they recommend their use. A more detailed explanation can be found in (Santafe et al. 2015).

4.4 Results and Discussion

In this section, the results of the experimentation are analyzed and discussed. First, a comparison between using a Multi-head or a Multi-channel convolution

architecture is performed (Section 4.4.1). Second, the difference between using a Conv1d or a LC1d as convolutional layers is analyzed (Section 4.4.2). Third, the performance of the architectures as the percentage of anomalies within the dataset decreases is analyzed (Section 4.4.3). Fourth, the impact that the window length has on the performance of the architectures is evaluated (Section 4.4.4). Fifth, the performance of the Multi-head architectures to transfer knowledge from one model to another to adequate them to changing scenarios is tested (Section 4.4.5). Note that only Multi-head Conv1d like architectures were considered in the last two experiments. Finally, the Multi-head CNN-RNN architecture is compared with traditional ML algorithms (Section 4.4.6).

4.4.1 Multi-Head vs Multi-Channel CNN-RNN

In this section, a comparison between our proposal (Multi-head CNN-RNN) and the Multi-channel CNN-RNN architecture existing in the literature (i.e., (Kanjo et al. 2019; Ordóñez and Roggen 2016; Tsironi et al. 2017)) is performed. The aim of the convolutional part of the proposed architecture is to extract the most relevant characteristics from sensor data. Thus, the performance of the entire architecture is analyzed based on how the features of sensors data are extracted, independently or as a group. Furthermore, the impact that the different recurrent layers have in these architectures is discussed. Note that in this experimentation, only the Conv1d layer was considered as the convolutional layer.

One of the main differences between both architectures lies on the number of features extracted from each window. Table 4.2 details the shape of the output vector regarding each layer and architecture. It also shows the total number of features extracted from each window, which refers to the last dimension of the output vector of the last layer. As shown, the number of extracted features per window is much higher for the Multi-head convolution. In fact, it increases linearly according to the number of sensors. Thus, it contains more features concerning each sensor. Moreover, the features of each sensor are ordered by sensor while in Multi-channel convolutions they are not.

Table 4.3 shows the comparison of their results, where the best scores for each recurrent layer are highlighted in bold. As it can be observed, there is a

significant difference between Multi-head and Multi-channel architectures in terms of g_mean. Figures 4.6a and 4.6b show the PRC of both Multi-head and Multi-channel architectures. The results demonstrate that Multi-head architectures are able to detect anomalies significantly better as they obtain higher precision and recall scores. Furthermore, the variance on the results from one iteration to another is higher for Multi-channel architectures. As recurrent layers are regarded, the Multi-head Conv1d-LSTM obtained the best result. However, the LSTM does not show a good result for the multi-channel architecture. It can also be shown that the difference between both convolutional architectures is larger in case that they are combined with the RNN layer. In fact, RNN achieved the worse results in both cases in comparison to the other recurrent layers. For Multi-head architectures, there is no significant difference between the other recurrent layers, while for Multi-channel architectures, the difference between recurrent layers increases slightly, with bi-directional layers achieving the best performance.

Considering the number of features used by both architectures, one could say that the main reason of Multi-head architectures obtaining better results in comparison to Multi-channel architectures, is due to the fact that the former has more information to make a decision. Consequently, another experiment was conducted in which the same number of extracted features as Multi-head architectures have was set to the Multi-channel architectures. To do so, $F_N = 400$ was set. The results obtained show an improvement of at most 2% for all the recurrent layers. Moreover, the training time increased by 20 as a consequence. Therefore, the main reason lies in the fact of extracting the features of each sensor independently rather than in extracting a larger number of features.

Thus, it can be observed how the proposed architecture outperforms the Multi-channel CNN-RNN existing in the literature.

4.4.2 Multi-Head Conv1D vs Multi-Head LC1d

In this section, Conv1d and LC1d layers are compared, both being Multi-head. The idea behind this study is to know how the different way in which these layers extract characteristics from input data impact on the performance of the entire architecture.

Layer type	Multi-head	Multi-Channel
Input	20 x (n_samples, 80, 100, 1)	(n_samples, 80, 100, 20)
Conv1d (1-4)	20 x (n_samples, 80, 100, 20)	(n_samples, 80, 100, 20)
Flatten	20 x (n_samples, 80, 2000)	(n_samples, 80, 2000)
Concatenate	(n_samples, 80, 40000)	-
Total features	40,000	2,000

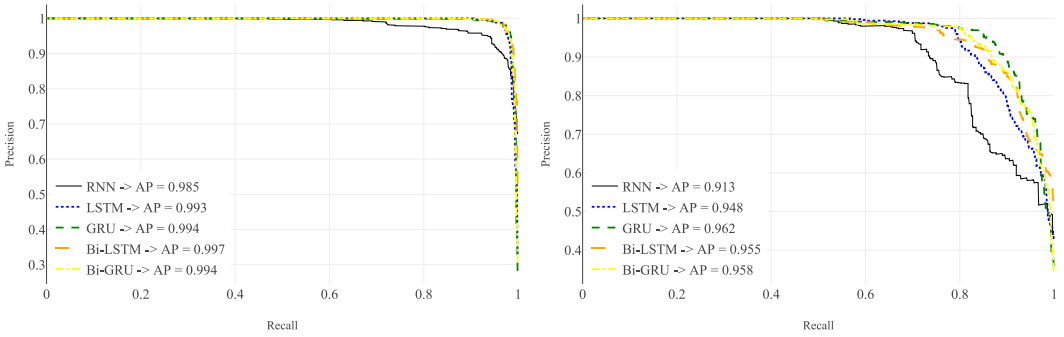
Table 4.2: Number of features extracted for each window by Multi-head and Multi-channel architectures. The shape of the output vector of each layer is detailed. For input layers, the dimensions refer to $(n\_samples, W_N, W_L, n\_channels)$. For Conv1d layers, they refer to $(n\_samples, W_N, W_L, F_N)$. Recall that $W_N = 80$, $W_L = 100$, and $F_N = 20$.

	Multi-head Conv1d	Multi-channel Conv1d
RNN	$\mathbf{0.961 \pm 0.021}$	0.838 ± 0.017
LSTM	$\mathbf{0.980 \pm 0.002}$	0.891 ± 0.023
GRU	$\mathbf{0.977 \pm 0.002}$	0.914 ± 0.020
Bi-LSTM	$\mathbf{0.978 \pm 0.004}$	0.924 ± 0.025
Bi-GRU	$\mathbf{0.977 \pm 0.004}$	0.929 ± 0.023

Table 4.3: Results of Multi-head and Multi-channel Conv1d architectures using the original dataset and g_mean as evaluation metric. The corresponding standard deviation is attached to each metric. The best values for each architecture are highlighted in bold.

In the industrial case study presented in Section 2.6, data comes from an elevator that has different phases during its journey. Here, the more critical phases of the journey are when the elevator initializes and ends the journey. These phases are specific of accelerating and braking actions. Conv1 layers use the same filters to extract characteristics from each window, while LC1d layers use a unique filter for each of the patches within a window. Thus, filters in LC1d layers do not share weights with each other and they only focus on their corresponding patch of the window.

Table 4.4 shows the results obtained for all the architectures. The best scores for each recurrent layer are highlighted in bold. Figures 4.6a and 4.7a depict the PRC of both Multi-head Conv1d and Multi-head LC1d architectures, respectively. Except



(a) Multi-head Conv1d **(b)** Multi-channel Conv1d

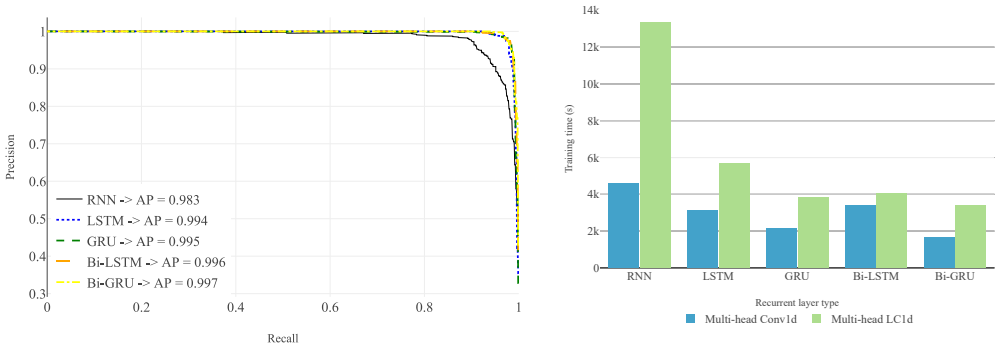
Figure 4.6: PRC of the different architectures using Multi-head and Multi-channel convolutions. Experiments conducted with the original dataset.

for the RNN, which obtained worse results for the LC1d architecture, no significant differences can be outlined between using Conv1d or LC1d layers. However, as observed in Figure 4.7b, Conv1d layer is, generally, two times faster at training time. In fact, Conv1d layer is more than three times faster than LC1D layer in case of combining them with the RNN layer. On the other hand, the time difference between both convolutional layers is reduced in case of combining them with Bi-LSTM.

Considering all this, we conclude that for this experiment, LC1d layer does not significantly improve the feature extraction by applying independent filters to each patch of the window. In addition, it is considerably slower than Conv1d layer.

	Multi-head Conv1d	Multi-head LC1d
RNN	$\mathbf{0.961 \pm 0.021}$	0.922 ± 0.015
LSTM	$\mathbf{0.980 \pm 0.002}$	0.979 ± 0.003
GRU	0.977 ± 0.002	$\mathbf{0.980 \pm 0.004}$
Bi-LSTM	$\mathbf{0.978 \pm 0.004}$	0.978 ± 0.005
Bi-GRU	0.977 ± 0.004	$\mathbf{0.977 \pm 0.003}$

Table 4.4: Results of Multi-head Conv1d and LC1d architectures using the original dataset and g_mean as evaluation metric. The corresponding standard deviation is attached to each metric. The best values for each architecture are highlighted in bold.



(a) PRC of Multi-head LC1d **(b)** Conv1d vs. LC1d training time

Figure 4.7: Results of Multi-head LC1d architectures. Experiments conducted with the original dataset.

4.4.3 Performance Analysis with Reduction of Anomalies

In many use cases, the percentage of anomalies is often lower than in the original dataset (20%). Thus, an experiment was conducted in which the percentage of anomalies was decreased to 15%, 10%, 5%, and 3%. In this study, all architectures of previous sections are included (Section 4.4.1 and 4.4.2). Hence, this experiment analyzes the performance of all architectures as the ratio of anomalies decreases.

Table 4.5 shows the results obtained. The results of the previous sections (20%) are also included in the table as a summary. The best AP and g_mean values for each dataset are highlighted in bold. As shown, the performance of the architectures decreases as the percentage of anomalies is reduced, although their performance maintains high for almost all the architectures. Thus, it is demonstrated that the proposed Multi-head CNN-RNN can also perform well in imbalanced data scenarios without any pre-processing. Focusing on each architecture, the reader can observe that Multi-channel convolutions obtained a poor performance in comparison to Multi-head convolutions since they obtained worse results in all the metrics and scenarios. Regarding both Multi-head Conv1d and LC1d architectures, they obtained similar results although the latter achieved slightly better scores in most of the cases. As recurrent layers are concerned, architectures that include the RNN layer got the worse results in almost all the scenarios.

To objectively analyze these results, Table 4.6 shows the average ranking calculated for all the architectures according to the g_mean and its corresponding APV, which was calculated by means of the Holm's test. Note that the results obtained for all the datasets were used to compute this test (datasets with 20%, 15%, 10%, 5%, and 3% of anomalies). It can be seen that the Multi-head LC1d-LSTM achieved the lowest value in the ranking and thus it is classified as the best architecture. It is worth pointing out that if a standard level of significance of $\alpha = 0.05$ is set, the null hypothesis of equality is rejected for all the Multi-channel architectures as they obtained lower APV. This fact supports the conclusion that Multi-head architectures outperform traditional Multi-channel architectures. Regarding Multi-head architectures, there is no significant difference between them. However, Multi-head architectures including the RNN layer obtained poor results compared to other Multi-head architectures. Therefore, its use is not recommended.

Table 4.6 does not show the differences between Multi-head architectures due to the fact that the statistical test computes the APV proportionally to the results obtained by all the architectures, being $APV = 1.0$ the value that indicates the highest level of equality. As Multi-channel architectures obtained poor results in comparison to Multi-head architectures, almost all of them obtained the highest score. Hence, another statistical test was performed including only Multi-head architectures to analyze their differences. Table 4.7 shows the results. It can be observed that the LC1d layer exhibits better results than Conv1d layer for almost all recurrent layers. The results also show that the architectures including the RNN layer obtained poor performance and thus they were rejected.

4.4.4 Analysis of the Window Length

On the proposed architecture, the window length is one of the parameters to take into account. For this reason, in this experiment, the performance of the architecture as the window length varies is analyzed. The results are measured in terms of network size (number of trainable parameters), training time and performance (metrics). The used range of values for the window length were as follows: $W_L = \{25, 50, 80, 100, 125, 160, 200\}$.

	% of anomalies		20%	15%	10%	5%	3%
Multi-head Conv1d	RNN	AP	0.985 ± 0.005	0.981 ± 0.004	0.970 ± 0.011	0.932 ± 0.011	0.912 ± 0.014
		g_mean	0.961 ± 0.021	0.957 ± 0.023	0.930 ± 0.029	0.881 ± 0.034	0.825 ± 0.016
	LSTM	AP	0.993 ± 0.002	0.992 ± 0.002	0.975 ± 0.004	0.941 ± 0.011	0.903 ± 0.023
		g_mean	$\mathbf{0.980 \pm 0.002}$	0.974 ± 0.004	0.949 ± 0.004	0.909 ± 0.021	0.830 ± 0.016
	GRU	AP	0.994 ± 0.001	$\mathit{0.994 \pm 0.001}$	0.987 ± 0.003	0.965 ± 0.006	0.929 ± 0.014
		g_mean	$\mathit{0.977 \pm 0.002}$	0.974 ± 0.003	0.964 ± 0.002	0.921 ± 0.013	0.857 ± 0.028
	Bi-LSTM	AP	0.997 ± 0.001	0.991 ± 0.002	0.983 ± 0.007	0.959 ± 0.007	0.929 ± 0.019
		g_mean	0.978 ± 0.004	0.971 ± 0.004	0.956 ± 0.028	0.927 ± 0.013	0.877 ± 0.030
	Bi-GRU	AP	0.994 ± 0.001	$\mathit{0.994 \pm 0.001}$	0.986 ± 0.002	0.968 ± 0.002	0.916 ± 0.006
		g_mean	0.977 ± 0.004	0.972 ± 0.003	0.958 ± 0.006	0.922 ± 0.008	0.870 ± 0.015
Multi-head LC1d	RNN	AP	0.984 ± 0.002	0.976 ± 0.002	0.964 ± 0.005	0.910 ± 0.016	0.840 ± 0.014
		g_mean	0.922 ± 0.015	0.907 ± 0.026	0.850 ± 0.022	0.829 ± 0.038	0.763 ± 0.054
	LSTM	AP	0.996 ± 0.001	0.992 ± 0.001	0.985 ± 0.003	0.974 ± 0.011	0.929 ± 0.016
		g_mean	0.979 ± 0.003	$\mathbf{0.977 \pm 0.002}$	0.959 ± 0.002	0.936 ± 0.012	$\mathbf{0.899 \pm 0.040}$
	GRU	AP	0.993 ± 0.001	0.992 ± 0.002	$\mathit{0.989 \pm 0.002}$	$\mathit{0.974 \pm 0.009}$	0.940 ± 0.019
		g_mean	0.980 ± 0.004	0.968 ± 0.004	$\mathbf{0.966 \pm 0.004}$	0.927 ± 0.011	0.897 ± 0.022
	Bi-LSTM	AP	0.992 ± 0.002	0.993 ± 0.001	0.986 ± 0.004	0.962 ± 0.009	$\mathit{0.946 \pm 0.013}$
		g_mean	0.978 ± 0.005	0.973 ± 0.002	0.960 ± 0.008	$\mathbf{0.941 \pm 0.016}$	0.882 ± 0.041
	Bi-GRU	AP	0.995 ± 0.001	0.993 ± 0.000	0.988 ± 0.003	0.964 ± 0.005	0.940 ± 0.009
		g_mean	0.977 ± 0.003	0.970 ± 0.004	0.955 ± 0.003	0.925 ± 0.010	0.874 ± 0.037
Multi-channel Conv1d	RNN	AP	0.913 ± 0.010	0.896 ± 0.008	0.859 ± 0.006	0.796 ± 0.015	0.657 ± 0.018
		g_mean	0.838 ± 0.017	0.837 ± 0.017	0.814 ± 0.016	0.801 ± 0.012	0.706 ± 0.025
	LSTM	AP	0.948 ± 0.010	0.946 ± 0.013	0.895 ± 0.012	0.812 ± 0.010	0.670 ± 0.015
		g_mean	0.891 ± 0.023	0.894 ± 0.011	0.829 ± 0.016	0.802 ± 0.029	0.663 ± 0.034
	GRU	AP	0.962 ± 0.013	0.955 ± 0.012	0.907 ± 0.028	0.822 ± 0.007	0.682 ± 0.036
		g_mean	0.914 ± 0.020	0.900 ± 0.031	0.867 ± 0.030	0.807 ± 0.018	0.700 ± 0.018
	Bi-LSTM	AP	0.955 ± 0.008	0.964 ± 0.006	0.924 ± 0.015	0.821 ± 0.013	0.661 ± 0.029
		g_mean	0.924 ± 0.025	0.923 ± 0.007	0.877 ± 0.022	0.780 ± 0.024	0.695 ± 0.022
	Bi-GRU	AP	0.958 ± 0.007	0.966 ± 0.002	0.921 ± 0.019	0.828 ± 0.029	0.656 ± 0.012
		g_mean	0.929 ± 0.023	0.922 ± 0.012	0.861 ± 0.014	0.793 ± 0.016	0.684 ± 0.028

Table 4.5: Comparison of all architectures as the ratio of anomalies decreases. The results were taken as averages over 10 runs. The corresponding standard deviation is reported as well. The best AP values for each dataset are highlighted in italic while the best g_mean values are highlighted in bold.

Architecture	g_mean ranking	$g\_mean\ APV$
Multi-head LC1d-LSTM	2.2	-
Multi-head LC1d-Bi-LSTM	3.1	1.0
Multi-head LC1d-GRU	3.2	1.0
Multi-head Conv1d-Bi-LSTM	4.8	1.0
Multi-head Conv1d-GRU	5.1	1.0
Multi-head Conv1d-LSTM	5.6	1.0
Multi-head Conv1d-Bi-GRU	5.8	1.0
Multi-head LC1d-Bi-GRU	6.2	1.0
Multi-head Conv1d-RNN	9	0.129676
Multi-head LC1d-RNN	11.4	0.010289
Multi-channel Conv1d-Bi-LSTM	11.8	0.006885
Multi-channel Conv1d-GRU	12	0.005836
Multi-channel Conv1d-Bi-GRU	12.2	0.004883
Multi-channel Conv1d-LSTM	13.8	0.000575
Multi-channel Conv1d-RNN	13.8	0.000575

Table 4.6: Average Friedman rankings and APVs using Holm's procedure in g_mean for all the architectures. The horizontal dashed line delimits the architectures rejected (located below the line) as a consequence of setting the level of significance to $\alpha = 0.05$.

The first insight was determined by the correlation between the network size and the required training time as the window length varies. Figure 4.8a shows how the size of the network increases as the length of the window grows. However, Figure 4.8b shows that, although the window length becomes bigger, the required training time decreases as the window length increases. This is due to the correlation between the number of windows (W_L) and the size of the feature maps resulting from applying convolutions over each window. The smaller W_L the smaller the size of the feature map and thus the smaller the network size. However, W_N increases. In contrast, the bigger W_L the bigger the size of the feature map and thus the bigger the network size. Nevertheless, W_N decreases. Therefore, the time required to train the models is given by the number of windows in which the time series are divided regardless of the network size. It can be observed that the architectures including the RNN layer have the smallest number of parameters. However, they are the slowest

Architecture	g_mean ranking	$g\_mean\ APV$
Multi-head LC1d-LSTM	2.2	-
Multi-head LC1d-Bi-LSTM	3.1	1.0
Multi-head LC1d-GRU	3.2	1.0
Multi-head Conv1d-Bi-LSTM	4.8	0.523576
Multi-head Conv1d-GRU	5.1	0.519621
Multi-head Conv1d-LSTM	5.6	0.379001
Multi-head Conv1d-Bi-GRU	5.8	0.360617
Multi-head LC1d-Bi-GRU	6.2	0.256997
Multi-head Conv1d-RNN	9	0.003068
Multi-head LC1d-RNN	11.4	0.000417

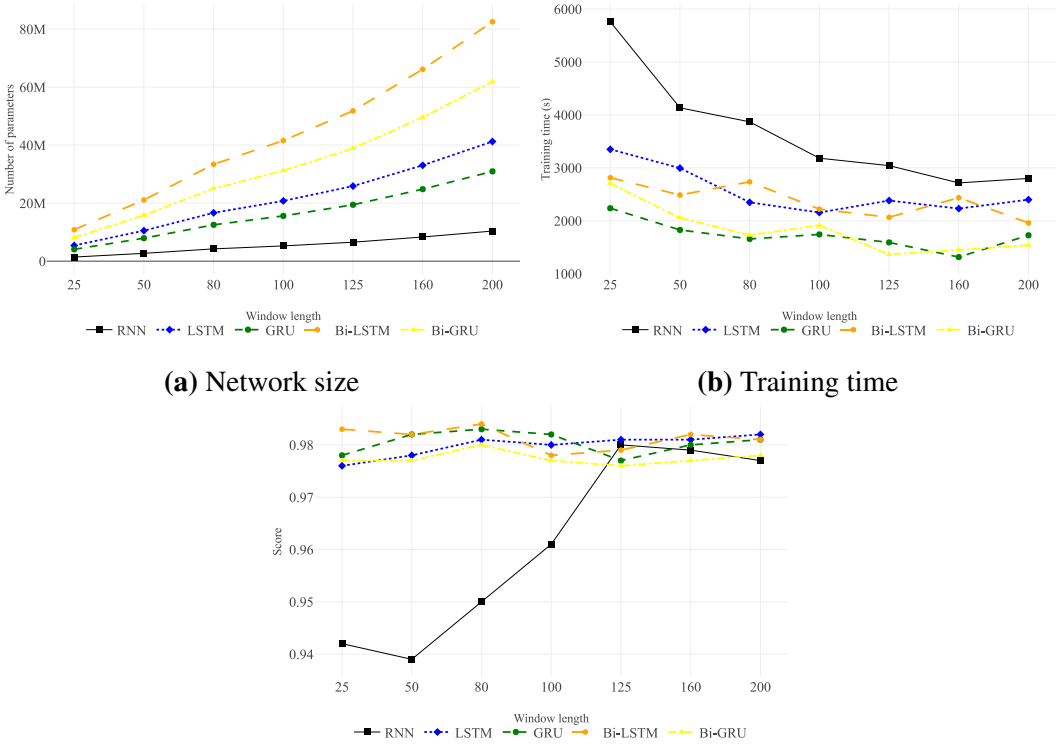
Table 4.7: Average Friedman rankings and APVs using Holm's procedure in g_mean for Multi-head architectures. The horizontal dashed line delimits the architectures rejected (located below the line) as a consequence of setting the level of significance to $\alpha = 0.05$.

architecture to converge. On the other hand, architectures that include Bi-LSTM and Bi-GRU layers have the largest number of parameters. This is due to the fact that they process data twice. However, they require as much time to train as others, being the latter one of the fastest at training time.

Figure 4.8c shows that, except for the RNN, the performance of the models remains stable regardless of the length of the window. The RNN achieved an improvement of at most 4% when $W_L = 125$. Overall, for this use case, the length of the window has no importance apart from the size of the network and the time required to train it. A trade-off between network size and training time must be found.

4.4.5 Transfer Ability

In this experiment, the ability of the proposed architecture to adapt to new sensor configurations by transferring the knowledge from one model to another is analyzed. To this end, the following experiments were conducted:



(a) Network size

(b) Training time

(c) g_mean

Figure 4.8: Performance of the Multi-head Conv1d architectures as the window length varies. Experiments conducted with the original dataset.

- First, three models were trained with 10, 15 and 20 sensor data, respectively. They were used as the base models.
- Then, two new models were generated based on previous models. For that, five new sensor data were added to the base models trained with 10 and 15 sensor data (10+5 and 15+5).
- Finally, another two models were trained by removing five sensor data to the base models trained with 20 and 15 sensor data (20-5 and 15-5).

To differentiate the models used in this experimentation, models generated from scratch (10, 15, and 20) were defined as base models. Models generated from base models, by adding or removing sensor data, were defined as transfer knowledge based (TKB) models. As long as TKB models were compared against

base models trained with the same number of sensors (i.e., a model trained with 15+5 sensors against a base model trained with 20 sensors), these models were defined as target models. Therefore, base models also actuate as target models. TKB models were compared against target models due to the fact that models trained with the same number of sensor data might obtain similar results. However, TKB models should require less time to train. Therefore, target models served as baselines. The comparison was carried out in terms of training time and performance (metrics).

Table 4.8 shows the performance of all the models. The best g_mean values for each group of sensors and recurrent layers are highlighted in bold. Figure 4.9 shows the time required to train them. As the performance of the architectures is concerned, it can be observed that TKB models with added sensor data obtained slightly worse results than their corresponding target models. However, they are up to two times faster than target models at training time. TKB models with removed sensor data achieved better results than target models. In fact, they obtained the best g_mean values in comparison to TKB models with added sensors and their corresponding target models. For TKB models with removed sensors, we cannot draw general conclusions about the improvement of training speed due to the fact that there were cases where TKB modes were faster than target models (GRU, Bi-LSTM, and Bi-GRU in 20-5 model), and vice versa (15-5 model).

Hence, it is demonstrated that the proposed architecture is able to adapt to dynamic scenarios.

# sensors	20	15+5	15	10+5	20-5	10	15-5
RNN	**0.961**	0.946	0.924	0.905	**0.934**	0.871	**0.886**
LSTM	**0.980**	0.954	0.923	0.907	**0.930**	0.873	**0.897**
GRU	**0.977**	0.957	0.933	0.922	**0.941**	0.886	**0.914**
Bi-LSTM	**0.978**	0.963	0.947	0.916	**0.952**	0.884	**0.910**
Bi-GRU	**0.977**	0.964	0.958	0.951	**0.963**	0.867	**0.926**

Table 4.8: g_mean of Multi-head Conv1d architectures, both base and TKB models. The best g_mean values for each group of sensors and recurrent layers are highlighted in bold. Experiments conducted with the original dataset.

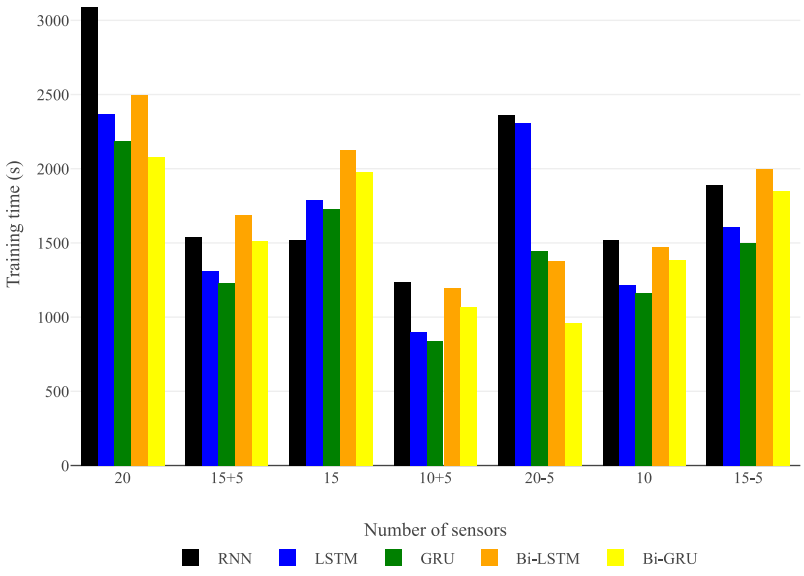


Figure 4.9: Training time of both base models and TKB models. They are all grouped by the number of sensors used.

4.4.6 Comparing with Traditional Algorithms

In this section, the Multi-head CNN-RNN is compared against traditional algorithms. The aim of this experimentation is to check whether the proposed architecture overcomes traditional algorithms. As mentioned in Section 4.3.3, the analyzed traditional algorithms are RF, CatBoost, LightGBM and ensemble. Recall that the AutoML was used to obtain the best model and its corresponding hyper-parameters automatically. Since in this experimentation the five datasets referring to the different percentage of anomaly ratios were used, the AutoML provides the best model for each dataset, which can be any of the listed algorithms. Moreover, recall that the AutoML performs several pre-processing steps to clean the data or extract the most important features, whereas the Multi-head CNN-RNN architecture works directly over raw data. To measure the impact of the pre-processing phase performed by the AutoML, other tests were conducted in which the best traditional algorithms were evaluated without pre-processing the data. To this extent, the hyper-parameters of

the models obtained by the AutoML were first extracted and then, the models were applied over raw data.

Table 4.9 shows the results obtained in this experimentation using the AutoML and the resulting traditional models applied directly over raw data. Recall that the results of the Multi-head CNN-LSTM are the same as in Table 4.5. Overall, the AutoML obtained the best results with the RF despite for the dataset with 15% of anomalies, in which the LightGBM was the winner. As it can be observed, the results obtained by the traditional algorithms resulting from the AutoML are worse than the ones obtained by the Multi-head CNN-LSTM for all the datasets in terms of AP and g_mean. The difference between the proposed architecture and the AutoML increases as the percentage of anomalies decreases, being the dataset with 3% of anomalies the one in which the difference is higher. Concretely, the gap between them is of 10% in terms of AP.

% of anomalies		20	15	10	5	3
Multi-head CNN-LSTM	AP	*0.993*	*0.992*	*0.975*	*0.941*	*0.903*
	g_mean	**0.980**	**0.974**	**0.949**	**0.909**	**0.830**
AutomML	AP	0.962	0.960	0.946	0.918	0.808
	g_mean	0.966	0.942	0.920	0.869	0.815
Raw ML	AP	0.911	0.879	0.854	0.715	0.672
	g_mean	0.968	0.950	0.916	0.869	0.818

Table 4.9: Comparison results between the Multi-head CNN-LSTM and traditional algorithms tested with and without the AutoML. The best AP values for each dataset are highlighted in italic while the best g_mean values are highlighted in bold.

As the impact of the pre-processing step performed by the AutoML, the table shows a significant difference between using the AutoML of using the resulting models without pre-processing the data. It can be seen that the g_mean is similar for all the datasets in both cases. However, focusing on the AP, the difference between them is higher, ranging from 10% to 20% depending on the dataset. Hence, we can conclude that the automatic pre-processing phase of the AutoML highly impacts on the final result. As the proposed Multi-head CNN-RNN architecture processes directly raw data, it makes more sense comparing it against traditional algorithms

applied also over raw data. Therefore, we can conclude that the proposed algorithm outperforms the evaluated traditional algorithms.

4.5 Conclusions

This chapter has presented a novel Multi-head CNN-RNN architecture for time series anomaly detection. The proposed architecture has demonstrated its potential in a real industrial scenario where anomalies are effectively detected on a service elevator based on multiple sensor data. The proposed architecture uses first the Multi-head CNN to extract the features of each sensor data on a fully independent basis to deal with heterogeneous data. Moreover, it processed the sensor data in a window-based method and thus it can focus the feature extraction in the different phases of the sensor data. Afterward, the RNN uses the extracted features to determine whether an anomaly occurred during the elevator journey or not. Finally, as this architecture can be implemented with several types of layers, all possible alternatives have been analyzed under different scenarios.

The experimental results have shown that our approach can detect anomalies over multi-time series effectively, since its performance maintains high even if the percentage of anomalies within the training dataset is reduced up to a 3%, outperforming traditional algorithms. Furthermore, it has been proven through statistical tests that the proposed Multi-head CNN architecture also outperforms the base Multi-channel CNN in all the tested scenarios, due to processing each sensor data independently. Regarding the recurrent layers, the RNN has shown the worse results in all the tests while others have obtained similar results. In this way, the use of the RNN layer is not recommended. However, a deep analysis must be conducted to know which layer performs best for the use case under consideration.

The experimentation has also demonstrated the ability of the proposed architecture to adapt to new sensor configurations as new models are successfully generated by transferring knowledge from one model to another. However, further research must be done to improve the performance and the required training time of the models generated by this method. The downside of this implementation is that when the sensor configuration changes, it is necessary to wait until there is enough data regarding the new sensors to train the model since it is a supervised learning. To

boost this process, TL could be used to transfer the ability to extract features from sensor data from one convolutional head to another. Nonetheless, little research has been conducted in this field and in TL techniques for time series and thus, it would be a good line of research for the future.

On the other hand, the proposed architecture has been validated with a dataset containing fixed-length time series. However, in some real use cases, they are not. Thus, further research must be done to analyze the performance of the proposed architecture at the time of processing time series with different frequencies. In our use case, the unique requirement would be to divide the time series into the same number of windows (W_L). For that matter, shorter time series would be filled by padding them until they reach the maximum length, which will be determined by the longest time series. Usually, the value used to pad the time series is masked to not take it into account at the time of computing the gradients. However, the methodology to mask padded values in one-dimensional convolutions is an ongoing research and therefore, it is another line of research for the future.

Finally, although the Multi-head CNN-RNN increases linearly according to the number of sensors, the training process could be parallelized. To this end, each convolutional head could be trained on a different GPU and the recurrent side of the architecture on another. This could reduce the training phase up to a point because the most computationally expensive step within the training phase lies in the recurrent part rather than on the convolutional side. This is because all the features coming from the convolutional heads are then concatenated and sent to the RNN, which might lead to a bottleneck. However, parallelizing the RNN is challenging because the order with which data is computed matters and therefore, it would be necessary to ensure that all the gradients were computed in chronological order.

By far, the greatest danger of Artificial Intelligence is that people conclude too early that they understand it.

Eliezer Yudkowsky

5

Interpretation of Deep Learning for Diagnosing Anomalies

The previous chapter has described how to detect anomalies using the proposed Multi-head CNN-RNN architecture. This architecture has demonstrated to achieve good results at detecting anomalies, but it is unable to point out where and when anomalies occurred. To solve this issue, this chapter presents a methodology to interpret the predictions of the Multi-head CNN-RNN architecture to diagnose the detected anomalies. First, the motivation of this contribution is exposed (Section 5.1). Next, the followed methodology and how it can be implemented within the Multi-head CNN-RNN architecture are described (Section 5.2). Afterward, the configuration of the conducted experimentation to validate this work is detailed (Section 5.3) and then the results obtained are analyzed (Section 5.4). Finally, the conclusions and future works resulting from this work are exposed (Section 5.5).

5.1 Motivation

DL algorithms have demonstrated during the last decade to be more effective than traditional algorithms in a wide range of domains. For instance, in the previous chapter, it has been shown how our proposal outperforms other traditional algorithms in the multi-time series anomaly detection field. However, it is yet not clear how DL algorithms learn or how they reach a decision as they lack interpretability.

This can be critical in many domains where classifying correctly a given observation might not be sufficient. This is the case of detecting anomalies in the industrial domain since identifying the root causes of the anomaly can be the key to fix it or to take further actions to prevent it in the future. Hence, in these scenarios, it is as important to detect an anomaly as it is to identify the reasons why it occurred. The interpretability of the model can also be critical at the time of building, evaluating, or tuning the model to analyze whether the model is learning the right features/patterns. These facts can lead researchers and practitioners to mistrust DL algorithms and, as a consequence, might cause them to stop using DL in certain domains (Ren et al. 2017). For instance, the proposed Multi-head CNN-RNN effectively detects anomalies in multi-time series, but it is not capable of pointing out where the anomaly lies or of stating the reasons why such a decision is made. Consequently, the maintenance operator has no clue as to where to initiate the troubleshooting and thus, this usually falls on the operator's experience.

The lack of interpretability of DL algorithms is a well-known but not yet solved issue. Recently, SHAP (Lundberg and S.-I. Lee 2017) emerged as a promising approach for explaining DL algorithms by calculating the impact that each input of the model has on the final decisions. Regarding our Multi-head CNN-RNN, this would help us to identify which sensor or group of sensors have more impact on the final decision. Nevertheless, we would not be aware of at which point in the time series the anomaly occurred. For this matter, the attention mechanism (Bahdanau et al. 2014) has attracted the interest of researchers in the DL field due to its ability to focus on the most relevant time-steps of the input data. Focusing on our Multi-head CNN-RNN architecture and its window-based time series processing, the attention mechanism would help us to identify which window or group of windows have more impact on the final decision.

To our knowledge, the performance and suitability of these mechanisms have not been yet proven to identify the reasons why an anomaly has occurred within a multi-time series. In the proposed approach, SHAP is used to identify from which sensors the anomaly is coming, whereas the attention mechanism is used to determine in which point of the time series occurred the anomaly. Thus, our motivation is to combine both methods and engage them to the Multi-head CNN-RNN in an effort to explain why it classifies an observation as anomalous, providing key information to the maintenance operator to fix the anomaly.

5.2 Improving the Interpretability of the Multi-Head CNN-RNN Architecture

This section describes the methodology followed to identify the reasons why the Multi-head CNN-RNN architecture detects an anomaly, thus improving its interpretability. First, the SHAP and how it can be implemented within the proposed architecture are described (Section 5.2.1). Next, the attention mechanism is described as well as its implementation (Section 5.2.2). Finally, a general overview of how both mechanisms are integrated within the Multi-head CNN-RNN architecture is detailed (Section 5.2.3).

5.2.1 SHAP for Identifying Anomalous Sensors

As mentioned, the SHAP is an algorithm that can explain the output of any ML model by measuring the importance that each input variable has in the final decision. In this work, it is used to measure the relevance that each sensor has at the time of detecting an anomaly. It is assumed that when an anomaly is detected, the sensor(s) from which the anomaly derives will be the most important. Hence, it is possible to tell the operator which sensor(s) or component(s) should be checked to analyze the anomaly.

To implement the SHAP into the Multi-head CNN-RNN model, some modifications must be done in the original working process. Figure 5.1 shows the modified working process. First, the Multi-head CNN-RNN model must be trained using the training dataset. Second, the SHAP model must be trained. Recall that the

SHAP model requires the user's model (the model that must be explained), and a portion of the training dataset to do so. However, it should be noted that the SHAP model must only be trained using the classification/regression model. Considering that the Multi-head CNN-RNN can be seen as the combination of two models in which the Multi-head CNN acts as the feature detector and the RNN as the classifier, only the latter model is required to train the SHAP. Therefore, once the Multi-head CNN-RNN model is trained, it must be divided into two parts: the Multi-head CNN, and the RNN. Regarding the training dataset, the input data of the RNN side of the architecture is not the original dataset but the original dataset processed by the Multi-head CNN. That is, the input data of the RNN is composed of a sequence of feature maps obtained through the convolutional part of the model. Hence, to train the SHAP, the Multi-head CNN is used to generate the training dataset, and the RNN model is used as the classification model.

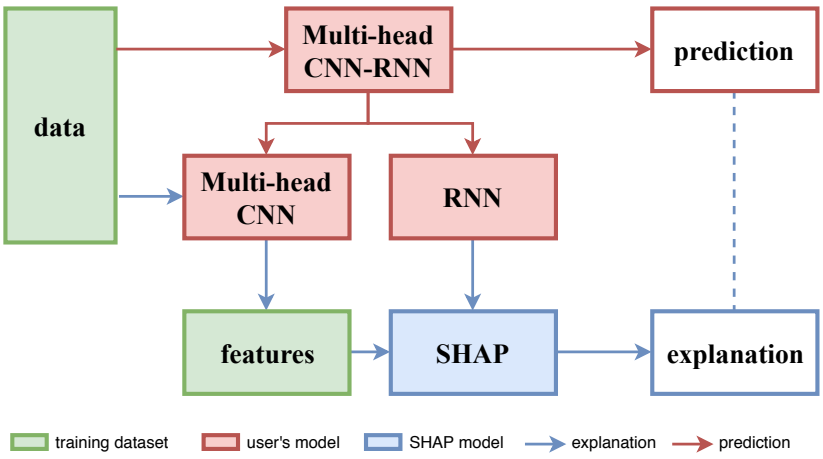


Figure 5.1: Implementation of the SHAP into the Multi-head CNN-RNN architecture.

Another fact to consider at the time of implementing the SHAP within this architecture, is the management of the output of the explainer. Figure 5.2 shows the process to manage the output values. The SHAP model returns a single SHAP value for each input variable. However, the input data of the RNN consists on a sequence of feature maps where each feature map (F_w) contains the extracted features of each window (w) of all sensors. Recall that each F_w is formed as the concatenation of the feature maps resulting from all the convolutional heads of the Multi-head CNN. In this way, the features of each sensor are sorted within each F_w. Consequently,

a SHAP value will be obtained for each feature of the F_w. Nonetheless, the main goal is to measure the importance of each sensor in the final decision. Thus, all the SHAP values corresponding to each sensor are then summed to obtain a single SHAP value per sensor.

To give an example of this process, let the number of sensors be equal to 20 and the size of the $F_w = 2,000$, thus, each sensor is represented by 100 ($\frac{2,000}{20}$) features, where the first 100 features correspond to the first sensor, the next 100 to the second sensor, and so forth. Therefore, 2.000 SHAP values will be obtained as a result, where each group of 100 values corresponds to each sensor. Finally, the values corresponding to each sensor are summed all together and thus, 20 SHAP values corresponding to the 20 sensors will be obtained.

It is noteworthy that this is only possible due to how the proposed Multi-head CNN-RNN independently processes sensor data, providing feature maps sorted by sensor. Recall that the Multi-channel CNN-RNN does extract features of all sensor data mixed all together and thus, it would not be possible to separate data by sensor after applying the convolution. Hence, this is another benefit of the proposed Multi-head CNN-RNN architecture.

5.2.2 Attention Mechanism for Identifying Anomalies in the Time Domain

As mentioned, the attention mechanism can help RNNs to pay attention to the most relevant time-steps of the input data, blurring less important ones. This is achieved by computing an attention vector that contains the probability values indicating how important each of the time-steps is. Consequently, each time-step is weighted so that time-steps with higher values will have more influence in the final decision. The attention vector, besides improving the performance of the network, it can be used to visualize the resulting probabilities to point out in which time-steps has the RNN focused to make the corresponding decision.

As the goal of the proposed Multi-head CNN-RNN architecture is to detect anomalies, it is assumed that the model will pay more attention to regions of the time series where anomalous behaviors exist. Thus, the attention mechanism is used to identify those regions and thus, to indicate to the maintenance operators at which

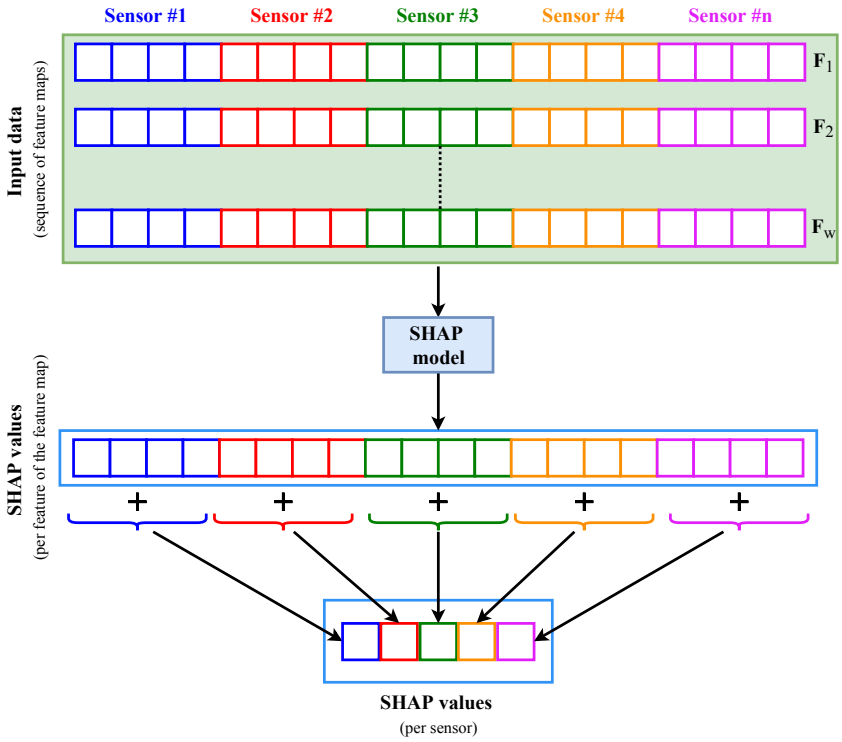


Figure 5.2: Implementation of the SHAP into the Multi-head CNN-RNN architecture.

point of the time series the anomaly arose. Recall that the proposed architecture process the sensor data through a sliding window and that each window represents a time-step. Consequently, if an anomaly falls within a given window, the attention mechanism should emphasize that window over the others.

To implement the attention mechanism within the Multi-head CNN-RNN architecture, it is set between the last LSTM layer and the final dense layer. Figure 5.3 shows the new RNN side of the architecture. As it can be observed, the attention layer provides a probability value for each time-step (window), which will be then used by the output layer to make the decision. Hence, once that the Multi-head CNN-RNN has made a prediction, the importance of each window can be obtained by extracting the attention vector from the attention layer.

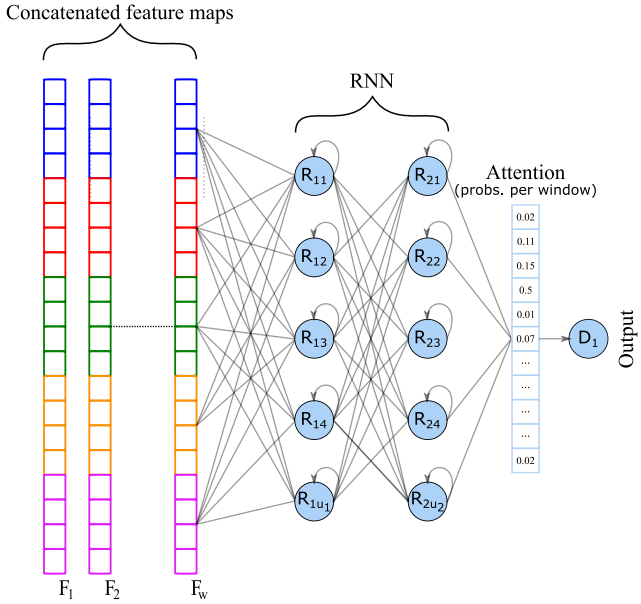


Figure 5.3: RNN side of the proposed architecture with the attention layer.

5.2.3 Combining Attention and SHAP to Explain DL Decisions

In the previous sections, how to implement both the SHAP and the attention mechanism separately have been described. Now, they must be implemented and engaged all together within the Multi-head CNN-RNN architecture in order to explain its outputs. Figure 5.4 shows this implementation.

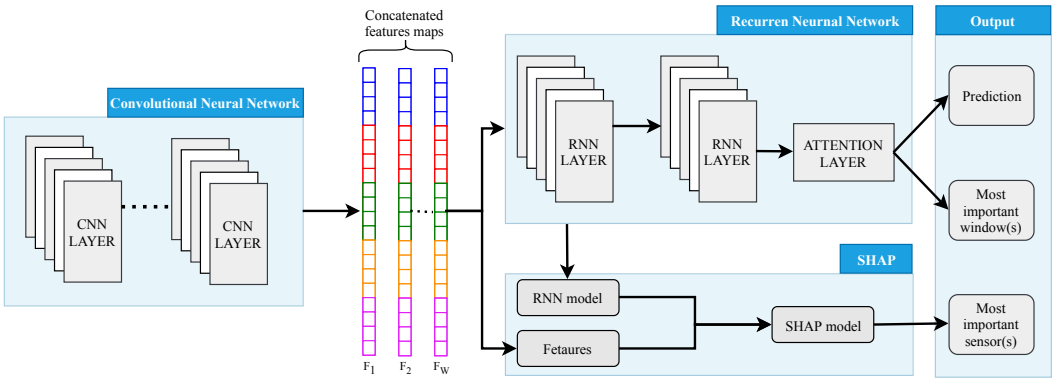


Figure 5.4: Implementation of SHAP and attention within the Multi-head CNN-RNN architecture.

To implement them, the attention mechanism does not have any impact on the architecture design other than appending a new layer after the last RNN layer. In contrast, the SHAP requires implementing an additional branch to the architecture. As far as both the SHAP and the RNN require the same input data, they are located in parallel, being both connected to the output of the convolutional part. They can be seen as two different and concurrent modules. The SHAP module is also connected to the RNN module to use the RNN model to train the SHAP model. Note that this connection is only used in the training phase since once the SHAP model is trained, it does not require the RNN model anymore to work. As a result of this implementation, the output of this architecture is threefold: the prediction of the Multi-head CNN-RNN model, the attention vector extracted from the attention layer, and the SHAP values resulting from the SHAP model. Therefore, besides the prediction itself, information about which window(s) and sensor(s) have been most important for the network to make such a prediction is provided.

It is worth pointing out that the SHAP module does not affect the output of the Multi-head CNN-RNN model, whereas the attention layer does. Therefore, the influence that the attention mechanism has in the performance of the Multi-head CNN-RNN model must be analyzed (see Section 5.4.2).

5.3 Experimental Framework

This section describes the set-up of the experimentation conducted to evaluate the use of the SHAP and the attention to explain the output of the Multi-head CNN-RNN. Thus, it shows the methodology followed to validate the explanations of the model (Section 5.3.1), and the parameters and the configuration of the used algorithms (Section 5.3.2).

5.3.1 Validation Methodology

As far as the output of both the attention vector and the explainer can be seen as a general recommendation rather than as a precise response, validating these explanations by means of traditional evaluation metrics (i.e., accuracy, precision, or recall) is not possible. Consequently, the validation was performed by visualizing

the resulting explanations and the raw sensor data corresponding to the classification of a given anomalous journey. Then, all the images were compared to verify whether the explanations match the ground truth or not. This comparison was conducted by a domain expert.

To validate this work, the industrial case study described in Section 2.6 was used. In this case, only the original dataset was used, that is, the dataset containing the 20% of anomalies. The goal of this experimentation is to evaluate whether the implemented interpretability mechanisms are suitable to identify in which sensors and region of the time series the anomalies lie. However, although the dataset contains three types of anomalies (point, context-specific, and collective) that come from different sensors, all samples are labeled in a binary way (normal, and anomalous). This means that there is no information regarding the causes of the anomalies. This is because when failures are introduced to the model as input parameters, the corresponding simulation is automatically labeled as anomalous, although the type and the root causes of the anomaly cannot be defined without being checked manually by the domain expert. Therefore, assigning an anomaly type and the causes of each anomaly would take much time an effort since the dataset contains about 2,800 (20% out of 14,000) anomalous simulations in the original dataset.

To generate a validation dataset, a small portion of the original dataset was extracted and labeled by the domain expert. This validation dataset is composed of 144 anomalous simulations, which are labeled by the expert depending on the error introduced in the model as an input parameter. Table 5.1 describes the errors introduced in the model and the sensors that might be affected as a consequence of the introduced failure. The fact that these sensors may be affected does not mean that all of them suffer anomalies in all cases (i.e., a fault of type 3 does not have to be reflected in all the described sensors but in some of them). Note that errors with label 1 and 2 can be of context-specific or collective anomalies, whereas errors with label 3 can also refer to point anomalies due to impacts in the cabin as a result of peaks in the guidance system. There were a total of 50 anomalous journeys containing point anomalies, in which one or more anomalies can be shown within each journey.

It should be noted that this dataset was only used to validate the interpretability mechanisms. Hence, the original dataset was used to train the Multi-head CNN-RNN and the SHAP.

Label	Introduced anomaly	#samples	Affected sensors
1	Reduction in the electric machine's inductance (L_q)	24	*Vq* and *Vd*
2	Reduction in permanent magnets generated flux (λ)	24	*Vq* and *Vd*
3	Malformations in the guiding system (GS)	96	*Ax, Ay, Az, Fc, FrictionCW FrictionCabin, Fsupport,* and *Iq*

Table 5.1: Summary of the anomalies within the validation dataset.

5.3.2 Parameters and Configuration of the Algorithms

In this section, the configuration parameters of the used algorithms are described. Regarding the Multi-head CNN-RNN, the used configuration was the same as the used in the previous chapter. As a reminder, Table 5.2 shows the configuration of the network. The only change in the architecture was the implementation of the attention layer between the last recurrent layer and the final dense layer. However, as the attention layer may impact on the performance of the model, the architecture was tested using one and two recurrent layers to analyze the real impact.

As mentioned in Section 3.2.2, the SHAP has several ways to compute the SHAP values depending on the model that wants to be explained. In this case, the two approaches suited for explaining DL models were used to train the SHAP model: *DeepExplainer*, and *GradientExplainer*. Although they compute the SHAP values differently, their implementation is equal in both cases as both approaches are provided by the SHAP library and no parameters must be configured.

In this experimentation, the Conv1d layer was used for the convolutional side of the architecture, whereas the LSTM and GRU layers were used for the recurrent

Parameter	Value
conv. layers	4
conv. filters	20
recurrent layers	1 and 2
recurrent units	128
attention layers	1
dropout	0.25
window length	100
window step	100 (No overlapping windows)
learning rate	0.00001
epochs	30 (with early stop)
batch size	50

Table 5.2: Parameter specification for the Multi-head CNN-RNN with attention.

side. Consequently, three different architectures were tested. Recall that these architectures were tested using one and two recurrent layers.

5.4 Results and Discussion

In this section, the results obtained in the experimentation are discussed. First, the impact of the attention layer in the anomaly detection architecture described in the previous chapter is analyzed (Section 5.4.1). Then, the ability of the SHAP (Section 5.4.2) and the attention layer to explain anomalies are analyzed (Section 5.4.3).

5.4.1 Impact of the Attention Mechanism on the Performance of the Multi-Head CNN-RNN Model

As interpreting methods used in this chapter are regarded, the attention mechanism does influence in the performance of the Multi-head CNN-RNN architecture as it is embedded within the model. As mentioned, the attention layer aims to pay attention to the most relevant time-steps and to blur the others to increase the performance of the model. Hence, the main goal of this experimentation was to

evaluate whether the attention layer can increase the ability of the proposed model to detect anomalies under several scenarios. As in the previous chapter, five datasets with different percentage of anomalies were tested: 20%, 15%, 10%, 5%, and 3%. Besides, the Multi-head CNN-RNN using one and two recurrent layers was analyzed, where the LSTM and GRU layers were considered. To better distinguish the used models, the Multi-head CNN-RNN architecture proposed in the previous chapter is referred to as the base model, whereas the models containing the attention layer are defined as the attention model. Table 5.3 shows the results of the experimentation.

As it can be observed, the overall difference between the base and attention models is not significant when the percentage of anomalies is high. However, the difference increases in higher imbalanced scenarios (5% and 3% of anomalies), in which attention models achieve better results. In this scenario, the attention model generally obtains a 2-3% (AP and g_mean) of improvement over the base model when the LSTM is used as a recurrent layer. When using the GRU, the difference with respect to the base model decreases although there is an improvement of 3% on the g_mean when the percentage of anomalies is reduced to 3% and two recurrent layers are used. As the number of recurrent layers is regarded, the performance obtained with one and two layers is very similar and thus, using one recurrent layer would be preferred since the size of the network would be smaller and less complex.

5.4.2 Interpreting Predictions with the Attention Mechanism

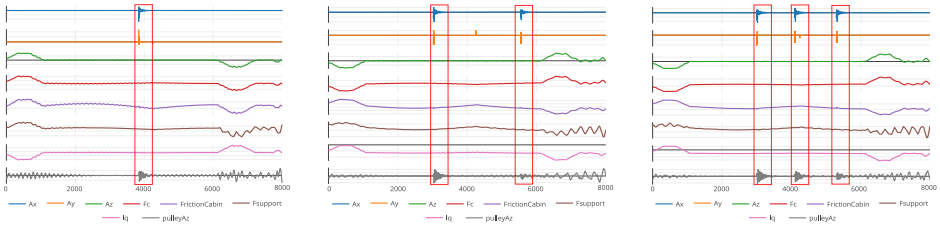
In this section, the ability of the attention mechanism to identify the time spans in which the anomalies occurred is evaluated. In this experimentation, point anomalies were differentiated from the rest because they have different characteristics. Point anomalies exhibit an abnormal behavior on specific time-steps of the time series whereas the others show an abnormal behavior throughout the entire time series. Hence, the attention mechanism would have to highlight concrete time-steps when detecting point anomalies, unlike in the rest of the anomalies in which the attention scores of the time-steps are expected to be more similar between them. As mentioned, there are a total of 144 anomalous journeys from which 50 of them contain point anomalies. Recall that each journey might contain one or more point anomalies.

% of anomalies			20%	15%	10%	5%	3%
Base model	LSTM	AP	0.993	0.992	0.975	0.941	0.903
		g_mean	**0.980**	**0.974**	0.949	0.909	0.830
	GRU	AP	*0.994*	*0.994*	*0.987*	0.965	0.929
		g_mean	0.977	**0.974**	0.964	0.921	0.857
Attention (1)	LSTM	AP	0.990	0.991	0.980	0.967	*0.931*
		g_mean	0.960	0.968	0.954	0.939	0.866
	GRU	AP	0.987	0.987	0.982	0.963	0.922
		g_mean	0.964	0.962	0.963	0.937	0.867
Attention (2)	LSTM	AP	0.993	0.990	0.985	*0.970*	0.923
		g_mean	0.967	0.969	**0.968**	**0.944**	0.880
	GRU	AP	*0.994*	*0.994*	0.986	0.962	0.924
		g_mean	0.974	0.966	0.966	0.929	**0.890**

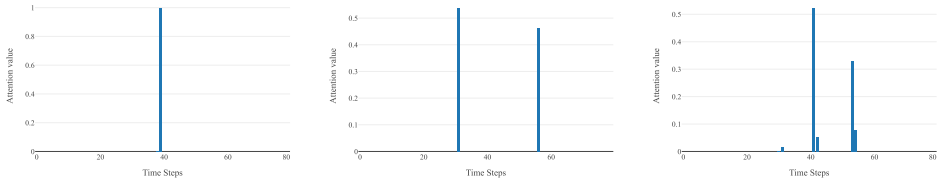
Table 5.3: Comparison of the base model and the attention model with one and two recurrent layers as the ratio of anomalies decreases. The results were taken as averages over 10 runs. The best AP values for each dataset are highlighted in italic while the best g_mean values are highlighted in bold.

First, the influence that the number of layers has on the model's interpretability is analyzed. Figure 5.5 shows three journeys containing point anomalies and their respective attention interpretations with one and two recurrent layers. Each of these journeys is defined as a point anomaly journey n (PAJ_n), where n refers to the number of point anomalies that each journey contains. Figures 5.5a, 5.5b, and 5.5c show the raw data of these three journeys, which have one, two, and three point anomalies, respectively. Hence, they are defined as PAJ_1, PAJ_2 and, PAJ_3. The second row of images (Figures 5.5d, 5.5e, and 5.5f) correspond to the interpretability of the journeys by means of the model containing a single recurrent layer, whereas the second row of images (Figures 5.5g, 5.5h, and 5.5i) depict the interpretability of the model with two recurrent layers. Note that although only the interpretability of three predictions is shown, they highly represent the rest of the journeys containing point anomalies.

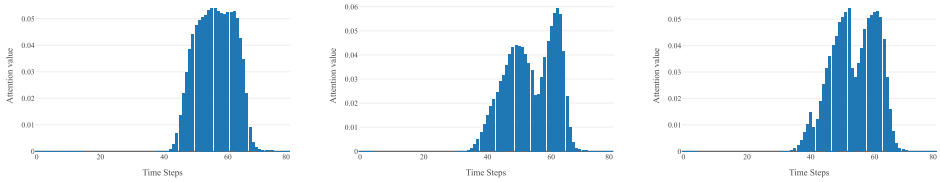
As it can be observed, the attention model with one recurrent layer can identify when the point anomalies occur as the focus is paid in the time-steps in which they arose. Figures 5.5d and 5.5e show that the model was able to identify the



(a) Row data (PAJ_1) **(b)** Row data (PAJ_2) **(c)** Row data (PAJ_3)



(d) Attention vector with 1 layer (PAJ_1) **(e)** Attention vector with 1 layer (PAJ_2) **(f)** Attention vector with 1 layer (PAJ_3)



(g) Attention vector with 2 layer (PAJ_1) **(h)** Attention vector with 2 layer (PAJ_2) **(i)** Attention vector with 2 layer (PAJ_3)

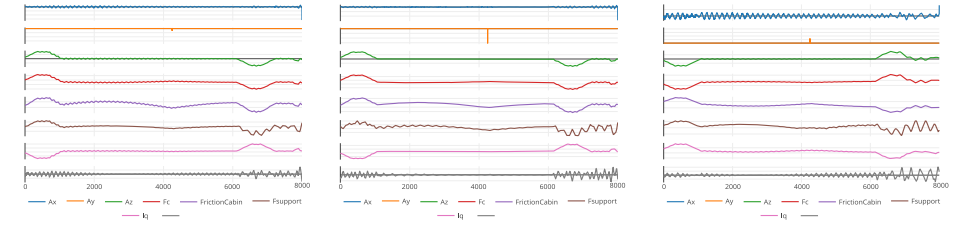
Figure 5.5: Comparison of the model's interpretability by means of the attention mechanism using one or two recurrent layers. The interpretability of three journeys containing point anomalies is depicted.

unique point anomaly occurred in PAJ_1 (Figure 5.5a), as well as the two anomalies shown in PAJ_2 (Figure 5.5b). Conversely, Figure 5.5f show that the model only identified two anomalies when there are actually three in PAJ_3 (Figure 5.5c). The reason lies in that the model only needed these two time-steps to classify the journey, thus blurring the rest of the time-steps. Therefore, the attention vector does not always show exactly all the point anomalies but in which time-steps it has focused to classify the journey as anomalous.

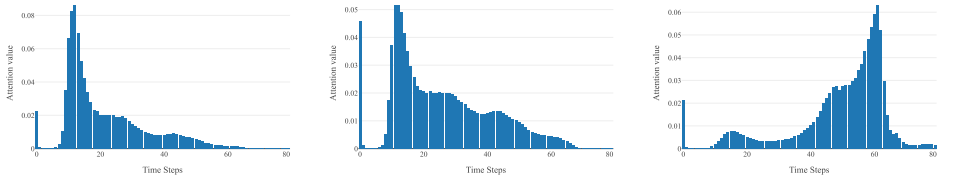
On the other hand, it can also be shown that the attention model with two recurrent layers is not able to identify correctly the point anomalies in any of the three journeys (Figures 5.5g, 5.5h, and 5.5i). They seem to pay attention to the period between point anomalies rather than to the specific point where they occur. The poor interpretability obtained by this model with respect to the model with one recurrent layer might be caused by the additional abstraction level achieved as a consequence of stacking a second layer. In this way, the attention is slightly blurred throughout the recurrent layers and thus, applying a single recurrent layer is recommended as it is more interpretable, besides achieving similar performance.

As the other anomalies are regarded (no point anomalies), Figure 5.6 shows another three anomalous journeys (first row) and their corresponding interpretation employing the attention mechanism (second row). As concluded before, the attention model with one recurrent layer is more interpretable and thus, only this model was used. Each of these journeys corresponds to a different anomaly type (label) and they are defined as anomalous journey x (AJ_x), where x refers to the type of anomaly (L_q, λ, or GS). Of these journeys, AJ_{L_q} and AJ_λ are uphill (Figures 5.6a and 5.6b), while AJ_{GS} is downhill (Figure 5.6c). Note that unlike point anomalies, these anomalies are not visible at a glance.

As in can be observed, the attention is more spread over the time-steps in all the journeys (Figures 5.6d, 5.6e, and 5.6f) since, unlike in point anomalies, there is no particular point on which to pay attention. Nevertheless, the attention shows interesting indications regarding where the anomalies might have occurred. In the first two journeys, the attention is paid mostly in the acceleration phase of the elevator, whereas in the third journey the braking phase is where the model paid more attention. Overall, the rest of the analyzed journeys follow a similar pattern and thus, we can conclude that the acceleration phase is the most critical when the elevator moves upward, while the braking phase is the most critical when the elevator moves down. However, the attention mechanism only indicates when the anomaly might have occurred but not which sensors caused it. Thus, the SHAP can shed light on this issue.



(a) Row data (AJ_{L_q}) **(b)** Row data (AJ_{λ}) **(c)** Row data (AJ_{GS})



(d) Attention (AJ_{L_q}) **(e)** Attention (AJ_{λ}) **(f)** Attention (AJ_{GS})

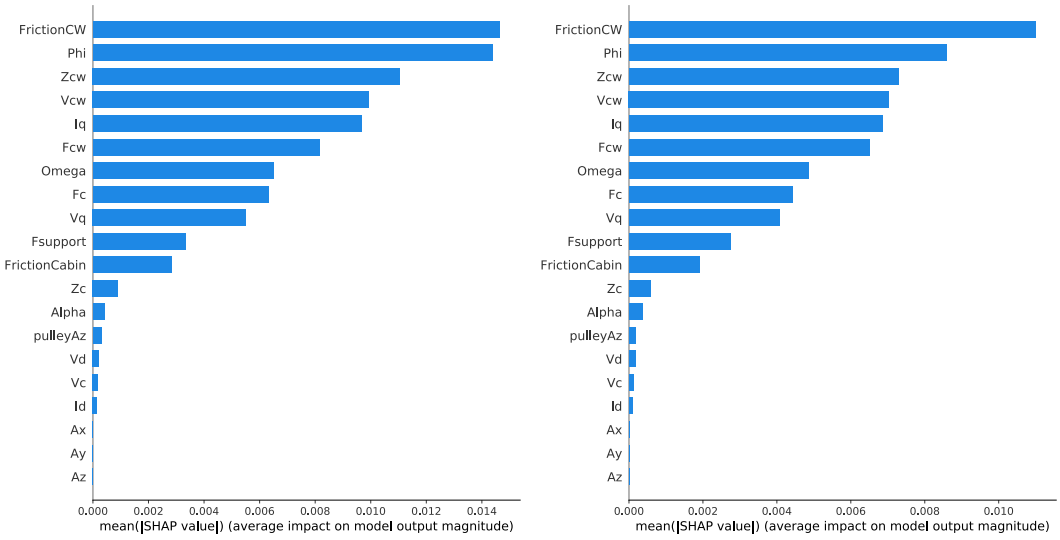
Figure 5.6: Model's interpretability by means of the attention mechanism using one recurrent layer. The interpretability of three journeys containing different anomalies (no point anomalies) is depicted.

5.4.3 Interpreting Predictions with the SHAP

In this experimentation, the ability of the SHAP to identify the sensors that most influences the model's output is evaluated. As in the previous section, the interpretability of point anomalies were measured separately from the rest since the behavior in both cases is different. Moreover, the same journeys as in the previous section were used to facilitate the comparison. In this way, PAJ_1, PAJ_2, and PAJ_3 were used as the journeys containing point anomalies, whereas AJ_{L_q}, AJ_{λ}, and AJ_{GS} were used as the journeys containing the other anomalies. Similarly, the attention model with one recurrent layer was used as it obtained better interpretability results. For the sake of reducing the number of experimentations, only the results obtained with the LSTM layer are shown as far as its interpretations are similar to the ones obtained with the GRU. As mentioned, both the *DeepExplainer* and the *GradientExplainer* were evaluated.

First, the differences between both explainers are assessed. As the interpretability is regarded, Figure 5.7 shows the feature importance obtained by means of the two

explainers in one of the journeys (AJ_x) in which a malfunction in the guiding system was simulated (it is a representative figure of all the validation dataset in terms of difference between both explainers). Recall that the sensors that might be affected as a consequence of this type of anomaly are Ax, Ay, Az, Fc, $FrictionCW$, $FrictionCabin$, $Fsupport$, and Iq. As it can be observed, both explainers identified the $FrictionCW$ as the sensor that most influences on the model's output and the rest of the sensors are classified in similar positions, based on their importance. This figure is representative of the rest of the interpretations and thus, we can conclude that both explainers obtain similar interpretability results despite they use different approaches to compute SHAP values. However, the *DeepExplainer* is near to two times faster than the *GradientExplainer* due to their internal functioning and thus, the former method is preferred. In this way, the rest of the experimentation was conducted using the *DeepExplainer* approach.

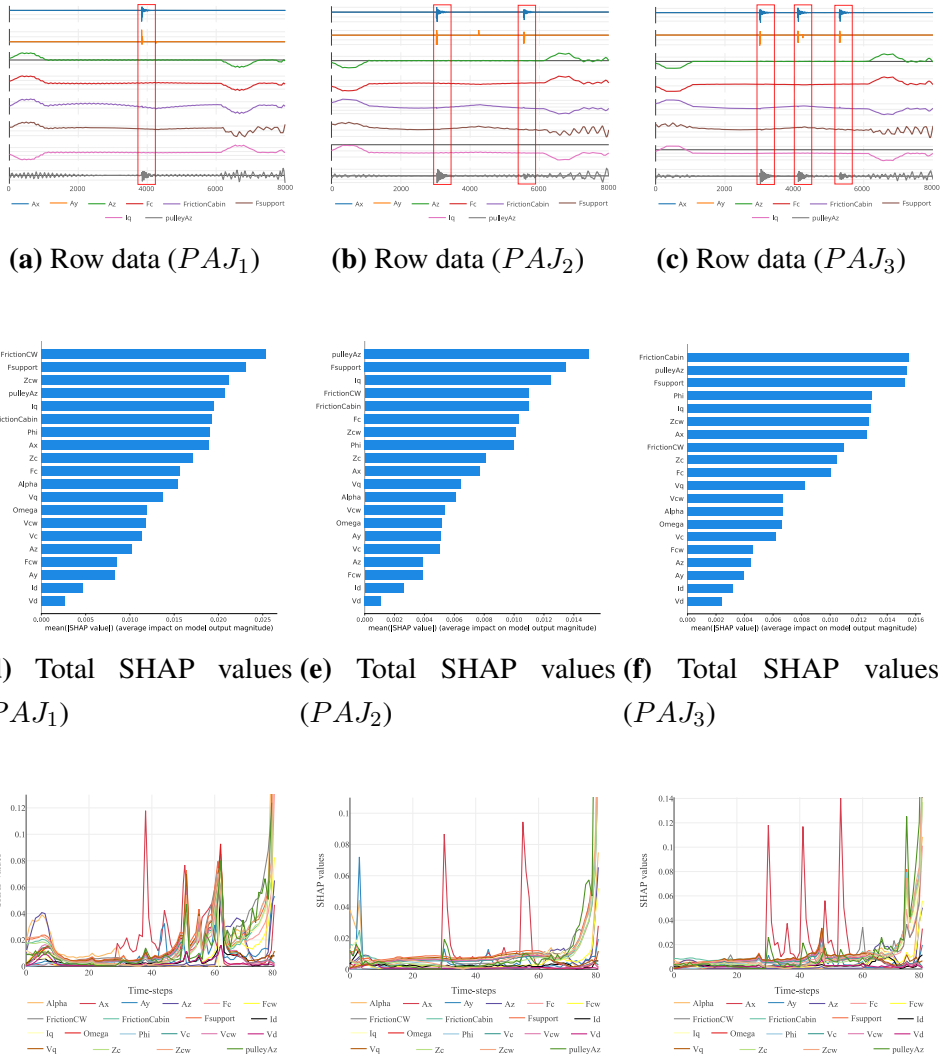


(a) *DeepExplainer* **(b)** *GradientExplainer*

Figure 5.7: Comparison of the interpretability results obtained by *DeepExplainer* and *GradientExplainer* approaches. The interpretations refer to a journey in which a malfunction in the guiding system is simulated (no point anomaly).

As the interpretation involving point anomalies is regarded, Figure 5.8 shows the results obtained by applying the *DeepExplainer* over the three journeys containing point anomalies. In this figure, the first row of images represents the sensor data of

the journeys, the second row refers to the overall SHAP values computed for each sensor, whereas the third row of images refers to the SHAP values computed for each time-step and sensor.



(a) Row data (PAJ_1) **(b)** Row data (PAJ_2) **(c)** Row data (PAJ_3)

(d) Total SHAP values (PAJ_1) **(e)** Total SHAP values (PAJ_2) **(f)** Total SHAP values (PAJ_3)

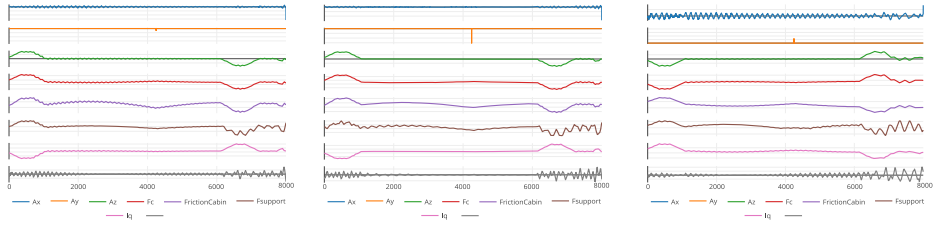
(g) SHAP values by time-step (PAJ_1) **(h)** SHAP values by time-step (PAJ_2) **(i)** SHAP values by time-step (PAJ_3)

Figure 5.8: Interpretability results of the *DeepExplainer*. The interpretability of three journeys containing point anomalies is depicted.

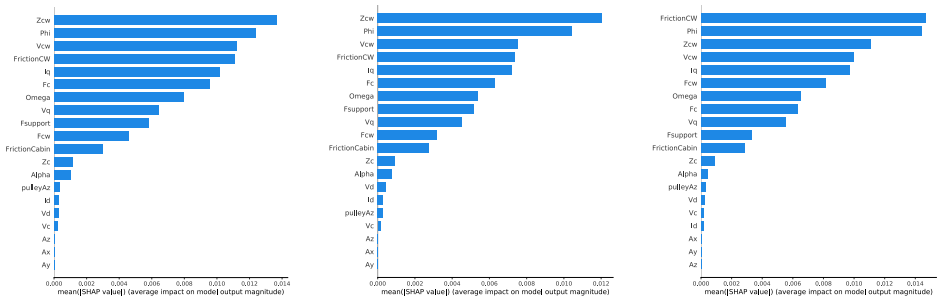
As it can be observed in Figures 5.8a, 5.8b, and 5.8c, the sensors in which the point anomalies can be visualized correspond to Ax, Ay, and $pulleyAz$. However, if the feature importance of each sensor is visualized (Figures 5.8d, 5.8e, and 5.8f), the $pulleyAz$ is the only sensor placed in the top positions, whereas Ax and Ay are placed lower in the ranking. Nonetheless, if the feature importance of each time-step is visualized (Figures 5.8d, 5.8e, and 5.8f), it can be shown that the Ax is the most relevant sensor in the time-steps in which the point anomalies occur. It can also be shown that the $pulleyAz$ is the second most relevant sensor, while the Ay shows no significant relevance. The reason why some of these sensors do not show overall high importance lies in the fact that they might not be relevant during a normal journey, but they become important when a point anomaly occurs. Hence, the values of these sensors in the time-steps in which the point anomalies occur are the ones used by the model to classify the journey as anomalous. In this way, it is shown that the SHAP is not only valid to identify the most relevant sensors for the model's output, but it is also valid to identify in which time-steps point anomalies occur. Moreover, it is capable of identifying all the peaks occurred during the journey, unlike the attention mechanism that might not identify all of them.

Referring to the other anomalies, Figure 5.9 shows their interpretability results obtained after applying the *DeepExplainer*. Following the same distribution, the first row of images represent the sensor data of the journeys, the second row of images refers to the overall SHAP values computed for each sensor, and the third row of images refers to the SHAP values computed for each time-step and sensor. Recall that Vd and Vq should be the sensors affected in AJ_{L_q} and AJ_{λ} (Figures 5.9a and 5.9b), whereas in AJ_{GS} (Figure 5.9c) the sensors affected are the same as in the point anomalies.

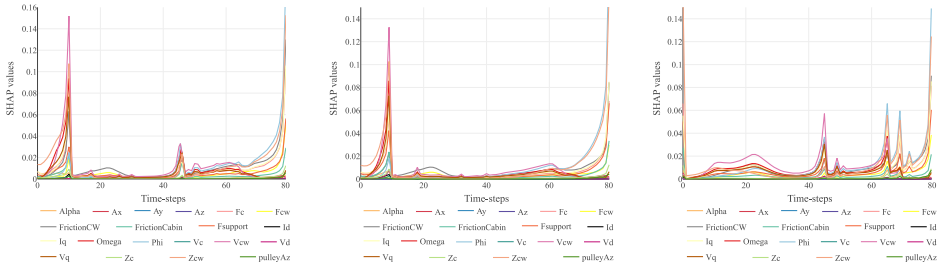
As it can be observed in Figures 5.9d and 5.9e, Vd and Vq sensors are ranked in the medium to bottom part of the classification, which means that according to the SHAP, they have little influence on the model's output. As the attention mechanism did, Figures 5.9g and 5.9h show a peak of focus on the acceleration phase of the journey, although none of the two sensors are placed on top of the feature importance classification, being Vq placed in the fourth position as the best-ranked sensor. Conversely, Figure 5.9f shows that $FrictionCW$ is the sensor that most influenced in the model's output according to the SHAP. Moreover, Iq



(a) Row data (AJ_{L_q}) **(b)** Row data (AJ_{λ}) **(c)** Row data (AJ_{GS})



(d) Total SHAP values (AJ_{L_q}) **(e)** Total SHAP values (AJ_{λ}) **(f)** Total SHAP values (AJ_{GS})



(g) SHAP values by time-step (AJ_{L_q}) **(h)** SHAP values by time-step (AJ_{λ}) **(i)** SHAP values by time-step (AJ_{GS})

Figure 5.9: Interpretability results of the *DeepExplainer*. The interpretability of three journeys containing different anomalies is depicted.

appears in the fifth position while sensors such as Ax, Ay, and Az show to be not relevant. The latter sensors seem to be relevant only when a point anomaly occurs, at least for the analyzed journeys. If the SHAP values are visualized by time-step (Figure 5.9i), it can be shown that there is a little focus on the braking phase of the

journey, although not as accentuated as the attention mechanism did. This behavior adjusts to the real scenario since the friction is present all through the journey.

5.5 Conclusions

This chapter has presented a novel methodology to interpret the predictions of the proposed Multi-head CNN-RNN model and thus, identify the root causes of the anomalies. To this end, the attention layer has been appended to the model to highlight which time-steps most influenced in the prediction, which indicates when the anomaly occurred. On the other hand, the SHAP has been used as an external interpretable model that measures the importance that each sensor has in the final decision, indicating where the anomaly occurred. In this way, we have been able not only to detect anomalies but also to point out where and when they occurred, thus giving to the maintenance operator valuable information to fix the anomaly and prevent it in the future. Despite we have adapted the SHAP to fit the Multi-head CNN-RNN model, the combination of the SHAP and the attention can be applied to any DL model, although it should be noted that the attention layer is only valid for recurrent layers.

The experimentation has shown how the attention layer is capable of successfully detecting point anomalies as it has highlighted the most relevant time-steps of the time series, which match with the time span in which the anomalies occurred. Is must be noted that it only accentuates the most important time-steps used to classify the journey, which may not match with all the point anomalies that actually occurred. In this way, the attention mechanism does not necessarily highlight all the point anomalies. As the remaining anomalies are regarded, the attention layer is not that useful because the anomalies appear throughout the entire time series. Nevertheless, it often highlights the initial and final time-steps, which refer to the acceleration and braking phases of the elevator that are the most critical, thus indicating that the anomalies fall within these phases. It is worth pointing out that the attention layer does impact on the performance of the model and that it achieves up to 2% of improvement with respect to the base Multi-head CNN-RNN, using one or two recurrent layers. However, the attention layer has achieved better interpretability results with one recurrent layer rather than with two, whereas the

overall performance is similar. Therefore, the architecture containing one recurrent layer has used as the best model.

On the other hand, the SHAP has shown generally good interpretability results at the time of identifying the most relevant sensors in the anomalies involving malfunctions on the guiding system. More specifically, it has achieved promising results at interpreting point anomalies as it has been able to identify the sensors that caused them, However, it has failed at interpreting anomalies involving a reduction in the electric machine's inductance (L_q) or the permanent magnets generated flux (λ). These poor results might be due to the specific architecture of the Multi-head CNN-RNN since the recurrent side has a feature map for each sensor as input data rather than a single variable for each sensor. This fact might hinder the process of identifying the most relevant input variables for each model's output. Hence, there is still room for improvement and further research must be done in this context. Hence, it would be a good line of research for the future, particularly for interpreting deeper DL architectures.

Besides, the SHAP has demonstrated that as the attention mechanism does, it is also capable of identifying the most relevant time-steps and thus, it can point out when point anomalies occur or in which part of the journey has the model paid more attention. In fact, it showed to be more precise at identifying the time-steps in which point anomalies occur. Therefore, the SHAP could only be used to accomplish the task of identifying when and where anomalies occur. Nevertheless, the use of the attention mechanism is recommended because it improves the model's performance and it can then be complementary to the SHAP.

As mentioned, this work has been validated using an expert validation methodology in which an expert has ensured that the interpretations matched the real root causes of the anomalies. Nevertheless, this is a manual and time-consuming task. Moreover, the interpretability of DL based models is becoming increasingly popular in several domains and thus, we have seen the need for designing and implementing a methodology to objectively validate the interpretations, rather than relying on an expert. This would be another interesting line of research for the future.

The intelligence consists not only in the knowledge but also in the skill to apply the knowledge into practice.

Aristotle

6

Large-Scale Monitoring System for Anomaly Detection

Previous chapters have shown how to generate a ML model to detect and to diagnose anomalies. However, this model must be deployed in production to provide real-time services to a wide range of industrial machines. Therefore, this chapter presents the Big Data architecture to monitor the status of industrial systems in real-time. This architecture is used to deploy in the cloud the anomaly detection and diagnosis models presented in the previous chapters in other to detect anomalies in real-time. The motivation of this contribution is first exposed (Section 6.1). Afterward, the components of the architecture and the flow of data followed from data gathering to data processing for detecting and diagnosing anomalies is described (Section 6.2). This contribution is validated against the industrial case study considered in this thesis, in which increasing volumes of data scenarios are simulated to validate the performance and scalability o the proposed Big Data architecture. Thus, the experimental framework is first defined (Section 6.3) and then, the obtained results are discussed (Section 6.4). Finally, the conclusions and future work are

exposed (Section 6.5). The content of this chapter is partially published in (Canizo et al. 2019c) and (Campos et al. 2018).

6.1 Motivation

An effective anomaly detection system is very important when monitoring the condition of industrial systems within the Industry 4.0. However, when it comes to real-time monitoring the condition of many industrial systems, the anomaly detection system might be useless if it is not deployed into the cloud and fed with the data coming from the CPSs, in order to early detect anomalies. Moreover, the information obtained regarding the condition of the CPSs must also be effectively served to the user for decision-making. Hence, there is a need for designing and implementing an efficient monitoring system that covers all the stages of the Industry 4.0, from data gathering to data processing in the cloud and the posterior data visualization.

However, the rapid growth and widespread use of information technologies, from individual sensors to cloud computing, has led to an increase in the volume of data that needs to be processed. Considering that CPSs can be monitored by multiple sensors and that they often operate 24/7, the generated data volume is too large to be processed with traditional technologies (Atat et al. 2018), thus causing delays at the time of detecting anomalies. This can be critical for decision-making processes, since obtaining the correct information at the correct moment is a key issue (Faiz and Edirisinghe 2009). Late detection of a fault can also be critical for an industrial machine and, consequently, for productivity.

As the data volume increases, the monitoring system must be horizontally scaled up to add more computational resources to keep ingesting, processing and storing the data in a fast manner. However, other issues can arise as more computational nodes are added to the existing server to spread the load across them, such as data partitioning (Xun et al. 2017) or the management of the computational resources, among others. Hence, scalability is one of the main challenges to these systems. On the other hand, fault-tolerance is another key aspect since in the industrial scenario it is common that errors such as network connectivity loses arise. Hence, the monitoring system must be capable of recovering itself if an error occurs. In this context, Big Data frameworks and cloud computing are particularly important since

they provide fast, scalable and fault-tolerant data processing capabilities for CPSs (Tao and Qi 2019).

Thus, the motivation of this work is to advance in the design of Big Data solutions that cover all the stages of the Industry 4.0. In this way, a real-time large-scale monitoring of CPSs can be conducted by providing the mechanisms to deploy the anomaly detection models into production.

6.2 Big Data Architecture for CPS Monitoring

This section details the proposed Big Data architecture for CPS real-time monitoring. First, the requirements and needs of the monitoring system are presented (Section 6.2.1). Then, the design of the proposed architecture to meet the exposed requirements is detailed (Section 6.2.2). Next, the management of the cloud and the used Big Data frameworks to make the monitoring system fast, scalable, and fault-tolerant are described (Section 6.2.3). Afterward, the functioning of the architecture to detect anomalies in real-time is detailed (Section 6.2.4). It is noteworthy that the proposed architecture is generic and thus some modifications are required to apply it to a specific domain, which are also described (Section 6.2.5).

6.2.1 Monitoring System Requirements

The main goal of this architecture is to process in real-time the data coming from industrial machines to monitor their operational state, providing actualized key information for the decision-making. Nonetheless, processing the large volumes of data that are continuously generated by these machines requires a Big Data infrastructure capable of dealing with these workloads. At this stage, several technological requirements arise, which are present throughout the entire vertical of the infrastructure, from local data collection, through data processing in the cloud, to the visualization of the state of the machines by the end-user (Niggemann et al. 2015). These requirements can be categorized into four main issues, which are briefly described below:

- **Data acquisition:** a system capable of gathering data from the industrial machines and sending it to the cloud is required. As far as some industrial

scenarios suffer from reduced connectivity or computational power, this requires a system that can gather data and sent it within these conditions. Furthermore, the cloud side must be able to manage thousands of messages per second without forming a bottleneck. Similarly, it must also be scalable to be able to handle data volume increments. Finally, the systems must be fault-tolerant to prevent data loss even if network issues arise. Otherwise, the monitoring cannot be properly carried out without all the relevant data.

- **Data processing:** a data processing engine that is capable of processing the streams of data coming from the industrial machines is required. Companies are not limited only to the processing of streaming data. In fact, they process their data periodically to perform advanced analytics to gain greater knowledge for decision-making. Therefore, the data processing system must allow both streaming and batch processing. To detect anomalies as early as possible and to not form bottlenecks, it must also be fast in terms of processing received data besides being scalable to manage increasing data volumes, and fault-tolerant to automatically recover from unexpected failures. As a key requirement, the data processing engine must be capable of connecting to existing ML frameworks to feed processed data into the deployed anomaly detection model.

- **Data persistence:** there is a need to store in the cloud all the raw data generated by the industrial machines. The database must be able to store huge volumes of data and thus, it must facilitate scaling its capacity as more data is saved. Due to the large volumes of data that must be continuously stored, it must have a high throughput in other not to form a bottleneck and collapse the monitoring systems. This database must provide an efficient search engine to query it for advanced analytics purposes or end-users' customized queries. Considering the large volumes of data stored in the database, the results of a given query must be provided without excessive delays. In addition to raw data, other static data must also be stored, such as information related to the industrial machines or to the company. Unlike raw data, this type of information is not expected to increase its volume significantly or to be

updated frequently. Nevertheless, this data must be remotely accessible at any time.

- **Data serving:** a system that provides services to query/push information from/to a user interface is required to easily check in real-time the operational status of the industrial machines. This system must provide mechanisms to handle immediate information (i.e., anomaly alerts or current machine status), and medium to long-term information (i.e., advanced analytics). In other words, it manages the connections between the cloud and the user interface.

All these requirements have been considered at the time of designing the proposed Big Data architecture.

6.2.2 Design of the Architecture

With previously described requirements in mind, the proposed Big Data architecture is divided into three main blocks, which correspond to a local data acquisition system (LDAS), a cloud platform, and a user interface. Figure 6.1 depicts the general overview of the architecture.

From a general perspective, the data flow of this architecture first consists in gathering the data from the machines and sending them to the cloud. Once in the cloud, data is 1) processed in real-time for detecting and diagnosing anomalies, 2) persisted in the database, and 3) processed periodically for conducting advanced analytics. If an anomaly is detected during the real-time processing, an alert is sent to the front-end so that the end-user is informed. Following, the LDAS, the cloud platform, and the front-end are analyzed in-depth (Sections 6.2.2.1, 6.2.2.2, and 6.2.2.3, respectively).

6.2.2.1 Local Data Acquisition System

The LDAS is part of the data acquisition system, concretely, the local side. The other part is located within the cloud platform. It is physically located in the same place as the industrial equipment, that is, it is deployed in the servers of the company (on-premise). This system is responsible for gathering data generated by the industrial machines and then sending it to the cloud. It is composed of a database

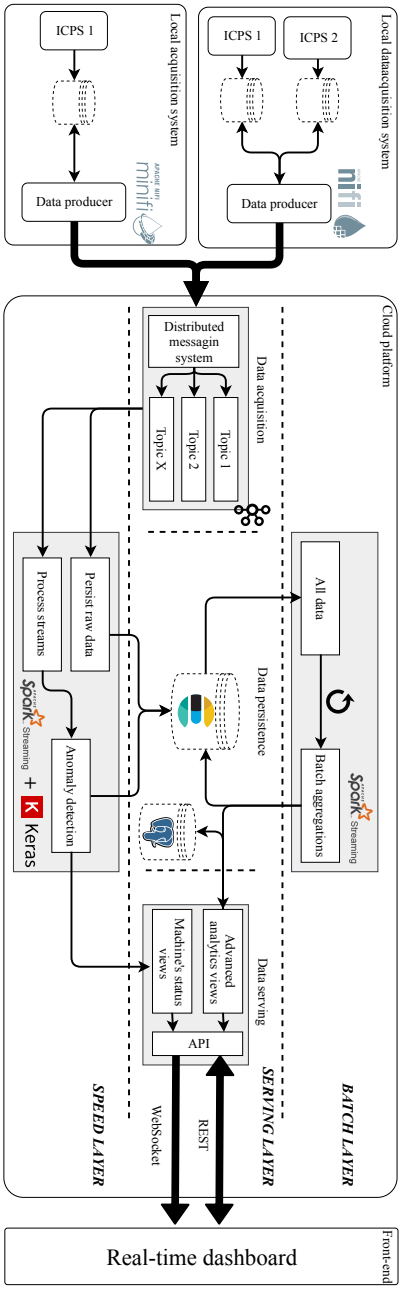


Figure 6.1: Architecture of the proposed CPS real-time monitoring system.

where the generated data is stored and of a data producer that pushes it to the cloud. Considering that this system has to be close to the machines, a different LDAS is required for each factory, building, or geographical location. Consequently, multiple LDASs might exist.

To this purpose, a hard drive is used to store the data generated by industrial machines. Then, Apache NiFi[22] was adopted to read the data stored in the hard drive and to send it to the cloud. NiFi is a real-time integrated data logistics and simple event processing platform that automates the movement of data between disparate data sources and systems, making data ingestion fast, easy, and secure. Also, NiFi offers a reduced version of it, called NiFi MiNiFi,[23] which is smaller and requires lower resource consumption. Thus, it is especially suited for scenarios in which the computational capacity is reduced. In this way, all the requirements concerning the data acquisition on the local side are met.

6.2.2.2 **Cloud Platform**

The cloud platform is the core block of the architecture since it is in charge of managing, processing, persisting, and serving all the data sent to the cloud. To meet all the previously described requirements, an implementation of the Lambda architecture (Darwish and Abu Bakar 2018) is used, which is composed of the following three layers:

- **Speed layer:** includes real-time data processing services, that is, it is responsible for processing in real-time the data sent by the LDASs and thus, for detecting and diagnosing anomalies. Moreover, it is in charge of persisting the data into the database.
- **Batch layer:** provides batch processing services, which means that the data is processed periodically with long time intervals. This is responsible for performing advanced analytics by aggregating historical data stored in the database. To implement both layers, SS was adopted to process the data as it meets all the requirements in terms of fast, scalable, and fault-tolerant data processing besides enabling the connection with customized ML models.

[22] https://nifi.apache.org/
[23] https://nifi.apache.org/minifi/index.html

In this way, micro-batch processing is used for the speed layer while batch processing with a long batch interval is used for the batch layer.

- **Serving layer:** comprises internally three sublayers which correspond to the data acquisition, persistence, and serving, as shown in Figure 4.1. They are summarized below:

  - **Serving sublayer:** is the layer that connects all the blocks of the architecture and the different layers within the cloud platform. At this point, the data acquisition sublayer holds the other side of the data acquisition systems, concretely, the cloud side. It is responsible for receiving all the data sent by the LDASs and thus, it can be seen as the gateway of the cloud platform. To this end, Kafka was adopted since it is a distributed messaging system that meets all the requirements in terms of fast, scalable, and fault-tolerant data acquisition.
  - **Persistence sublayer:** contains the databases to store all the information corresponding to the monitoring systems. Taking into account the requirements, there are two distinguishable types of data: continuous streams of raw data, and static data. Both types are of different nature and they have, therefore, different needs. Since no database engine suits the requirements of both types of data, two databases are used. Thus, a NoSQL database engine is used to store streaming data and a relational database is used to store static data. The reason why a NoSQL database has been selected is that they offer higher throughput and scalability than relational databases. To this end, Elasticsearch[24] was adopted as it is a distributed, document-oriented, RESTful search and analytics engine that is capable of persisting data and fulfilling the established requirements (Tong Zachary 2015). Moreover, Elasticsearch is also a mature and robust technology that has been successfully adopted in other domains (Andreassen et al. 2015; Balalaie et al. 2016; D. Chen et al. 2017; Langi et al. 2015). As an alternative, InfluxDB is also suitable although it was discarded as it must be paid in case more than one node

[24]https://www.elastic.co/products/elasticsearch

is used. On the other hand, PostgreSQL was adopted as a relational database as it is a mature and powerful storage engine. In this way, all the requirements corresponding to the data persistence are satisfied.

- **Serving sublayer:** includes the services regarding the connection between the cloud platform and the front-end to transfer data from one block to the other, in both directions. To implement these services, two components are used: a generic REST API to query the database from the dashboard to request data regarding advanced analytics or historical data, and a WebSocket that enables direct messages to be sent to the dashboard, which is used to send anomaly alerts. These are responsible for data serving, which is located in the cloud within the serving layer. To provide fault-tolerant and scalability capabilities to the dashboard, a micro-service solution based on the 12-factor app approach[25] was adopted. Therefore, the requirements regarding the data serving are satisfied.

In this way, a Big Data architecture capable of processing the data coming from the CPSs in a fast, scalable, and fault-tolerant manner is achieved.

6.2.2.3 **Front-End**

The front-end is the visual block of the architecture. Although all the logic of the monitoring systems lies on the cloud platform, the user interface is equally important as it shows at a glance all the current information involving the status of the monitored industrial machines. Figure 6.2 shows a screen-shot of the dashboard. The front-end is not only limited to visualizing the status of the machines but it also allows to query the database for advanced analytics or for making customized queries to acquire more knowledge about the performance of the machines. Hence, the front-end is a key component at the time of decision-making as it provides a general view of all the systems of a company. Note that the front-end is also deployed within the cloud platform.

[25] https://12factor.net/

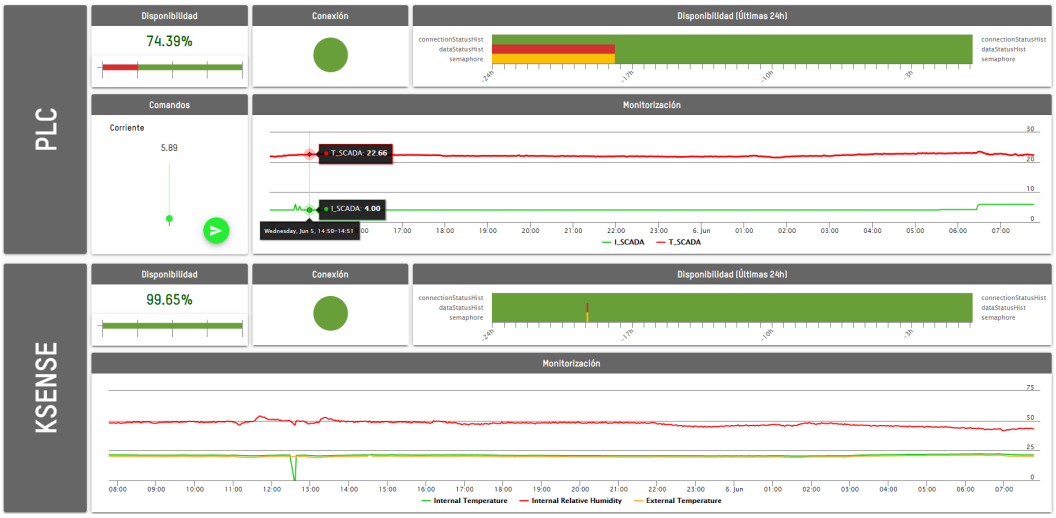


Figure 6.2: Dashboard of the monitoring systems.

6.2.3 Cloud Management

As mentioned, almost all the components of the proposed Big Data architecture run in the cloud. In this context, the cloud refers to an infrastructure where multiple interconnected computational nodes work together and thus, they can be viewed as a single computer. Therefore, the used frameworks make use of the computational resources provided by the cluster to run. On the other hand, the used frameworks require more than one working node running at the same time to guaranty that they keep running even if an unexpected error occurs. In a typical scenario, there is a master working node and one or more slave nodes as a replica. However, managing how the resources of the cluster are used and how the different nodes are synchronized within the cluster can be a complicated task besides being hard to maintain.

In other to facilitate the management of the computational resources, a cloud manager such as Apache Mesos[26] is used. Among other computational resources, Mesos abstracts CPU, memory, and storage away from the machines (physical or virtual), enabling fault-tolerant and elastic distributed systems (Hindman et al. 2011). Mesos is in charge of dynamically managing the resources used by the frameworks within the cloud, specifying where and how they have to be executed. In the same

[26]https://mesos.apache.org/

way, it allows to dynamically add or reduce the resources that a framework can use at run-time, depending on the current needs.

The design of the cloud management is shown in Figure 6.3. Mesos has three master nodes and three agent nodes. The former are responsible for managing the cloud and the latter for executing the scheduled tasks. Only one of the master nodes is active (the leader) while the others are in standby mode as a replica in case some of the leaders fail.

Mesos, Elasticsearch, and Kafka have a master node and two slave nodes. This allows a threefold replication of the services. Therefore, if an error were to occur in one of the nodes, then the data would still be available in the remaining live nodes. Furthermore, this avoids the single point of failure problem (Ranjithprabhu and Sasirega 2014), that is, there is no point at which if something fails, the entire system stops working.

However, when some of the working nodes fail, neither the cloud manager nor the frameworks are capable of synchronizing themselves to select a new leader or master node. To this end, Apache Zookeeper[27] is used. Zookeeper is a centralized service that is used to maintain the configuration information and naming, to provide distributed synchronization and group services (Hunt et al. 2010). Therefore, if any master node fails, then Zookeeper would be responsible for selecting a new master node. Thus, the application could continue working properly. In this architecture, Zookeeper has three working nodes since the selection of new master nodes is made in quorum by all the nodes, and three is the minimum number to form a quorum.

As described, NiFi is deployed locally. Thus, if network problems were to arise, the gathering of new data would continue and submission of the data to the cloud would take place when the network is able to recover its normal behavior. Despite not being deployed on the cloud, NiFi can restart itself at the point it had reached before a failure occurred. To do this, it uses a checkpointing mechanism to ensure that no events are lost. Moreover, NiFi itself is a scalable framework. This guarantees scalability and fault-tolerant services.

[27] https://zookeeper.apache.org/

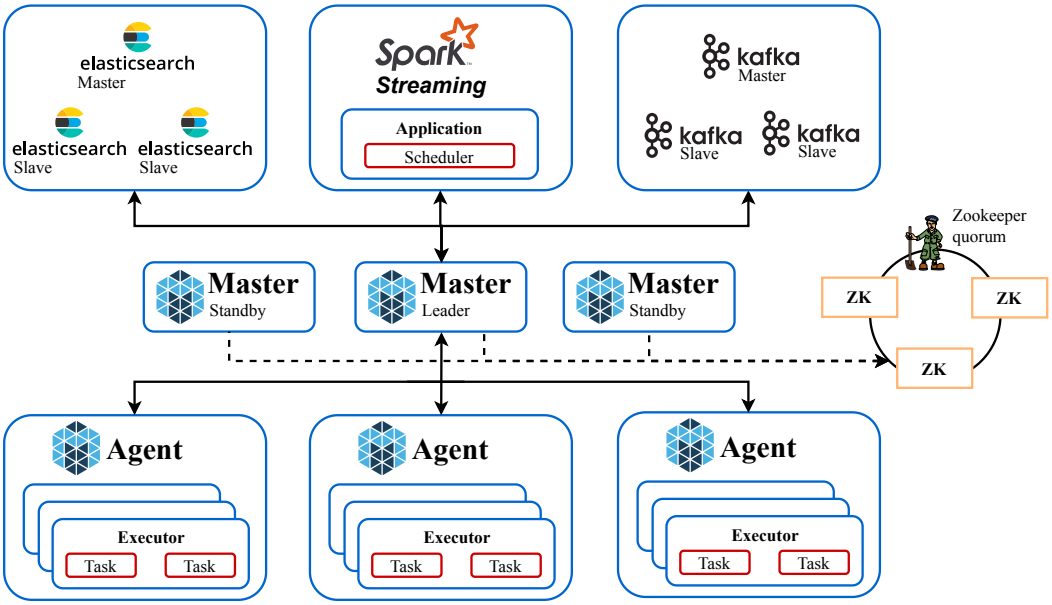


Figure 6.3: Configuration of the cloud platform.

6.2.4 Real-Time Anomaly Detection

This section describes the data-flow followed to perform the real-time monitoring of the industrial systems and to detect anomalies. Figure 6.4 depicts this followed procedure. This process starts when data is gathered from industrial machines through the LDAS. Internally, the LDAS gathers data from each industrial machine and stores it in a hard drive. Concurrently, the data producer periodically queries the local storage to check whether new data is available. If so, the LDAS publishes new data within Kafka topics. Note that a different topic is used for each machine. Therefore, each LDAS can send data to one or many topics depending on how many machines it manages.

Once the local side of the data acquisition system publishes data on Kafka, the real-time processing engine (SS) subscribes to the corresponding topics to read the messages. At this point, the tasks of Apache Spark are twofold and are executed concurrently: 1) to persist the received data into Elasticsearch, and 2) to process data and apply the corresponding anomaly detection model to detect anomalies. To this end, two DStreams are used for each topic to execute both tasks in parallel. It is noteworthy that if the data volume increases in the future, then the number of

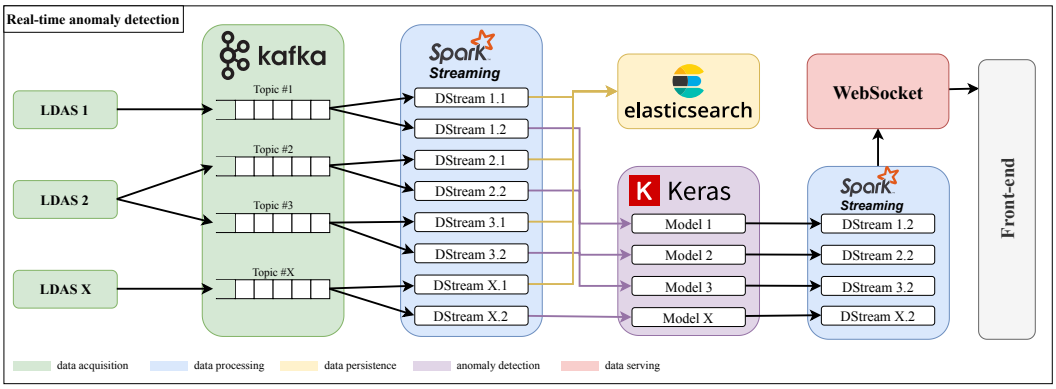


Figure 6.4: Process followed in the proposed architecture to detect anomalies in real-time.

partitions per topic can be increased. Similarly, the number of DStreams can also be increased proportionally to the number of partitions. Consequently, the data ingestion throughput can be increased.

After processing the data, the anomaly detection model is applied to detect anomalies. However, as multiple measurements are typically required as input parameters of the anomaly detection model, it could be the case that not all of them are received within a given batch. This may occur due to connection losses or delays, or because the data to be sent is too large and may take some time to arrive. It is, therefore, necessary to wait until all data is received. However, Spark does not support the persistence of values between batches as a default. Consequently, a stateful method was adopted. In this way, previous data can keep into memory within each batch. This method takes into consideration a *key*, a *value*, and a *state* as input parameters. The *key* is a unique identifier for classifying the data; the *value* is the data received within a batch, classified by *key*, and the *state* is the parameter through which data can be persisted in memory between batches.

Algorithm 1 shows the method followed. As it can be observed, data received within a batch is first grouped by *key*. Here, the *value* is the data received within a batch grouped by *key*, while the *state* refers to an array whose size is determined by the amount of data required. Thus, this array is filled each time that a batch is processed. If not all the data is received, then it returns to receive new data. When all the data is received, the *value* of the stateful method is returned and the anomaly

detection model is then applied. At this point, the state is removed to free memory space, since this data is not used again in this context. The result of the model is then analyzed and an alert is sent to the front-end in case an anomaly is detected.

Algorithm 1 Stateful algorithm for anomaly detection.

```
1:  receive data from kafka;
2:  group received data by key;                          ▷ key=machineNumber
3:  execute statefulMethod;
4:  if statefulMethod returns value then
5:      isAnomaly = applyAnomalyDetectionModel(value)
6:      if statefulMethod == True then
7:          send alert to the front-end;
8:      end if
9:  end if
10:
11: function STATEFULMETHOD(key, value, state)
12:     arrayData = getPreviousState(state);
13:     fill arrayData with new received data;
14:     check if all data is received;
15:     if all data received then
16:         remove state;
17:         return arrayData;
18:     else
19:         update the current state;
20:     end if
21: end function
```

To correct the maintenance process, alerts are also sent to the Technical Assistance department where a maintenance assistant analyzes the failure and plans the corresponding corrective actions to be made, if needed. If the failure can be remotely fixed, then the assistant will start the process using a Virtual Private Network (VPN). Otherwise, the assistant will launch a maintenance order, in which the assistant will have to physically fix the fault.

6.2.5 Implementation of the Architecture on a Specific Domain

This solution can be implemented in any industrial domain as it is composed of generic frameworks. However, some modifications must be made to implement this architecture in a specific domain, since each one has its requirements and needs.

At an architectural level, the LDAS is the only component that might be changed because each industrial system is different and thus, an ad-hoc system must be implemented to gather the data generated by it and to store this data into the local storage. Then, depending on the available computational resources and the quality of the connectivity, it is necessary to choose between using NiFi or MiNiFi, or both. Moreover, the number of LDASs has to be defined to be capable of sending the data of all the machines of a given company. The frequency with which data is sent to the cloud must also be established depending on the specifications and requirements of the domain. Using the industrial case study of this thesis as an example, a LDAS should be installed for each building, where each LDAS would gather data of one or more elevators, depending on the number of elevators within the building. Regarding the selection of the technology to send data to the cloud, both NiFi and its reduced version could be used depending on the resources of each building. For instance, a normal building with private flats, where computational resources are generally reduced, MiNiFi would be used, whereas a building of a company could use NiFi instead since greater computing resources are expected to be available. As for the frequency of data sending, data would be sent after each elevator journey. The rest of the architecture, that is, the cloud platform and the front-end should be equal for any domain.

As the data flow of the architecture is regarded, although the used frameworks and the process followed to monitor the industrial machines is the same for all the domains, the data structure is also ad-hoc for each specific domain. That is, the data required to monitor the industrial systems is different for each domain. Besides, in the cloud platform, the number of topics used to classify the received data, and the number of DStreams used by SS must also be defined depending on the current requirements and the expected volume of data to be processed.

6.3 Experimental Framework

This section describes the configuration and properties related to the experimentation followed to validate the proposed Big Data architecture. Note that this experimentation does only cover the performance of the architecture rather than the anomaly detection model. Although the architecture has many components, this experimentation only covers the components involving the anomaly detection task, which correspond to the data acquisition, processing, and persistence components. First, a criterion is defined to objectively define the objective of the experimentation (Section 6.3.1). Afterward, the evaluation metrics (Section 6.3.2), and the conducted scalability tests are described (Section 6.3.3).

6.3.1 Evaluation Criterion

To evaluate the performance of the proposed Big Data architecture in terms of data processing speed and scalability, the following criterion is defined to determine whether it successfully passes the established tests:

> "*The developed real-time monitoring system is capable of detecting anomalies by processing data generated by industrial machines in a stable way, when the data volume is equal to the data generated in the current scenario and under conditions considered for future scenarios*."

Since the monitoring system processes data in batch, stable means that it requires less time to process batch data than the duration of a batch (i.e., five seconds). Hence, as the data coming from the elevator increases in volume, the monitoring system would have to maintain stable by scaling its computational resources and, consequently, processing more data within the same period of time.

6.3.2 Evaluation Metrics

In this section, the metrics used to measure the performance and the scalability of the developed real-time monitoring system are analyzed. Different parameters provided by SS are used to validate the criterion:

- **Input rate:** the number of messages received per second.
- **Scheduling delay:** the time for which a batch waits in a queue until the processing of previous batches is finished. For example, assuming that the monitoring agent reads from the source with a frequency of five seconds and that the given batch took seven seconds to compute, then means the agent is two seconds behind $(7 - 5 = 2)$, thus making the scheduling delay two seconds long.
- **Processing time:** the time to process each batch of data.
- **Delay time:** the time spent to complete all the streaming jobs of a batch. This is calculated by summing the scheduling delay and the processing time. Following the same example, if the agent is already two seconds behind and the processing of the next batch takes a further seven seconds, then the data will be processed with a total delay of nine seconds $(2+7 = 9)$. These metrics are calculated for each batch. Therefore, since this parameter is calculated from the previous ones, it is used to measure the success of the test.

6.3.3 Scalability Tests

To verify the defined criterion, three tests were conducted using the industrial case study concerning the elevators. The goal of the experimentation is to assess the scalability and performance of the architecture as the data volume of data to be processed increases in volume. Therefore, each test varied from the others in terms of input rate. In addition, the computational resources provided were also modified in the last test. The details of each test are as follows:

- **Realistic test:** currently, the number of monitored elevators is low, meaning that the data volume generated is quite low. Consequently, 40 messages per second were set as input rate, which was similar to the current real scenario.
- **High input rate test:** as far as it was expected to increase the number of monitored elevators and, as a consequence, the volume of data to be processed, the goal of this test was to find the maximum input rate supported by the proposed architecture given the computational resources used in the first test.

Thus, the data volume was increased to bring the monitoring system to its limit. To this end, the input rate was progressively increased until the processing time reached the duration of a batch and the total delay started to increase. This occurred when the input rate was set to $4,000$ messages per second.

- **Scalability test:** to evaluate whether the proposed architecture can scale its performance to process more data by adding more computational resources, the same input rate as used as for the high input rate test was established. However, the computational resources of the application were increased for this test. Therefore, the monitoring system would have to be able to process this volume of data in a stable way.

As proof of concept, all the tests were executed using a single Kafka topic to send the data to the cloud. On the other hand, two DStreams were used, one to store the data in the database, and the other to process the data and detect anomalies. Defined by domain experts, the total delay must be lower than a threshold of 20 s for realistic and scalability tests to be considered as successful. The duration of each test was two days, that is, $34,560$ batches. Table 6.1 shows the configuration used for executing the tests. More specifically, the resources used in the drivers and the executors.

	Driver		Executors		
	# cores	Memory	Amount	# cores	Memory
Realistic & high input rate	2	4 GB	1	4	11 GB
Scalability	3	8 GB	3	6	11 GB

Table 6.1: Computational resources used for the experimentation.

6.4 Results and Discussion

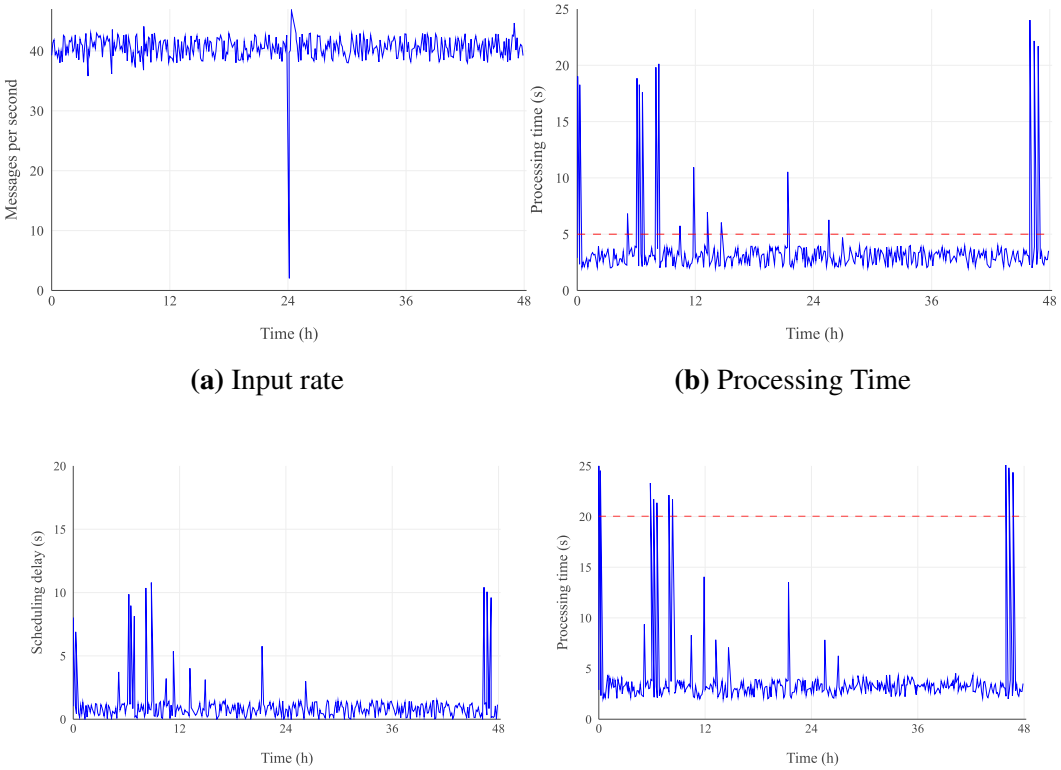
In this section, the results of the experimentation are analyzed and discussed. The aim of this experimentation is to evaluate the performance and scalability of the proposed Big Data architecture. To this end, the performance under a realistic scenario is first measured, then the maximum capacity of the architecture to process the

data coming from the CPSs is shown and finally, the scalability of the architecture is assessed by adding more resources in order to deal with the data volume increments.

The results of the realistic test are shown in Figure 6.5. These results demonstrate that the monitoring system is stable during the computation of the gathered data (Figure 6.5a). It is worth pointing out that the used system to push data to the cloud was not designed for real-time purposes. Therefore, the input rate was not stable throughout time due to system overheads. As shown in Figure 6.5b, the processing time remains lower than the batch period. Moreover, as depicted in Figure 6.5c, the scheduling delay is almost zero, which means that almost no batches are enqueued before being processed. This implies that the data processing is performed in real-time and that the total delay (Figure 6.5d) is made up of processing time. Overall, the monitoring system requires 3.38 s, on average, between gathering the data coming from elevators and providing a result indicating whether an anomaly is detected or not. Taking into account the condition defining success and the obtained results, the realistic test was passed satisfactorily.

Similarly, the results of the high input test are represented in Figure 6.6. These results show that the limit of the monitoring system, for the given computational resources, is 4,000 messages per second, as it can be seen in Figure 6.6a. As shown in Figure 6.6b, the processing time borders on the batch time, as its average is 6.23 s. Each time the processing time exceeds the batch time threshold, this implies a delay time. Moreover, the scheduling delay (Figure 6.6c) increases due to the execution of other data processing tasks, such as persisting data. This combination makes the total delay (Figure 6.6d) too large to be considered as a fast response since detecting an anomaly so late would be critical. It is worth noting that this test was stopped after two hours since the results obtained during this time span showed the upward trend of the total delay. If the test had not been stopped, the delay would have continued to rise with the similar pattern shown in Figure 6.6.

Once the maximum input rate for the given computational resources was known, the scalability test was executed. The results are presented in Figure 6.7. These results confirm the scalability of the developed monitoring system, as it was able to process the volume of data established in the previous test (Figure 6.7a) in a stable way. As shown in Figure 6.7b, the increase of computational resources implied faster data processing than in the high input test. Therefore, almost no batches are



(a) Input rate

(b) Processing Time

(c) Scheduling Delay

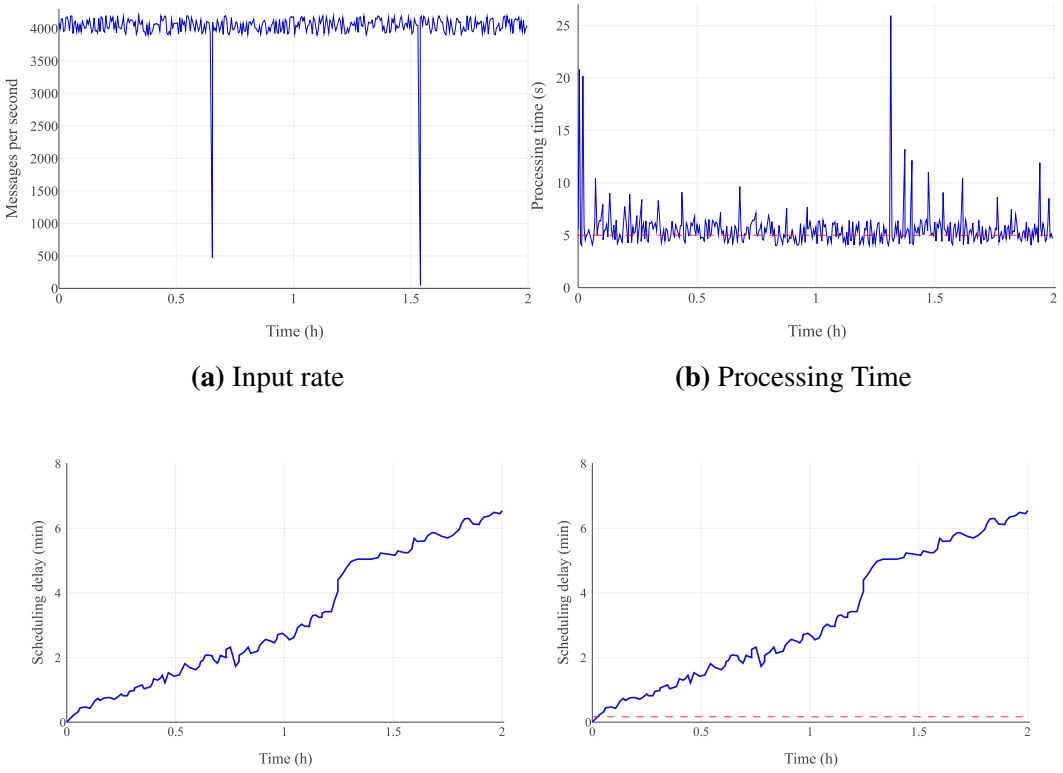
(d) Total Delay

Figure 6.5: Performance results of the realistic test. The red dashed line indicates in b) the batch processing time threshold, and d) the total delay threshold.

enqueued (Figure 6.7c) and, consequently, the total delay is almost equal to the processing time, as shown in Figure 6.7d. Thus, according to the condition defined for success, the scalability test was passed successfully.

Table 6.2 shows a summary of the results obtained in the three tests. As it can be observed, these tests show that the monitoring system satisfies all the requirements of the defined criterion: 1) the system is capable of detecting anomalies, 2) the data processing is stable under the current and future scenarios, as shown in the realistic and the high input rate tests and 3) the monitoring system is scalable, as it can handle future demand for data volume by scaling its computational resources, as shown in the high input rate and the scalability tests.

The implementation of this solution, for this particular use case, has lead to several enhancements in the maintenance service. The combination of a fast, scalable,



(a) Input rate

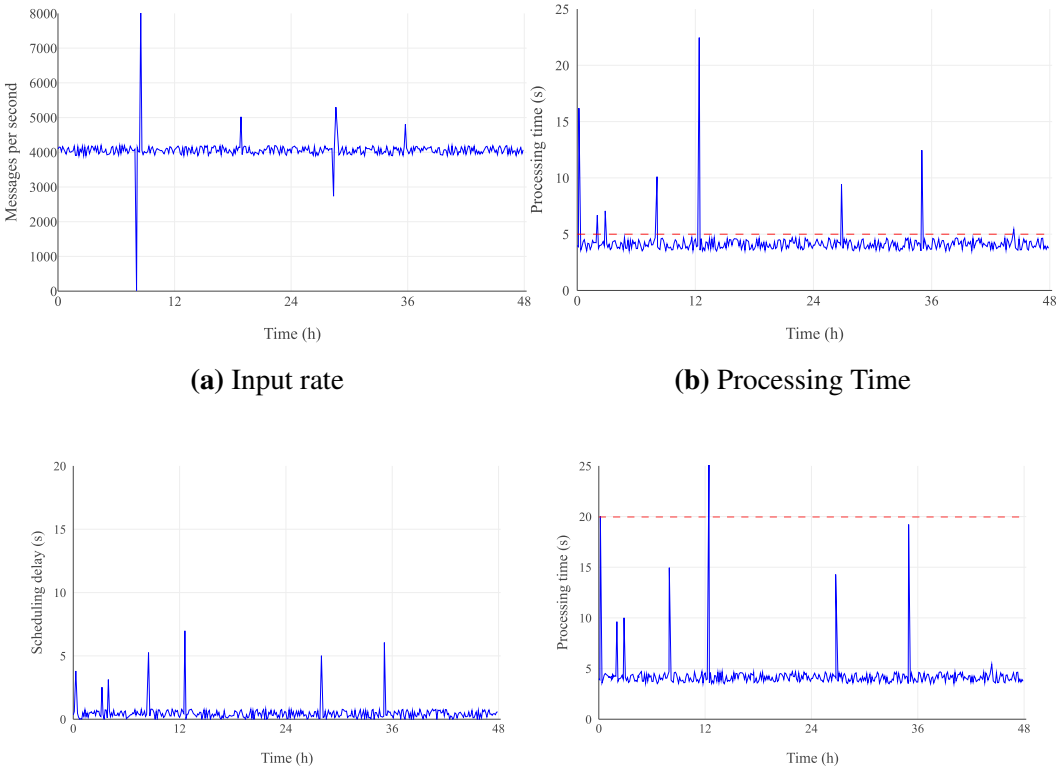
(b) Processing Time

(c) Scheduling Delay

(d) Total Delay

Figure 6.6: Performance results of the high input test. The red dashed line indicates in b) the batch processing time threshold, and d) the total delay threshold.

and fault-tolerant real-time monitoring system with an effective feedback system to manage the anomalies has improved the availability of the elevators and the maintenance operations. However, as it has only been a short time since this solution was implemented, there is an absence of qualitative and quantitative results from an empirical application and/or validation. Therefore, it is difficult to evaluate the potential of the proposed solution with respect to its usability and/or usefulness for industry adoption. This is a general problem when implementing this type of solution in the industry (Khurum and Gorschek 2009). Preliminary studies made by the clients show the adequacy and the correctness of this implementation. Nonetheless, an exhaustive analysis of the monitoring system will be done once the system has been in production long enough to obtain sufficient quantitative data to measure the real gain.



(a) Input rate

(b) Processing Time

(c) Scheduling Delay

(d) Total Delay

Figure 6.7: Performance results of the scalability test. The red dashed line indicates in b) the batch processing time threshold, and d) the total delay threshold.

6.5 Conclusions and Future Work

This chapter has presented a Big Data solution for the fast, scalable, and fault-tolerant monitoring of CPSs which has been validated on a real industrial scenario where several service elevators are monitored. To this extent, the large-scale monitoring systems is composed of 1) a local data acquisition system that gathers data from CPSs and sends it to the cloud, 2) a remote data acquisition system in charge of managing the data sent to the cloud, 3) a stream data processing engine that processes in real-time the received data and feeds it into the developed ML model to detect anomalies, and 4) a dashboard in which the condition of all the CPSs and advanced analytics can be visualized in real-time. All this process being fast, scalable, and fault-tolerant. In this way, the proposed solution has demonstrated the

		Realistic test	High input test	Scalability test
Input Rate	Avg	40 msg/s	4,000 msg/s	4,000 msg/s
Processing time	Min	1.56 s	4.42 s	3.71 s
	Avg	2.68 s	6.23 s	4.28 s
	Max	24.31	25.83 s	23.56 s
Scheduling Delay	Min	0.00 s	4.66 s	0.00 s
	Avg	0.63 s	223.00 s	0.52 s
	Max	31.49 s	380.00 s	4.15 s
Total delay	Min	1.63 s	9.12 s	4.23 s
	Avg	3.38 s	229.00 s	4.69 s
	Max	25.03 s	380.00 s	25.06 s

Table 6.2: Scalability results of the three tests.

potential of Big Data technologies in an industrial scenario where the volumes of data generated are very large, and unexpected failures must be managed without affecting the proper operation of the monitoring system.

The experimental results obtained in the industrial use case has shown that the application exceeds the current needs of the monitoring system since the data processing remains stable for the data volume generated in the current scenario. Although the limit of this application was reached in the high input rate test, its scalability has been demonstrated in the scalability test, since it allows processing this volume (or more) of data to be processed simply by adding more computational resources.

The proposed Big Data solution has been developed using open source and generic Big Data frameworks and thus, it can be deployed in any public or private cloud. Moreover, its architecture is generic, meaning that it can be applied to any industrial domain by conducting minimum modifications in the way in which data is gathered from the industrial machines and in the data structure, since these are domain-specific and they must be adapted to the current scenario.

If you think you made it, you are at the wrong place, never stop.

Chris Pardo

7

Conclusions and future work

This final chapter summarizes the main contributions and future lines resulted from this work and validates the hypotheses defined at the beginning of the thesis in Section 1.3. First, the results obtained and the contributions performed are presented (Section 7.1). Afterward, the future lines of research resulting from this thesis are exposed (Section 7.2).

7.1 Results and Contributions

In this dissertation, a large-scale system for detecting and diagnosing anomalies in heterogeneous multi-time series within the Industry 4.0 domain has been proposed. Concretely, this system uses DL techniques to detect anomalies along with interpretability methods to diagnose them. Finally, the system uses Big Data and cloud computing technologies to deploy the proposed anomaly detection model in a large-scale production scenario.

Therefore, the proposed anomaly detection system covers all the stages of the Industry 4.0, which range from gathering data generated by the CPSs and sending it to the cloud, to processing the data in real-time to detect and diagnose anomalies, and finally to visualizing the extracted information to support the decision-making. To

this extent, the system is composed of three modules, which match the contributions described throughout this dissertation. Their characteristics are summarized below:

- **Anomaly Detection in Heterogeneous Multi-Sensor Systems:** this module is in charge of detecting anomalies in industrial heterogeneous multi-sensor systems by using a DL based approach (Canizo et al. 2019b). The proposed architecture is composed of a convolutional and a recurrent part. The former is responsible for automatically extracting features from raw sensor data, thus removing the need for pre-processing it, whereas the latter is responsible for identifying temporal patterns and for classifying the events. The novelty of the proposed architecture relies on that sensors are processed in independent convolutional heads, which entails the following advantages:
  - Focus the feature extraction on individual sensors rather than mixing the features of all sensors, thus improving this task.
  - Adjust each convolutional head to the specific characteristics of each heterogeneous sensor.
  - Adapt to changing sensor configurations as it facilitates adding, modifying, or removing convolutional heads to the generated model.
- **Interpretation of Deep Learning for Diagnosing Anomalies:** this module is responsible for diagnosing the anomalies detected by the anomaly detection model (Canizo et al. 2019a). To this end, various algorithms are combined to interpret the output of the DL model. The SHAP measures the input variables that most influences the model's output, whereas the attention mechanism highlights the time-steps in which the model has focused to classify the time series. Hence, the anomalies can be diagnosed and valuable information can be provided to the maintenance operator to fix it. Note that interpreting this architecture is only possible due to the fact that the sensors are processed individually and thus, the data of each sensor can be traced throughout the network.
- **Large-Scale Monitoring System for Anomaly Detection:** this is the module in charge of providing the architecture to real-time monitoring the condition of the CPSs in the cloud and detecting anomalies at large-scale (Canizo

et al. 2019c). To this end, the monitoring system is composed of the following components:

- A LDAS that gathers data from CPSs and sends it to the cloud.
- A cloud data acquisition system that manages the data received.
- A real-time data processing engine that processes the coming streams of data and feeds the generated anomaly detection and diagnosis model.
- A data persistence system that stores all the data involving the monitoring processes.
- A real-time dashboard in which the information extracted as well as the anomaly alerts can be visualized for decision-making.

This architecture allows deploying the anomaly detection model to detect and diagnose anomalies in real-time as it provides fast, scalable, and fault-tolerant data processing due to the use of Big Data and cloud computing technologies.

As an additional contribution, the **CloudFMI Distributed Simulation Framework** has been designed and developed, which is freely available for the use of the scientific community. Given a physical model, this framework enables running in parallel multiple simulations of the model with different parameters. For this matter, the framework uses cloud computing technology to provide a fast, scalable, and fault-tolerant processing engine. Hence, this framework facilitates running a large number of simulations in less time.

The three main modules have been tested individually in an industrial scenario in which a service elevator monitored by multiple sensors has been used. The used data was generated by means of a physical model developed by domain experts that mimics the real behavior of the elevator and the CloudFMI Distributed Simulation Framework. Overall, there were 14,000 journeys of which the 20% were anomalous. To test the contribution in more imbalanced scenarios, other four datasets containing 15%, 10%, 5%, and 3% of anomalies were generated. In this way, the anomaly detection system has been extensively evaluated throughout non-parametric tests under different scenarios in which three types of anomalies have been considered: point, context-specific, and collective. Moreover, the anomaly diagnosis module has

been validated using an expert validation approach. Finally, considering the large volumes of data that can be generated in a real scenario, a scalability test has been performed to prove the performance and scalability of the developed monitoring system.

The following conclusions have been drawn from the entire dissertation work, grouped by the three contributions to which they belong.

Anomaly Detection in Heterogeneous Multi-Sensor Systems:

1. DL techniques have been shown well suited for the industrial domain. Concretely, the combination of CNNs and RNNs have demonstrated to be effective at detecting point, context-specific, and collective anomalies in heterogeneous multi-sensors systems.

2. The Multi-head CNN-RNN architecture has proven to be able to operate directly over raw sensor data due to the automatic feature extraction performed by the convolutional side of the architecture, outperforming in this way traditional ML algorithms.

3. Processing each sensor data in an independent way improves the feature extraction of individual sensors, leading to increased performance at detecting anomalies. Moreover, this fact provides flexibility to the CNN-RNN architecture to adjust to the requirements of each sensor data.

4. TL based techniques are valid to boost the training phase of DL based architectures, particularly when the model must be adapted to changing sensor configurations. Furthermore, TL techniques can yield the anomaly detection model in scenarios in which not enough data is available.

Interpretation of Deep Learning for Diagnosing Anomalies:

5. Interpretability methods have shown promising results at explaining how DL models reach a decision in order to diagnose anomalies in multi-sensor systems.

6. The attention mechanism is a valid interpretability method to identify which time spans of the time series the model has focused to reach a decision.

Therefore, it is capable of identifying when anomalies occur. The attention mechanism is particularly effective in scenarios in which specific time-steps are more significant than the others, as is the case of point anomalies.

7. The SHAP is a valid interpretability method to identify the sensors that most influence in the output of the model, thus indicating the sensors causing the anomalies. Furthermore, it has shown to be valid at identifying when anomalies occurred, outperforming in some cases the attention mechanism.

8. The performance of the interpretability methods decreases as the number of stacked layers increases. The reason lies in that more abstract representations of input data is obtained as it flows deeper within the network and thus, it is more difficult to trace the functioning of the network. Therefore, deeper DL architectures are less interpretable.

Large-Scale Monitoring System for Anomaly Detection:

9. Big Data and cloud computing technologies have demonstrated to be suitable for ingesting, processing, and persisting the large volumes of data generated by CPSs in a fast, scalable, and fault-tolerant way. Furthermore, they enable deploying the anomaly detection and diagnosis models and feed them in real-time in order to early detect anomalies.

10. Big data and cloud computing technologies enable horizontally scaling the resources of the monitoring system to support future demanding increased loads, thus facilitating monitoring industrial systems at large-scale.

7.1.1 Hypotheses Validation

Three hypotheses were stated in Section 1.3. Considering the design, implementation, and evaluations conducted throughout this dissertation, this section analyses each of the contributions and briefly discusses whether the stated hypotheses have been validated or not.

Hypothesis 1:

> "*The use of DL methods in which the sensors are processed individually can improve the heterogeneous multi-time series anomaly detection task, besides dealing with changing scenarios*."

To validate this hypothesis, the proposed DL architecture has been compared under different scenarios against traditional algorithms such as RF, CatBoost, LightGBM, and ensemble methods against an existing CNN-RNN architecture used as a baseline. The proposed architecture has outperformed all the other methods at detecting anomalies in heterogeneous multi-time series, thus validating this hypothesis.

Hypothesis 2:

> "*the use of interpretability models can help to understand why DL algorithms reach a decision to diagnose an anomaly.*."

To validate this hypothesis, a small subset of anomalous journeys has been extracted from the original dataset and labeled by a domain expert. This data has been then used to check whether the interpretations obtained using both the SHAP and the attention mechanism were able to correctly diagnose the detected anomalies. The obtained results have demonstrated that, overall, both methods are capable of diagnosing anomalies by interpreting the output of the model. Hence, it can be said that this hypothesis has been validated.

Hypothesis 3:

> "*The use of Big Data and cloud computing technologies can help real-time monitoring CPSs at large-scale*."

To validate this hypothesis, a scalability test has been conducted in which the capability of the proposed Big Data architecture to efficiently ingest, process, and persist data within the cloud has been evaluated. The test has consisted first in assessing the architecture in a realistic scenario to show that it is capable to efficiently monitoring the CPSs considering the current volume of data. Then,

the limit of the architecture has been identified by increasing the volumes of data until the architecture has not been able to efficiently manage the data. Finally, its scalability has been tested by increasing the computational resources to deal with the increased volumes of data in an efficient manner. The obtained results have demonstrated that the proposed monitoring system is able to efficiently scale up to monitor CPS at large-scale. Therefore, it can be said that this hypothesis has been validated.

7.1.2 Limitations

This section discusses the limitations that the proposed anomaly detection system have:

- **Scalability of the anomaly detection model:** as the size of the Multi-head CNN-RNN increases linearly with the number of sensors, one of the biggest limitations of this architecture refers to the scalability. Concretely, the problem arises in the input of the RNN, which is where all the features resulting from the convolutional heads are concatenated and thus, the input size of the RNN will become bigger as more convolutional heads are added. This fact could form a bottleneck and the network could be very slow at training time, besides requiring more computational resources.

- **Adaptability to changing scenarios:** despite the Multi-head CNN-RNN is capable of adapting to changing scenarios by adding, modifying, or adding convolutional heads to the existing architecture, it requires historical data regarding the new sensors to train the new model. Therefore, the new model cannot be trained immediately, but it is necessary to wait until enough data is generated to be able to train it.

- **Interpretability of DL:** although it has been demonstrated that the SHAP and the attention mechanism are capable of diagnosing anomalies by interpreting the output of the DL model, actually they have not been able to diagnose all the anomaly types considered in this dissertation. Moreover, it has been shown that the interpretations have been worsened when more layers are stacked to

the model. Hence, good interpretability results can be obtained provided that few RNN layers are used.

The resolution of these limitations is contemplated in the future work presented in the following section.

7.2 Future Research Lines

This section describes the future research lines opened by this thesis. They range from improving the training phase of the anomaly detection model to the interpretability of DL models to diagnose the anomalies.

7.2.1 Scaling the Multi-Head CNN-RNN Architecture

Although the Multi-head CNN-RNN architecture uses independent CNNs to process each sensor data and thus, each convolutional head could be easily parallelized, the bottleneck is formed at the time of feeding the RNN side of the architecture. This is because all the data resulting from the convolutional heads is centralized in the entrance of the RNN. In this way, the higher the number of sensors the bigger the input of the RNN.

There are typically two main ways to parallelize DL architectures:

- **Data parallelism:** consists in partitioning the data into multiple devices. Consequently, each device has a copy of the entire model and trains it only with the local data to finally compute the gradients by communicating with all the devices.
- **Model parallelism:** consists in partitioning the model itself in a way in which each part is executed in a different device.

In this case, the data parallelism method seems to be not useful since the bottleneck problem would not be addressed. Consequently, model parallelism should be used. To this end, each convolutional head could be easily executed on different devices. However, the scalability issue lies in the difficulty of the RNNs to be parallelized since they have to compute the time-steps in sequential order

because the chronology in time series matters. As each RNN (LSTM and GRU) layer has a state cell that flows from the initial to the final time-step to remember events occurred during time, distributing this process will break the chain and thus, the memory could be lost. Moreover, distributing the computation could lead to disorderly compute the gradients, which will result in bad performance of the network.

One way to distribute RNNs across multiple devices is to run each layer on separate devices. Therefore, when a layer finishes its computation it sends the results to the subsequent layer so that the first layer can start to compute the next batch of data. Nevertheless, this is a way to alleviate the sequential computation and thus, further research must be conducted to parallelize RNNs.

7.2.2 Boosting the Training by Using Transfer Learning

TL is a widely used technique to boost the training of DL architectures and to reduce the amount of data required (Guo et al. 2017; Shelhamer et al. 2017). This is particularly used in the image classification field in which a pre-trained CNN is used as an already trained image feature extractor (Mehdipour Ghazi et al. 2017). However, little research has been done to date in TL for time series classification.

In this case, TL would be especially beneficial at the time of adapting the Multi-head CNN-RNN architecture to new sensor configurations, since fewer data would be required to train the network to fit the new scenario. The technique to extract the most meaningful features for signals should be similar to the one used for images as the only difference is that 1D convolutions are used instead of 2D. However, this task would require large volumes of signal data, as in Mask R-CNN (He et al. 2017) or YOLO (Redmon and Farhadi 2018) which need huge amounts of images for training. Another alternative that would require less data is to use a clustering method to group sensors by similarity so that the convolutional head that is most similar to the new sensor could be used as a pre-trained CNN, as in (Saeedi and Gebremedhin 2018).

7.2.3 Improving the Interpretability of DL

Although the SHAP and the attention mechanism have shown promising results at interpreting the output of the proposed Multi-head CNN-RNN architecture to diagnose anomalies, the truth is that there is still room for improvement. On the one hand, the SHAP has been able to identify correctly the sensors causing some of the anomalies, but it has failed to recognize others. On the other hand, the attention mechanism has shown that it is not able to identify all the point anomalies occurred during the time series. Moreover, it has achieved good interpretability results provided that the network is shallow and thus, further research must be done in this sense to enhance the interpretability of DL architectures to diagnose anomalies, particularly for deeper DL models.

A possible future research line would be to use of fuzzy logic in combination with the SHAP and the attention mechanism. In this way, the explanation of both interpretability methods could be enhanced by setting some fuzzy rules that add more value to the interpretations. As an example, fuzzy logic could be used to determine whether an anomaly has occurred as a consequence of SHAP values of a given sensor being higher than expected.

APPENDIX

A

Acronyms

APV adjusted p-value

ANN Artificial Neural Networks

AP Average Precision

BN Batch Normalization

CNN Convolutional Neural Network

CPS Cyber-Physical System

DL Deep Learning

DNN Deep Neural Network

DS Distributed Simulation

DStream Discretized Stream

DT Decision Trees

geometric mean g_mean

A. Acronyms

GRU Gated Recurrent Unit

IoT Internet of Things

IaaS Infrastructure as a Service

LC1d Locally Connected 1D

LDAS local data acquisition system

LSTM Long-Short Term Memory

ML Machine Learning

MLP Multilayer Perceptron

PaaS Platform as a Service

PCA Principal Component Analysis

RF Random Forest

RDBMS Relational Database Management Systems

ReLU Rectified Linear Units

RNN Recurrent Neural Network

SaaS Software as a Service

SHAP SHapley Additive exPlanations

SS Spark Streaming

SSS Stratified Shuffle Split

SQL Structured Query Language

SVM Support Vector Machines

TL Transfer Learning

TKB transfer knowledge based

APPENDIX

B

CloudFMI Distributed Simulation Framework

B.1 Architecture of the CloudFMI Distributed Simulation Framework

The CloudFMI Distributed Simulation Framework is designed for executing multiple simulations in a parallel and distributed way, reducing the time required to simulate multiple configurations of the same model. Figure B.1 shows an overview of the proposed framework. The simulations are carried out in two phases. First, the SGA configures the experimentation set-up and stores its information in the central database. In this experimentation set up, the different configurations with which the model will be executed are defined. To this end, it utilizes data specified by the user such us input and output parameters. At this point, the SGA also labels each simulation based on the values defined by the user.

Second, the SEA executes the simulations scheduled by the SGA in a sequential manner and stores their output in the database each time a simulation is completed. To do this, the SEA queries the central database to get the configuration parameters of a simulation and executes it. The SEA keeps executing simulations until there are

no pending simulations to run in the database. To execute multiple simulations in parallel, multiple SEAs can be instantiated. These SEAs will run in parallel and thus, as many simulations as agents will be executed concurrently. Each SEA is stateless and fully independent from other running SEAs. It only accesses the database for new parameter configurations to execute. This makes the framework highly scalable as its limit is only given by the available computational resources. See Section B.3 for scalability tests.

It is worth to point out that this framework can execute any FMI compliant model. This is due to the fact that the control over the model's deployment is a key aspect in the use of simulation during the system's operation. Therefore, it should be ensured that the physics-based model can be seamlessly connected with existing platforms, and that it protects the modeler's intellectual property. The FMI provides an attractive solution to ensure a seamless deployment of the model. The FMI is an industrial model ex-change standard aimed at combining models from different vendors, as well as combining these models with custom made code. The model is compiled as a black-box called *Functional Mock-up Unit* (FMU), which may be called by any tool compliant with the FMI. Consequently, it preserves the model's intellectual property, and allows a seamless integration in different platforms. Next, the components of the framework are described, i.e. the SGA, the database, and the SEA.

B.1.1 Simulation Generator Agent

The SGA is responsible of configuring the experimentation environment, which includes configuring the database and generating the experimentation set up. To this purpose, it requires three configuration files (JSON) specified by the user:

- **Input_params**: contains the name of the input parameters and a set of values for each one. These values represent the different conditions to be tested in the simulations.
- **Anomalous_params**: specifies which of the values defined in the input_-params file represent an anomaly in the simulation. This file is optional as it is only used for labeling the simulations.

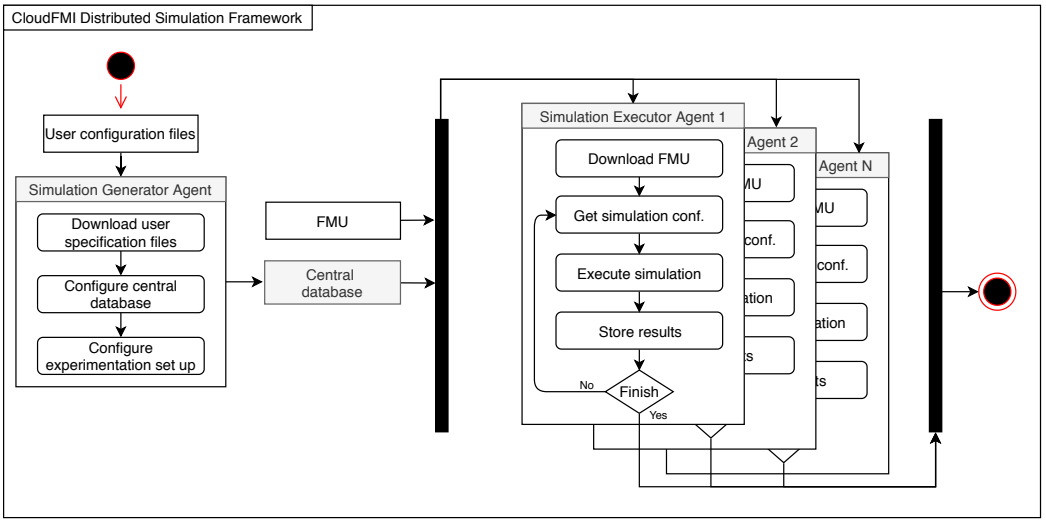


Figure B.1: Architecture of the CloudFMI Distributed Simulation Framework. From left, the SGA configures the experimentation set-up based on user specifications and stores it in a central database. Then, 1 to N SEAs execute in parallel the simulations scheduled in the central database.

- **Output_params**: defines the name of the output parameters. It is used to let the user select which of all the output parameters wants to store in the database.

Hence, the SEA initially configures the database by creating the three previously described tables. It uses input and output parameter files to generate the schema of each table as it can change depending on the parameters specified by the user. Afterward, it defines two stored procedures that will be used by the SEA at execution time. These stored procedures are used to obtain the configuration data of new simulations to be executed and to check whether failed simulations exist. See Section B.1.3 for more details.

Once the database is configured, the SEA proceeds to configure the set up of the experimentation. To accomplish it, it uses the parameters specified in the *input_params* file to generate all possible combinations between the values of all input parameters. Note that each combination refers to a single simulation. Then, the information of the *anomalous_params* file is used to label each simulation as "normal" or "anomalous" (binary classification). Moreover, the status of all

the simulations is set to "Not executed". Finally, each simulation is stored in the database as a new row. In case the user prefers to generate the experimentation set up by its own, the SGA lets a way for uploading it in a Comma Separated Values (CSV) file.

B.1.2 Central Database

The database is the central component of the framework as it stores the configuration of all the simulations to be conducted and their results.

The database engine must satisfy some requirements of the proposed framework. First, it has to be a mature and reliable engine that enables safely storing the simulation's information. Moreover, it must allow querying the database to search and analyze simulations. Thus, maintaining the relationship between the tables that compose the database is a key factor. Regarding the volume of data to be stored in the database, it is expected to be low as the current requirements of the experimentation do not imply simulating more than tens of thousands of simulations. Finally, as well as the other tools used to develop this framework, the database engine must be open source in order to avoid licensing fees at the time of deploying the solution.

Considering all this, PostgreSQL[28] was selected as the database engine since it is an open source object-relational database that satisfies all the stated requirements.

The database contains three tables: *experimentation_config*, *simulation_results*, and *simulation_failure_registry*. Figure B.2 shows the UML diagram of the database. The first table is used to store the experimentation set up, that is, the configuration of each simulation to be executed. Hence, each row of the table refers to a single simulations and each of the columns refer to input parameters (conditions of a given simulation). In addition, other columns were defined to add valuable information to each simulation. Thus, a simulation ID, a timestamp when each simulation starts and finishes, a status, and a label are also included. Note that the status indicates the current state of a simulation, which can be "*Not Executed*", "*Executing*", "*Executed*", or "*Failed*". The label can refer to "*normal*" or "*anomalous*".

[28] https://www.postgresql.org

The *simulation_results* table is used to store the results obtained on each simulation. In addition to the results obtained, the corresponding simulation ID, the identifier of the SEA, and the required execution time is assigned. As the size of each output parameter might be large, their values are stored in binary format to improve the performance of the database.

The *simulation_failure_registry* table contains the registry of the failures occurred during the simulations. Thus, every time a failure occurs, it is registered in the database indicating the simulation identification, the timestamp when it occurred, the identifier of the SEA that executed the corresponding simulation, and a brief description of the failure.

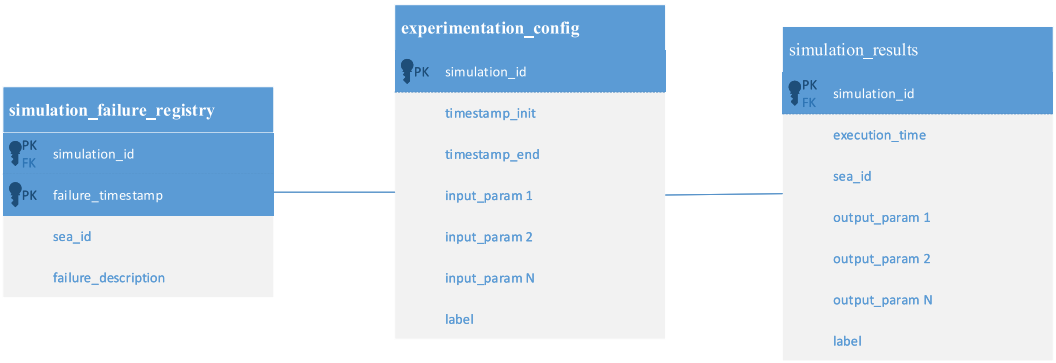


Figure B.2: UML diagram of the database.

B.1.3 Simulation Executor Agent

The SEA is responsible for executing all the simulations scheduled by the SGA. Figure B.3 shows the flow diagram that follows the SEA to execute all the simulations. First, it queries the database to get an available simulation to be executed. To consider a simulation as available, its status must be "*Not executed*" and its failure counter must not exceed the maximum permitted value. Note that the maximum number of allowed failures per simulation is specified by the user at the time of deploying the framework. If the query obtains a positive result, it marks the current simulation as "*Executing*" and initializes the execution of the simulation using the corresponding configuration. When the simulation is completed, the SEA marks the simulation as "*Executed*" and stores the obtained results in the database.

Every time the SEA completes a simulation, it queries the database again to get new simulations until all of them are completed.

However, during the execution of the simulations unexpected errors that prevent the correct operation of the SEA might occur. It may also be the case where a simulation exceeds the time-limit for completion. This time limit is specified by the user and we refer to it as a time-out error. To make the SEA fault-tolerant, the current simulation is automatically canceled as soon as an error is caught. Hence, the simulation is marked in the database as "*Failed*" and its counter of failures is increased by one. Furthermore, a description of the failure is set to let the user know the error's root cause. Then it follows its workload.

When a SEA does not find in the database more simulations to be executed, it checks if failed simulations exist. In such a case, the corresponding simulations are marked again as "*Not Executed*" provided that they do not exceed the maximum number of failures. Consequently, the SEA tries to execute them again. In contrast, if all the simulations finished successfully or the failed simulations exceed the maximum number of failures, the SEA stops working as the experimentation is considered as finished.

To carry out multiple simulations in parallel, multiple SEAs can be executed at the same time. Thus, we can have 1 to N number of SEAs running concurrently. Hence, the time required to complete the entire experimentation can be reduced linearly as the number of agents increases. As many SEAs as there are simulations can be instantiated provided that sufficient computational resources are available. Thus, the framework is highly scalable. However, having multiple SEAs running in parallel lead to a race condition problem S. Li et al. 2018 when more than one agent tries to manipulate the same table of the database at the same time. If this happens, it could lead to malfunctions on the entire experimentation such as setting incorrect status to simulations or executing the same simulation several times. In this case, querying the database to get new configuration parameters and checking for failed simulations to reset them are the conflicting operations. To avoid this kind of problems, the SEAs use locking transactions, which means that only one agent can perform operations at the same time. Note that locking a table for updating a single row implies no delay on the operation of a SEA as relational databases support thousands of transactions per second.

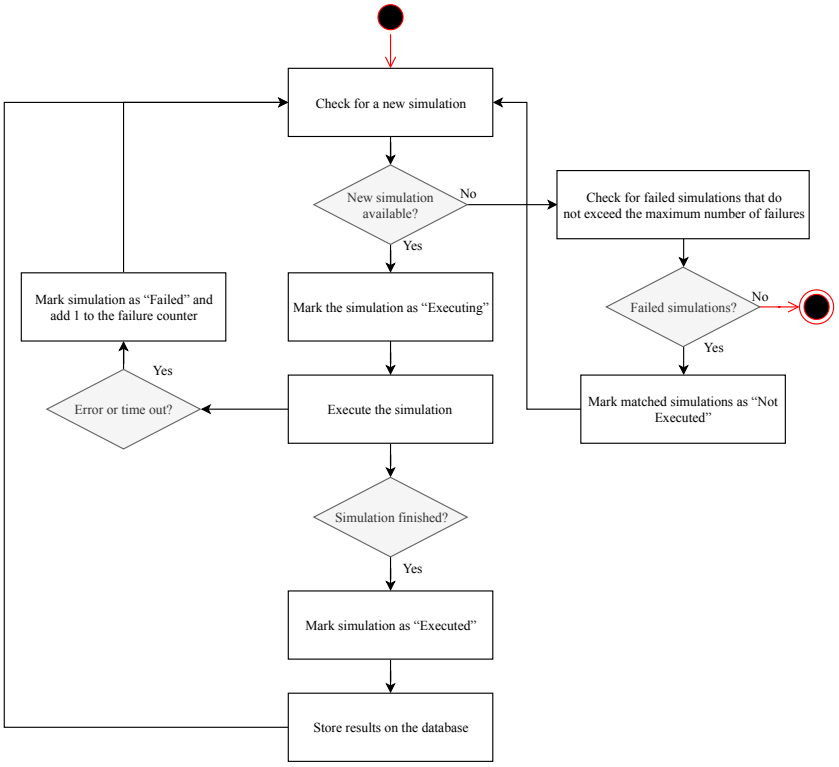


Figure B.3: Workload diagram of the SEA.

B.1.4 Deployment

The CloudFMI Distributed Simulation Framework can be easily deployed in any public or private cloud. To this end, the framework is divided into three layers: application, virtualization, and infrastructure. Figure B.4 depicts these layers.

The application layer contains the components of the framework, that is, the database, the SGA, and the SEA. These applications are responsible for executing the business logic of the framework. However, it is assumed that these applications will run on different machines than in which they are developed and tested. This might cause incompatibility errors or applications to behave differently from how they were developed.

To avoid this problem, each component of the framework is encapsulated in a Docker[29] container. Docker is a lightweight open source virtualization engine that enables packing an application with all the parts it needs, such as libraries and other

[29]https://www.docker.com

dependencies, and shipping it all into one package called container. In this way, the database, the SGA, and the SEA are packaged into independent containers and thus they can be executed in any Linux machine regardless of their configuration. At this point, the framework could be executed in a cluster. However, the orchestration of the containers and the computational resources of the cluster would have to be managed manually. This task is time-consuming and often challenging to accomplish.

Therefore, within the infrastructure layer, a container orquestrator such as Kubernetes[30], DC/OS[31], or Amazon EMR[32] is used to automate the deployment, scaling, and management of the containerized applications. In this way, at the time of deploying the applications, the orchestration of the containers and the required computational resources are managed in an automatic way for the user. Provided that the resources are available, the user only needs to specify the number of parallel SEAs, and Kubernetes automatically deploys the agents.

Although the container orquestrator manages the containers in an automated way, the applications do not know how to connect with each other. Within the proposed framework, both the SGA and the SEA require connecting to the database to make queries and to store information related to the simulations. Therefore, the information they need to connect to the database is introduced into both containers as environment variables. This information involves the IP, port, user name and password of the database. Furthermore, the SEA requires two additional variables: the time-limit and the maximum number of failures per simulation. The former is used to set a time-out, and the latter determines the maximum number of times a simulation can be canceled due to an error before being considered as non-executable.

It is worth to point out, that the configuration files specified by the user and the compiled model are not packed within the Docker containers. In this way, the user can modify the configuration files as needed without having to modify the application. Moreover, different models can be used for each experimentation. Thus, several models and scenarios can be simulated by only modifying these files. To do so, they must be remotely accessible by the SGA as it downloads them from a

[30] https://kubernetes.io
[31] https://dcos.io
[32] https://aws.amazon.com/emr/?nc1=h_ls

remote server such as Amazon S3[33] service. Hence, the path to the configuration files is passed to the SGA as argument variables. In the same way, the path to the compiled model is passed to the SEA.

Both environment and argument variables are introduced by the user at the time of deploying the applications in the cloud.

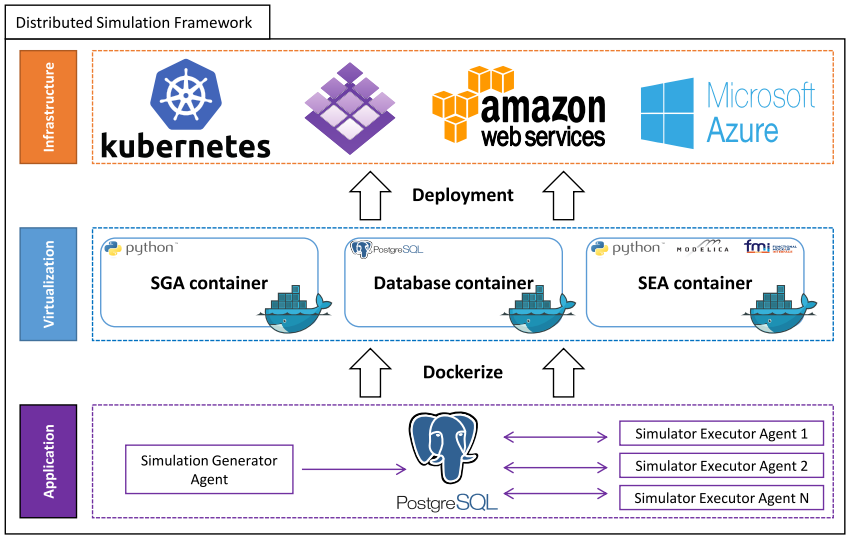


Figure B.4: Deployment of the Distributed Simulation Framework.

B.2 Experimental Framework

This section describes the configuration and properties related to the experimentation followed in this chapter. We show the measures used to benchmark the scalability of the framework in Section B.2.1, the different physical models used (Section B.2.2), the parameters used for each model (Section B.2.3), and the infrastructure where the framework is deployed (Section B.2.4). The code used is in this experimentation is publicly available (https://github.com/ikerlan2015/CloudFMI-Distributed-Simulation-Framework).

[33] https://aws.amazon.com/s3/?nc1=h_ls

B.2.1 Performance Measures

In this section, we describe the metric used to measure the parallel scaling performance of the proposed framework. Namely, speedup is selected as performance metric. This metric is widely used on parallel computing benchmarks, since it indicates how efficient an application is at the time of using increasing numbers of parallel processing elements Archibald et al. 2018; Atashpendar et al. 2018; Y. Zhou et al. 2018.

As standard distributed computing methods, the parallel scaling performance of the framework can be measured in terms of speedup (S), which indicates the improvement in speed of execution for a given task by using more computational resources. A program is considered to scale linearly if the speedup (in terms of work units completed per unit time) is equal to the number of processing elements used (N). The $speedup$ corresponding to a given N is computed as:

$$S_N = \frac{t_1}{t_N} \tag{B.1}$$

where t_1 denotes the execution time required to complete a work unit with one processing element, and t_N denotes the execution time to complete the same work unit with N processing elements. In this context, a work unit refers to an experiment where a fixed number of simulations are executed, and a processing element refers to a SEA.

The ideal scenario would be to achieve $S_N = N$ although a lower score is typically obtained due to system overheads caused by multiple processes making system or database calls, among others.

B.2.2 Used FMI Models

In this section, we describe the physics-based models used in this experimentation. Hence, the proposed framework is evaluated using two different models developed in *Modelica* and compiled according to the FMI. Next, a brief description of each model is presented:

- **Elevator model:** refers to the model used to generate the synthetic data considered in this thesis. This model mimics the behavior of an elevator. Each

simulation of this model requires a computational time of approximately 350 s (mean value), and studying each combination of faults leads to a significantly large number of simulations. Additionally, certain combinations of parameters could make the model diverge. Therefore, with this model the generation of synthetic measurements to feed a supervised machine learning strategy requires a framework which allows: (i) simulating several instances in parallel, (ii) labeling each simulation according to the fault combination, and (iii) stopping the simulation in case of divergent simulations.

- **Academic model:** this is a model used to share the use of the presented platform. It consists of five spring-mass systems connected together. As such, this model has 5 degrees of freedom, corresponding to the position of each mass, s_i. This model has 10 parameters that may be modified to create multiple model configurations in the SEA, i.e. the stiffness and mass of each spring-mass system. This model is depicted in figure B.5. Note that this model is only designed for people to test the proposed framework as the elevator model is private. Therefore, all performed experiments are executed using the elevator model.

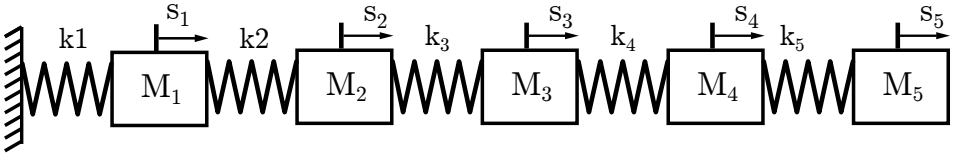


Figure B.5: Academic spring-mass system.

B.2.3 Parameters of the Elevator Model

The presented physics-based model can simulate different scenarios depending on the defined parameters. This section describes the model's input and output and their range of values used in this experimentation. Table B.1 shows the input parameters used for the elevator model.

In total, five input parameters and 20 outputs are used. Note that for the elevator model, each output represents a time series of 8.000 samples and thus, the simulation's result consists of 160.000 data points ($8.000 \cdot 20$). For the sake of reducing

the time required to run all the experiments, only 128 simulations are executed for each experiment. Therefore, in this experiment, a work unit (w) refers to executing 128 simulations.

As the scalability of the framework is concerned, the database is the unique component likely to be affected by the number of SEAs accessing to it continuously. However, the execution time of the simulations varies significantly depending on the specified input values, being $5s$ and $650s$ the fastest and slowest execution times, and 170 s the standard deviation. Therefore, the probability of multiple SEAs accessing to the database at the same time is considerably reduced. This situation hinders the assessment of whether the database is a bottleneck. To be able to evaluate this, a set of 128 simulations with identical input values is generated, which results in the same execution time for all of them. In this case, multiple SEAs will access the database at the same time. Thus, in the experimentation, both sets of simulations are used.

Description	Values	Unit
Quadrature axis inductance	[0.0261, 0.0267]	H
Flux linkage	[1.64, 1.66]	Wb
Cabin load	[0, 150, 300]	Kg
Coulomb's friction coefficient	[0.5, 0.7]	-
Travel distance and journey direction (uphill or downhill)	[1, -1]	floor

Table B.1: Input parameter of the elevator model.

B.2.4 Deployment Configuration

This section describes the framework's deployment. Two different hosts are tested: (i) a private on-premise cloud, where DC/OS is used as container orchestration manager, and (ii) a public cloud hosted in Amazon Web Services (AWS), where Kubernetes is used as container orchestration manager.

The deployment of the framework is similar for both clouds although different container orchestration managers are used. The main difference between them is the computational resources that are available in each one. Table B.2 summarizes

the resources of each cluster. As it can be observed, the are more computational resources available in the public cloud. This is due to the fact that computational resources are fixed for the private cloud, and more would have to be bought to increase them. Conversely, AWS offers on-demand resources meaning that we can scale our public cloud as much as required in an easy way. We used five "*m4.4xlarge*" instances to conduct the experimentation (check the official web page[34] for more details).

Table B.3 shows the resources used by the database, the SGA, and the SEA in each of the clusters. As shown, the computational resources used by the components is higher for the public cloud, meaning that they are more powerful in terms of computational capacity. It must be noted that, for the private cloud, all the components are deployed in it. In contrast, for the public cloud, the database is deployed outside of the cluster meaning that all available resources are dedicated to execute the SEAs. Nonetheless, the database is also deployed in AWS by using the Amazon Relational Database Service[35] (RDS), which facilitates the deployment of a relational database. In this way, we deployed two databases with different computational resources to test the scalability of the framework. We used a "*db.t2.small*" instance for the former and a "*db.m4.xlarge*" for the latter (check the official web page[36] for more details).

The other difference between them relies on the way the components of the framework are executed due to the use of different container orchestration manager. In DC/OS, SEAs are run as services meaning that every time a SEA finishes its workload, the system will try to restart the service continuously. This is due to the fact that a service is designed for being always active and available. Thus, the services must be deleted manually once they complete their workload. In contrast, SEAs are run as jobs in Kubernetes. Unlike services, jobs stop themselves automatically once they complete their task. They only try to recover themselves if they stop working due to an error.

To deploy the framework, the first step is to deploy the central database and to upload the model and the user configuration files to a repository accessible by the cluster. Then, the SGA must be deployed to configure the central database based on

[34]https://aws.amazon.com/ec2/instance-types/?nc1=h_ls

[35]https://aws.amazon.com/rds/?nc1=h_ls

[36]https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html

user's specifications. Finally, the SEA must be deployed by indicating the number of agents required to run in parallel, among other database connection parameters (Check the GitHub[37] repository for detailed information).

	CPU		RAM
	model	cores	Amount
On-premise	Intel(R) Xeon(R) CPU E5-2640 v3 2.6 GHz	36	66 GB
AWS	m4.4xlarge (x5)	80	320 GB

Table B.2: Computational resources of both on-premise and AWS clusters.

	On-premise		AWS	
	CPUs	RAM	CPUs	RAM
PostgreSQL	1	2 GB	1	2 GB
			4	16 GB
SGA	1	2 GB	1	2 GB
SEA	1	2 GB	2	4 GB

Table B.3: Computational resources used by the components of the framework. Note that if N number of SEAs are used the required computational resources will also be multiplied by N.

B.3 Results and Discussion

In this section we analyze and discuss the results of the experimentation. We first perform a scalability test in terms of execution time reduction (Section B.3.1). Afterward, we perform a scalability test in terms of increasing the volume of work to be done (Section B.3.2).

[37] https://github.com/ikerlan2015/CloudFMI-Distributed-Simulation-Framework

B.3.1 Scalability Test: Reducing Execution Time

This test evaluates the scalability of the proposed framework from the perspective of reducing the execution time required to execute a work unit (W) as the number of SEAs (N) is increased (computational units). In an ideal scenario, the required execution time should decrease proportionally to N. In this experimentation, two sets of 128 simulations are executed where the first set involves executing simulations with different input values and the other with identical input values. Bear in mind that the execution of a set of 128 simulations is considered as a single W. In all the tests, N is increased exponentially up to 32, that is, $N = \{1, 2, 4, 8, 16, 32\}$.

Figure B.6 shows the speedup and efficiency obtained for both set of simulations in the on-premise and AWS clusters. Regarding the test with the set of different simulations in the on-premise cluster, Figure B.6a shows that the speedup of the framework keeps near to the ideal speedup although it starts to distance from the ideal line when $n \geq 16$. This deviation is caused by the randomness with which a SEA selects the next simulation to run. That is, each SEA queries the database to obtain the first simulation that has not been yet executed. However, it is not possible to know which simulation will be obtained as a results of the query. This selection is influenced by what the other SEAs have executed before. The selection is also influenced by the time required to execute a given simulation, which varies depending on the specified input parameters. Note that it is not possible to know how much time will require a simulation to run. As a consequence, SEAs running longer simulations will execute a fewer number of simulation than SEAs running shorter ones. Considering this fact, the combination of the simulations executed by a given SEA might not be the most optimal in terms of execution time (i.e., a given SEA executes the longest simulations meaning that it might be working for long time while other SEAs have already finished their work and are idle). Hence, it makes the framework deviates from the ideal speedup since the slowest SEA determines the total execution time required by the framework to complete a work unit.

On the other hand, when all simulations take the same amount of time to run, each SEA should execute the same number of simulations. Therefore, each SEA should take similar time. It can be seen that the speedup slightly deviates from

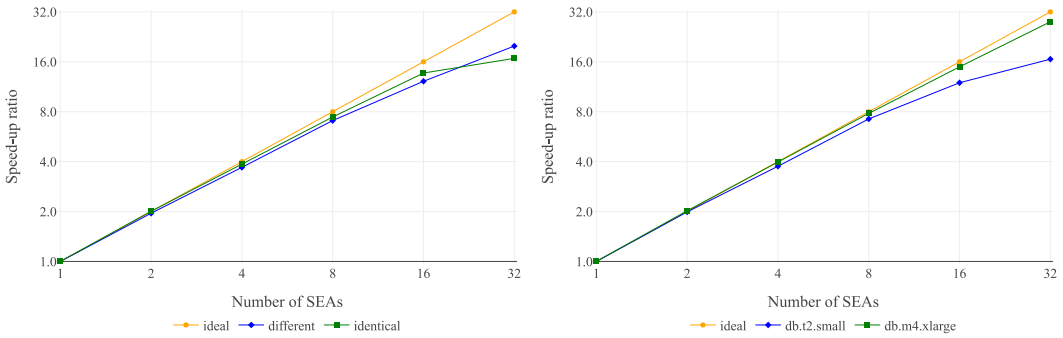
the ideal scenario when $N = 16$ and that it highly deviates when $N = 32$. In this case, the deviation is caused by multiple SEAs accessing simultaneously to the database, forming a bottleneck. Note that since simulations takes the same time to execute, SEAs try to read and write into the database continuously at the same time. The analysis of the log files showed that the insertions in the database were the underlying cause of the delay. This process is computationally expensive as much data must be stored and, as a consequence, inserting multiple simulations concurrently reduces the database's efficiency.

To evaluate whether the framework can scale-up correctly, two additional tests were conducted in AWS as it facilitates scaling the computational resources as needed. The only difference between these tests was the computational resources used by the database. We used similar computational resources to those used in the on-premise cluster for the first test, while we scaled its resources for the second. Note that only the set of identical simulations was used. As shown in Figure B.6b, the obtained speedup is similar to the test conducted in the on-premise cluster. Table B.4 summarizes the time required to insert the results into the database depending on the number of SEAs running in parallel and the type of database used. It can be seen how the time required increases significantly as more SEAs are executed concurrently when little resources are set to the database. However, obtained results demonstrates that the proposed framework can efficiently scale-up if the computational resources of the database are also scaled correspondingly. Insertions time detailed in the table supports this conclusion as the time required to insert data into the database maintains stable regardless of the number of SEAs running in parallel.

B.3.2 Scalability Test: Increasing Workload

Another way to assess the scalability is to measure the ability of the framework to efficiently execute more work units as the number of SEAs is increased proportionally. That is, $W = N = \{1, 2, 4, 8, 16, 32\}$. Hence, the time required to execute a given N should be the same for any case, at least in an ideal scenario.

To evaluate the scalability from this point of view, two tests were conducted in AWS using *db.t2.small* and *db.m4.xlarge* instance for the database. Figure B.7



(a) **(b)**

Figure B.6: Speedup test (reducing execution time) conducted in (a) on-premise cluster using both different and identical set of simulations, and (b) AWS cluster using different computational resources for the database (*db.t2.small* and *db.m4.xlarge* instances) and only using the identical set of simulations.

shows the speedup for the set of different and identical simulations. As it can be observed, the speedup maintains near the ideal scenario for any N meaning that the proposed framework efficiently executes more work units as the number of SEAs is increased. Therefore, it is demonstrated that the proposed framework is also suitable for experimentations in which the workload must be increased.

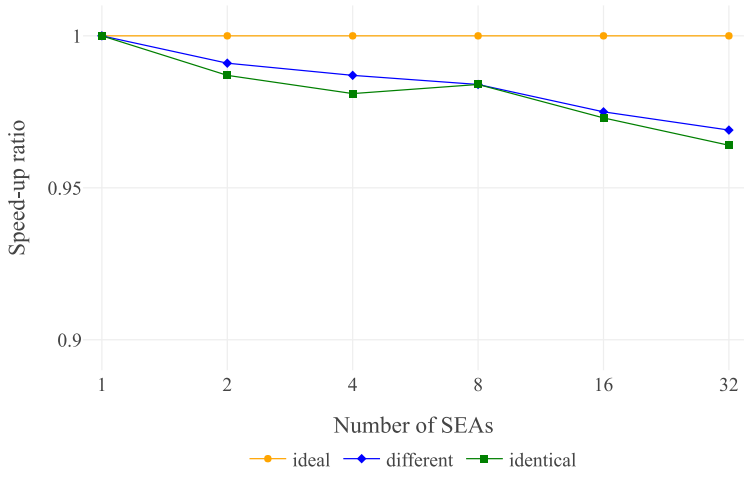


Figure B.7: Speedup test (increasing workload) conducted in AWS with both different and identical set of simulations.

# SEAs	Database type	Min	Max	Mean	Variance
1	db.t2.small	0.454 s	1.168 s	0.513 s	0.004 s
	db.m4.xlarge	**0.440 s**	**0.798 s**	**0.488 s**	**0.002 s**
2	db.t2.small	0.437 s	1.763 s	0.525 s	0.013 s
	db.m4.xlarge	**0.124 s**	**0.873 s**	**0.470 s**	**0.002 s**
4	db.t2.small	0.435 s	8.275 s	0.656 s	0.497 s
	db.m4.xlarge	**0.424 s**	**0.779 s**	**0.462 s**	**0.002 s**
8	db.t2.small	0.441 s	12.557 s	1.035 s	2.177 s
	db.m4.xlarge	**0.420 s**	**0.823 s**	**0.466 s**	**0.002 s**
16	db.t2.small	0.446 s	15.919 s	2.303 s	7.180 s
	db.m4.xlarge	**0.425 s**	**1.189 s**	**0.525 s**	**0.016 s**
32	db.t2.small	0.445 s	20.436 s	5.767 s	15.692 s
	db.m4.xlarge	**0.431 s**	**1.902 s**	**0.715 s**	**0.078 s**

Table B.4: Comparison of the time required for inserting simulation results into the database as the number of both SEAs running in parallel and resources of the database increase. Test conducted in AWS using a *db.t2.small* and *db.m4.xlarge* instance for the database. The best scores for each number of SEAs running in parallel are highlighted in bold.

Bibliography

Abbasi, Maryam et al. (2018). «NoSQL Scalability Performance Evaluation over Cassandra». In: *International Conference Europe Middle East & North Africa Information Systems and Technologies to Support Learning*. Springer, pp. 512–520.

Abdi, Lida and Sattar Hashemi (2016). «To Combat Multi-Class Imbalanced Problems by Means of Over-Sampling Techniques». In: *IEEE Transactions on Knowledge and Data Engineering* 28.1, pp. 238–251. ISSN: 1041-4347. DOI: 10.1109/TKDE.2015.2458858.

Agrawal, Divyakant, Amr El Abbadi, Sudipto Das, and Aaron J Elmore (2011). «Database scalability, elasticity, and autonomy in the cloud». In: *International Conference on Database Systems for Advanced Applications*. Springer, pp. 2–15.

Ahmad, Subutai, Alexander Lavin, Scott Purdy, and Zuha Agha (2017). «Unsupervised real-time anomaly detection for streaming data». In: *Neurocomputing* 262, pp. 134–147. ISSN: 09252312. DOI: 10.1016/j.neucom.2017.04.070.

Aljawarneh, Shadi A. and Radhakrishna Vangipuram (2018). «GARUDA: Gaussian dissimilarity measure for feature representation and anomaly detection in Internet of things». In: *The Journal of Supercomputing*. ISSN: 0920-8542. DOI: 10.1007/s11227-018-2397-3.

Alsheikh, Mohammad Abu, Dusit Niyato, Shaowei Lin, Hwee-pink Tan, and Zhu Han (2016). «Mobile big data analytics using deep learning and apache spark». In: *IEEE Network* 30.3, pp. 22–29. ISSN: 0890-8044. DOI: 10.1109/MNET.2016.7474340.

Andreassen, Odd, Cedric Charrondière, and Alicia De Dios Fuente (2015). «Monitoring Mixed-Language Applications with Elastic Search, Logstash and Kibana (ELK)». In: *15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2015)*, WEPGF041. DOI: 10.18429/JACoW-ICALEPCS2015-WEPGF041.

Antsaklis, Panos, Vijay Gupta, Bill Goodwine, and Karl-Erik Johansson (2012). «Control of Cyber-Physical Systems Workshop». In: *London CPS Workshop*.

Archibald, Blair, Patrick Maier, Ciaran McCreesh, Robert Stewart, and Phil Trinder (2018). «Replicable parallel branch and bound search». In: *Journal of Parallel and Distributed Computing* 113, pp. 92–114. ISSN: 07437315. DOI: 10.1016/j.jpdc.2017.10.010.

BIBLIOGRAPHY

Atashpendar, Arash, Bernabé Dorronsoro, Grégoire Danoy, and Pascal Bouvry (2018). «A scalable parallel cooperative coevolutionary PSO algorithm for multi-objective optimization». In: *Journal of Parallel and Distributed Computing* 112, pp. 111–125. ISSN: 07437315. DOI: 10.1016/j.jpdc.2017.05.018.

Atat, Rachad et al. (2018). «Big Data Meet Cyber-Physical Systems: A Panoramic Survey». In: *IEEE Access* 6, pp. 73603–73636. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2878681.

Atzori, Luigi, Antonio Iera, and Giacomo Morabito (2010). «The internet of things: A survey». In: *Computer networks* 54.15, pp. 2787–2805.

Babiceanu, Radu F and Remzi Seker (2016). «Big Data and virtualization for manufacturing cyber-physical systems: A survey of the current status and future outlook». In: *Computers in Industry* 81, pp. 128–137. ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2016.02.004.

Bach, Sebastian et al. (2015). «On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation». In: *PLOS ONE* 10.7, pp. 1–46. DOI: 10.1371/journal.pone.0130140.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). «Neural Machine Translation by Jointly Learning to Align and Translate». In: *arXiv preprint arXiv:1409.0473*.

Baheti, Radhakisan and Helen Gill (2011). «Cyber-physical systems». In: *The impact of control technology* 12, pp. 161–166.

Balalaie, Armin, Abbas Heydarnoori, and Pooyan Jamshidi (2016). «Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture». In: *IEEE Software* 33.3, pp. 42–52. ISSN: 0740-7459. DOI: 10.1109/MS.2016.64.

Bankó, Zoltán and János Abonyi (2012). «Correlation based dynamic time warping of multivariate time series». In: *Expert Systems with Applications* 39.17, pp. 12814–12823. ISSN: 09574174. DOI: 10.1016/j.eswa.2012.05.012.

Bao, Yuequan, Zhiyi Tang, Hui Li, and Yufeng Zhang (2018). «Computer vision and deep learning–based data anomaly detection method for structural health monitoring». In: *Structural Health Monitoring: An International Journal*, p. 147592171875740. ISSN: 1475-9217. DOI: 10.1177/1475921718757405.

Barnett, Vic. and Toby. Lewis (1994). *Outliers in statistical data*. Wiley, p. 584. ISBN: 9780471930945.

Battaïa, Olga, Alena Otto, Fabio Sgarbossa, and Erwin Pesch (2018). «Future trends in management and operation of assembly systems: from customized assembly systems to cyber-physical systems». In: *Omega (United Kingdom)* 78, pp. 1–4. DOI: 10.1016/j.omega.2018.01.010.

Bazar, Cristina, Cosmin Sebastian Iosif, et al. (2014). «The transition from rdbms to nosql. a comparative analysis of three popular non-relational solutions: Cassandra, mongodb and couchbase». In: *Database Systems Journal* 5.2, pp. 49–59.

Bengio, Yoshua (2009). «Learning Deep Architectures for AI». In: *Foundations and Trends in Machine Learning* 2.1, pp. 1–127. ISSN: 1935-8237. DOI: 10.1561/2200000006.

— (2012). «Deep Learning of Representations for Unsupervised and Transfer Learning». In: *Unsupervised and Transfer Learning - Workshop held at ICML 2011*. Bellevue, Washington, USA, pp. 17–36.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). «Learning long-term dependencies with gradient descent is difficult». In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181.

Benkabou, Seif-Eddine, Khalid Benabdeslem, and Bruno Canitia (2018). «Unsupervised outlier detection for time series by entropy and dynamic time warping». In: *Knowledge and Information Systems* 54.2, pp. 463–486. ISSN: 0219-1377. DOI: 10.1007/s10115-017-1067-8.

Bianchi, Filippo Maria, Enrico De Santis, Antonello Rizzi, and Alireza Sadeghian (2015). «Short-Term Electric Load Forecasting Using Echo State Networks and PCA Decomposition». In: *IEEE Access* 3, pp. 1931–1943. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2015.2485943.

Biljanovicc, Petar. (2011). «Fault detection methods: A literature survey». In: *Proceedings of the 34th International Convention MIPRO*. Opatija, Croatia: IEEE. ISBN: 9781457709968.

Bontemps, Loic, Van Loi Cao, James McDermott, and Nhien-An Le-Khac (2016). «Collective Anomaly Detection based on Long Short Term Memory Recurrent Neural Network». In: *International Conference on Future Data and Security Engineering*, pp. 141–152. ISBN: 978-3-319-48057-2. DOI: 10.1007/978-3-319-48057-2{_}9.

Bosman, Hedde HWJ, Giovanni Iacca, Arturo Tejada, Heinrich J. Wörtche, and Antonio Liotta (2017). «Spatial anomaly detection in sensor networks using neighborhood information». In: *Information Fusion* 33, pp. 41–56. ISSN: 15662535. DOI: 10.1016/j.inffus.2016.04.007.

Boss, Greg, Padma Malladi, Dennis Quan, Linda Legregni, and Harold Hall (2007). «Cloud computing». In: *IBM white paper* 321, pp. 224–231.

Boyd, Stephen (2010). «Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers». In: *Foundations and Trends® in Machine Learning* 3.1, pp. 1–122. ISSN: 1935-8237. DOI: 10.1561/2200000016.

Campos, Jaime et al. (2018). «The use of Relational and NoSQL Databases in Industrial Asset Management». In: *The 13Th World Congress on Engineering Asset Management (WCEAM)*.

Camps, Julia et al. (2018). «Deep learning for freezing of gait detection in Parkinson's disease patients in their homes using a waist-worn inertial measurement unit». In: *Knowledge-Based Systems* 139, pp. 119–131. ISSN: 09507051. DOI: 10.1016/j.knosys.2017.10.017.

Canizo, Mikel, Angel Conde, and Enrique Onieva (2019a). «Interpreting a Deep Learning model for industrial multi-sensor systems anomaly diagnosis». In: *IEEE Transactions on Industrial Informatics*.

Canizo, Mikel, Isaac Triguero, Angel Conde, and Enrique Onieva (2019b). «Multi-head CNN–RNN for multi-time series anomaly detection: An industrial case study». In: *Neurocomputing* 363, pp. 246–260. ISSN: 09252312. DOI: 10.1016/j.neucom.2019.07.034.

Canizo, Mikel et al. (2019c). «Implementation of a Large-Scale Platform for Cyber-Physical System Real-Time Monitoring». In: *IEEE Access* 7, pp. 52455–52466. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2911979.

BIBLIOGRAPHY

Cao, Hong, Xiao-Li Li, David Yew-Kwong Woon, and See-Kiong Ng (2013). «Integrated Oversampling for Imbalanced Time Series Classification». In: *IEEE Transactions on Knowledge and Data Engineering* 25.12, pp. 2809–2822. ISSN: 1041-4347. DOI: 10.1109/TKDE.2013.37.

Carcillo, Fabrizio et al. (2018). «SCARFF : A scalable framework for streaming credit card fraud detection with spark». In: *Information Fusion* 41, pp. 182–194. ISSN: 15662535. DOI: 10.1016/j.inffus.2017.09.005.

Cauteruccio, Francesco et al. (2019). «Short-long term anomaly detection in wireless sensor networks based on machine learning and multi-parameterized edit distance». In: *Information Fusion* 52, pp. 13–30. ISSN: 15662535. DOI: 10.1016/j.inffus.2018.11.010.

Cenedese, Angelo, Michele Luvisotto, and Giulia Michieletto (2017). «Distributed Clustering Strategies in Industrial Wireless Sensor Networks». In: *IEEE Transactions on Industrial Informatics* 13.1, pp. 228–237. ISSN: 1551-3203. DOI: 10.1109/TII.2016.2628409.

Cha, Young-Jin, Wooram Choi, and Oral Büyüköztürk (2017). «Deep learning-based crack damage detection using convolutional neural networks». In: *Computer-Aided Civil and Infrastructure Engineering* 32.5, pp. 361–378.

Chandola, Varun, Arindam Banerjee, and Vipin Kumar (2009). «Anomaly detection: A survey». In: *ACM Computing Surveys* 41.3, 15:1–15:58. DOI: 10.1145/1541880.1541882.

Chandra, Deka Ganesh (2015). «BASE analysis of NoSQL database». In: *Future Generation Computer Systems* 52, pp. 13–21.

Chang, Wo L. and Nancy Grady (2015). *NIST Big Data Interoperability Framework: Volume 1, Definitions*. Tech. rep. Gaithersburg, MD: National Institute of Standards and Technology. DOI: 10.6028/NIST.SP.1500-1.

Chatham, Mark. (2012). *Structured query language by example - volume i : data query language.* Lulu Com. ISBN: 1291199519.

Chattopadhyay, Pratik, Lipo Wang, and Yap-Peng Tan (2018). «Scenario-Based Insider Threat Detection From Cyber Activities». In: *IEEE Transactions on Computational Social Systems* 5.3, pp. 660–675. ISSN: 2329-924X. DOI: 10.1109/TCSS.2018.2857473.

Chauhan, Sucheta and Lovekesh Vig (2015). «Anomaly detection in ECG time signals via deep long short-term memory networks». In: *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, pp. 1–7. ISBN: 978-1-4673-8272-4. DOI: 10.1109/DSAA.2015.7344872.

Chen, Dequan et al. (2017). «Real-Time or Near Real-Time Persisting Daily Healthcare Data Into HDFS and ElasticSearch Index Inside a Big Data Platform». In: *IEEE Transactions on Industrial Informatics* 13.2, pp. 595–606. ISSN: 1551-3203. DOI: 10.1109/TII.2016.2645606.

Chen, Shu-Ching (2019). «Multimedia Deep Learning». In: *IEEE MultiMedia* 26.1, pp. 5–7. ISSN: 1070-986X. DOI: 10.1109/MMUL.2019.2897471.

Chen, Zhehan, Xiaohua Zhang, and Ketai He (2017). «Research on the Technical Architecture for Building CPS and Its Application on a Mobile Phone Factory». In: *2017 5th International Conference on Enterprise Systems (ES)*. IEEE, pp. 76–84. ISBN: 978-1-5386-0936-1. DOI: 10.1109/ES.2017.20.

Cheng, Guo-Jian, Li-Ting Liu, Xin-Jian Qiang, and Ye Liu (2016). «Industry 4.0 Development and Application of Intelligent Manufacturing». In: *2016 International Conference on Information System and Artificial Intelligence (ISAI)*, pp. 407–410. DOI: 10.1109/ISAI.2016.0092.

Cho, Kyunghyun et al. (2014). «Learning phrase representations using RNN encoder-decoder for statistical machine translation». In: *arXiv preprint arXiv:1406.1078*.

Choi, Dong-Kyu, Joong-Hwa Jung, Hyung-Woo Kang, and Seok-Joo Koh (2017). «Cluster-based CoAP for message queueing in Intemet-of-Things networks». In: *2017 19th International Conference on Advanced Communication Technology (ICACT)*. IEEE, pp. 584–588.

Cinar, Yagmur Gizem, Hamid Mirisaee, Parantapa Goswami, Eric Gaussier, and Ali Aït-Bachir (2018). «Period-aware content attention RNNs for time series forecasting with missing values». In: *Neurocomputing* 312, pp. 177–186. ISSN: 09252312. DOI: 10.1016/j.neucom.2018.05.090.

Colombo, Armando Walter, Stamatis Karnouskos, and Thomas Bangemann (2014). «Towards the Next Generation of Industrial Cyber-Physical Systems». In: *Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach*. Ed. by Armando W Colombo et al. Cham: Springer International Publishing, pp. 1–22. ISBN: 978-3-319-05624-1. DOI: 10.1007/978-3-319-05624-1{_}1.

Cutler, Thomas R (2014). «The internet of manufacturing things». In: *Industrial Engineer* 46.8, pp. 37–41.

Darwish, Tasneem S. J. and Kamalrulnizam Abu Bakar (2018). «Fog Based Intelligent Transportation Big Data Analytics in The Internet of Vehicles Environment: Motivations, Architecture, Challenges, and Critical Issues». In: *IEEE Access* 6, pp. 15679–15701. DOI: 10.1109/ACCESS.2018.2815989.

Das, Samarjit (1994). *Time series analysis*. Princeton University Press, Princeton, NJ.

Datta, Anupam, Shayak Sen, and Yair Zick (2016). «Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems». In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 598–617. ISBN: 978-1-5090-0824-7. DOI: 10.1109/SP.2016.42.

Dean, Jeffrey and Sanjay Ghemawat (2008). «MapReduce: simplified data processing on large clusters». In: *Communications of the ACM* 51.1, pp. 107–113. DOI: 10.1145/1327452.1327492.

Demšar, Janez (2006). «Statistical Comparisons of Classifiers over Multiple Data Sets». In: *Journal of Machine Learning Research* 7, pp. 1–30. ISSN: 1532-4435.

Dey, Rahul and Fathi M. Salemt (2017). «Gate-variants of Gated Recurrent Unit (GRU) neural networks». In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, pp. 1597–1600. ISBN: 978-1-5090-6389-5. DOI: 10.1109/MWSCAS.2017.8053243.

Diez-Olivan, Alberto, Jose A. Pagan, Ricardo Sanz, and Basilio Sierra (2017). «Data-driven prognostics using a combination of constrained K-means clustering, fuzzy modeling and LOF-based score». In: *Neurocomputing* 241, pp. 97–107. ISSN: 09252312. DOI: 10.1016/j.neucom.2017.02.024.

BIBLIOGRAPHY

Dobbelaere, Philippe and Kyumars Sheykh Esmaili (2017). «Kafka versus RabbitMQ». In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems - DEBS '17*. New York, New York, USA: ACM Press, pp. 227–238. ISBN: 9781450350655. DOI: 10.1145/3093742.3093908.

Doshi-Velez, Finale and Been Kim (2017). «Towards A Rigorous Science of Interpretable Machine Learning». In: *arXiv preprint arXiv:1702.08608*.

Dromard, Juliette, Gilles Roudiere, and Philippe Owezarski (2017). «Online and Scalable Unsupervised Network Anomaly Detection Method». In: *IEEE Transactions on Network and Service Management* 14.1, pp. 34–47. ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2627340.

Du, Yuheng et al. (2018). «A Distributed Message Delivery Infrastructure for Connected Vehicle Technology Applications». In: *IEEE Transactions on Intelligent Transportation Systems* 19.3, pp. 787–801. ISSN: 1524-9050. DOI: 10.1109/TITS.2017.2701799.

Eidson, John C, Edward A Lee, Slobodan Matic, Sanjit A Seshia, and Jia Zou (2012). «Distributed Real-Time Software for Cyber-Physical Systems». In: *Proceedings of the IEEE* 100.1, pp. 45–59. ISSN: 0018-9219. DOI: 10.1109/JPROC.2011.2161237.

Erfani, Sarah M., Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie (2016). «High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning». In: *Pattern Recognition* 58, pp. 121–134. ISSN: 00313203. DOI: 10.1016/j.patcog.2016.03.028.

Esteban, Ekaitz, Oscar Salgado, Aitzol Iturrospe, and Inge Isasa (2016). «Model-based approach for elevator performance estimation». In: *Mechanical Systems and Signal Processing* 68-69, pp. 125–137. ISSN: 10961216. DOI: 10.1016/j.ymssp.2015.07.005.

Faiz, Rim and Eran A. Edirisinghe (2009). «Decision Making for Predictive Maintenance in Asset Information Management». In: *Interdisciplinary Journal of Information, Knowledge, and Management* Volume 4, p23. DOI: 10.28945/696.

Faroughi, Azadeh and Reza Javidan (2018). «CANF: Clustering and anomaly detection method using nearest and farthest neighbor». In: *Future Generation Computer Systems* 89, pp. 166–177. ISSN: 0167739X. DOI: 10.1016/j.future.2018.06.031.

Fei, Jingjing and Shiliang Sun (2019). «Online Anomaly Detection with Sparse Gaussian Processes». In: *arXiv preprint arXiv:1905.05761*.

Fernández-Rodríguez, Jorge Y., Juan A. Álvarez-García, Jesús Arias Fisteus, Miguel R. Luaces, and Victor Corcoba Magaña (2017). «Benchmarking real-time vehicle data streaming models for a smart city». In: *Information Systems* 72, pp. 62–76. ISSN: 03064379. DOI: 10.1016/j.is.2017.09.002.

Ferreira, Luís et al. (2013). «Cloudlet Architecture for Dashboard in Cloud and Ubiquitous Manufacturing». In: *Procedia CIRP* 12, pp. 366–371. ISSN: 2212-8271. DOI: 10.1016/J.PROCIR.2013.09.063.

Feurer, Matthias et al. (2015). «Efficient and Robust Automated Machine Learning». In: *Advances in neural information processing systems*, pp. 2962–2970.

Filonov, Pavel (2016). «Multivariate Industrial Time Series with Cyber-Attack Simulation: Fault Detection Using an LSTM-based Predictive Data Model.» In: *CoRR* abs/1612.0.

Fisher, Aaron, Cynthia Rudin, and Francesca Dominici (2018). «All Models are Wrong but many are Useful: Variable Importance for Black-Box, Proprietary, or Misspecified Prediction Models, using Model Class Reliance». In: *arXiv preprint arXiv:1801.01489*.

Fisher, Wendy D., Tracy K. Camp, and Valeria V. Krzhizhanovskaya (2017). «Anomaly detection in earth dam and levee passive seismic data using support vector machines and automatic feature selection». In: *Journal of Computational Science* 20, pp. 143–153. ISSN: 18777503. DOI: 10.1016/j.jocs.2016.11.016.

Fu, Tak-chung (2011). «A review on time series data mining». In: *Engineering Applications of Artificial Intelligence* 24.1, pp. 164–181. ISSN: 0952-1976. DOI: 10.1016/J.ENGAPPAI.2010.09.007.

Furht, Borivoje and Armando Escalante (2010). *Handbook of cloud computing*. Vol. 3. Springer.

Gantz, John and David Reinsel (2011). «Extracting value from chaos». In: *IDC iview* 1142.2011, pp. 1–12.

García, Salvador, Alberto Fernández, Julián Luengo, and Francisco Herrera (2010). «Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power». In: *Information Sciences* 180.10, pp. 2044–2064. ISSN: 0020-0255. DOI: 10.1016/J.INS.2009.12.010.

García, Salvador, Alberto Fernández, Julían Luengo, and Francisco Herrera (2009). «A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability». In: *Soft Computing* 13.10, pp. 959–977. ISSN: 1432-7643. DOI: 10.1007/s00500-008-0392-y.

García, Salvador and Francisco Herrera (2008). «An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons». In: *Journal of Machine Learning Research* 9.Dec, pp. 2677–2694. ISSN: ISSN 1533-7928.

Garcia-Ceja, Enrique, Carlos E. Galván-Tejada, and Ramon Brena (2018). «Multi-view stacking for activity recognition with sound and accelerometer data». In: *Information Fusion* 40, pp. 45–56. ISSN: 15662535. DOI: 10.1016/j.inffus.2017.06.004.

Gerostathopoulos, Ilias et al. (2016). «Self-adaptation in software-intensive cyber–physical systems: From system goals to architecture configurations». In: *Journal of Systems and Software* 122, pp. 378–397. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2016.02.028.

Gessert, Felix, Wolfram Wingerath, Steffen Friedrich, and Norbert Ritter (2017). «NoSQL database systems: a survey and decision guidance». In: *Computer Science-Research and Development* 32.3-4, pp. 353–365.

Ghomi, Einollah Jafarnejad, Amir Masoud Rahmani, and Nooruldeen Nasih Qader (2017). «Load-balancing algorithms in cloud computing: A survey». In: *Journal of Network and Computer Applications* 88, pp. 50–71.

Goldstein, Markus and Seiichi Uchida (2016). «A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data». In: *PLOS ONE* 11.4. Ed. by Dongxiao Zhu, e0152173. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0152173.

BIBLIOGRAPHY

Golik, Pavel, Patrick Doetsch, and Hermann Ney (2013). «Cross-entropy vs. squared error training: a theoretical and experimental comparison». In: *14th Annual Conference of the International Speech Communication Association*. Lyon, France, pp. 1756–1760.

Gołuch, Piotr, Janusz Kuchmister, Kazimierz Ćmielewski, and Henryk Bryś (2018). «Multi-sensors measuring system for geodetic monitoring of elevator guide rails». In: *Measurement* 130, pp. 18–31. ISSN: 02632241. DOI: 10.1016/j.measurement.2018.07.077.

Gonzalez, Mikel, Oscar Salgado, Jan Croes, Bert Pluymers, and Wim Desmet (2016). «Model-based virtual sensing approaches for the estimation of forces in guiding systems». In: *ISMA 2016*, pp. 1203–1216. ISBN: 9789073802940.

Gonzalez-Diaz, Ivan (2019). «DermaKNet: Incorporating the Knowledge of Dermatologists to Convolutional Neural Networks for Skin Lesion Diagnosis». In: *IEEE Journal of Biomedical and Health Informatics* 23.2, pp. 547–559. ISSN: 2168-2194. DOI: 10.1109/JBHI.2018.2806962.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

Greff, Klaus, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber (2017). «LSTM: A Search Space Odyssey». In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10, pp. 2222–2232. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2016.2582924.

Grieco, Luigi .A. et al. (2014). «IoT-aided robotics applications: Technological implications, target domains and open issues». In: *Computer Communications* 54, pp. 32–47. ISSN: 0140-3664. DOI: 10.1016/J.COMCOM.2014.07.013.

Guo, Yuchen, Guiguang Ding, Jungong Han, and Yue Gao (2017). «Zero-Shot Learning With Transferred Samples». In: *IEEE Transactions on Image Processing* 26.7, pp. 3277–3290. ISSN: 1057-7149. DOI: 10.1109/TIP.2017.2696747.

Gupta, Manish, Jing Gao, Charu C. Aggarwal, and Jiawei Han (2014). «Outlier Detection for Temporal Data: A Survey». In: *IEEE Transactions on Knowledge and Data Engineering* 26.9, pp. 2250–2267. ISSN: 1041-4347. DOI: 10.1109/TKDE.2013.184.

Hamamoto, Anderson Hiroshi, Luiz Fernando Carvalho, Lucas Dias Hiera Sampaio, Taufik Abrão, and Mario Lemes Proença (2018). «Network Anomaly Detection System using Genetic Algorithm and Fuzzy Logic». In: *Expert Systems with Applications* 92, pp. 390–402. DOI: 10.1016/j.eswa.2017.09.013.

Hanin, Boris (2018). «Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients?» In: *Advances in Neural Information Processing Systems 31*. Ed. by S Bengio et al. Curran Associates, Inc., pp. 582–591.

Happ, Daniel, Niels Karowski, Thomas Menzel, Vlado Handziski, and Adam Wolisz (2017). «Meeting IoT platform requirements with open pub/sub solutions». In: *Annals of Telecommunications* 72.1-2, pp. 41–52.

Harrison, Robert, Daniel Vera, and Bilal Ahmad (2016). «Engineering Methods and Tools for Cyber 2013;Physical Automation Systems». In: *Proceedings of the IEEE* 104.5, pp. 973–985. ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2510665.

Hashemian, Hash M. and Wendell C. Bean (2011). «State-of-the-Art Predictive Maintenance Techniques*». In: *IEEE Transactions on Instrumentation and Measurement* 60.10, pp. 3480–3492. ISSN: 0018-9456. DOI: 10.1109/TIM.2009.2036347.

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick (2017). «Mask R-CNN». In: *arXiv preprint arXiv:1703.06870*.

Hindman, Benjamin et al. (2011). «Mesos: A Platform for Fine-grained Resource Sharing in the Data Center». In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. Berkeley, CA, USA: USENIX Association, pp. 295–308.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). «Long Short-Term Memory». In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). «Multilayer feedforward networks are universal approximators». In: *Neural Networks* 2.5, pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8.

Hu, Min et al. (2018). «Detecting Anomalies in Time Series Data via a Meta-Feature Based Approach». In: *IEEE Access* 6, pp. 27760–27776. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2840086.

Hu, Yu et al. (2018). «A novel attention-based hybrid CNN-RNN architecture for sEMG-based gesture recognition». In: *PLOS ONE* 13.10. Ed. by Huiguang He, e0206049. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0206049.

Hua, Xiaoqiang, Yongqiang Cheng, Hongqiang Wang, Yuliang Qin, and Yubo Li (2017). «Geometric means and medians with applications to target detection». In: *IET Signal Processing* 11.6, pp. 711–720. ISSN: 1751-9675. DOI: 10.1049/iet-spr.2016.0547.

Hunt, Patrick, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed (2010). «ZooKeeper: Wait-free Coordination for Internet-scale Systems.» In: *USENIX annual technical conference*. Vol. 8, p. 9.

Hyde, Richard, Plamen Angelov, and A.Rob MacKenzie (2017). «Fully online clustering of evolving data streams into arbitrarily shaped clusters». In: *Information Sciences* 382-383, pp. 96–114. ISSN: 00200255. DOI: 10.1016/j.ins.2016.12.004.

Iglesias, Aitziber et al. (2017). «Product Line Engineering of Monitoring Functionality in Industrial Cyber-Physical Systems: A Domain Analysis». In: *Proceedings of the 21st International Systems and Software Product Line Conference*. Sevilla, Spain, pp. 195–204. DOI: 10.1145/3106195.3106223.

Ignatov, Andrey (2018). «Real-time human activity recognition from accelerometer data using Convolutional Neural Networks». In: *Applied Soft Computing* 62, pp. 915–922. ISSN: 15684946. DOI: 10.1016/j.asoc.2017.09.027.

Ioffe, Sergey and Christian Szegedy (2015). «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.» In: *32nd International Conference on Machine Learning*. Lille, France: JMLR.org, pp. 448–456.

Isermann, Rolf (2006). *Fault-Diagnosis Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-24112-6. DOI: 10.1007/3-540-30368-5.

Jansen, Christoph, Stephan Hodel, Thomas Penzel, Martin Spott, and Dagmar Krefting (2018). «Feature relevance in physiological networks for classification of obstructive sleep apnea». In: *Physiological measurement*.

Kagerman, Henning and Wolfgang Wahlster (2013). *Recommendations for implementing the strategic initiative INDUSTRIE 4.0 April 2013 Securing the future of German manufacturing industry Final report of the Industrie 4.0 Working Group*. Tech. rep. Forschungsunion.

Kaji, Deepak A et al. (2019). «An attention based deep learning model of clinical events in the intensive care unit». In: *PloS one* 14.2, e0211057.

Kalavri, Vasiliki and Vladimir Vlassov (2013). «MapReduce: Limitations, Optimizations and Open Issues». In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, pp. 1031–1038. ISBN: 978-0-7695-5022-0. DOI: 10.1109/TrustCom.2013.126.

Kanarachos, Stratis, Stavros-Richard G. Christopoulos, Alexander Chroneos, and Michael E. Fitzpatrick (2017). «Detecting anomalies in time series data via a deep learning algorithm combining wavelets, neural networks and Hilbert transform». In: *Expert Systems with Applications* 85, pp. 292–304. ISSN: 09574174. DOI: 10.1016/j.eswa.2017.04.028.

Kang, Hyoung Seok et al. (2016). «Smart manufacturing: Past research, present findings, and future directions». In: *International Journal of Precision Engineering and Manufacturing-Green Technology* 3.1, pp. 111–128.

Kanjo, Eiman, Eman M.G. Younis, and Chee Siang Ang (2019). «Deep learning analysis of mobile physiological, environmental and location sensor data for emotion detection». In: *Information Fusion* 49, pp. 46–56. ISSN: 15662535. DOI: 10.1016/j.inffus.2018.09.001.

Karim, Fazle, Somshubra Majumdar, Houshang Darabi, and Shun Chen (2018). «LSTM Fully Convolutional Networks for Time Series Classification». In: *IEEE Access* 6, pp. 1662–1669. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2779939.

Karpathy, Andrej, Justin Johnson, and Li Fei-Fei (2015). «Visualizing and understanding recurrent networks». In: *arXiv preprint arXiv:1506.02078*.

Kermany, Daniel S. et al. (2018). «Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning». In: *Cell* 172.5, pp. 1122–1131. ISSN: 00928674. DOI: 10.1016/j.cell.2018.02.010.

Khurum, Mahvish and Tony Gorschek (2009). «A systematic review of domain analysis solutions for product lines». In: *Journal of Systems and Software* 82.12, pp. 1982–2003. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2009.06.048.

Kiirikki, J. and M. Haag (2013). «Ubiquitous Assembly Cell Concept and Requirements». In: *Procedia CIRP* 12, pp. 157–162. ISSN: 2212-8271. DOI: 10.1016/J.PROCIR.2013.09.028.

Kim, Been, Oluwasanmi Koyejo, and Rajiv Khanna (2016). «Examples are not enough, learn to criticize! Criticism for Interpretability». In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2280–2288.

Kim, Tae-Young and Sung-Bae Cho (2018). «Web traffic anomaly detection using C-LSTM neural networks». In: *Expert Systems with Applications* 106, pp. 66–76. ISSN: 09574174. DOI: 10.1016/j.eswa.2018.04.004.

Kingma, Diederik P and Jimmy Ba (2014). «Adam: A Method for Stochastic Optimization». In: *CoRR* abs/1412.6.

Kondo, Ruho, Shunsuke Yamakawa, Yumi Masuoka, Shin Tajima, and Ryoji Asahi (2017). «Microstructure recognition using convolutional neural networks for prediction of ionic conductivity in ceramics». In: *Acta Materialia* 141, pp. 29–38.

Krawczak, Maciej and Grażyna Szkatuła (2014). «An approach to dimensionality reduction in time series». In: *Information Sciences* 260, pp. 15–36. ISSN: 00200255. DOI: 10.1016/j.ins.2013.10.037.

Kreps, Jay, Neha Narkhede, Jun Rao, et al. (2011). «Kafka: A distributed messaging system for log processing». In:

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). «ImageNet classification with deep convolutional neural networks». In: *Proceedings of the 25th International Conference on Neural Information Processing Systems*. Lake Tahoe, Nevada, USA: Curran Associates Inc., pp. 1097–1105.

Laney, Doug (2001). «3D data management: Controlling data volume, velocity and variety». In: *META group research note* 6.70, p. 1.

Langi, Pingkan P I, Widyawan, Warsun Najib, and Teguh Bharata Aji (2015). «An evaluation of Twitter river and Logstash performances as elasticsearch inputs for social media analysis of Twitter». In: *2015 International Conference on Information Communication Technology and Systems (ICTS)*, pp. 181–186. DOI: 10.1109/ICTS.2015.7379895.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). «Deep learning». In: *Nature* 521.7553, pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539.

Lecun, Yann, Leon Bottou, Yosua Bengio, and Patrick Haffner (1998). «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. ISSN: 00189219. DOI: 10.1109/5.726791.

Lee, Jay, Behrad Bagheri, and Hung-An Kao (2015). «A cyber-physical systems architecture for industry 4.0-based manufacturing systems». In: *Manufacturing Letters* 3, pp. 18–23.

Lee, Jay, Hung-An Kao, and Shanhu Yang (2014). «Service innovation and smart analytics for industry 4.0 and big data environment». In: *Procedia Cirp* 16, pp. 3–8.

Lee, Kun Chang, Namho Chung, and Jeongeun Byun (2015). «Understanding continued ubiquitous decision support system usage behaviour». In: *Telematics and Informatics* 32.4, pp. 921–929. ISSN: 0736-5853. DOI: 10.1016/J.TELE.2015.05.001.

Leitão, Paulo, Armando Walter Colombo, and Stamatis Karnouskos (2016). «Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges». In: *Computers in Industry* 81, pp. 11–25. DOI: 10.1016/j.compind.2015.08.004.

Li, Feng, Beng Chin Ooi, M. Tamer Özsu, and Sai Wu (2014). «Distributed data management using MapReduce». In: *ACM Computing Surveys* 46.3, pp. 1–42. ISSN: 03600300. DOI: 10.1145/2503009.

Li, Jinbo, Witold Pedrycz, and Iqbal Jamal (2017). «Multivariate time series anomaly detection: A framework of Hidden Markov Models». In: *Applied Soft Computing* 60, pp. 229–240. ISSN: 15684946. DOI: 10.1016/j.asoc.2017.06.035.

Li, Sizhao et al. (2018). «Race-Condition-Aware and Hardware-Oriented Task Partitioning and Scheduling Using Entropy Maximization». In: *IEEE Transactions on Parallel and Distributed Systems* 29.7, pp. 1589–1604. ISSN: 1045-9219. DOI: 10.1109/TPDS.2017.2784829.

Lin, Sangdi and George C. Runger (2018). «GCRNN: Group-Constrained Convolutional Recurrent Neural Network». In: *IEEE Transactions on Neural Networks and Learning Systems* 29.10, pp. 4709–4718. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2017.2772336.

Lipovetsky, Stan and Michael Conklin (2001). «Analysis of regression in game theory approach». In: *Applied Stochastic Models in Business and Industry* 17.4, pp. 319–330. ISSN: 1524-1904. DOI: 10.1002/asmb.446.

Lipton, Zachary C (2018). «The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery». In: *Queue* 16.3, 30:31–30:57. ISSN: 1542-7730. DOI: 10.1145/3236386.3241340.

Liu, Ruonan, Guotao Meng, Boyuan Yang, Chuang Sun, and Xuefeng Chen (2017). «Dislocated Time Series Convolutional Neural Architecture: An Intelligent Fault Diagnosis Approach for Electric Machine». In: *IEEE Transactions on Industrial Informatics* 13.3, pp. 1310–1320. ISSN: 1551-3203. DOI: 10.1109/TII.2016.2645238.

Lundberg, Scott M, Gabriel G Erion, and Su-In Lee (2018). «Consistent individualized feature attribution for tree ensembles». In: *arXiv preprint arXiv:1802.03888*.

Lundberg, Scott M and Su-In Lee (2017). «A Unified Approach to Interpreting Model Predictions». In: *Advances in Neural Information Processing Systems 30*. Ed. by I Guyon et al. Curran Associates, Inc., pp. 4765–4774.

Ly, Linh Thao, Fabrizio Maria Maggi, Marco Montali, Stefanie Rinderle-Ma, and Wil M.P. van der Aalst (2015). «Compliance monitoring in business processes: Functionalities, application, and tool-support». In: *Information Systems* 54, pp. 209–234. ISSN: 03064379. DOI: 10.1016/j.is.2015.02.007.

Magnoni, Luca (2015). «Modern messaging for distributed sytems». In: *Journal of Physics: Conference Series*. Vol. 608. 1. IOP Publishing, p. 12038.

Maillo, Jesus, Sergio Ramírez, Isaac Triguero, and Francisco Herrera (2017). «kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data». In: *Knowledge-Based Systems* 117, pp. 3–15. ISSN: 09507051. DOI: 10.1016/j.knosys.2016.06.012.

Malhotra, Pankaj (2016). «LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection.» In: *CoRR* abs/1607.0.

Malhotra, Pankaj, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal (2015). «Long Short Term Memory Networks for Anomaly Detection in Time Series». In: *European Symposium on Artificial Neural Networks*, pp. 22–24. ISBN: 9782875870148. DOI: 10.14722/ndss.2015.23268.

Markou, Markos and Sameer Singh (2003). «Novelty detection: a review—part 1: statistical approaches». In: *Signal Processing* 83.12, pp. 2481–2497. ISSN: 01651684. DOI: 10.1016/j.sigpro.2003.07.018.

— (2006). «A Neural Network-Based Novelty Detector for Image Sequence Analysis». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.10, pp. 1664–1677. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2006.196.

Masdani, Muhammad V and Denny Darlis (2018). «A comprehensive study on MQTT as a low power protocol for internet of things application». In: *IOP Conference Series: Materials Science and Engineering* 434, p. 012274. ISSN: 1757-899X. DOI: 10.1088/1757-899X/434/1/012274.

Mehdipour Ghazi, Mostafa, Berrin Yanikoglu, and Erchan Aptoula (2017). «Plant identification using deep neural networks via optimization of transfer learning parameters». In: *Neurocomputing* 235, pp. 228–235. ISSN: 09252312. DOI: 10.1016/j.neucom.2017.01.018.

Meijer, Erik (2011). «The world according to LINQ». In: *Queue* 9.8, p. 60.

Miller, Tim (2019). «Explanation in artificial intelligence: Insights from the social sciences». In: *Artificial Intelligence* 267, pp. 1–38. ISSN: 00043702. DOI: 10.1016/j.artint.2018.07.007.

Miotto, Riccardo, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley (2017). «Deep learning for healthcare: review, opportunities and challenges». In: *Briefings in bioinformatics* 19.6, pp. 1236–1246.

Mohamed, Mohamed A, Obay G Altrafi, and Mohammed O Ismail (2014). «Relational vs. nosql databases: A survey». In: *International Journal of Computer and Information Technology* 3.03, pp. 598–601.

Mohammadian Rad, Nastaran et al. (2018). «Deep learning for automatic stereotypical motor movement detection using wearable sensors in autism spectrum disorders». In: *Signal Processing* 144, pp. 180–191. ISSN: 01651684. DOI: 10.1016/j.sigpro.2017.10.011.

Molnar, Christoph (2019). *Interpretable Machine Learning*.

Moniz, Nuno, Paula Branco, and Luís Torgo (2017). «Resampling strategies for imbalanced time series forecasting». In: *International Journal of Data Science and Analytics* 3.3, pp. 161–181. ISSN: 2364-415X. DOI: 10.1007/s41060-017-0044-3.

Monostori, L et al. (2016). «Cyber-physical systems in manufacturing». In: *CIRP Annals - Manufacturing Technology* 65.2, pp. 621–641. ISSN: 0007-8506. DOI: https://doi.org/10.1016/j.cirp.2016.06.005.

Montavon, Grégoire, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller (2017). «Explaining nonlinear classification decisions with deep Taylor decomposition». In: *Pattern Recognition* 65, pp. 211–222. ISSN: 00313203. DOI: 10.1016/j.patcog.2016.11.008.

Nguyen Thi, Nga, Van Loi Cao, and Nhien-An Le-Khac (2017). «One-Class Collective Anomaly Detection Based on LSTM-RNNs». In: *Transactions on Large-Scale Data- and Knowledge-Centered Systems*. Vol. 36, pp. 73–85. DOI: 10.1007/978-3-662-56266-6{_}4.

BIBLIOGRAPHY

Nguyen, Anh, Jason Yosinski, and Jeff Clune (2016). «Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks». In: *arXiv preprint arXiv:1602.03616*.

Niggemann, Oliver et al. (2015). «Data-Driven Monitoring of Cyber-Physical Systems Leveraging on Big Data and the Internet-of-Things for Diagnosis and Control.» In: *International Workshop on the Principles of Diagnosis*, pp. 185–192.

Olah, Chris, Alexander Mordvintsev, and Ludwig Schubert (2017). «Feature Visualization». In: *Distill* 2.11, e7. ISSN: 2476-0757. DOI: 10.23915/distill.00007.

Olah, Chris et al. (2018). «The building blocks of interpretability». In: *Distill* 3.3, e10.

Ordóñez, Francisco and Daniel Roggen (2016). «Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition». In: *Sensors* 16.1, p. 115. ISSN: 1424-8220. DOI: 10.3390/s16010115.

Pang, Jingyue, Datong Liu, Yu Peng, and Xiyuan Peng (2017). «Anomaly detection based on uncertainty fusion for univariate monitoring series». In: *Measurement* 95, pp. 280–292. ISSN: 02632241. DOI: 10.1016/j.measurement.2016.10.031.

Paragliola, Giovanni and Antonio Coronato (2018). «Gait Anomaly Detection of Subjects with Parkinson's Disease Using a Deep Time Series-based Approach». In: *IEEE Access*, pp. 1–1. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2882245.

Park, Jaehui and Su-young Chi (2016). «An implementation of a high throughput data ingestion system for machine logs in manufacturing industry». In: *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 117–120. DOI: 10.1109/ICUFN.2016.7536997.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2012). «Understanding the exploding gradient problem». In: *CoRR, abs/1211.5063* 2.

Pereira, Diogo Augusto, Wagner Ourique de Morais, and Edison Pignaton de Freitas (2018). «NoSQL real-time database performance comparison». In: *International Journal of Parallel, Emergent and Distributed Systems* 33.2, pp. 144–156. ISSN: 1744-5760. DOI: 10.1080/17445760.2017.1307367.

Pimentel, Marco A.F., David A. Clifton, Lei Clifton, and Lionel Tarassenko (2014). «A review of novelty detection». In: *Signal Processing* 99, pp. 215–249. ISSN: 0165-1684. DOI: 10.1016/J.SIGPRO.2013.12.026.

Pop, Florin and Nik Bessis (2013). «Energy-efficient and fault tolerant methods for message delivery in Internet of Things». In: *2013 11th RoEduNet International Conference*. IEEE, pp. 1–6. ISBN: 978-1-4673-6116-3. DOI: 10.1109/RoEduNet.2013.6511735.

Pourbabaee, Bahareh, Mehrsan Javan Roshtkhari, and Khashayar Khorasani (2017). «Deep Convolutional Neural Networks and Learning ECG Features for Screening Paroxysmal Atrial Fibrillation Patients». In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–10. ISSN: 2168-2216. DOI: 10.1109/TSMC.2017.2705582.

Preece, Alun (2018). «Asking 'Why' in AI: Explainability of intelligent systems - perspectives and challenges». In: *Intelligent Systems in Accounting, Finance and Management* 25.2, pp. 63–72. ISSN: 1055615X. DOI: 10.1002/isaf.1422.

Puggini, Luca and Seán McLoone (2018). «An enhanced variable selection and Isolation Forest based methodology for anomaly detection with OES data». In: *Engineering Applications of Artificial Intelligence* 67, pp. 126–135. ISSN: 09521976. DOI: 10.1016/j.engappai.2017.09.021.

Qian, Yanmin, Mengxiao Bi, Tian Tan, and Kai Yu (2016). «Very Deep Convolutional Neural Networks for Noise Robust Speech Recognition». In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24.12, pp. 2263–2276. ISSN: 2329-9290. DOI: 10.1109/TASLP.2016.2602884.

Qiu, Xuan, Hao Luo, Gangyan Xu, Runyang Zhong, and George Q. Huang (2015). «Physical assets and service sharing for IoT-enabled Supply Hub in Industrial Park (SHIP)». In: *International Journal of Production Economics* 159, pp. 4–15. ISSN: 0925-5273. DOI: 10.1016/J.IJPE.2014.09.001.

Rahimi, M. Reza, Jian Ren, Chi Harold Liu, Athanasios V. Vasilakos, and Nalini Venkatasubramanian (2014). «Mobile Cloud Computing: A Survey, State of Art and Future Directions». In: *Mobile Networks and Applications* 19.2, pp. 133–143. ISSN: 1383-469X. DOI: 10.1007/s11036-013-0477-4.

Ramírez-Gallego, Sergio, Bartosz Krawczyk, Salvador García, and Francisco Woźniak Michałand Herrera (2017). «A survey on data preprocessing for data stream mining: Current status and future directions». In: *Neurocomputing* 239, pp. 39–57.

Ranjithprabhu, K and D Sasirega (2014). «Eliminating Single Point of Failure and Data Loss in Cloud Computing». In: *International Journal of Science and Research (IJSR)* 3, pp. 335–337. ISSN: 2319-7064.

Redmon, Joseph and Ali Farhadi (2018). «YOLOv3: An Incremental Improvement». In: *arXiv preprint arXiv:1804.02767*.

Ren, Donghao, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D. Williams (2017). «Squares: Supporting Interactive Performance Analysis for Multiclass Classifiers». In: *IEEE Transactions on Visualization and Computer Graphics* 23.1, pp. 61–70. ISSN: 1077-2626. DOI: 10.1109/TVCG.2016.2598828.

Ribeiro, Marco Túlio, Sameer Singh, and Carlos Guestrin (2016). «"Why Should I Trust You?": Explaining the Predictions of Any Classifier». In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, CA, USA, pp. 1135–1144. DOI: 10.1145/2939672.2939778.

Rosenblatt, F (1958). «The perceptron: A probabilistic model for information storage and organization in the brain.» In: *Psychological Review* 65.6, pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). «Learning Representations by Back-propagating Errors». In: *Nature* 323.6088, pp. 533–536. DOI: 10.1038/323533a0.

Saabas, Ando (2014). *Interpreting random forests — Diving into data*. URL: http://blog.datadive.net/interpreting-random-forests/.

Saeedi, Ramyar and Assefaw Gebremedhin (2018). «A Signal-Level Transfer Learning Framework for Autonomous Reconfiguration of Wearable Systems». In: *IEEE Transactions on Mobile Computing*, pp. 1–1. ISSN: 1536-1233. DOI: 10.1109/TMC.2018.2878673.

Saito, Takaya and Marc Rehmsmeier (2015). «The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets». In: *PLOS ONE* 10.3. Ed. by Guy Brock, e0118432. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0118432.

Sakamura, Ken and Noboru Koshizuka (2005). «Ubiquitous computing technologies for ubiquitous learning». In: *IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE'05)*. IEEE, pp. 11–20.

Salamon, Justin and Juan Pablo Bello (2017). «Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification». In: *IEEE Signal Processing Letters* 24.3, pp. 279–283. ISSN: 1070-9908. DOI: 10.1109/LSP.2017.2657381.

Samuelsson, Moa (2016). *Anomaly Detection In Time Series Data: a practical implementation for pulp and paper industry*.

Santafe, Guzman, Iñaki Inza, and Jose A. Lozano (2015). «Dealing with the evaluation of supervised classification algorithms». In: *Artificial Intelligence Review* 44.4, pp. 467–508. ISSN: 0269-2821. DOI: 10.1007/s10462-015-9433-y.

Schirrmeister, Robin Tibor et al. (2017). «Deep learning with convolutional neural networks for EEG decoding and visualization». In: *Human Brain Mapping* 38.11, pp. 5391–5420. ISSN: 10659471. DOI: 10.1002/hbm.23730.

Schmidhuber, Jürgen (2015). «Deep learning in neural networks: An overview». In: *Neural Networks* 61, pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

Schuster, M. and K.K. Paliwal (1997). «Bidirectional recurrent neural networks». In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681. ISSN: 1053587X. DOI: 10.1109/78.650093.

Scime, Luke and Jack Beuth (2018). «Anomaly detection and classification in a laser powder bed additive manufacturing process using a trained computer vision algorithm». In: *Additive Manufacturing* 19, pp. 114–126. ISSN: 22148604. DOI: 10.1016/j.addma.2017.11.009.

Serdio, Francisco et al. (2014). «Fault detection in multi-sensor networks based on multivariate time-series models and orthogonal transformations». In: *Information Fusion* 20, pp. 272–291. ISSN: 15662535. DOI: 10.1016/j.inffus.2014.03.006.

Serpanos, D (2018). «The Cyber-Physical Systems Revolution». In: *Computer* 51.3, pp. 70–73. DOI: 10.1109/MC.2018.1731058.

Shahrivari, Saeed (2014). «Beyond Batch Processing: Towards Real-Time and Streaming Big Data». In: *Computers* 3.4, pp. 117–129. ISSN: 2073-431X. DOI: 10.3390/computers3040117.

Shelhamer, Evan, Jonathan Long, and Trevor Darrell (2017). «Fully Convolutional Networks for Semantic Segmentation». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4, pp. 640–651. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2572683.

Sheskin, David J (2007). *Handbook of Parametric and Nonparametric Statistical Procedures*. 4th ed. Chapman & Hall/CRC. ISBN: 978-1439858011.

Shipmon, Dominique T. (2017). «Time Series Anomaly Detection; Detection of anomalous drops with limited features and sparse examples in noisy highly periodic data.» In: *CoRR* abs/1708.0.

Shrikumar, Avanti, Peyton Greenside, and Anshul Kundaje (2017). «Learning Important Features Through Propagating Activation Differences». In: *CoRR* abs/1704.0.

Shvachko, Konstantin, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. (2010). «The hadoop distributed file system.» In: *MSST*. Vol. 10, pp. 1–10.

Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman (2013). «Deep inside convolutional networks: Visualising image classification models and saliency maps». In: *arXiv preprint arXiv:1312.6034*.

Smilkov, Daniel, Nikhil Thorat, Been Kim, Fernanda B Viégas, and Martin Wattenberg (2017). «SmoothGrad: removing noise by adding noise». In: *CoRR* abs/1706.0.

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller (2014). «Striving for Simplicity: The All Convolutional Net». In: *arXiv preprint arXiv:1412.6806*.

Stripling, Eugen, Bart Baesens, Barak Chizi, and Seppe vanden Broucke (2018). «Isolation-based conditional anomaly detection on mixed-attribute data to uncover workers' compensation fraud». In: *Decision Support Systems* 111, pp. 13–26. ISSN: 01679236. DOI: 10.1016/j.dss.2018.04.001.

Štrumbelj, Erik and Igor Kononenko (2014). «Explaining prediction models and individual predictions with feature contributions». In: *Knowledge and Information Systems* 41.3, pp. 647–665. ISSN: 0219-1377. DOI: 10.1007/s10115-013-0679-x.

Sturm, Irene, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller (2016). «Interpretable deep neural networks for single-trial EEG classification». In: *Journal of Neuroscience Methods* 274, pp. 141–145. ISSN: 01650270. DOI: 10.1016/j.jneumeth.2016.10.008.

Sundararajan, Mukund, Ankur Taly, and Qiqi Yan (2017). «Axiomatic Attribution for Deep Networks». In: *arXiv preprint arXiv:1703.01365*.

Suthakar, Uthayanath, Luca Magnoni, David Smith, and Akram Khan (2019). «Optimised Lambda Architecture for monitoring scientific infrastructure». In: *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1. ISSN: 1045-9219. DOI: 10.1109/TPDS.2017.2772241.

Tanenbaum, Andrew S and Maarten Van Steen (2007). *Distributed systems: principles and paradigms*. Prentice-Hall.

Tang, Xiaoyan, Weiming Zeng, Yuhu Shi, and Le Zhao (2018). «Brain activation detection by modified neighborhood one-class SVM on fMRI data». In: *Biomedical Signal Processing and Control* 39, pp. 448–458. ISSN: 17468094. DOI: 10.1016/j.bspc.2017.08.021.

Tao, Fei and Qinglin Qi (2019). «New IT driven service-oriented smart manufacturing: Framework and characteristics». In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49.1, pp. 81–91. DOI: 10.1109/TSMC.2017.2723764.

Theissler, Andreas (2017). «Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection». In: *Knowledge-Based Systems* 123, pp. 163–173. ISSN: 09507051. DOI: 10.1016/j.knosys.2017.02.023.

Tong Zachary, Clinton Gormley (2015). *Elasticsearch: {The} {Definitive} {Guide}*. ISBN: 978-1-4493-5854-9.

BIBLIOGRAPHY

Toshniwal, Durga (2009). «Feature extraction from time series data». In: *Journal of Computational Methods in Sciences and Engineering* 9.s1, S99–S110. ISSN: 18758983. DOI: 10.3233/JCM-2009-0239.

Triguero, Isaac, Mikel Galar, Humberto Bustince, and Francisco Herrera (2017). «A first attempt on global evolutionary undersampling for imbalanced big data». In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 2054–2061. ISBN: 978-1-5090-4601-0. DOI: 10.1109/CEC.2017.7969553.

Triguero, Isaac, Daniel Peralta, Jaume Bacardit, Salvador García, and Francisco Herrera (2015). «MRPR: A MapReduce solution for prototype reduction in big data classification». In: *Neurocomputing* 150, pp. 331–345. ISSN: 09252312. DOI: 10.1016/j.neucom.2014.04.078.

Triguero, Isaac et al. (2018). «Transforming big data into smart data: An insight on the use of the k-nearest neighbors algorithm to obtain quality data». In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, e1289. ISSN: 19424787. DOI: 10.1002/widm.1289.

Tsironi, Eleni, Pablo Barros, Cornelius Weber, and Stefan Wermter (2017). «An analysis of Convolutional Long Short-Term Memory Recurrent Neural Networks for gesture recognition». In: *Neurocomputing* 268, pp. 76–86. ISSN: 09252312. DOI: 10.1016/j.neucom.2016.12.088.

Turner, Carlton Reid, Alfonso Fuggetta, Luigi Lavazza, and Alexander L Wolf (1999). «A conceptual basis for feature engineering». In: *Journal of Systems and Software* 49.1, pp. 3–15. DOI: 10.1016/S0164-1212(99)00062-X.

Turner, Ryan (2016). «A Model Explanation System: Latest Updates and Extensions». In: *CoRR* abs/1606.0.

Wang, Lihui, Martin Törngren, and Mauro Onori (2015). «Current status and advancement of cyber-physical systems in manufacturing». In: *Journal of Manufacturing Systems* 37, pp. 517–527. ISSN: 0278-6125. DOI: 10.1016/J.JMSY.2015.04.008.

Wang, Limin, Sheng Guo, Weilin Huang, Yuanjun Xiong, and Yu Qiao (2017). «Knowledge Guided Disambiguation for Large-Scale Scene Classification With Multi-Resolution CNNs». In: *IEEE Transactions on Image Processing* 26.4, pp. 2055–2068. ISSN: 1057-7149. DOI: 10.1109/TIP.2017.2675339.

Wang, Peng et al. (2016). «Semantic expansion using word embedding clustering and convolutional neural network for improving short text classification». In: *Neurocomputing* 174, pp. 806–814. ISSN: 09252312. DOI: 10.1016/j.neucom.2015.09.096.

Wang, Wei, Jiqiang Liu, Georgios Pitsilis, and Xiangliang Zhang (2018). «Abstracting massive data for lightweight intrusion detection in computer networks». In: *Information Sciences* 433-434, pp. 417–430. ISSN: 00200255. DOI: 10.1016/j.ins.2016.10.023.

Weinberg, Graham V. (2017). «Geometric mean switching constant false alarm rate detector». In: *Digital Signal Processing* 69, pp. 1–10. ISSN: 10512004. DOI: 10.1016/j.dsp.2017.06.015.

Weiss, Karl, Taghi M. Khoshgoftaar, and DingDing Wang (2016). «A survey of transfer learning». In: *Journal of Big Data* 3.1, p. 9. ISSN: 2196-1115. DOI: 10.1186/s40537-016-0043-6.

Wilber, Laura (2012). *A Practical Guide to Big Data: Opportunities, Challenges & Tools*. Tech. rep. EXALEAD.

Wu, Chunyang, Mark J. F. Gales, Anton Ragni, Penny Karanasou, and Khe Chai Sim (2018). «Improving Interpretability and Regularization in Deep Learning». In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.2, pp. 256–265. ISSN: 2329-9290. DOI: 10.1109/TASLP.2017.2774919.

Wu, Junfeng, Li Yao, and Bin Liu (2018). «An overview on feature-based classification algorithms for multivariate time series». In: *2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, pp. 32–38. ISBN: 978-1-5386-4301-3. DOI: 10.1109/ICCCBDA.2018.8386483.

Wu, Xindong, Xingquan Zhu, Gong-Qing Wu, and Wei Ding (2013). «Data mining with big data». In: *IEEE transactions on knowledge and data engineering* 26.1, pp. 97–107.

Xiao, Cao, Edward Choi, and Jimeng Sun (2018). «Opportunities and challenges in developing deep learning models using electronic health records data: a systematic review». In: *Journal of the American Medical Informatics Association* 25.10, pp. 1419–1428. ISSN: 1067-5027. DOI: 10.1093/jamia/ocy068.

Xun, Yaling, Jifu Zhang, Xiao Qin, and Xujun Zhao (2017). «FiDoop-DP: Data Partitioning in Frequent Itemset Mining on Hadoop Clusters». In: *IEEE Transactions on Parallel and Distributed Systems* 28.1, pp. 101–114. DOI: 10.1109/TPDS.2016.2560176.

Yang, Wenguang, Chao Liu, and Dongxiang Jiang (2018). «An unsupervised spatiotemporal graphical modeling approach for wind turbine condition monitoring». In: *Renewable Energy* 127, pp. 230–241. ISSN: 09601481. DOI: 10.1016/j.renene.2018.04.059.

Yoon, Heebum, Seungryong Kim, Taekho Nam, and JongWon Kim (2017). «Dynamic flow steering for IoT monitoring data in SDN-coordinated IoT-Cloud services». In: *2017 International Conference on Information Networking (ICOIN)*, pp. 625–627. DOI: 10.1109/ICOIN.2017.7899572.

Yue, Tao, Shaukat Ali, and Bran Selic (2015). «Cyber-physical System Product Line Engineering: Comprehensive Domain Analysis and Experience Report». In: *Proceedings of the 19th International Conference on Software Product Line*. SPLC '15. New York, NY, USA: ACM, pp. 338–347. ISBN: 978-1-4503-3613-0. DOI: 10.1145/2791060.2791067.

Zaharia, Matei, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica (2012). «Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters.» In: *HotCloud* 12, p. 10.

Zhang, Binghua et al. (2015). «Application of Synthetic NDVI Time Series Blended from Landsat and MODIS Data for Grassland Biomass Estimation». In: *Remote Sensing* 8.1, p. 10. ISSN: 2072-4292. DOI: 10.3390/rs8010010.

Zhang, Haijun, Jingxuan Li, Yuzhu Ji, and Heng Yue (2017). «Understanding Subtitles by Character-Level Sequence-to-Sequence Learning». In: *IEEE Transactions on Industrial Informatics* 13.2, pp. 616–624. ISSN: 1551-3203. DOI: 10.1109/TII.2016.2601521.

Zhang, Kai, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang (2017). «Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising». In: *IEEE Transactions on Image Processing* 26.7, pp. 3142–3155. ISSN: 1057-7149. DOI: 10.1109/TIP.2017.2662206.

Zhang, Quan-shi and Song-chun Zhu (2018). «Visual interpretability for deep learning: a survey». In: *Frontiers of Information Technology & Electronic Engineering* 19.1, pp. 27–39. ISSN: 2095-9184. DOI: 10.1631/FITEE.1700808.

Zhang, Yingfeng et al. (2015). «Real-time information capturing and integration framework of the internet of manufacturing things». In: *International Journal of Computer Integrated Manufacturing* 28.8, pp. 811–822.

Zhao, Bendong, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu (2017). «Convolutional neural networks for time series classification». In: *Journal of Systems Engineering and Electronics* 28.1, pp. 162–169. ISSN: 10044132. DOI: 10.21629/JSEE.2017.01.18.

Zhao, Bin, Xuelong Li, Xiaoqiang Lu, and Zhigang Wang (2018). «A CNN–RNN architecture for multi-label weather recognition». In: *Neurocomputing* 322, pp. 47–57. ISSN: 09252312. DOI: 10.1016/j.neucom.2018.09.048.

Zhao, Shuai, Mayanka Chandrashekar, Yugyung Lee, and Deep Medhi (2015). «Real-time network anomaly detection system using machine learning». In: *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, pp. 267–270. DOI: 10.1109/DRCN.2015.7149025.

Zhao, Wanqing, Hangzai Luo, Jinye Peng, and Jianping Fan (2017). «Spatial pyramid deep hashing for large-scale image retrieval». In: *Neurocomputing* 243, pp. 166–173. ISSN: 09252312. DOI: 10.1016/j.neucom.2017.03.021.

Zheng, Jinde, Haiyang Pan, and Junsheng Cheng (2017). «Rolling bearing fault detection and diagnosis based on composite multiscale fuzzy entropy and ensemble support vector machines». In: *Mechanical Systems and Signal Processing* 85, pp. 746–759. ISSN: 08883270. DOI: 10.1016/j.ymssp.2016.09.010.

Zheng, Pai et al. (2018). «Smart manufacturing systems for Industry 4.0: Conceptual framework, scenarios, and future perspectives». In: *Frontiers of Mechanical Engineering* 13.2, pp. 137–150. DOI: 10.1007/s11465-018-0499-5.

Zhong, Ray Y., Xun Xu, Eberhard Klotz, and Stephen T. Newman (2017). «Intelligent Manufacturing in the Context of Industry 4.0: A Review». In: *Engineering* 3.5, pp. 616–630. ISSN: 20958099. DOI: 10.1016/J.ENG.2017.05.015.

Zhou, Bolei, David Bau, Aude Oliva, and Antonio Torralba (2018). «Interpreting Deep Visual Representations via Network Dissection». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2018.2858759.

Zhou, Dao et al. (2016). «Distributed Data Analytics Platform for Wide-Area Synchrophasor Measurement Systems». In: *IEEE Transactions on Smart Grid* 7.5, pp. 2397–2405. ISSN: 1949-3053. DOI: 10.1109/TSG.2016.2528895.

Zhou, Yi, Fazhi He, Neng Hou, and Yimin Qiu (2018). «Parallel ant colony optimization on multi-core SIMD CPUs». In: *Future Generation Computer Systems* 79, pp. 473–487. ISSN: 0167739X. DOI: 10.1016/j.future.2017.09.073.

Zikopoulos, Paul, Chris Eaton, et al. (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media.