View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repository@Nottingham

Multi-Head CNN-RNN for Multi-Time Series Anomaly Detection: An industrial case study

Mikel Canizo[a], Isaac Triguero[b], Angel Conde[a], Enrique Onieva[c]

[a] *Ikerlan Technology Research Centre, P^o. J. M^a. Arizmendiarrieta, 2. 20500, Arrasate-Mondragón*
[b] *The Automated Scheduling Optimisation and Planning Research Group, School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, United Kingdom*
[c] *Deusto Institute of Technology (DeustoTech) & University of Deusto, Avenida de las Universidades 24, 48007, Bilbao*

Abstract

Detecting anomalies in time series data is becoming mainstream in a wide variety of industrial applications in which sensors monitor expensive machinery. The complexity of this task increases when multiple heterogeneous sensors provide information of different nature, scales and frequencies from the same machine. Traditionally, machine learning techniques require a separate data preprocessing before training, which tends to be very time-consuming and often requires domain knowledge. Recent deep learning approaches have shown to perform well on raw time series data, eliminating the need for pre-processing. In this work, we propose a deep learning based approach for supervised multi-time series anomaly detection that combines a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) in different ways. Unlike other approaches, we use independent CNNs, so-called convolutional heads, to deal with anomaly detection in multi-sensor systems. We address each sensor individually avoiding the need for data pre-processing and allowing for a more tailored architecture for each type of sensor. We refer to this architecture as Multi-head CNN-RNN. The proposed architecture is assessed against a real industrial case study, provided by an industrial partner, where a service elevator is monitored. Within this case study, three type of anomalies are considered: point, context-specific, and collective. The experimental results show that the proposed architecture is suitable for multi-time series anomaly detection as it obtained promising results on the real industrial scenario.

Keywords: Deep Learning, Anomaly detection, Convolutional Neural Networks, Recurrent Neural Networks, Multi-sensor systems, Industry 4.0

*Corresponding author
Email address: mcanizo@ikerlan.es (Mikel Canizo)

1. Introduction

Time series anomaly detection is a very relevant field in computer science and data mining [1, 2, 3]. It has become a necessity in the industrial scenario as undetected failures can lead to a critical damage [4]. Industrial machinery is prone to failure, meaning that an effective anomaly detection can improve system availability and reliability. This directly affects the productivity and reduces the operation and maintenance costs [5]. Thus, much research in this subject can be found in multiple real industrial scenarios such as automotive [6], manufacturing [7], energy [8], or industrial sensor networks [9].

An anomaly can be defined as an unusual pattern that does not conform to expected behavior. According to [1], time series anomalies can be categorized into three types: point, contextual, and collective. Point anomalies refer to a single instance of data being anomalous. Contextual anomalies are context-specific, that is, a given behavior might be common in a concrete scenario but abnormal on another. Collective anomalies refer to multiple instances of data that individually may have no relevance, but which as a group of events may become an anomaly.

In recent years, there have been advances in this topic due to the evolution of Industry 4.0 and the Internet of Things [10]. Technology has provided companies with more efficient and reliable monitoring systems [11]. In this way, industrial machinery is equipped with multiple sensors, which form a multi-sensor system [12]. These systems make data collection simpler and therefore, more data is now available in greater quantity and quality. As a consequence, there has been a proliferation of machine learning techniques for time series anomaly detection [6, 13, 14, 15].

Industrial machines are complex and often use a high number of sensors. In addition, many of these machines perform actions composed of multiple events, which makes it difficult to detect anomalies since it must be considered in which part of the time series each event occurs. Therefore, there might be different behaviors within a time series. It can be seen as context-specific anomalies within a single action. On the other hand, complex industrial machines have heterogeneous sensor systems meaning that they might be of a different nature, and thus, measuring differently scaled real value data or collecting data at different frequencies [16]. This fact often implies a previous data pre-processing to clean the data, extract meaningful features or reduce the dimensionality to convert this data into smart/usable data [17]. This is typically a very time-consuming task and it may require domain knowledge, that is, have knowledge about the characteristics of the data or about what an anomaly looks like. Managing data from heterogeneous sensor networks is a well known issue and many works can be found in the literature to deal with it, such as the use of a collaborative sparse representation framework [18], a deep multimodal encoder [19], a multi-view stacking method [20], or an ensemble pruning system [21].

Besides these challenges, there are some other inherent issues such as the fact that the boundary between normal and anomalous instances is often very thin, or that the data might contain noise due to sensor malfunctioning or

wrong measurements that may look similar to an anomaly. The imbalanced data problem [22] is also a common issue in anomaly detection scenarios since usually there are very few anomalous observations available and large volumes of normal observations. Variability is another key issue in industrial systems as it is common to change their sensor configuration [23]. Depending on the requirements of the moment, sensors are installed, modified or removed. As the sensor configuration changes, the model has to continuously adapt to the current configuration. In most cases, this involves retraining the model from scratch, which can take a lot of time and many computational resources. Recently, researchers focus on transfer learning (TL) [24] to transfer knowledge from one model to another to avoid having to create a new model from scratch. However, normally all sensor data is treated as a whole [25, 26, 27] and therefore the flexibility to adapt the model to new sensor configurations is limited.

In recent years, Deep Learning (DL) approaches have become the state of the art in time series modeling [28]. In particular, the combination of a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) have shown promising results in multi-time series classification problems [29, 30, 31] as they are able to work directly over raw data and thus no pre-processing nor additional domain knowledge is required. However, little research has been done in the anomaly detection domain using the CNN-RNN architecture. Hence, our aim is to investigate new techniques for supervised multi-time series anomaly detection based on this architecture. In this paper, we propose a new CNN-RNN architecture where the CNN is used to extract meaningful features from raw sensor data and the RNN is applied to learn temporal patterns. Unlike other approaches [26, 32, 33], we utilize an independent CNN to process each sensor data. Throughout the paper, we refer to each convolution as a convolutional head, thus forming a Multi-head CNN. Processing each sensor data on independent CNN 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. Furthermore, instead of processing the entire time series resulting from sensor data, they are divided into smaller portions using a sliding window. 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. We will refer to the proposed architecture as Multi-head CNN-RNN.

The main contributions of this work are described below:

- We introduce a new DL architecture specifically designed for supervised anomaly detection on multi-sensor systems. Little research has been done in this domain on CNN-RNN architecture and therefore, we aim to investigate new variations of this architecture to improve the anomaly detection. Moreover, our proposal works directly over raw data and thus no

pre-processing nor additional domain knowledge is required.

- As sensor data might be of a different nature on multi-sensor systems, we use independent convolutions for each one. Hence, each convolution can be adapted to the requirements of each sensor.
- To learn the specific events that occur within a time series, we propose a window based approach where each window can focus on each phase of the time series.
- To generate new models as the sensor configuration of an industrial system varies, we take advantage of the Multi-head convolutional architecture to generate a new model by transferring knowledge from one model to another, which is inspired in TL. As convolutional heads are fully independent of each other, we can easily add or remove heads if more sensors are installed or removed.
- Finally, since there exist many CNN and RNN layer types, we conduct a deep experimentation to analyze how they perform under different scenarios.

To analyze the performance of the proposed architecture, we conduct an extensive experimental study on a real industrial use case. The dataset contains about 14,000 simulations of a service elevator where 20 sensors are used to collect the data. The dataset is a two-class dataset which contains an anomaly rate of 20%. Regarding the anomalies, the dataset contains the three types, that is, point, context-specific, and collective anomalies.

This paper is structured as follows. Section 2 analyzes related works. Section 3 describes the proposed DL architecture. Section 4 details the experimental set up. Section 5 discusses the results obtained in the experimentation. Section 6 presents the conclusions and future work.

2. Background

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. Chandola et. al. presented a survey on anomaly detection that covers different domains [1]. Fu et al. presented another survey focused on data mining techniques [2].

In the anomaly detection field, as in other machine learning 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 [34]: supervised, semi-supervised, and unsupervised. In time series anomaly detection, the capability to detect anomalies in complex scenarios is another key factor as there are techniques that only consider univariate time series [35]. 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.

Within unsupervised techniques, multiple time series are typically managed by clustering them [36] or by calculating distances between time series [37]. Recently, artificial neural network (ANN) based approaches have arisen such as a Hierarchical Temporal Memory [13], a Restricted Boltzmann Machine [38], or a combination of Auto-Encoders and RNNs [39], where multiple time series are fed all together in the ANN. Although unsupervised techniques are more flexible than the others [40], most of them rely on the assumption that normal instances are far more frequent than abnormal ones [34]. However, this assumption does not always hold true. Moreover, since there are no labels in unsupervised learning, it is near impossible to get a reasonably objective measure of how accurate the algorithm is.

Within semi-supervised techniques, one-class Support Vector Machine is one of the most used in this category [41, 42]. However, modeling a normal region that captures all normal behavior is extremely difficult and the boundary between normal an abnormal is often blurred [34]. Recent approaches use DL algorithms such as Long-Short Term Memory (LSTM) to model 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 [43, 44, 45, 46, 47, 48]. However, all of these works require defining an error threshold to determine if the error is large enough to be considered an anomaly. 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 [49]. As in unsupervised methods, these all works also manage multiple time series by feeding them all together in the network.

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. Nevertheless, some approaches can be found such as the use of Support Vector Machines [50, 15], ensemble methods [51], or DL algorithms [52, 53]. The downside of these classification-based algorithms is that they suffer from the imbalanced data problem [54] since in the anomaly detection field there is much more data related to normal behavior than to the anomalous. Hence, over-sampling [55] and under-sampling [56] 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 [57, 58]. The difficulty increases in the multi-time series domain as values of a given time series might be affected by the others. Furthermore, the majority of analyzed techniques require previous data pre-processing to reduce dimensionality or to extract relevant features, among others. This often requires domain knowledge and it is time-consuming. As discussed before, all the time series are typically processed all together which, in the case of heterogeneous

multi-time series, 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.

DL techniques, concretely CNN and RNN, have become the state of art on time series modeling [28]. Furthermore, their combination has attracted the attention of many researchers as it is capable of working directly on raw sensor data in multi-sensor environments [29], thus addressing one of the described challenges as no pre-processing is needed. The CNN-RNN architecture has achieved promising results in various domains such as speech recognition [30], gesture recognition [31], weather recognition [59], or emotion detection [60]. However, the imbalanced data problem and the feature extraction in heterogeneous multi-time series are still challenges to be addressed. Therefore, the motivation of this work is to investigate new supervised techniques for anomaly detection in multi-time series by implementing an adapted CNN-RNN architecture that processes all the time series individually, and can perform well in imbalanced data scenarios.

3. A Multi-head CNN-RNN architecture for multi-sensor systems

This section details the proposed Multi-head CNN-RNN architecture. We first introduce the proposed architecture (Section 3.1). Afterward, we provide a detailed explanation of both CNN (Section 3.2) and RNN (Section 3.2) sides of the architecture.

3.1. General overview of the Multi-head CNN-RNN

Figure 1 shows the general overview of the Multi-head CNN-RNN architecture. This architecture combines convolutional and recurrent layers.

The Multi-head convolution is a CNN where each time series is processed on a fully independent convolution, so-called convolutional heads. It is responsible for extracting meaningful features from sensor data. In many industrial scenarios, machines have installed multiple sensors that are independent of each other and thus they might not be correlated. Often, these sensors conform an heterogeneous sensor system meaning that they might capture data of different natures and real value scales, or even at different frequencies. Hence, it is reasonable to treat them in an independent way. 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 extraction of characteristics would be done according to the entire time series, thus leaving possible key features of each phase uncaptured.

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 behaviour 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 consideration. In this article, we analyze all the possibilities to find the architecture that best adapts to our use case. See Sections 4 and 5 for more details.

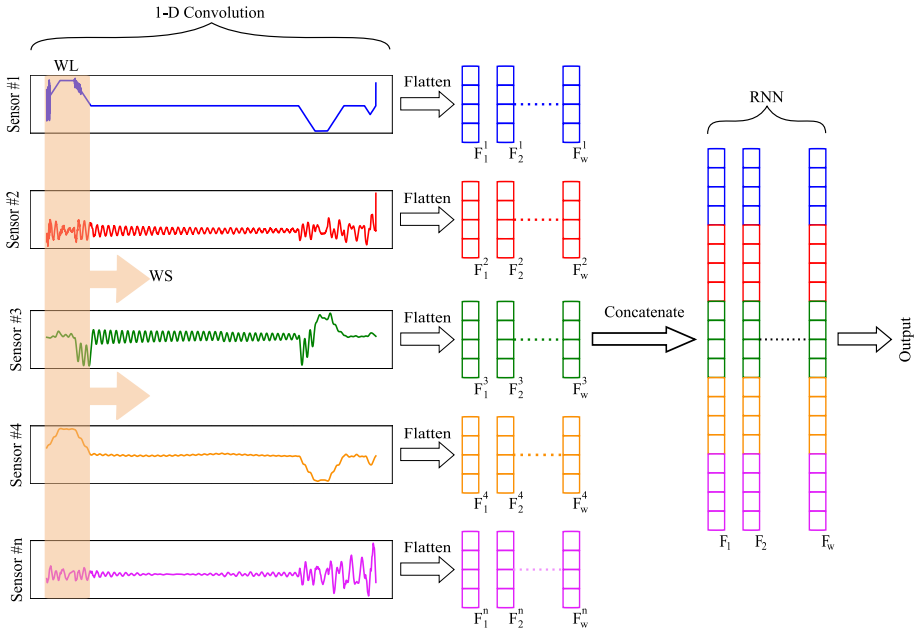


Figure 1: Multi-head CNN-RNN architecture for multi-time series anomaly detection. From the left, data coming from sensors are individually processed by independent convolutional heads by means of a window W of length W_L. The window slides over the time series with a step of size W_S. A feature map 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 recurrent layers yield the classification outcome by processing all the windows chronologically.

3.2. 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 [32, 33, 26]. Throughout this paper, 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 in the number of parameters of convolutional architectures. The number of parameters of each layer on a traditional Multi-channel CNN is computed using Equation 1.

$$p = F_N \cdot K_S \cdot P_{PL} + bias \tag{1}$$

where F_N denotes the number of filters, K_S the kernel size, P_{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 [29]. 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 on Equation 2.

$$W_N = \frac{S_L - W_L}{W_S} + 1 \tag{2}$$

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 3.

$$input\_dim = (n\_samples, W_N, W_L, n\_channels) \tag{3}$$

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

3.3. RNN

RNNs are DL architectures with the ability to remember the 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 a 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.

$$input\_dim = (n\_samples, W_N, P_{PL}) \tag{4}$$

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. Experimental framework

This section describes the configuration and properties related to the experimentation followed in this article. We show the measures used to benchmark

the performance of the models (Section 4.1), the industrial use case and the properties of the dataset used (Section 4.2), the parameters and the base classifiers (Section 4.3), and finally, a description of the non-parametric statistical methods used to compare the results obtained (Section 4.4).

4.1. Performance measures

In this section, we analyze the metrics used to measure the performance of the DL architectures studied in the experimentation. As this paper 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) [61, 62, 63].

As standard classification methods, the performance of these classifiers can be measured in terms of precision and recall. Nonetheless, Precision-Recall Curve (PRC) is used since it is a plot that summarizes the trade-off between the precision and the recall using different probability thresholds. Often, Receiver Operating Characteristic (ROC) curve is used instead of the PRC [64]. However, it is demonstrated that the PRC is more informative at the time of evaluating binary classifiers on imbalanced datasets [54]. 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, an Average Precision (AP) is 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 5.

$$AP = \sum_{n} (R_n - R_{n-1}) P_n \tag{5}$$

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

In imbalanced data scenarios, the geometric mean is another extensively used metric [65, 66, 67]. It is the geometric mean between precision and recall. It is computed as in Equation 6.

$$g\_mean = \sqrt{\frac{TP}{TP + FN} \cdot \frac{TN}{TN + FP}} \tag{6}$$

These metrics are calculated from a confusion matrix, which displays the crossing correct and wrong predictions between pairs of categories (classes) [68]. It represents True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) undertaken by the system.

4.2. Industrial case study

The proposed architecture is validated in a real industrial scenario in which the operating status of a service elevator is monitored. The elevator is monitored

by 20 sensors that record data at a frequency of 500 Hz, thus obtaining 20 univariate time series which values are of different scales. Since obtaining the sufficient amount of real data that matches all possible scenarios of normal and anomalous activity is hard and time-consuming, we have generated a dataset by means of a physical model, developed by domain experts, that represents accurately the behavior of the service elevator. Hence, we have been able to simulate a huge volume of different service elevator journeys. In this way, the dataset used in our experiments comprises a set of simulations of a service elevator. Each simulation represents an elevator journey of fixed length, both uphill or downhill. Each simulation is composed of 20 univariate time series corresponding to 20 sensors that record data at a frequency of 500 Hz, according to the real scenario. Table I summarizes them. Each time series has a fixed length of 8,000 data points. Overall, there are about 14,000 instances within the dataset corresponding to two type of classes: "*normal*" as the negative, and "*anomaly*" as the positive.

The physical model used to mimic the real behavior of the elevator includes all its relevant subsystems such as the cabin, the counterweight, the guiding system, the electric machine, or the driving pulley. Each of these subsystems is highly configurable to enable simulating several scenarios, which varies depending on the input parameters introduced to the model. To generate anomalies, different faults have been included as input parameters. Namely, faults in the guiding system (reduced lubrication, misalignment) and in the electric machine (de-magnetization, lose of inductance) were 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 were included in the model.

The dataset contains a 20% of positive instances. 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 are labeled by domain experts.

To test the behavior of the different architectures under different imbalanced ratios, four different datasets are generated based on the original one. These new datasets contain a 15%, 10%, 5%, and 3% of anomalies, respectively. The selection of anomalies for each dataset is done randomly.

Regarding the training of the architectures analyzed in this paper, all the datasets are split into train/validation/test sets, with a ratio of 60/20/20%. The considered datasets are partitioned using a Stratified Shuffle Split (SSS) cross-

validator (obtained from scikit-learn[1]), which splits the dataset into train/test sets and returns stratified randomized folds. As SSS only splits the dataset into two sets, we first partition the dataset into train/test with a ratio of 60/40. Afterward, the test set is divided into two equally sized sets to generate the validation set. The folds generated by the SSS preserve the percentage of samples for each class. In the experimentation, the SSS is applied 10 times for each model and thus, each of them is 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.

Description
Angular acceleration of the pulley
Lateral acceleration of cabin on X, Y, and Z axis
Tension on the cabin's and counterweight cable
Cabin and counterweight friction
Force on the support of the machine-pulley
Direct and quadrature power
Angular speed of the pulley
Angular position of the pulley
Vertical acceleration of the pulley
Cabin and Counterweight speed
Direct and quadrature voltage
Cabin and counterweight position

Table I: Summary of the variables of the dataset. Note that they are grouped by sensor type. There are a total of 20 variables.

4.3. Parameters and base classifiers

In this section we describe the architecture of all the neural networks used in this experiments. As described in Section 3, the proposed architecture has a one-dimension convolutional part followed by a recurrent part. Regarding the convolutional side, three types of layers are considered: *Multi-channel Conv1d*, *Multi-head Conv1d*, and *Multi-head LC1d*. Recently, the Multi-channel architecture has been used in several works [29, 31, 60] and thus we use it as baseline. On the other hand, five types of recurrent layers are examined: *RNN* [28], *LSTM* [69], *GRU* [70], *Bi-LSTM*, and *Bi-GRU*. All possible combinations between convolutional and recurrent layers are analyzed in the experimentation.

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

Thus, there are a total of 15 architectures. A brief description of each layer is presented below:

- **Convolutional layers**:
  - **Conv1d**: A one-dimension convolutional layer is mainly used to process 1D vector data such is the case of signal processing. 1D convolutions slide the kernel over the input vector, that is, moving the kernel in one dimension [71].
  - **LC1D**: Is similar to Conv1d. However, instead of using a sliding kernel that moves over the entire vector, LC1D uses multiple static kernels. Thus, it uses a single kernel for each different patch of the input [72]. Although this layer is not purely a convolutional layer but behaves similarly to it, in this paper we include the LC1d layer within the group of convolutions.
  - **Multi-head and Multi-channel**: They are variants of previously described convolutional layers. Multi-head convolutions use independent single-channel convolution branches to process each sensor data. The branches are defined as heads. In contrast, Multi-channel convolutions use a single convolutional head with multiple channels. There are as many channels as there are sensor data.
- **Recurrent layers:**
  - **RNN**: Is designed to recognize patterns in sequences of data. Unlike other types of layers, it takes as input not just the current input, but also what it has perceived previously in time. Thus, its output is not only influenced by actual events but also by past events. However, RNNs suffer from the vanishing and exploiting gradients problem and thus they can be hard to train.
  - **LSTM**: Is a variation of the RNN layer that solves the vanishing/exploding gradient problem. It uses a state cell that runs straight down the entire chain, with only some minor linear interactions. Thus, the information flows along it unchanged. The LSTM is able to remove or add information to the state cell. For that matter, it uses input, output, and forget gates.
  - **GRU**: It can be considered as a simplification of the LSTM. The main difference between them is how they modify the state cell. GRU uses two gates instead of three. Therefore, it uses an update and a reset gate. Although GRU is simpler than LSTM, it can achieve the same or better results in some use cases [73].
  - **Bi-LSTM and Bi-GRU**: They add a bi-directional modality to LSTM and GRU layers [74]. These layers process input data in chronological order. However, in bi-directional mode, data is processed in both chronological and reverse order.

The configuration of the architectures is divided into two groups: Multi-head and Multi-channel. The former uses independent single-channel convolutional heads to process each time series separately (Figure 2a). The latter uses a single convolutional head with multiple channels to process all the time series (Figure 2b). 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 architecture remains the same. Note that the Multi-channel architecture serves of comparison approach.

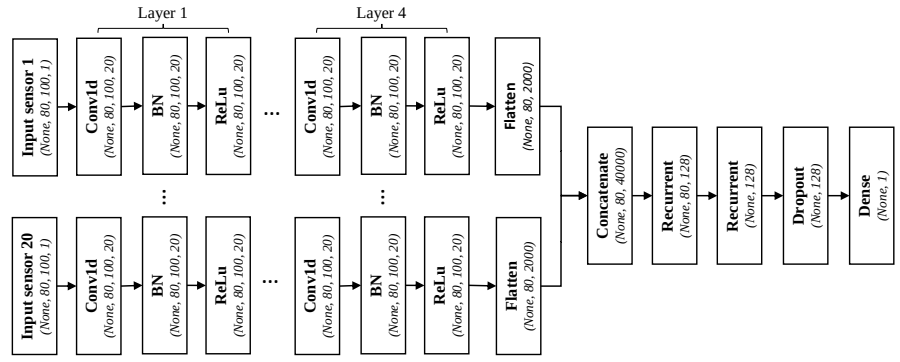
The convolutional part of the architectures is composed of four stacked one-dimensional convolutional layers. Each convolution applies a *Batch Normalization* (BN) layer [75] followed by a *ReLU* [76] activation layer. The inclusion of the BN layer is particularly important as it reduces the internal covariance shift. This brings a regularization effect between batches and makes training faster [77]. Unlike other CNN's for time series classification [78, 32, 79], no subsampling 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.

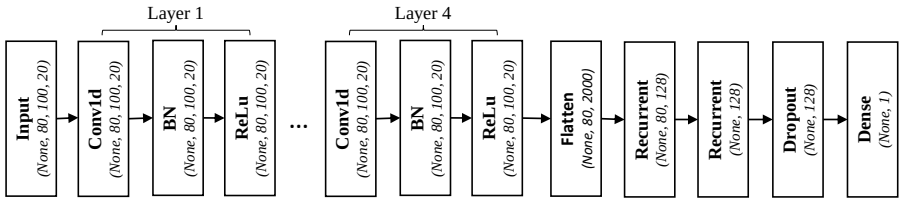
The training process of these architectures is conducted in a fully-supervised way using back-propagation to adjust gradients from the final dense layer to the initial convolutional layers. As architectures are designed for a binary classification task, Binary cross-entropy [80] loss function is used. Regarding the optimizer, Adam [81] is used since it obtained the most stable results among those we tested. To find the best values for hyper-parameters, a grid search is used. Table II 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 are used and gradients are computed after each batch. As architectures containing LC1d layers require more computational resources, batches of 10 instances are used to avoid memory problems at the time of training. Furthermore, a maximum epoch number is set to 30. However, an early stop method is used to stop the training process in case it converges before reaching the defined maximum number of epochs. In this way, the training set is used to train the architectures while the validation set is used to tune them. Finally, the test set is used to check their performance against unseen data.

All the experiments have been executed on a single computer with the following characteristics:

- GPU: Titan V



(a) Multi-head



(b) Multi-channel

Figure 2: 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.

Parameter	Value
conv. filters	20
recurrent units	128
dropout	0.25
window length	100 (It varies on Section 5.4)
window step	100 (No overlapping windows)
learning rate	0.00001
epochs	30 (with early stop)
batch size	50 (10 for LC1d architectures)

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

- 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

4.4. Statistical test for performance comparison

In order to provide statistical support to the results obtained in the experimentation, a hypothesis testing technique is used [82]. In particular, a non-parametric test is used since parametric tests might lose credibility because the initial conditions guaranteeing their reliability may not be satisfied (i.e., independence, normality, and homoscedasticity) [83].

In this paper, we use these tests in cases where multiple datasets are used (Section 5.3). For these cases, a Friedman test [82, 84] is used to analyze whether statistical differences exist between them. Furthermore, Holm post hoc test [84] is used to identify which of the architectures are distinctive among a 1 x n comparison. In this way, a given hypothesis can be rejected at a specified level of significance α (We will use $\alpha = 0.05$). Hence, the adjusted p-value (APV) is computed for each comparison which denotes the lowest level of significance of a hypothesis to be rejected. In addition, the architectures are classified by means of a ranking that determines how good are each of them in comparison to the others. The positions of the ranking are assigned by computing the average performance obtained by each architecture in all the tested datasets. The architecture obtaining the best APV will have the best ranking.

These tests are widely used in the field of machine learning as can be shown in [82, 84, 85, 86], where they recommend their use. A more detailed explanation can be found in [87].

5. Results and Discussion

In this section we analyze and discuss the results of the experimentation. First, we perform a comparison between using a Multi-head or a Multi-channel convolution architecture (Section 5.1). Second, we analyze the difference between using a Conv1d or a LC1d as convolutional layers (Section 5.2). Third, we analyze the performance of the architectures as the percentage of anomalies within the dataset decreases (Section 5.3). Fourth, we evaluate the impact that the window length has on the performance of the architectures (Section 5.4). Finally, we test the performance of Multi-head architectures to transfer knowledge from one model to another to adequate them to changing scenarios (Section 5.5). Note that only Multi-head Conv1d like architectures are considered in the last two experiments.

5.1. 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., [29, 31, 60]) is performed. The aim of the convolutional part of the proposed architecture is to extract the most relevant characteristics from sensor data.

Thus, we analyze the performance of the entire architecture based on how the features of sensors data are extracted, independently or as a group. We also discuss the impact that the different recurrent layers have in these architectures. Note that in this experimentation, we only consider the Conv1d layer as convolutional layer.

One of the main differences between both architectures is the number of features extracted from each window. Table III details the shape of the output vector for 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 IV 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 3a and 3b 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 achieves 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 take a decision. Consequently, we conducted another experiment in which we matched for Multi-channel architectures the same number of extracted features as Multi-head architectures have. To do so, we set $F_N = 400$. 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 is the fact of extracting the features of each sensor independently rather than in extracting a larger number of features.

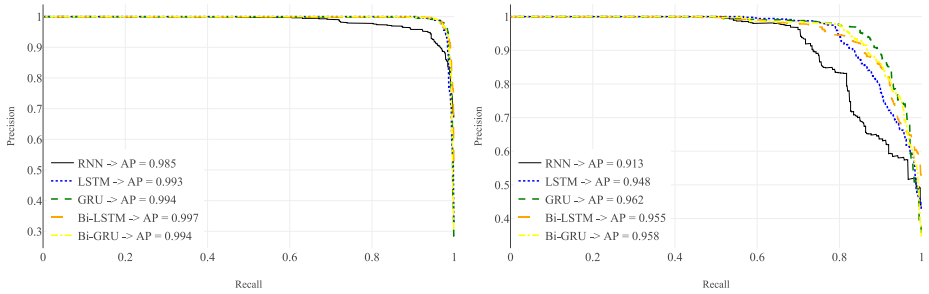
Thus, we can observe how the proposed architecture outperforms the Multi-channel CNN-RNN existing in the literature.

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 III: 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 IV: 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.



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

Figure 3: PRC of the different architectures using Multi-head and Multi-channel convolutions. Experiments conducted with the original dataset.

5.2. Multi-head Conv1D vs Multi-head LC1d

In this section we compare Conv1d and LC1d layers, 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.

In the industrial case study presented in this paper, 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

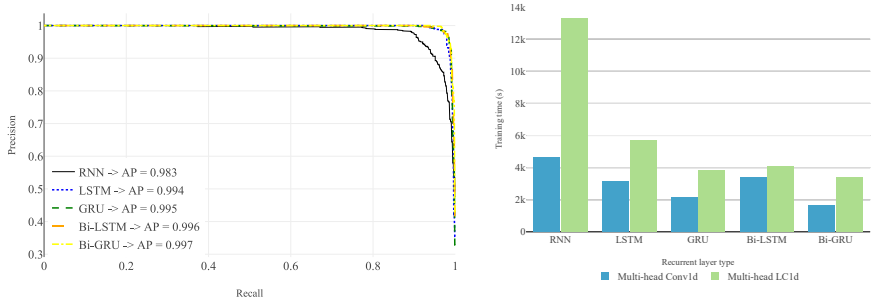
use a unique filter for each of the patches within a window. Thus, filters in LC1d layers do not share weights with each other and they only focus on their corresponding patch of the window.

Table V shows the results obtained for all the architectures. The best scores for each recurrent layer are highlighted in bold. Figures 3a and 4a depict the PRC of both Multi-head Conv1d and Multi-head LC1d architectures, respectively. Except for 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 4b, Conv1d layer is, generally, x2 times faster at training time. In fact, Conv1d layer is more than x3 times faster than LC1D layer in case of combining them with RNN. 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 V: 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: Results of Multi-head LC1d architectures. Experiments conducted with the original dataset.

5.3. Performance analysis with reduction of anomalies

In many use cases, the percentage of anomalies is often lower than in our original dataset (20%). Thus, we have conducted an experiment where the percentage of anomalies is decreased to 15%, 10%, 5%, and 3%. In this study, all architectures of previous sections (Section 5.1 and 5.2) are included. Hence, this experiment analyzes the performance of all architectures as the ratio of anomalies decreases.

Table VI 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 previous pre-processing. Focusing on each architectures, 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 VII shows the average ranking calculated for all the architectures according to *g_mean* metric and its corresponding APV, which is calculated by means of the Holm's test. Note that the results obtained for all the datasets are 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 to point out that if we set a standard level of significance of $\alpha = 0.05$, 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 comparing to other Multi-head architectures. Therefore, we do not recommend its use.

Table VII 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, we performed another statistical test including only Multi-head architectures to analyze their differences. Table VIII 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 shown that the architectures including the RNN layer obtained poor performance and thus they are rejected.

% 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	*0.994 ± 0.001*	0.987 ± 0.003	0.965 ± 0.006	0.929 ± 0.014
		g_mean	0.977 ± 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	*0.994 ± 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	*0.989 ± 0.002*	*0.974 ± 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	*0.946 ± 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 VI: Comparison of all architectures as the ratio of anomalies decreases. The results were taken as averages over 10 runs. We report the corresponding standard deviation 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 VII: 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 $\alpha = 0.05$

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 VIII: 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 $\alpha = 0.05$

5.4. Analysis of the window length

On the proposed architecture, the window length is one of he parameters to take into account. For this reason, in this experiment we analyze the performance of the architecture as the window length varies. 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 are as follows: $W_L = \{25, 50, 80, 100, 125, 160, 200\}$.

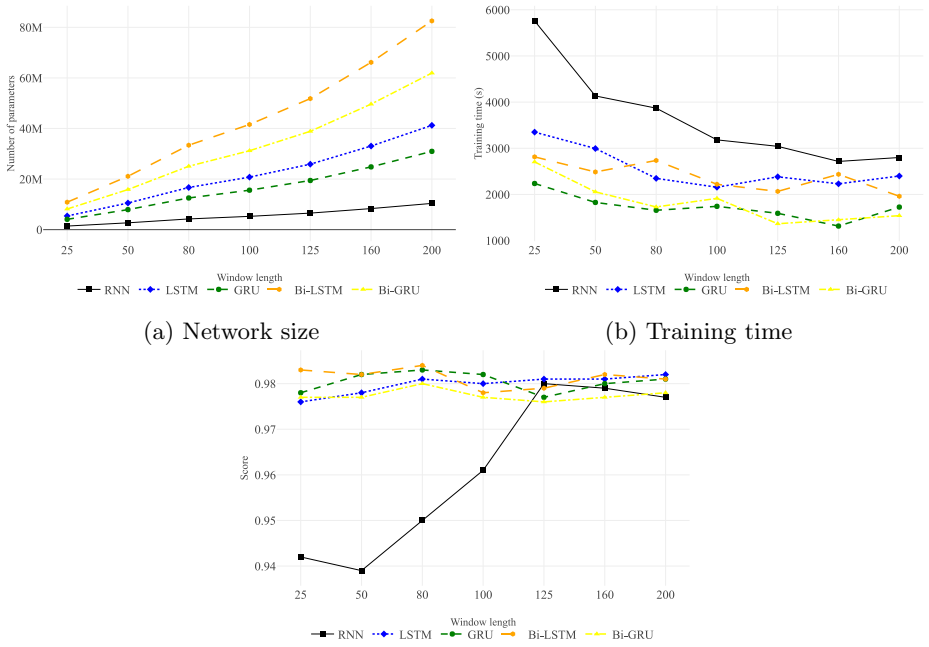
The first insight is determined by the correlation between the network size and the required training time as the window length varies. Figure 5a shows

how the size of the network increases as the length of the window grows. However, Figure 5b 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 architecture that includes the RNN layer has the smallest number of parameters. However, it is the slowest 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 5c shows that, except for the RNN, the performance of the models remains stable regardless of the length of the window. The RNN achieves 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.

5.5. Transfer ability

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 this experiment, the ability of the proposed architecture to adapt to these changes is analyzed. Instead of training a new model from scratch, this model is generated based on an already trained model, which has been trained with other sensor configuration. Thus, less time and computational resources would be required. In this approach, we benefit from the Multi-head architecture where each sensor data is processed on a fully independent convolutional head. Therefore, it is easy to add, remove or modify a convolutional head on a trained model. As an example, let's say we have a model trained with multiple sensor data. Now, a new sensor is installed. Hence, a new architecture is generated by adding a convolutional head to the previous one. Next, the knowledge (weights) of the convolutional heads corresponding to the previous sensor data is transferred to the new architecture. The weights of these convolutional heads are frozen as they are already trained. Hence, at the time of training the new architecture, only the additional convolutional head is trained in addition to the recurrent part of the architecture. This is inspired by TL as the underlying idea is to transfer knowledge from one model to another [24]. However, in TL only the first layers of the trained network are typically used to transfer knowledge from one model to another. Then, the new network is trained by freezing or fine-tuning these layers. Conversely, in our proposal we



(a) Network size

(b) Training time

(c) *g_mean*

Figure 5: Performance of the Multi-head Conv1d architectures as the window length varies. Experiments conducted with the original dataset.

take all the convolutional layers, which are then always frozen, corresponding to each of the sensors that are transferred from one model to another.

To demonstrate the performance of the proposed architecture to adjust to new sensor configurations, the following experiments are conducted:

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

To differentiate the models used in this experimentation, we define as base models those that have been generated from scratch (10, 15, and 20). Models generated from base models, by adding or removing sensor data, are defined as transfer knowledge based (TKB) models. As long as TKB models are 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), we

define these models as target models. Therefore, base models also actuate as target models. We compare TKB models 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 serve as baselines. The comparison is carried out in terms of training time and performance (metrics).

Table IX 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 6 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 x2 times faster than target models at training time. TKB models with removed sensor data achieve better results than target models. In fact, they obtain 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 are cases where TKB modes are 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.

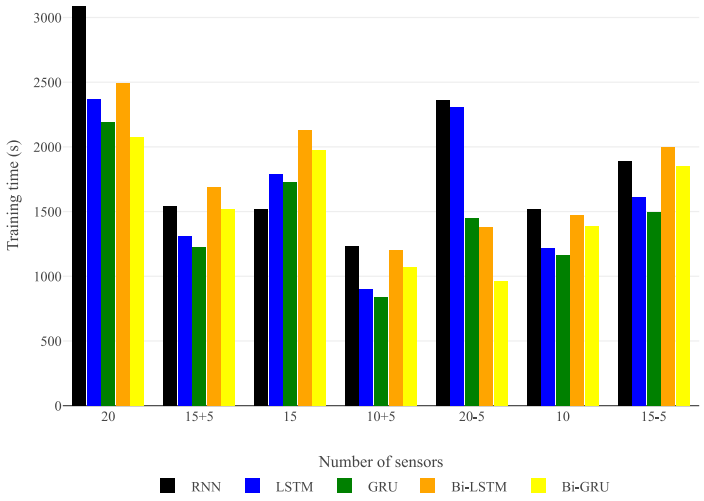


Figure 6: Training time of both base models and TKB models. They are all grouped by the number of sensors used.

# sensors	20	15+5	15	10+5	20-5	10	15-5
RNN	$\mathbf{0.961 \pm 0.021}$	0.946 ± 0.013	0.924 ± 0.013	0.905 ± 0.017	$\mathbf{0.934 \pm 0.013}$	0.871 ± 0.020	$\mathbf{0.886 \pm 0.008}$
LSTM	$\mathbf{0.980 \pm 0.002}$	0.954 ± 0.011	0.923 ± 0.011	0.907 ± 0.011	$\mathbf{0.930 \pm 0.011}$	0.873 ± 0.054	$\mathbf{0.897 \pm 0.010}$
GRU	$\mathbf{0.977 \pm 0.002}$	0.957 ± 0.019	0.933 ± 0.010	0.922 ± 0.007	$\mathbf{0.941 \pm 0.005}$	0.886 ± 0.017	$\mathbf{0.914 \pm 0.009}$
Bi-LSTM	$\mathbf{0.978 \pm 0.004}$	0.963 ± 0.021	0.947 ± 0.018	0.916 ± 0.011	$\mathbf{0.952 \pm 0.004}$	0.884 ± 0.023	$\mathbf{0.910 \pm 0.010}$
Bi-GRU	$\mathbf{0.977 \pm 0.004}$	0.964 ± 0.008	0.958 ± 0.023	0.951 ± 0.007	$\mathbf{0.963 \pm 0.002}$	0.867 ± 0.016	$\mathbf{0.926 \pm 0.013}$

Table IX: *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.

6. Conclusions

We have 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 a several types of layers, all possible alternatives have been analyzed under different scenarios.

The experimental results have shown that our approach is able to detect anomalies over multi-time series in an effective way, since its performance maintains high even if the percentage of anomalies within the training dataset is reduced up to a 3%. Furthermore, we have proven through statistical tests that the proposed Multi-head CNN architecture outperforms the traditional 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. Little research has been conducted in 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, we have validated the proposed architecture 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.

Acknowledgement

The authors wish to express their thanks to the Basque Government for their financial support of this research through the Elkartek program under the TEKINTZE project (Grant agreement No. KK-2018/00104). Any opinions, findings and conclusions expressed in this article are those of the authors and do not necessarily reflect the views of funding agencies.

References

[1] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Computing Surveys 41 (3) (2009) 15:1–15:58. doi:10.1145/1541880.1541882.

[2] T.-c. Fu, A review on time series data mining, Engineering Applications of Artificial Intelligence 24 (1) (2011) 164–181. doi:10.1016/J.ENGAPPAI.2010.09.007.

[3] R. A. Ariyaluran Habeeb, F. Nasaruddin, A. Gani, I. A. Targio Hashem, E. Ahmed, M. Imran, Real-time big data processing for anomaly detection: A Survey, International Journal of Information Management- doi:10.1016/j.ijinfomgt.2018.08.006.

[4] J. Yan, Y. Meng, L. Lu, L. Li, Industrial Big Data in an Industry 4.0 Environment: Challenges, Schemes, and Applications for Predictive Maintenance, IEEE Access 5 (2017) 23484–23491. doi:10.1109/ACCESS.2017.2765544.

[5] H. M. Hashemian, W. C. Bean, State-of-the-Art Predictive Maintenance Techniques*, IEEE Transactions on Instrumentation and Measurement 60 (10) (2011) 3480–3492. doi:10.1109/TIM.2009.2036347.

[6] A. Theissler, Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection, Knowledge-Based Systems 123 (2017) 163–173. doi:10.1016/j.knosys.2017.02.023.

[7] L. Scime, J. Beuth, Anomaly detection and classification in a laser powder bed additive manufacturing process using a trained computer vision algorithm, Additive Manufacturing 19 (2018) 114–126. doi:10.1016/j.addma.2017.11.009.

[8] W. Yang, C. Liu, D. Jiang, An unsupervised spatiotemporal graphical modeling approach for wind turbine condition monitoring, Renewable Energy 127 (2018) 230–241. doi:10.1016/j.renene.2018.04.059.

[9] A. Cenedese, M. Luvisotto, G. Michieletto, Distributed Clustering Strategies in Industrial Wireless Sensor Networks, IEEE Transactions on Industrial Informatics 13 (1) (2017) 228–237. doi:10.1109/TII.2016.2628409.

[10] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, M. Guizani, Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges, IEEE Wireless Communications 24 (3) (2017) 10–16. doi:10.1109/MWC.2017.1600421.

[11] R. Y. Zhong, X. Xu, E. Klotz, S. T. Newman, Intelligent Manufacturing in the Context of Industry 4.0: A Review, Engineering 3 (5) (2017) 616–630. doi:10.1016/J.ENG.2017.05.015.

[12] P. Gołuch, J. Kuchmister, K. Ćmielewski, H. Bryś, Multi-sensors measuring system for geodetic monitoring of elevator guide rails, Measurement 130 (2018) 18–31. doi:10.1016/j.measurement.2018.07.077.

[13] S. Ahmad, A. Lavin, S. Purdy, Z. Agha, Unsupervised real-time anomaly detection for streaming data, Neurocomputing 262 (2017) 134–147. doi:10.1016/j.neucom.2017.04.070.

[14] S. Torkamani, V. Lohweg, Survey on time series motif discovery, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 7 (2) (2017) e1199. doi:10.1002/widm.1199.

[15] J. Zheng, H. Pan, J. Cheng, Rolling bearing fault detection and diagnosis based on composite multiscale fuzzy entropy and ensemble support vector machines, Mechanical Systems and Signal Processing 85 (2017) 746–759. doi:10.1016/j.ymssp.2016.09.010.

[16] F. Cauteruccio, G. Fortino, A. Guerrieri, A. Liotta, D. C. Mocanu, C. Perra, G. Terracina, M. Torres Vega, Short-long term anomaly detection in wireless sensor networks based on machine learning and multi-parameterized edit distance, Information Fusion 52 (2019) 13–30. doi:10.1016/j.inffus.2018.11.010.

[17] I. Triguero, D. García-Gil, J. Maillo, J. Luengo, S. García, F. Herrera, Transforming big data into smart data: An insight on the use of the k-nearest neighbors algorithm to obtain quality data, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery (2018) e1289doi:10.1002/widm.1289.

[18] M. Dao, N. H. Nguyen, N. M. Nasrabadi, T. D. Tran, Collaborative Multi-Sensor Classification Via Sparsity-Based Representation, IEEE Transactions on Signal Processing 64 (9) (2016) 2400–2415. doi:10.1109/TSP.2016.2521605.

[19] Z. Liu, W. Zhang, S. Lin, T. Q. Quek, Heterogeneous Sensor Data Fusion By Deep Multimodal Encoding, IEEE Journal of Selected Topics in Signal Processing 11 (3) (2017) 479–491. doi:10.1109/JSTSP.2017.2679538.

[20] E. Garcia-Ceja, C. E. Galván-Tejada, R. Brena, Multi-view stacking for activity recognition with sound and accelerometer data, Information Fusion 40 (2018) 45–56. doi:10.1016/j.inffus.2017.06.004.

[21] J. Cao, W. Li, C. Ma, Z. Tao, Optimizing multi-sensor deployment via ensemble pruning for wearable activity recognition, Information Fusion 41 (2018) 68–79. doi:10.1016/j.inffus.2017.08.002.

[22] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, F. Herrera, Learning from Imbalanced Data Sets, Springer, 2018. doi:10.1007/978-3-319-98074-4.

[23] A. Iglesias, H. Lu, C. Arellano, T. Yue, S. Ali, G. Sagardui, 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, 2017, Volume A, Sevilla, Spain, September 25-29, 2017, 2017, pp. 195–204. doi:10.1145/3106195.3106223.

[24] K. Weiss, T. M. Khoshgoftaar, D. Wang, A survey of transfer learning, Journal of Big Data 3 (1) (2016) 9. doi:10.1186/s40537-016-0043-6.

[25] R. Saeedi, A. Gebremedhin, A Signal-Level Transfer Learning Framework for Autonomous Reconfiguration of Wearable Systems, IEEE Transactions on Mobile Computing (2018) 1–1doi:10.1109/TMC.2018.2878673.

[26] N. Mohammadian Rad, S. M. Kia, C. Zarbo, T. van Laarhoven, G. Jurman, P. Venuti, E. Marchiori, C. Furlanello, Deep learning for automatic stereotypical motor movement detection using wearable sensors in autism spectrum disorders, Signal Processing 144 (2018) 180–191. doi:10.1016/j.sigpro.2017.10.011.

[27] W. Qian, S. Li, J. Wang, A New Transfer Learning Method and its Application on Rotating Machine Fault Diagnosis Under Variant Working Conditions, IEEE Access 6 (2018) 69907–69917. doi:10.1109/ACCESS.2018.2880770.

[28] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. doi:10.1038/nature14539.

[29] F. Ordóñez, D. Roggen, Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition, Sensors 16 (1) (2016) 115. doi:10.3390/s16010115.

[30] Y. Qian, M. Bi, T. Tan, K. Yu, Very Deep Convolutional Neural Networks for Noise Robust Speech Recognition, IEEE/ACM Transactions on Audio, Speech, and Language Processing 24 (12) (2016) 2263–2276. doi:10.1109/TASLP.2016.2602884.

[31] E. Tsironi, P. Barros, C. Weber, S. Wermter, An analysis of Convolutional Long Short-Term Memory Recurrent Neural Networks for gesture recognition, Neurocomputing 268 (2017) 76–86. doi:10.1016/j.neucom.2016.12.088.

[32] B. Zhao, H. Lu, S. Chen, J. Liu, D. Wu, Convolutional neural networks for time series classification, Journal of Systems Engineering and Electronics 28 (1) (2017) 162–169. doi:10.21629/JSEE.2017.01.18.

[33] R. Liu, G. Meng, B. Yang, C. Sun, X. Chen, Dislocated Time Series Convolutional Neural Architecture: An Intelligent Fault Diagnosis Approach for Electric Machine, IEEE Transactions on Industrial Informatics 13 (3) (2017) 1310–1320. doi:10.1109/TII.2016.2645238.

[34] M. Samuelsson, Anomaly Detection In Time Series Data: a practical implementation for pulp and paper industry, Ph.D. thesis, Chalmers University of Technology (2016).

[35] J. Pang, D. Liu, Y. Peng, X. Peng, Anomaly detection based on uncertainty fusion for univariate monitoring series, Measurement 95 (2017) 280–292. doi:10.1016/j.measurement.2016.10.031.

[36] A. Diez-Olivan, J. A. Pagan, R. Sanz, B. Sierra, Data-driven prognostics using a combination of constrained K-means clustering, fuzzy modeling and LOF-based score, Neurocomputing 241 (2017) 97–107. doi:10.1016/j.neucom.2017.02.024.

[37] S.-E. Benkabou, K. Benabdeslem, B. Canitia, Unsupervised outlier detection for time series by entropy and dynamic time warping, Knowledge and Information Systems 54 (2) (2018) 463–486. doi:10.1007/s10115-017-1067-8.

[38] W. Yang, C. Liu, D. Jiang, An unsupervised spatiotemporal graphical modeling approach for wind turbine condition monitoring, Renewable Energy 127 (2018) 230–241. doi:10.1016/j.renene.2018.04.059.

[39] W. Lu, Y. Cheng, C. Xiao, S. Chang, S. Huang, B. Liang, T. Huang, Unsupervised Sequential Outlier Detection With Deep Architectures, IEEE Transactions on Image Processing 26 (9) (2017) 4321–4330. doi:10.1109/TIP.2017.2713048.

[40] M. Goldstein, S. Uchida, A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data, PLOS ONE 11 (4) (2016) e0152173. doi:10.1371/journal.pone.0152173.

[41] M. Hu, Z. Ji, K. Yan, Y. Guo, X. Feng, J. Gong, X. Zhao, L. Dong, Detecting Anomalies in Time Series Data via a Meta-Feature Based Approach, IEEE Access 6 (2018) 27760–27776. doi:10.1109/ACCESS.2018.2840086.

[42] X. Tang, W. Zeng, Y. Shi, L. Zhao, Brain activation detection by modified neighborhood one-class SVM on fMRI data, Biomedical Signal Processing and Control 39 (2018) 448–458. doi:10.1016/j.bspc.2017.08.021.

[43] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long Short Term Memory Networks for Anomaly Detection in Time Series, in: European Symposium on Artificial Neural Networks, 2015, pp. 22–24. doi:10.14722/ndss.2015.23268.

[44] S. Chauhan, L. Vig, 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, 2015, pp. 1–7. doi:10.1109/DSAA.2015.7344872.

[45] P. Malhotra, LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection., CoRR abs/1607.0.

[46] L. Bontemps, V. L. Cao, J. McDermott, N.-A. Le-Khac, Collective Anomaly Detection based on Long Short Term Memory Recurrent Neural Network, in: International Conference on Future Data and Security Engineering, 2016, pp. 141–152. doi:10.1007/978-3-319-48057-2_9.

[47] P. Filonov, Multivariate Industrial Time Series with Cyber-Attack Simulation: Fault Detection Using an LSTM-based Predictive Data Model., CoRR abs/1612.0.

[48] N. Nguyen Thi, V. L. Cao, N.-A. Le-Khac, One-Class Collective Anomaly Detection Based on LSTM-RNNs, in: Transactions on Large-Scale Data- and Knowledge-Centered Systems, Vol. 36, 2017, pp. 73–85. doi:10.1007/978-3-662-56266-6_4.

[49] D. T. Shipmon, Time Series Anomaly Detection; Detection of anomalous drops with limited features and sparse examples in noisy highly periodic data., CoRR abs/1708.0.

[50] B. Zhang, L. Zhang, D. Xie, X. Yin, C. Liu, G. Liu, Application of Synthetic NDVI Time Series Blended from Landsat and MODIS Data for Grassland Biomass Estimation, Remote Sensing 8 (1) (2015) 10. doi:10.3390/rs8010010.

[51] P. Chattopadhyay, L. Wang, Y.-P. Tan, Scenario-Based Insider Threat Detection From Cyber Activities, IEEE Transactions on Computational Social Systems 5 (3) (2018) 660–675. doi:10.1109/TCSS.2018.2857473.

[52] G. Paragliola, A. Coronato, Gait Anomaly Detection of Subjects with Parkinson's Disease Using a Deep Time Series-based Approach, IEEE Access (2018) 1–1doi:10.1109/ACCESS.2018.2882245.

[53] Y. Bao, Z. Tang, H. Li, Y. Zhang, Computer vision and deep learning–based data anomaly detection method for structural health monitoring, Structural Health Monitoring: An International Journal (2018) 147592171875740doi:10.1177/1475921718757405.

[54] T. Saito, M. Rehmsmeier, The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets, PLOS ONE 10 (3) (2015) e0118432. doi:10.1371/journal.pone.0118432.

[55] L. Abdi, S. Hashemi, To Combat Multi-Class Imbalanced Problems by Means of Over-Sampling Techniques, IEEE Transactions on Knowledge and Data Engineering 28 (1) (2016) 238–251. doi:10.1109/TKDE.2015.2458858.

[56] I. Triguero, M. Galar, H. Bustince, F. Herrera, A first attempt on global evolutionary undersampling for imbalanced big data, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 2054–2061. doi:10.1109/CEC.2017.7969553.

[57] H. Cao, X.-L. Li, D. Y.-K. Woon, S.-K. Ng, Integrated Oversampling for Imbalanced Time Series Classification, IEEE Transactions on Knowledge and Data Engineering 25 (12) (2013) 2809–2822. doi:10.1109/TKDE.2013.37.

[58] N. Moniz, P. Branco, L. Torgo, Resampling strategies for imbalanced time series forecasting, International Journal of Data Science and Analytics 3 (3) (2017) 161–181. doi:10.1007/s41060-017-0044-3.

[59] B. Zhao, X. Li, X. Lu, Z. Wang, A CNN–RNN architecture for multi-label weather recognition, Neurocomputing 322 (2018) 47–57. doi:10.1016/j.neucom.2018.09.048.

[60] E. Kanjo, E. M. Younis, C. S. Ang, Deep learning analysis of mobile physiological, environmental and location sensor data for emotion detection, Information Fusion 49 (2019) 46–56. doi:10.1016/j.inffus.2018.09.001.

[61] T.-Y. Kim, S.-B. Cho, Web traffic anomaly detection using C-LSTM neural networks, Expert Systems with Applications 106 (2018) 66–76. doi:10.1016/j.eswa.2018.04.004.

[62] A. H. Hamamoto, L. F. Carvalho, L. D. H. Sampaio, T. Abrão, M. L. Proença, Network Anomaly Detection System using Genetic Algorithm and Fuzzy Logic, Expert Systems with Applications 92 (2018) 390–402. doi:10.1016/j.eswa.2017.09.013.

[63] H. H. Bosman, G. Iacca, A. Tejada, H. J. Wörtche, A. Liotta, Spatial anomaly detection in sensor networks using neighborhood information, Information Fusion 33 (2017) 41–56. doi:10.1016/j.inffus.2016.04.007.

[64] S. Kanarachos, S.-R. G. Christopoulos, A. Chroneos, M. E. Fitzpatrick, Detecting anomalies in time series data via a deep learning algorithm combining wavelets, neural networks and Hilbert transform, Expert Systems with Applications 85 (2017) 292–304. doi:10.1016/j.eswa.2017.04.028.

[65] J. Camps, A. Sama, M. Martín, D. Rodríguez-Martín, C. Pérez-López, J. M. Moreno Arostegui, J. Cabestany, A. Català, S. Alcaine, B. Mestre, A. Prats, M. C. Crespo-Maraver, T. J. Counihan, P. Browne, L. R. Quinlan, G. O. Laighin, D. Sweeney, H. Lewy, G. Vainstein, A. Costa, R. Annicchiarico, A. Bayés, A. Rodríguez-Molinero, Deep learning for freezing of gait detection in Parkinson's disease patients in their homes using a waist-worn inertial measurement unit, Knowledge-Based Systems 139 (2018) 119–131. doi:10.1016/j.knosys.2017.10.017.

[66] X. Hua, Y. Cheng, H. Wang, Y. Qin, Y. Li, Geometric means and medians with applications to target detection, IET Signal Processing 11 (6) (2017) 711–720. doi:10.1049/iet-spr.2016.0547.

[67] G. Weinberg, Geometric mean switching constant false alarm rate detector, Digital Signal Processing 69 (2017) 1–10. doi:10.1016/j.dsp.2017.06.015.

[68] L. Wang, S. Guo, W. Huang, Y. Xiong, Y. Qiao, Knowledge Guided Disambiguation for Large-Scale Scene Classification With Multi-Resolution CNNs, IEEE Transactions on Image Processing 26 (4) (2017) 2055–2068. doi:10.1109/TIP.2017.2675339.

[69] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, J. Schmidhuber, LSTM: A Search Space Odyssey, IEEE Transactions on Neural Networks and Learning Systems 28 (10) (2017) 2222–2232. doi:10.1109/TNNLS.2016.2582924.

[70] R. Dey, F. M. Salemt, Gate-variants of Gated Recurrent Unit (GRU) neural networks, in: 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), IEEE, 2017, pp. 1597–1600. doi:10.1109/MWSCAS.2017.8053243.

[71] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. doi:10.1038/nature14539.

[72] W. Zhao, H. Luo, J. Peng, J. Fan, Spatial pyramid deep hashing for large-scale image retrieval, Neurocomputing 243 (2017) 166–173. doi:10.1016/j.neucom.2017.03.021.

[73] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, in: NIPS 2014 Workshop on Deep Learning, December 2014, 2014.

[74] M. Schuster, K. Paliwal, Bidirectional recurrent neural networks, IEEE Transactions on Signal Processing 45 (11) (1997) 2673–2681. doi:10.1109/78.650093.

[75] S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift., in: 32nd International Conference on Machine Learning, JMLR.org, Lille, France, 2015, pp. 448–456.

[76] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet classification with deep convolutional neural networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems, Curran Associates Inc., Lake Tahoe, Nevada, USA, 2012, pp. 1097–1105.

[77] K. Zhang, W. Zuo, Y. Chen, D. Meng, L. Zhang, Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising, IEEE Transactions on Image Processing 26 (7) (2017) 3142–3155. doi:10.1109/TIP.2017.2662206.

[78] A. Ignatov, Real-time human activity recognition from accelerometer data using Convolutional Neural Networks, Applied Soft Computing 62 (2018) 915–922. doi:10.1016/j.asoc.2017.09.027.

[79] B. Pourbabaee, M. J. Roshtkhari, K. Khorasani, Deep Convolutional Neural Networks and Learning ECG Features for Screening Paroxysmal Atrial Fibrillation Patients, IEEE Transactions on Systems, Man, and Cybernetics: Systems (2017) 1–10doi:10.1109/TSMC.2017.2705582.

[80] P. Golik, P. Doetsch, H. Ney, Cross-entropy vs. squared error training: a theoretical and experimental comparison, in: 14th Annual Conference of the International Speech Communication Association, Lyon, France, 2013, pp. 1756–1760.

[81] D. P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, CoRR abs/1412.6.

[82] S. García, A. Fernández, J. Luengo, F. Herrera, A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability, Soft Computing 13 (10) (2009) 959–977. doi:10.1007/s00500-008-0392-y.

[83] D. J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, 4th Edition, Chapman & Hall/CRC, 2007.

[84] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power, Information Sciences 180 (10) (2010) 2044–2064. doi:10.1016/J.INS.2009.12.010.

[85] J. Demšar, Statistical Comparisons of Classifiers over Multiple Data Sets, Journal of Machine Learning Research 7 (2006) 1–30.

[86] S. García, F. Herrera, An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons, Journal of Machine Learning Research 9 (Dec) (2008) 2677–2694.

[87] G. Santafe, I. Inza, J. A. Lozano, Dealing with the evaluation of supervised classification algorithms, Artificial Intelligence Review 44 (4) (2015) 467–508. doi:10.1007/s10462-015-9433-y.