Università degli Studi di Padova

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics of Data

Anomaly Detection of Multivariate Time Series for Industrial Machinery

Thesis supervisor:
Prof. Gian Antonio Susto

Candidate:
Nicola Bee

Thesis co-supervisor:
PhD Diego Tosato
PhD Chiara Masiero

Academic Year 2023/2024

Abstract

In industrial contexts, anomaly detection is crucial for identifying deviations from normal operating conditions, ensuring proactive maintenance, minimising downtime, and optimising the reliability and efficiency of industrial processes. Advanced machinery, constantly monitored by diverse sensors, generates multiple temporal sequences of data that can be analysed for evaluating performance. Within this setting, this thesis delves into a real-world scenario, focusing on the analysis of multivariate time series data produced by a set of filling machines for dairy products. Initially, the study conducts an in-depth analysis of the signals generated by the machinery and dedicates itself to collecting and preprocessing a dataset for following analyses. Subsequently, the Temporal Fusion Transformer (TFT), a cutting-edge deep learning model, is employed to effectively capture the complex temporal patterns inherent in industrial process signals and detect anomalies within the dataset. By addressing the challenge of dealing with intricate real-world data, this research aims to unravel their latent complexities and enhance anomaly detection precision through the utilisation of advanced deep learning models.

Contents

Chapter 1

Introduction

Industrial machinery are the backbone of modern manufacturing processes, driving efficiency, productivity, and innovation across various industries.These equipments play a critical role in ensuring smooth operations and delivering high-quality products to consumers. However, the reliable performance of industrial machinery is not guaranteed, as they are susceptible to various anomalies that can compromise their functionality, safety, and efficiency. Anomalies, defined as deviations from normal patterns or behaviours, can manifest in various forms within industrial contexts, ranging from equipment malfunctions to safety hazards and security breaches. For this reason, detecting those deviations is an important task to enhance the understanding of the apparatus, enabling data-driven decisions regarding hardware improvement or maintenance prediction.

One of the key challenges in detecting anomalies in industrial machinery lies in the nature of the signals they produce during operation. Typically, these signals are complex, multivariate time series data encompassing various sensor readings, including pressure, vibration frequency, temperature, and electrical currents, among others. For this reason, anomaly detection in industrial context requires advanced techniques that can deal with such intricate signals by leveraging, for example, advanced deep learning models tailored to identify anomalies.

Multivariate time series anomaly detection in industrial machinery is essential for several reasons. Firstly, anomalies in machinery signals can signify potential malfunctions or failures, which, if left undetected, can lead to costly downtime, production delays, and equipment damage. By detecting anomalies early, maintenance teams can intervene promptly, preventing critical failures and minimising disruptions to production schedules. Secondly, industrial machinery often operates under harsh and demanding conditions, leading to wear and tear over time.

Anomaly detection allows for the early identification of degradation or deterioration in machinery performance, enabling proactive maintenance and extending the lifespan of equipment. This predictive maintenance approach helps optimise maintenance schedules, reduce repair costs, and maximise operational efficiency.

In this thesis, a multivariate anomaly detection analysis is conducted on a real-world case involving an industrial machine responsible for dairy product packaging, focusing specifically on the signals produced by the cap-applicator module. The study delved into the complexities associated with creating a real-world model and outlined the various steps involved. In this chapter, the goals and challenges of the present study are delineated. Specifically, we outline the overarching objectives that motivate our research endeavours and identify the key challenges that must be addressed to achieve these aims successfully. In Ch.2, a comprehensive literature review is conducted to assess the current state-of-the-art models available in the field. This review aims to provide a thorough examination of existing methodologies in anomaly detection and time series forecasting. By examining the cutting-edge advancements in the literature, this section seeks to establish a foundational understanding of the landscape of available models, thereby informing the subsequent discussions and analyses within this thesis. In Ch.3 the methodology used for processing the data and fulfilling the task is described. The investigation begins with data collection, where sensor readings from the cap applicator module are gathered from cloud storage. Next, preprocessing steps is employed to clean and prepare the collected data for further steps. Subsequently, a multivariate time-series forecasting model, called Temporal Fusion Transformer, is developed, trained with the preprocessed data, and utilized for anomaly detection. Lastly, the performance of the model is evaluated in Ch.4.

1.1 Problem Definition

1.1.1 Machine Description

The machines under investigation in our study are packaging and filling machines employed in the food and beverage sector, specifically designed for the accurate filling of aseptic cartons with various liquids, including both consumables such as milk, eggs, and dairy products, as well as industrial solutions like detergents and chemicals. These machines, all produced by the same manufacturer, are available in both single-line and dual-line production configurations. The production process consists of numerous systematic steps. Initially, a stack of cartons is gently introduced into the machine, where each carton is individually retrieved and placed onto a rolling conveyor system. Here, the carton undergoes the bottom forming phase, during which a series of mechanical processes, including molding into a three-dimensional shape through mechanical presses, occur before its bottom is folded and securely welded.

For machines equipped with cap-applicator modules, an additional step ensues, wherein caps are seamlessly affixed onto the carton tops. The caps are transferred from an external cap warehouse through a tube system to the point of application inside the machine. Here, they are positioned on the inside of the carton in order to be welded. This task is accomplished through the application of ultrasound technology, where a specialised hollow metal cylinder, known as a sonotrode, melds the plastic cap to the cardboard container, ensuring a secure seal. The precise wave signals are generated by a generator, which transmits energy to the transducer. The transducer converts this energy into ultrasonic mechanical waves, which are subsequently transmitted to the sonotrode for application. This micro welding fuses the cap flange to the cardboard, creating a hermetic connection. The correct execution of welding is verified through established testing procedures.

Following these initial stages, the cartons proceed through a thorough disinfection process, where they are enveloped in a fine mist of specialised cleaning agents. Once sanitised, the designated liquid for filling is precisely metered and dispensed, and then dropped in one go to comple the filling process. Finally, as the carton reaches the end of the conveyor belt, the top is hermetically sealed, and the completed package is smoothly ejected from the machine.

1.1.2 Goal

The primary objective of this study is to develop an effective method for detecting anomalies within industrial signals, specifically tailored to real-world scenarios, with a particular focus on the dairy packaging machinery available to us. This goal is pursued through the implementation and testing of state-of-the-art deep learning models. The underlying premise of this research is that industrial machinery, responsible for production lines, typically operates in repetitive and predictable patterns. Consequently, we anticipate that by forecasting the signals produced by the machinery, we can accurately predict its standard behaviour. In contrast, anomalies are expected to exhibit less predictability, resulting in a greater discrepancy between real data and forecasted values. This work emphasizes the development of a high-level solution applicable to various machines, thus maintaining a broad dataset encompassing all the available machines of a certain version. Consequently, the model avoids incorporating solutions tailored specifically for individual machines or training on unbalanced datasets. It is worth noting that this study represents the first exploration of this dataset. As such, it serves as a pioneering analysis of the data's properties, without precedent upon which to build.

1.1.3 Challenges of the Dataset:

Studying a real case scenario can lead to intrinsic challenges and difficulties related to the dataset characteristics or to the methodology. Some of those challenges are listed in this section.

Unsupervised nature of anomaly detection Anomaly detection is typically an unsupervised task due to the inherent difficulty in obtaining labelled anomalies. Unlike supervised learning tasks, where the model is trained on labelled data, anomaly detection often relies on identifying patterns or deviations in data that do not conform to expected behaviour. Labelling anomalies requires expert domain knowledge and is often subjective, making it challenging to create a comprehensive labelled dataset for training. As a result, unsupervised methods that focus on detecting deviations from normal patterns without explicit labelling of anomalies are commonly employed in anomaly detection tasks.

Ambiguity in anomaly definition Finding a unique definition of what anomalies are is non-trivial, especially in the industrial sector. In effect, what may appear anomalous in certain settings could be entirely normal under different operating conditions. Moreover, anomalies might be of different kinds, such as single events far from the standard behaviour or collective drift during a certain time interval.

Lack of clean dataset Building a model capable of correctly identifying anomalies in new datasets becomes inherently challenging when the dataset used for training contains itself untracked anomalies. In such a scenario, there is a risk that the model may inadvertently learn to anticipate the presence of anomalies within the time series data. This situation can lead to both false negatives, where genuine anomalies go undetected because the model perceives them as normal data points, and false positives, where the model incorrectly identifies standard data points as anomalies due to discrepancies between the expected and actual data behaviours.

Uncharted analysis territory The complexity of the study undertaken in this study is underscored by the lack of prior exploration and analysis of the dataset under consideration. To date, the dataset remains indeed largely unexplored, with limited existing knowledge or precedents available for reference. Furthermore, the absence of established machine learning models tailored to this specific dataset compounds the challenge, requiring the development of novel approaches to effectively extract insights and detect anomalies.

Diverse machine models The analysis of the data aims to be applicable to a broad range of machinery. However, the dataset comprises signals from various machine versions, resulting in distinct signals for each machine. This poses a significant challenge for the selected anomaly detection model, as it necessitates a consistent set of signals across all machines. Therefore, training a single AI architecture to accommodate different models is inherently difficult. Additionally, machines may feature either a single or double production line, resulting in sensor signals being recorded either once or for both lines in two separate variables (`_A` and `_B`).

Variation in recipe settings Industrial machinery frequently operates using different configurations, referred to as "recipes," tailored to specific customer requirements. These recipe settings can significantly change the behaviour of the machine. Moreover, similar configurations are recorded with different recipe IDs for different customers, making the actions of the machines less consistent.

Variable and fragmented production The machines have an inconstant and unpredictable duration of the production state. This leads to inconsistencies in the length of the useful segments within the dataset. As a consequence, the data collected may encompass periods of differing durations, making it challenging to establish uniform intervals for analysis. These fluctuations in production durations contribute to the complexity of the dataset, requiring adaptable methodologies capable of accommodating such variability.

Misalignment between alarm logs and anomalies The industrial machinery under analysis automatically logs alarms as they occur, storing them in a dedicated database. However, it is important to note that alarms do not always correspond to anomalies within the dataset. In some instances, these two phenomena can be entirely disconnected. For example, an alarm may trigger without any corresponding anomaly in the sensor data, or conversely, an anomaly in the sensor readings may occur without triggering an alarm in the system.

Chapter 2

State of the art

2.1 Time Series

Time series, a fundamental concept in the realm of statistics and data analysis, represents a sequence of data points ordered chronologically. Unlike traditional datasets, where each observation is independent, time series data exhibits a temporal dependency, wherein each data point is associated with a specific time index. At its core, time series data comprises a sequence of observations collected at equally spaced intervals over a defined period. Each observation, typically representing a measurement or value recorded at a specific time point, is arranged chronologically, creating a sequential dataset where the ordering of observations is significant. This unique characteristic allows analysts to explore and understand patterns, trends, and behaviours that evolve over time, making time series analysis an invaluable tool across various domains such as industrial processes [1], healthcare [2], meteorology [3] and beyond. Time series can encompass various types of data, including numerical, categorical, or even textual, depending on the context of the analysis. These data sequences may exhibit different patterns and structures, ranging from simple trends and seasonal variations to more complex dependencies and irregular fluctuations.

Time series can be divided into univariate or multivariate, depending on the number of variables that are recorded simultaneously.

- **Univariate time series** consists of a single variable or data stream observed or measured over time, in which each data point represents the value of that variable at a specific time instance. Univariate time series analysis focuses solely on understanding and modelling the patterns, trends, and behaviours inherent in that single variable.

- **Multivariate time series**, in contrast, involve multiple variables or data streams observed or measured simultaneously over time. Each observation in the dataset consists of values for multiple variables recorded at the same time point. For example multivariate time signals could be recorded by monitoring the temperature, pressure, and millimetres of rain in a certain area, obtaining three standalone signals describing physical quantities that could be interconnected in a non-trivial manner. Multivariate time series analysis aims, therefore, to explore and model the relationships, dependencies, and interactions among these variables over time. This type of analysis enables a deeper understanding of complex systems where the behavior of one variable may influence or be influenced by others.

A time series is said to be stationary if its statistical properties do not change over time, i.e. if the mean, variance, and autocorrelation structure remain constant. This property, ideal for modelling and analysis, is hard to find in a real-world scenario, where volatile features interfere and nonstationarity is encountered. One primary cause can be seasonality, which manifests as repeating patterns or fluctuations within a specific period, such as daily, weekly, monthly, *etc.* For instance, consider the daily operation of machinery in an industrial setting: if a particular machine is active only during the day, there could be seasonality in the signals registered by the machine, with production levels fluctuating predictably throughout the day. Furthermore, the nonstationarity might be due to change points, that is, specific time instances or periods within a time series where the underlying data distribution experiences significant shifts or changes. For example, if machinery operates with different settings, there will be a change point in the time series that registers the machine's state whenever the settings are changed. Lastly, time series data may exhibit concept drift [4] when a progressive change in the environment in which data are produced may lead to changes in the underlying statistical distribution of a data stream over time. For example, if an industrial machine ages or undergoes wear and tear, its performance may degrade gradually over time, leading to changes in the behavior of the machine captured by sensor signals.

2.2 Anomaly Detection

Anomaly detection, also known as outlier detection, is a vital task in data analysis and machine learning. It encompasses the task of identifying data instances that exhibit substantial deviations from the typical patterns observed within a dataset. More precisely, an anomaly can be defined as an *observation that deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism* [5]. This field has witnessed extensive exploration across various domains, finding applications in critical areas such as fraud detection, public health, structural defect detection, and image processing [6]. In recent years, the advent of deep learning methodologies has revolutionised anomaly detection, enabling the development of sophisticated models capable of learning intricate representations from complex data types such as temporal sequences [7], spatial distributions [8], and graph structures [9]. This evolution has boosted the exploration of deep anomaly detection methods, showcasing remarkable advancements in addressing intricate detection challenges encountered in real-world scenarios.

In particular, anomaly detection has a relevant role in the context of Industry 4.0. This branch, often referred to as the fourth industrial revolution, addresses the paradigm shift in manufacturing and production marked by the integration of advanced digital technologies such as Internet of Things (IoT), artificial intelligence, big data analytics, and automation, that aim to create smart, interconnected systems for optimising processes and enhance efficiency [10]. In this context, anomaly detection offers powerful tools for identifying irregularities within the vast volumes of data generated by sensors and IoT devices in smart factories. By continuously monitoring sensor data, production metrics, and other critical parameters, anomaly detection enables the early detection of equipment malfunctions [11], process inefficiencies, or potential security threats [12], facilitating interventions to maintain operational integrity and minimise disruptions. This approach lays the foundation for predictive maintenance strategies, where maintenance interventions are planned based on actual equipment condition rather than predetermined schedules. Moreover, anomaly detection plays a crucial role in quality control processes [13], ensuring product consistency and reliability amidst the complexities of modern manufacturing processes. By analysing data from various stages of the production, deviations from established quality standards or specifications can indeed be promptly detected, allowing for timely interventions to rectify issues, preventing the production of defective products and minimising waste.

In particular, one of the most common signal types in Industry 4.0 is the time series. This stems from the fact that sensors commonly generate time series data to monitor the real-time condition of industrial machinery. For this reason, numerous methods for spotting outliers in this data type have been developed [14].

2.2.1 Anomaly Taxonomy

The term "anomaly" has neither a univocal nor a precise definition, but is instead a general term that includes different types of deviation from the standard behaviour. This means that different types of anomalies, each offering unique challenges and insights, can be identified [15].

- **Point Anomalies** are characterised by individual data points that deviate significantly from the norm. This kind of anomaly could signify noise, errors, or rare events. Several techniques exist for identifying anomalies of this kind, exploiting the fact that the anomaly is significantly distant from the other data points. In time series analysis, the classical approach involves establishing upper and lower control limits based on historical data and assessing whether a specific data point falls outside these limits.

- **Contextual anomalies** depend on the surrounding circumstances or contextual information for their identification. Those are points that do not exceed the control limits but show unusual behaviour with respect to normal data. For time series this translates, for example, to a sudden change in a periodic or predictable trend.

- **Collective anomalies**, also known as group anomalies, involve subsets of data exhibiting aberrant behaviour collectively. Those points might appear to exhibit normal behaviour when examined individually, but they raise suspicions when considered collectively. Since they are not easily recognisable at first glance, long-term contexts are particularly important in detecting them.

- **Series anomalies**, or abnormal time series, are anomalies that specifically occur in multivariate time series when the relationships of a certain time series with other time series in the dataset are significantly different from those observed between the remaining series.

Other types of classifications specifically designed for time series are possible, such as the one proposed in [16], that consider different scenarios of anomalous behaviour for data over time. This taxonomy is visible in Tab.2.1

Category	**Description**
Missing	Most data are missing and the frequency drops to 0
Outlier	One or more outliers appear in the image as spikes
Minor vibration	Response oscillates with a small amplitude compared to normal sensor data
Square	Vibration response oscillates abnormally within a limiting range
Trend	The data has an unexpected and obvious non-stationary and monotonous trend
Drift	The data has an unexpected and obvious non-stationary trend with random drift

Table 2.1: Bao et al. detailed classification of anomalies in time series

2.2.2 Types of Supervision

Anomaly detection methods vary in their approach depending on the availability of labelled anomalies.

Supervised Supervised anomaly detection algorithms require labelled instances of both normal and anomalous behaviour for training. However, obtaining a fully labelled dataset is not common in anomaly detection and can be a challenging task. Consequently, these methods are rarely employed, and there is limited research focus on them. As a result, there are no algorithms specifically designed for this task [17], but existing classifiers such as random forests [18] or neural networks are often utilised.

Semi-supervised Semi-supervised algorithms combine elements of both supervised and unsupervised approaches, utilising a combination of labelled and unlabelled data to identify anomalies. In a semi-supervised setting, the algorithm typically receives a small amount of labelled data , along with a larger amount of unlabelled data. There are several types of semi-supervised techniques, for example, *incomplete supervision*, where only a subset of training data is given with labels, *inexact supervision*, where the training data are given with only coarse-grained labels, and *inaccurate supervision*, where the given labels are not always ground-truth [19]. For instance, some incomplete semi-supervised models for anomaly detection are trained only on normal samples, and detect anomalies that deviate from the normal representations learned during the training process [20].

Unsupervised Unsupervised anomaly detection algorithms operate without the need for labelled data and aim to identify anomalies based solely on the characteristics of the data itself. These algorithms seek to detect instances that deviate significantly from the norm without prior knowledge of what constitutes normal or anomalous behaviour, finding points that do not follow the underlying probability distribution. The strength of these methods lies in the fact that, since they do not need labels, they can be applied to any type of raw data, making them suitable for scenarios where labelled anomalies are unavailable. Moreover, the abundance of different algorithms provides a wide range of options to choose from, based on the specific characteristics of the data and the requirements of the application. This diversity allows for flexibility in selecting the most appropriate approach for a given problem domain, enhancing the versatility and effectiveness of unsupervised anomaly detection in practical applications. However, there are some cons to consider. First of all, unsupervised algorithms may flag normal instances, that deviate significantly from the majority of the data, as anomalies. For this reason, interpreting the results and distinguishing between true anomalies and normal data can be challenging, especially when the context is complex and the definition of an anomaly has some degree of uncertainty. On the other hand, algorithms may produce false positives, leading to a higher rate of false alarms and decreased confidence in the detected anomalies. In this framework, determining appropriate threshold values that balance the trade-off between false positives and false negatives for identifying anomalies can be difficult and requires careful consideration of the domain.

2.2.3 Algorithms

As previously emphasised, unsupervised methods are the preferred choice in scenarios where labelled data is scarce, that is a common occurrence in many practical applications. In the context of this study, which centres on anomaly detection within time series data, the forthcoming section will undertake a thorough examination of classical methods and deep learning algorithms tailored specifically for this purpose.

Classical Methods

Z-score One of the initial methods for anomaly detection, though simplistic, is the Z-score (also known as the standard score). This score is computed for each data point by subtracting the mean of the time series and then dividing by the standard deviation. Anomalies are commonly flagged as data points whose Z-scores surpass a predefined threshold. Hence, this approach proves useful for pinpointing point anomalies within univariate time series. However, it has some limitations in capturing more complex anomalies or patterns in multivariate or highly dynamic time series data.

Distance-based Distance-based methods offer another avenue for anomaly detection. These methods exploit the distances between data points, identifying outliers as those that deviate significantly from their neighbouring points. They are applicable to time series analysis, where data points may exhibit stationarity, clustering around certain values, or gradual changes without abrupt jumps. Operating in multidimensional spaces, these algorithms naturally handle multivariate time series data. An example of such algorithms is DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [21]. Renowned for its capacity to detect clusters of varying shapes and sizes within a dataset while remaining robust to noise, DBSCAN differs from traditional clustering algorithms like K-means [22], since it does not require a priori specification of the number of clusters. Instead, it groups closely packed points based on their density, forming clusters around regions of high density separated by areas of low density. Isolated points in low density regions are automatically mapped as noise or anomalies. Sometimes, these methods employ feature extraction to obtain more informative and clusterable data. Additionally, they may utilise kernel methods to map the time series into a higher-dimensional space [23]. This process aids in capturing non-linear relationships and complexities within the data, enhancing the clustering performance.

Isolation Forest Isolation Forest [24] is an anomaly detection method based on the concept of isolating anomalies within a dataset by constructing specific decision trees called isolation trees. At each step, the algorithm randomly selects a feature and a split value within the range of that feature, recursively partitioning the data until all points are isolated or a maximum tree depth is reached. The number of partitions needed to isolate a data point serves as its isolation depth. Multiple isolation trees are constructed independently, with each tree contributing to the anomaly detection process. An anomaly score is then calculated for each data point based on the average isolation depth across all trees. Points with lower average isolation depths are considered anomalies. This approach leverages the intuition that anomalies are typically isolated more quickly than normal data points, as they deviate from the majority of the dataset. Isolation Forest technique offers scalability and is intrinsically multivariate, making it suitable for time series analysis [25]. However, its performance can be sensitive to the choice of hyperparameters, such as the number of trees in the ensemble and the maximum tree depth. Suboptimal hyperparameter selection may lead to decreased detection accuracy.

ARIMA Autoregressive Integrated Moving Average (ARIMA) models [26] are widely used statistical methods renowned for their effectiveness in time series forecasting. These methods are also recognised for their utility in anomaly detection tasks, leveraging their ability to capture temporal dependencies and patterns within sequential data. ARIMA models are particularly useful when the data exhibit non-stationarity since they are able to capture both seasonality and drift. The name ARIMA itself provides insights into its key components:

AR Auto-regressive: This component captures the relationship between an observation and a number of lagged observations (i.e., its own past values), modelling the dependency of the current value on its previous values. Given a random variable X at time step t, the autoregressive part of order p is given by: $AR(p) : X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + \varepsilon_t$ where $\{\phi_i\}_{i=1}^{p}$ are auto-correlation coefficients and ε is white noise.

I Integrated: This component refers to differencing the time series data to make it stationary. Differencing involves subtracting consecutive observations to remove trends or seasonality. Hence, a point at timestep t is $X_t = X_T - X_{t-1}$. The order d account for the number of times a difference is applied to make data stationary.

MA Moving Average: This component accounts for the dependency between an observation and a residual error from a moving average model applied to lagged observations, capturing short-term fluctuations in the data. The moving average part at time step t of order q is given by $MA(q) : X_t = \varepsilon_t - \theta_1\varepsilon_{t-1} - \theta_2\varepsilon_{t-2} - \ldots - \theta_q\varepsilon_{t-q}$ where $\{\theta_i\}_{i=1}^{q}$ are moving-average coefficients and ε_t denotes a model prediction error at time step t.

Despite the model's power and suitability for many applications, it is essential to properly tune the model parameters and validate its performance to ensure accurate forecasting results. Additionally, ARIMA can only analyse univariate time series; for multivariate series, a variant exists called VARIMA (Vector Autoregressive Integrated Moving Average) [27].

In addition to the aforementioned techniques, several other algorithms can serve the same purpose. For example, time series can be analysed in the frequency domain [28] leveraging methods such as the Fourier transform, the wavelet transform [29] or the Power Spectral Density (PSD). Moreover some methods can be borrowed from other areas of research, such as Saliency Detection [30]. This technique, traditionally utilised for computer vision, focuses on identifying the most salient or significant features within the data that deviate from the norm. By highlighting these salient features, anomalies can be detected based on their deviation from the expected patterns or distributions, providing valuable insights into potential anomalies within the time series data [31].

Deep Learning Techniques

Traditionally, anomaly detection has relied on statistical methods, machine learning algorithms, and domain-specific rules to identify deviations from normal behaviour in datasets. While these approaches have shown effectiveness in many scenarios, they often struggle to handle the complexities and nuances present in modern data environments. On the other hand, deep learning techniques have shown promising results and improved detection performance on complex datasets with respect to shallow methods [32]. By leveraging neural networks with multiple layers and innovative architectures, deep learning models can indeed automatically learn useful features from raw data and perform end-to-end anomaly detection. In the context of time series, there are three conceptual paradigms that can be applied [33], namely Deep Learning for Feature Extraction, Learning Feature Representations of Normality,and End-to-end Anomaly Score Learning.

Deep Learning for Feature Extraction This paradigm involves utilising deep learning architectures to automatically extract relevant features from time series data. Specifically, it aims to extract a low-dimensional feature representation from data in higher dimensions that is potentially not linearly separable. The anomaly scoring, i.e., the process of assigning a label to each value indicating whether the data are anomalous or not, is independent of the feature extraction. The use of deep learning is appropriate since these algorithms have shown better capability in extracting meaningful features [34]. This class of methods can leverage well-known models, such as AlexNet [35] or ResNet [36], for feature extraction, including the use of their pre-trained version if the data is suitable. Other research instead focuses on training deep feature extraction models from scratch, leveraging architectures like the auto-encoder for feature extraction [37]. These algorithms have shown good utility, especially when used to detect anomalies in high-dimensional data, such as video samples, where dimensionality reduction can aid in reducing the complexity of the problem while extracting valuable information. Anomaly scoring is computed on the extracted features using traditional machine learning methods, such as one-class SVMs [38] or unsupervised classification methods like clustering [39]. Since many anomaly scoring methods are readily available, various combinations of feature-extractor and anomaly-scorer can be explored. However, having the two processes disjointed can lead to suboptimal anomaly scores.

Learning Feature Representations of Normality The methods in this category blend feature learning with anomaly scoring. One way of implementing such algorithms is by optimising a generic feature learning objective function. Despite these functions are not specifically designed for anomaly detection, they can still capture key underlying data regularities. The key idea is that if the model learns to reconstruct or forecast data, standard data will be predicted with more precision than anomalous data. Consequently, outliers will yield a higher reconstruction/prediction error, highlighting their atypical nature. For example, auto-encoders (AE) compress data into a low-dimensional space that only retains the relevant features. This property can be exploited by evaluating the reconstruction error, which will be higher for anomalies, that are indeed hardly captured by the model. Some variants, like sparse AE [40] or variational AE [41] might enhance the robustness of anomaly detection. Moreover, the auto-encoder structure can be implemented using many different species of layers. For example, *Gugulothu et al.* [42] combine an auto-encoder with a recurrent neural network for detecting anomalies in high-dimentional timeseries, while *Kieu et al.* [7] use an ensemble of recurrent auto-encoder for outlier detection. Nevertheless, the feature representations derived from auto-encoders may exhibit bias due to infrequent patterns and the existence of outliers or anomalies within the training dataset.

Another idea for identifying anomalies is by using a Generative Adversarial Network (GAN) [43]. This model operates on a dual neural network architecture comprising a generator and a discriminator. The generator synthesises new data instances by processing random noise as input, whereas the discriminator evaluates these samples, discerning between genuine data from the training set and counterfeit data from the generator. Through an adversarial training process, the generator and discriminator engage in a competitive interplay: the generator strives to produce samples indistinguishable from authentic data, while the discriminator attempt to accurately classify real and fake samples. This dynamic feedback loop continues iteratively until equilibrium is ideally reached, where the generator generates realistic samples that confound the discriminator. The detection of anomalies by using this framework is based on the idea that normal data can be better generated from the feature space, and so it is expected that anomalies will have fewer highly similar generated counterparts. There are several famous examples of GANs used for this purpose, such as AnoGAN [44] or GANomaly [20]. In the time series realm, GANs are used for detecting anomalies in multivariate time series [45, 46].

A third approach for harnessing the reconstruction error, particularly in time series data, involves utilising advanced forecasting models, such as Recurrent Neural Networks (RNNs) or Transformer architectures. These models excel at predicting future values of the time series data for multiple time steps ahead. Anomalies are identified when the reconstruction error surpasses a predefined threshold. This method capitalises on the ability of these models to capture intricate and long-term temporal dependencies [47, 48].

Other methods leverage self-supervised learning, that is, evaluating the conformity of data instances to normality by examining their alignment with an ensemble of predictive models, where each model is trained to forecast one feature based on the remaining features in the dataset. These methods are based on cross-feature analysis and are designed to check the consistency of the different series, finding anomalies where data shows inconsistency [49].

A second macro-class of algorithms is Anomaly Measure-dependent Feature Learning. This category of methods focuses on acquiring feature representations tailored to optimise a specific anomaly measure already in place. This includes distance-based anomaly detection, which trains neural networks to extract features designed to enhance the performance of certain distance-based anomaly measures [50]. Additionally, it encompasses deep one-class classification techniques [51], which leverage neural networks in conjunction with one-class Support Vector Machines [52]. Lastly, this category includes deep clustering-based algorithms, which learn to encode data in a manner conducive to easy clustering, distinguishing them from anomalies [53].

End-to-end Anomaly Score Learning This methodology involves training deep learning models in an end-to-end fashion to directly predict anomaly scores for each data point by simultaneously learning the feature extraction and the anomaly scores into a single cohesive framework. Rather than relying on pre-existing deep learning models for feature extraction, this category of methods integrates discriminative or order information directly into the anomaly scoring network, therefore enhancing the optimization of anomaly scores. For example, ranking models aim to directly learn how to prioritise or rank items based on their relevance to a given context by optimising a scoring function that assigns a numerical score to each item in a given set, with higher scores indicating greater relevance or priority [54]. Other strategies use a prior distribution for encoding and guiding anomaly score learning. For example, *Pang et al.* [55] use a Gaussian prior to encode the anomaly scores and enable their direct optimization. Lastly, some researches instead use the likelihood maximization of the events for anomaly learning, assuming that anomalies are low-probability events.

2.3 Time Series Forecasting

In anomaly detection for time series data, a prevalent and effective technique involves forecasting the series and inspecting the reconstruction error by comparing the predicted time series with the original data [15]. This approach has demonstrated its efficacy in various domains due to its ability to capture deviations from expected patterns. Consequently, below follows a brief review of both established and cutting-edge deep learning models tailored for time series forecasting.

2.3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [56] have emerged as a successful class of neural network architectures uniquely designed to handle sequential data with temporal dependencies. Unlike traditional feedforward neural networks, RNNs possess the ability to retain memory of past inputs, allowing them to capture and process sequential information effectively. This distinctive capability has made RNNs indispensable in various domains, including natural language processing [57], time series analysis [58], and more.

The overall structure of an RNN resembles that of standard neural networks: it is composed of an input, a hidden, and an output layer, and exploits weights, biases, and activation functions to make predictions. However, this model also features a feedback loop that retains the value obtained by the neuron after the activation function and combines it with the input of the next time step. This is called a hidden state, which serves as memory to retain information from previous time steps. The hidden state is updated recursively using the current input and the previous hidden state. Despite the input sequence potentially having multiple time steps, the number of parameters to train does not increase, as they are shared across all time steps. Hence, RNNs can handle sequences of variable lengths, making them suitable for a wide range of sequential data tasks. The outcome of the output layer contains, for every instant, the prediction for the following time step, enabling the process of sequential data both input-wise and output-wise. Therefore, multiple combinations of types of input and output are possible. For instance, singular input with sequential output finds utility in image captioning, while sequential input with a single output suits document classification. Moreover, when both input and output are sequential, RNNs can meticulously analyse video content frame by frame or forecast temporal trends in time series.

However, training RNNs poses notable challenges, primarily the issue of vanishing gradients. As each time step effectively acts as a layer in a feedforward network, training an RNN for extended temporal sequences results in exponentially diminishing gradients and information decay over time. This leads to smaller steps in the gradient descent algorithm, i.e. a much slower learning of the optimal parameter values. On the other hand, if feedback weights are greater than 1, there is a risk of a mirror issue called exploding gradient. To mitigate this problem, the common solution is leverage the gating mechanisms, prominently embodied by Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM) cells. These gating mechanisms enable RNNs to discern when to retain or discard temporal information, thereby enhancing the vanishing gradient.

Long Short-Term memory

Long Short-Term Memory (LSTM) [59] addresses the vanishing gradient problem by introducing gated mechanisms that regulate the flow of information through the network and capture long-range dependencies in sequential data. The LSTM structure consists of multiple memory cells interconnected through various gating mechanisms, allowing it to effectively retain and manipulate information over extended time intervals. In particular, the model utilises two different feedback lines, one for short-term memory and the other for long-term memory, connected by three different gates. The long-term memory is stored inside the cell state, flowing through the entire network and being modified only by linear interactions (summation and multiplication). It can retain information over long sequences and is regulated by the gates. On the other hand, short-term memory is stored inside the hidden state. This is used as input through the LSTM cell and is updated at each time step depending on its value, the input of the network, and the cell state. The gates of the LSTM cell include the forget, input, and output gates, as described below.

- **The Forget Gate** determines the proportion of information from the previous cell state to discard or retain. It takes the previous value of the short-term memory and the current input as input, sums them (using appropriate weights and bias), and produces a value between 0 and 1 through a sigmoid function. For instance, a value of 1 indicates that all long-term memory will be retained, while a value of 0 will erase all previous memory. The cell state is then scaled by this proportion.
- **The Input Gate** determines which new information should be added to the cell state. It is composed of two similar sub-modules: one designed to compute a "potential long-term memory" value that will be summed to the new long-term memory, and the other for computing the percentage of this memory that should be remembered. Both modules take in input the hidden state and the network's input and perform operations similar to the forget gate. However, in one case, the activation function is the hyperbolic tangent, obtaining a value between -1 and 1, while in the other, the sigmoid function is used again. The results of the two sub-modules are multiplied together and then summed to the cell state.
- **The Output Gate** determines which parts of the cell state should be returned as the new hidden state for the short-term memory while the cell state remains untouched. The module takes input, for the third time, the current input and the hidden state value, summing them and applying a sigmoid function. The cell state value is processed with a *tanh* function and then multiplied by the result of the previous computation, obtaining an updated value for the hidden state.

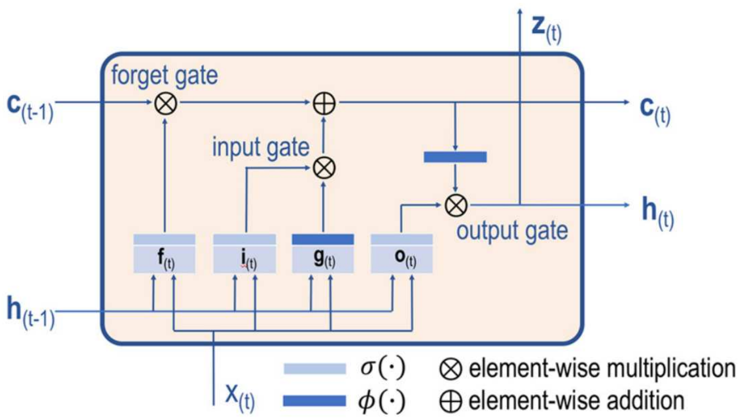


Figure 2.1: LSTM cell diagram [60]

Gated Residual Unit

Similar to LSTM, Gated Residual Unit (GRU) networks [61] incorporate gated mechanisms to control the flow of information and address the vanishing gradient problem. GRU cells have a simplified architecture compared to LSTM, with fewer parameters, making them computationally more efficient. In particular, GRU does not have the cell state but uses the hidden state to transmit information itself. The unit is composed by two different gates: the reset gate and the update gate.

- **The Reset Gate** controls the degree to which the previous hidden state should be ignored when computing the new candidate hidden state. It is similar to the LSTM's forget gate: it takes in input the input data x_t and the hidden state h_{t-1}, combining them and applying a sigmoid activation function. The result r_t determine the percentage of past information that should be discarded before computing the new candidate hidden state. This value r_t is multiplied element-wise by the hidden state h_{t-1} and summed to the input x_t; the hyperbolic tangent activation function is then applied to obtain a candidate hidden state $\tilde{h}_t$ in the range between -1 and 1.

- **The Update Gate** determines how much of the past information should be retained and how much of the new information should be added to the current hidden state. It takes input from the previous hidden state h_{t-1} and the current input x_t and produces an update gate vector using a sigmoid activation function. The update gate value z_t ranges between 0 and 1, where 1 indicates keeping all the previous information and 0 indicates ignoring it entirely. The final hidden state is computed by combining the

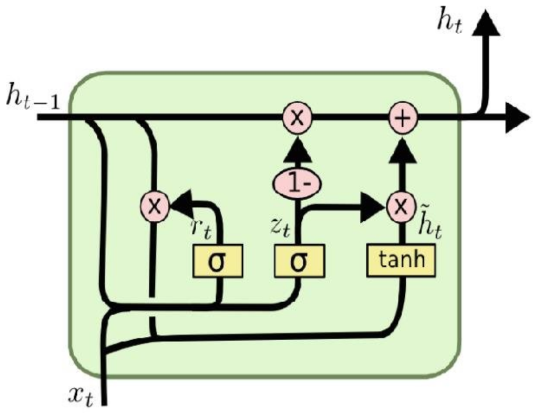


Figure 2.2: GRU cell diagram [62]

previous hidden state h_{t-1} and the new candidate hidden state $\tilde{h}_t$ based on the update gate z_t. It interpolates between the previous hidden state and the new candidate hidden state, with the update gate determining the balance between the two. The final hidden state h_t is computed as $h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t$.

2.3.2 Transformer

Transformers have emerged as a powerful architecture in the domain of artificial intelligence and machine learning, particularly renowned for their exceptional performance in natural language processing tasks. Originally introduced in 2017 by Vaswani et al. [63], transformers have since become a cornerstone in various applications beyond text processing, including time series forecasting. Unlike traditional recurrent neural networks, which rely on sequential processing or local receptive fields, transformers leverage a self-attention mechanism to capture global dependencies within a sequence of data. This self-attention mechanism allows transformers to effectively model complex relationships across different positions in the input sequence, making them particularly adept at capturing long-range dependencies.

The transformer architecture, developed originally for sequence-to-sequence word-related tasks, is made of the the following building blocks:

Word Embeddings In order to facilitate the integration of input and output words into numerical data for utilisation in neural networks, a pivotal step involves converting them into numerical representations. Word embedding, a commonly employed technique in neural networks, serves this purpose effectively.

The underlying principle of word embedding leverages a simple neural network architecture wherein each word and symbol within the target vocabulary corresponds to a distinct input node. Notably, an additional input node is designated to represent the symbol EOS, denoting *end of sentence* or *end of sequence*. Given the potential variability in the vocabulary, encompassing words, word fragments, and symbols, each input is referred to as a token. The same word embedding network is re-utilised for every input word or symbol, ensuring uniformity across input sentences of varying lengths.

Positional Encoding Positional encoding is a fundamental technique employed by Transformers to preserve the sequential order of words within a sentence. The position of a word is encoded with a sequence of values having the same dimension as the word embedding. Numerical values denoting word order are derived from a sequence generated by alternating sine and cosine functions with increasing wavelength. This way of selecting the values ensures that a distinct vector is assigned to each possible position. To integrate positional information with word embeddings, the positional values are summed to the embedding values, enabling Transformers to effectively retain and utilise information regarding word order.

Self-Attention Mechanism The self-attention mechanism is the core component of the transformer architecture. It allows the model to weigh the importance of each element in the input sequence with respect to every other element, including itself, enabling it to capture long-range dependencies and relationships within the sequence. To compute the self-attention weights, the input embeddings are transformed into three sets of vectors: query, key, and value. These transformations are achieved through learned linear projections.

- The query vectors represent the elements of the sequence for which we want to compute attention weights.
- The key vectors represent the elements that we want to compare against to determine the relevance of each query.
- The value vectors represent the information associated with each element in the sequence.

Therefore we first do a dot product between the query and the transpose of the key matrix. The output of this dot product can be called an attention filter. At the start of the training process, the contents of the attention filter are more or less random numbers, but once the training process is done, they take on more meaningful values that are, in fact, attention scores. To ensure that more similar words exert greater influence on the encoding of the word under consideration, the calculated similarities are passed through a softmax function. The softmax function preserves the order of input values while transforming them into probabilities, thus determining the relative contribution of each input word to the encoding of the word being examined. The attention weights are hence used to compute a weighted sum of the corresponding value vectors, where elements with higher attention weights contribute more to the final output. The resulting vector represents the output of the attention mechanism for each query. Eventually, the embedded words are summed to their attention value, computing the residual connection. This process ensures that each word's representation is informed by its relationship with other words in the sentence, facilitating comprehensive understanding and modelling of word associations within the Transformer architecture.

Multi-Head Attention In practice, the transformer employs multi-head attention, where the self-attention mechanism is applied multiple times in parallel, each with its own set of query, key, and value projections. This allows the model to attend to different aspects of the input sequence simultaneously, enhancing its ability to capture diverse patterns and relationships.

Decoder The decoder part of the transformer has an analogous structure to the encoder block, i.e. a word embedding network, a positional encoding, a multi-head attention layer, and finally a residual connection operation. It has to be emphasised that the target dictionary undergoes its own word embedding. For example, if the task is translating words from English to Italian, the input word embedding would encode the English dictionary, while the output word embedding would encode the Italian one.

Encoder-Decoder Attention The self-attention mechanism keeps track of how the different elements are related within a sequence. Similarly, the encoder-decoder attention is a set of queries, keys, and values that learn the relationships between the input and the output sequence. This is a crucial step in order to correctly preserve the important information contained in the input and correctly predict the output sequence. The set of vectors utilised for encoder-decoder attention differs from those employed in self-attention, as it learns a distinct set of weights. On top of the encoder-decoder attention layer, another residual connection is performed in order to keep the embedded information.

Final Fully Connected Layer The final output is produced by a fully connected layer, i.e. a single set of weights and biases applied to the input vector. The output layer has the same dimension as the output dictionary; hence, the output values are processed with a softmax activation function in order to derive which word (or value) has the highest probability.

2.3.3 Temporal Fusion Transformer

The Temporal Fusion Transformer (TFT) [64] represents a cutting-edge deep learning model specifically tailored for time series forecasting. Leveraging the architecture of transformers, it aims to achieve remarkable accuracy while providing a thorough understanding of temporal dynamics. This is achieved through careful organisation of its components and the introduction of novel strategies. The TFT is engineered for multi-horizon forecasting, meaning it can predict outcomes for multiple time steps ahead. Additionally, it operates in a multivariate manner, capable of processing input from and forecasting outcomes for multiple time series simultaneously. Furthermore, the TFT is characterised by its probabilistic nature. Instead of generating single data points for each time step, it produces a set of quantiles that represent a probability distribution for the forecasted point. To capture both local and long-range dependencies, the model integrates various strategies across its architecture, including LSTM and self-attention mechanisms. In pursuit of interpretability, the TFT implements features such as multi-head attention. This aids users in identifying globally-important variables for the prediction task, significant events, and persistent temporal patterns.

Input type The model is designed to accommodate diverse inputs, processing them through distinct structures. Specifically, it incorporates past inputs, future inputs, and static metadata. Past inputs represent the data points from the time series that are already known, i.e., the historical observations collected. Future inputs denote the events that are anticipated to occur in the future. Static metadata encompasses supplementary information associated with the time series that remains constant over time. To illustrate this concept with an example, consider a chain of retail store's business that seeks to forecast sales of a particular product. Past inputs would include historical sales data for the product over previous

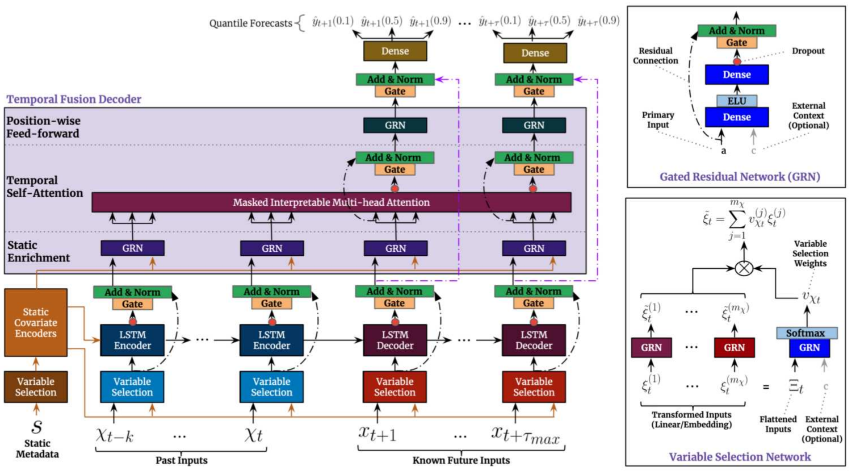


Figure 2.3: TFT architecture as depicted in the original paper [64]

periods, perhaps daily, weekly, or monthly records. Additionally, it might incorporate time series data on factors such as pricing fluctuations or inventory levels. Future inputs could encompass upcoming holidays or special events that are expected to impact sales. Static metadata might instead include information such as store locations, size, *etc.*, i.e. details that remain constant over time and provide context for understanding sales patterns.

Model overview The TFT model is made of multiple different building blocks designed to effectively capture the information contained in the provided time series. Time series are passed to a Variable Selection Network that determines the relevance of each variable. Data is then processed by a sequence-to-sequence layer, i.e. an LSTM encoder-decoder, and by a temporal self-attention decoder that utilises interpretable multi-head attention, to capture both the dependencies from close data and far ones. The result of the model is a quantile forecast prediction. In contrast to other time-series forecasting architectures, TFT is meticulously designed to incorporate information from static metadata; therefore, it utilises different encoders to generate four distinct context vectors from the static covariance. These vectors are strategically integrated into various parts of the temporal fusion decoder, where static variables play a crucial role in processing, namely contexts for temporal variable selection, local processing of temporal features, and enhancement of temporal features with static information.

Gated Resudual Network One of the most important building blocks of the TFT is the gating mechanism. This component is designed to provide the model with the flexibility to apply non-linear processing only where necessary. Such a feature is particularly important when the precise relationship between exogenous inputs and targets is unknown in advance, making it challenging to anticipate which variables are relevant. The gating mechanism is performed by the Gated Residual Network (GRN), which is the building block of TFT designed to address this challenge.

The GRN takes a primary input a and an optional context vector c and yields a processed output as follows:

$$\text{GRN}_\omega(\boldsymbol{a}, \boldsymbol{c}) = \text{LayerNorm}\left(\boldsymbol{a} + \text{GLU}_\omega\left(\boldsymbol{\eta}_1\right)\right), \tag{2.1}$$

$$\boldsymbol{\eta}_1 = \boldsymbol{W}_{1,\omega}\boldsymbol{\eta}_2 + \boldsymbol{b}_{1,\omega}, \tag{2.2}$$

$$\boldsymbol{\eta}_2 = \text{ELU}\left(\boldsymbol{W}_{2,\omega}\boldsymbol{a} + \boldsymbol{W}_{3,\omega}\boldsymbol{c} + \boldsymbol{b}_{2,\omega}\right), \tag{2.3}$$

$$\text{GLU}_\omega(\boldsymbol{\gamma}) = \sigma\left(\boldsymbol{W}_{4,\omega}\boldsymbol{\gamma} + \boldsymbol{b}_{4,\omega}\right) \odot \left(\boldsymbol{W}_{5,\omega}\boldsymbol{\gamma} + \boldsymbol{b}_{5,\omega}\right) \tag{2.4}$$

Where $\boldsymbol{\eta}_1,\ \boldsymbol{\eta}_2 \in \mathbb{R}^{d_{\text{model}}}$ are two intermediate layers and $W_{\cdot,\omega}$ and $\boldsymbol{b}_{\cdot,\omega}$ are weights and biases of the model. ELU is the Exponential Linear Unit [65] activation function that acts as an identity function when the argument is much greater than zero, and linearly when the output is much less than zero. GLU is instead the Gated Linear Unit [66], an activation function that allows the suppression of any parts of the architecture that are not required for a given dataset, providing adaptive depth and network complexity to accommodate a wide range of datasets and scenarios. It exploit the sigmoid $\sigma(\cdot)$ and the element-wise Hadamard product $\odot$. During training, dropout is applied before the gating layer and layer normalisation to enhance the model's robustness and generalization capabilities. Eventually, the GRN perform Layer Normalisation [67] to stabilise the training of neural networks by normalising the inputs to neurons within each layer, thereby reducing the impact of internal covariate shift and improving the network's convergence and generalization performance.

Variable Selection Network One of the challenges when dealing with multivariate time series is understanding which variables are relevant for a specific case. The TFT is specifically designed to provide instance-wise variable selection through one of its building blocks, known as the Variable Selection Network, which is utilised for analysing both static and time-dependent data. This approach not only identifies the most influential variables for prediction but also eliminates unnecessary noisy inputs, thereby enhancing performance. Given that many real-world time series datasets contain features with limited predictive value, variable selection optimises model performance by focusing learning capacity on the most relevant variables. Static, past, and future inputs use three different Variable Selection Networks. Categorical variables are represented as features with entity embedding [68], while continuous variables undergo a linear transformation so that they are mapped into a d_{model}-dimentional vector. Each of these transformed inputs $\xi_{\mathbf{t}}^{(\mathbf{i})}$ is stored in a flattened vector $\mathbf{\Xi_t} = \left[\xi_{\mathbf{t}}^{(\mathbf{1})}, \ldots, \xi_{\mathbf{t}}^{(\mathbf{n})}\right]$. Both $\mathbf{\Xi_t}$ and the context vector c_s, obtained by the static covariate encoder, are taken to obtain the vector of variable selection weights $v_{\chi_t} = \text{Softmax}\left(\text{GRN}_{\text{v}_\chi}\left(\mathbf{\Xi_t}, \text{c}_\text{s}\right)\right)$. Each transformed input is also processed by its own GRN as $\tilde{\xi}_t^{(j)} = \text{GRN}_{\tilde{\xi}(j)}\left(\boldsymbol{\xi}_t^{(j)}\right)$. At last, the processed features are weighted by their variable selection weights and combined: $\tilde{\xi}_t = \sum_{j=1}^{m_\chi} v_{\chi_t}^{(j)} \tilde{\xi}_t^{(j)}$

Interpretable Multi-Head Attention In transformer-based architecture the multi-head attention mechanism [63] scales values $\mathbf{V}$ according to the relationship between keys $\mathbf{K}$ and queries $\mathbf{Q}$ as: $\text{Attention}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = A(\mathbf{Q}, \mathbf{K})\mathbf{V}$ in which $A(\cdot)$ is a normalisation function. The standard multi-head attention mechanism trains multiple heads in parallel, each conducting its own attention computation simultaneously. The results of these computations are then concatenated and multiplied by head-specific weights. However, this approach lacks interpretability. Therefore, TFT adopts an interpretable multi-head attention mechanism. This solution utilises a shared value matrix and additive averaging of attention matrices instead of concatenation. This enhances interpretability by ensuring that attention values are not disregarded.

Temporal Fusion Decoder The decoder part of the TFT is composed of multiple different layers. Firstly,data is enhanced levering by the local context by utilising an LSTM encoder-decoder [59]. Past data is fed to the encoder, while future data is progressively passed to the decoder. Static metadata influences local processing through context vectors from static covariate encoders. A gated skip connection is also employed for enhanced connectivity. The following layer is called Static Enrichment Layer and utilises shared-weight gated residual networks to enrich temporal features with static information from covariate encoders. Afterwards, interpretable multi-head attention is performed to grasp long-range dependencies effectively. All enriched temporal features are consolidated into a single matrix, and multi-head attention is applied at each forecast time. To preserve causal information flow, decoder masking is applied. Eventually, an additional shared-weight gated residual network is applied across the layer. The output of the model is a prediction of different percentiles at each future time step, called quantile forecasts; during the training the quantile loss is minimised, summed across all quantile outputs [69].

Chapter 3

Model

3.1 Data Acquisition

The initial step of this data analysis involves collecting the necessary data from the online dataset where it is stored. Crafting well-constructed queries is crucial for accessing relevant information within the dataset and also for performing a portion of the preprocessing and cleaning directly during the fetching phase. All the industrial machines of our interest automatically store data coming from their sensors in an Influx dataframe. InfluxDB is an open-source time-series database created by InfluxData and designed to handle high write and query loads while providing high availability and scalability. It uses a SQL-like query language called InfluxQL, which allows for complex queries on time-stamped data in real-time. Machine metadata, such as the serial number or the model name, is instead stored on a Mongo database, a widely used open-source NoSQL database management system that provides a flexible, scalable, and high-performance solution for storing and managing data.

The process of storing data involves several steps. Initially, the sensors bring the signals to the PLC, or Programmable Logic Controller, which is the programmable device controlling the machine. Variables are sent to an edge node using the Modbus protocol, commonly used in industrial automation for PLC communication. The edge node is a network node usually located near devices or sensors that collect data, and it is used to process or transmit data directly to the cloud or other processing systems. This node transmits data to the cloud via MQTTs, a lightweight messaging protocol for device communication with bandwidth or network restrictions, often used in IoT applications. In particular, MQTTs differs from MQTT since it adds security through SSL/TLS encryption for data exchange. Besides the node, there's an edge gateway, a concentrator of connections to and from the machine, functioning as a firewall and managing information routing for security. Once it reaches the cloud, namely AWS (Amazon Web Services), a well-known cloud-services platform, data is processed using a protocol with a centralised broker for scalability. Subsequently, the pipeline receiving data streams transforms them into KPIs, or Key Performance Indicators, which are used to evaluate process, project, or organisational performance. The architecture is serverless, designed to execute necessary functions without direct server management. Servers are hence able to adjust their numbers based on machine traffic conditions. For instance, at 7 in the morning, when machines activate, the architecture expands tenfold before returning to its normal size. Metadata information, which contains customer details, is instead exported from the company's local database to MongoDB through periodic calls.

Data are collected using a specifically designed Python function that can retrieve them according to some parameters, such as the list of desired variables or the start and stop times for data retrieval. Firstly, the function exploits a Python library called `influxdb-client`, which handles the connection with the server and facilitates data transfer. The process begins by initialising an `InfluxDBClient` object. This object is instantiated by providing specific information, notably the URL of the InfluxDB server API, which identifies the database's location, an authentication token, and the name of the organisation. Once the client object is instantiated, it is possible to initialised a query client with the `.query_api()` method. The query structure is written in InfluxQL language, and it is stored inside a multi-line string. Writing the query is completed at run-time using the built-in `.format()` method with the addition of information provided by the user or collected from the metadata. The script retrieves metadata about machines from a MongoDB database using the `MongoClient` function of the `pymongo` package. It begins by establishing a connection to the database by passing a token to the client. Then the machines collection is accessed within the specified database, and documents are retrieved if the `activation_date` field exists, i.e. if the machine has been put into production. For each machine document found, it is stored in a dictionary where the serial number acts as the key. This dictionary is exploited for checking which machines are active and which are their respective models. Subsequently, the InfluxDB query is finalised by incorporating user-supplied details. Finally, it is executed once for each distinct machine, either within a user-defined list or across all potential machines. Initially, the query filters data corresponding to the provided serial number. Then, it selects specific fields referring to the different signal variables and pivots the data, organising it by time while creating distinct columns for each field. A separate table is generated for each variable to apply a time aggregation function, allowing for data point resampling across broader time windows. This involves applying functions such as mean, max, or selecting the initial value. The choice of function for each variable can be tailored according to specific requirements. Afterward, the query removes all unnecessary columns, sorts them by ascending time, and combines all data tables into a single one using the union operation. Eventually, the table is yielded as a Pandas dataframe. This process is repeated for each required machine's serial, and the final output has the shape of a dictionary of Pandas dataframes.

A similar function is performed for fetching data related to the alarm logs. Since the query does not perform any resampling, it turns out to be much simpler. Data are selected within a certain range and filtered depending on the serial number of the different machines. Then they are filtered again depending on the code associated with the alarms that we want to collect. At last data, the are pivoted and returned in a Pandas dataframe for each machine.

Metadata are collected using a function named `get_machines()`, which is designed to retrieve machine information from a MongoDB database. The function first establishes a connection to the database using the pymongo library, then, within a try-except block, accesses a specific collection named "machines" and queries for documents where the `activationDate` field exists. For each document retrieved from the query, the function extracts the serial number of the machine and stores it as the key in a dictionary, while the entire document is stored as the value. Finally, the function ensures that the MongoDB client connection is closed, and it returns the result dictionary containing the metadata of machines retrieved from the database. A similar function can instead acquire the metadata of the alarms, that are fetched from a different collection.

3.1.1 Signal Description

The collected dataset is organised into Pandas dataframes. Various relevant signals can be gathered. In particular, the `ST` signal represents the status of the machine at a certain timestamp. This signal is a 32-bit integer that uses each bit to store different information, such as whether production is active, if the machine is running slower than expected, or if the cleaning procedure of the machine is in progress. Since the signal is collected as an integer, it needs to be converted into a binary representation for a correct analysis of the information contained.

Another relevant signal is `RA`, which represents the identification number of the selected recipe. The format is a 32-bit integer that records which recipe is selected, indicating how the machine should behave according to preset parameters. Different recipes are selected for different carton sizes or products to be packed. Unfortunately, the recipe numbers vary for each customer, rendering them unreliable for providing information about the machine's behavior.

Signal	**Parameter**	**Unit**
PVCA	Current Welding Energy	$\%\times10$
PVCF	Sonotrode Operative Frequency	Hz
PVCP	Current Welding Power	W
PVCPP	Current Welding Power Percent	%
PVWT	Welding Duration Set Point (if no energy control)	ms
PVWE	Applied Energy at Last Sealed Cap	J
PVWP	Applied Power at Last Sealed Cap	$\%\times10$
SPWE	Welding Energy Set Point (when energy control)	J
SCAC	Caps applicator counter	$h\times10$

Table 3.1: Cap applicator parameters

The cap-applicator module record many measurements acquired from the different sensors present in the machinery. Tab.3.1 shows the characteristics of the different signals. These signals encompass a range of parameters critical for monitoring and controlling the welding process. They include measurements related to the current welding operation, such as the percentage of welding energy utilised, the supplied power, its percentage compared to the maximum possible value, and the sonotrode frequency. Additionally, another set of signals tracks the settings used for the previous welding of a cap on the carton, including the applied energy and power percentage. Finally, other variables record the machine settings in terms of welding duration or energy value. Parameters are typically differentiated into line A parameters, denoted by the suffix "_A", and line B parameters, distinguished by the suffix "_B", with the exception of the SCAC variable, which tracks the number of caps applied globally.

Since this work is primarily focused on the cap applicator, no other signals were collected at this stage, facilitating the development of a playground model for initial analysis and exploration, which, however, has all the desired characteristics. This opens up to the possibility of expanding the model by collecting and integrating new variables. Additionally, the volume of collected data, consisting solely of signals from the cap applicator, already raises concerns about the resource requirements for training the deep learning model. This necessitates enhancing the allocated resources for further explorations.

Alarm Description The machinery is capable of generating alarm signals when various issues are detected. The alarm variable `ASR` is an integer code of three to four digits that uniquely corresponds to a particular warning. These signals are saved in real-time on the cloud platform in the same manner as sensor signals. Each alarm signal is saved along with its `ASR` code, the timestamp of when it was recorded, and the machinery that produced it. The alarm signals for the cap applicator encompass a wide spectrum of severity, ranging from minor issues such as cap shortages or absences in the slit or caps not loaded on the positioner to more serious anomalies like alarms received from the ultrasonic generator, which likely indicate a component failure.

3.2 Data Visualization

After gathering the data, an important step consists to perform proper data visualisation in order to collect visual information about the dataset that will be useful for subsequent analysis, comparison with machinery specialists, and decision-making processes. To fulfil this requirement, a dashboard was developed using Bokeh [70], a Python library designed for data visualisation that provides an array of tools for generating interactive plots. The dashboard, visible in Fig.3.1 and 3.2 is composed of two different tabs, one for detailed data visualisation and another for the comparison between industrial signals and the occurrence of alarm events. The dashboard has the following characteristics:

- A `TextInput` bar for selecting the folder in which data is stored
- A `MultiSelect` widget for visualising multiple plots at the same time
- A `RadioButtonGroup` that allows to select a specific signal to visualise
- Another `RadioButtonGroup` for selecting signals coming from production line A, production line B or both.
- A resample `Button` that plots a mean signal by resampling the data in a time-window given by the user inside a `TextInput` bar.

- The main plot, displaying the visual representation of the data time series. Each distinct data file is depicted using unique colours, while the two separate production lines of the same machine are distinguished by different point shapes (circles or triangles). When the resample function is enabled, the dots become more translucent, and a delicate line representing the moving average is superimposed. Furthermore, the axis labels and graph title adjust dynamically based on the selected signal and are updated with the appropriate name and unit of measure. The interactive plot includes default tools such as panning and zooming, enhancing its usability.

- An additional smaller plot, displaying histograms of the data points for each production line of every machine. Histograms belonging to the same machine are depicted with the same color but distinct filling patterns. These histograms feature horizontally oriented bars and share the y-axis with the main plot, ensuring that when one is moved, the other adjusts accordingly.

- A `DataTable`, present in the second tab, that displays the alarms recorded for a particular data interval. The table reports the alarm codes, their descriptions, and the number of times each has occurred. By selecting one or more lines, the corresponding alarms are shown as vertical lines on the main plot, with their identifying codes next to each line.

3.3 Preliminary Analysis

After the development of a visualisation dashboard, it is possible to undertake a comparative assessment with the experts within the organisation to discern the potential insights retrievable from the signals. Consequently, a significant segment of the initial endeavour has been dedicated to ascertaining whether indications of machinery malfunction are discernible within the signals. Notably, the signals exhibit a scattered distribution, exhibiting erratic or volatile behaviour characterised by abrupt fluctuations or irregular movements, thus lacking a discernible regular pattern.

Preliminary analyses encompass the inspection of correlation matrices and Principal Component Analysis (PCA) on the signals. Correlation matrices allow for the exploration of linear relationships between variables, providing insights into potential dependencies among signal components. Meanwhile, PCA seeks to

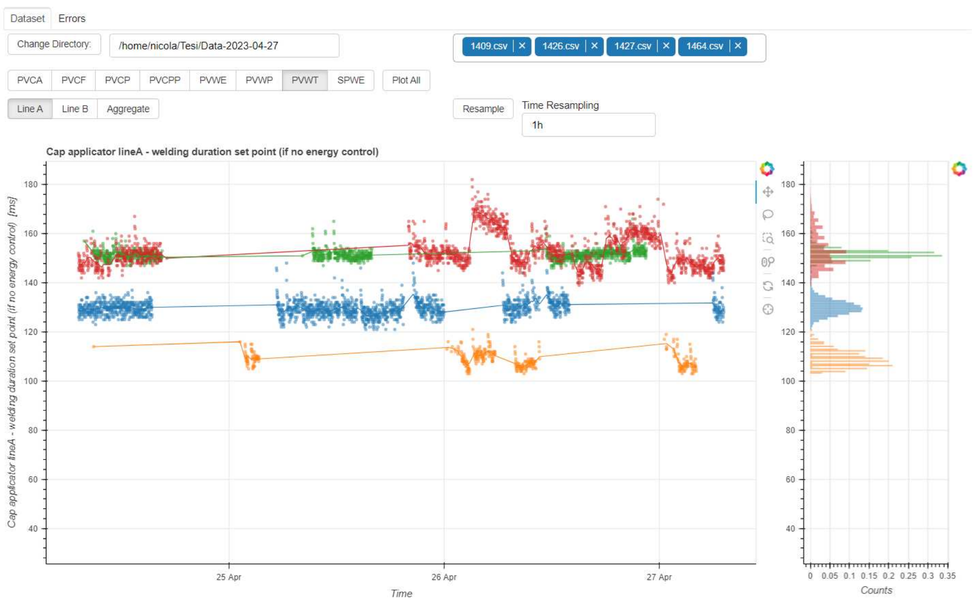
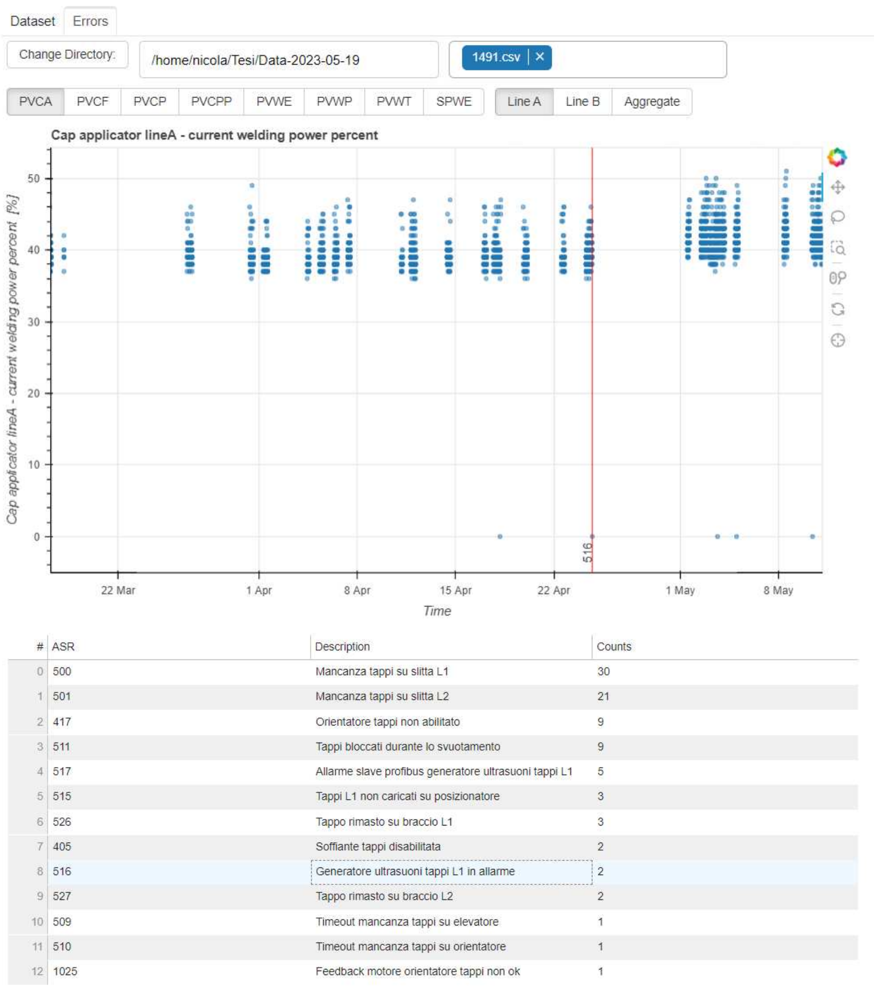

Dataset
Errors
Change Directory:
/home/nicola/Tesi/Data-2023-04-27
1409.csv
1426.csv
1427.csv
1464.csv
PVCA
PVCF
PVCP
PVCPP
PVWE
PVWP
PVWT
SPWE
Plot All
Line A
Line B
Aggregate
Resample
Time Resampling
1h
Cap applicator lineA - welding duration set point (if no energy control)
Cap applicator lineA - welding duration set point (if no energy control) [ms]
25 Apr
26 Apr
27 Apr
Time
Counts

Figure 3.1: Dashboard for advanced time-series visualisation

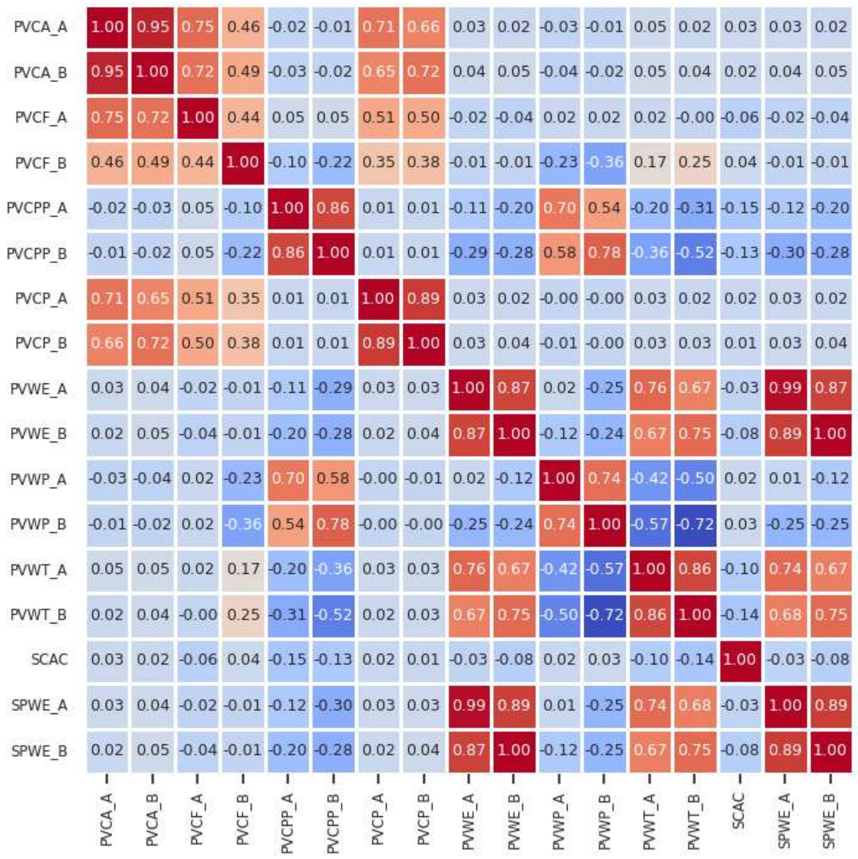

Dataset
Errors
Change Directory:
/home/nicola/Tesi/Data-2023-05-19
1491.csv
PVCA
PVCF
PVCP
PVCPP
PVWE
PVWP
PVWT
SPWE
Line A
Line B
Aggregate
Cap applicator lineA - current welding power percent
Cap applicator lineA - current welding power percent [%]
516
22 Mar
1 Apr
8 Apr
15 Apr
22 Apr
1 May
8 May
Time

#	ASR	Description	Counts
0	500	Mancanza tappi su slitta L1	30
1	501	Mancanza tappi su slitta L2	21
2	417	Orientatore tappi non abilitato	9
3	511	Tappi bloccati durante lo svuotamento	9
4	517	Allarme slave profibus generatore ultrasuoni tappi L1	5
5	515	Tappi L1 non caricati su posizionatore	3
6	526	Tappo rimasto su braccio L1	3
7	405	Soffiante tappi disabilitata	2
8	516	Generatore ultrasuoni tappi L1 in allarme	2
9	527	Tappo rimasto su braccio L2	2
10	509	Timeout mancanza tappi su elevatore	1
11	510	Timeout mancanza tappi su orientatore	1
12	1025	Feedback motore orientatore tappi non ok	1

Figure 3.2: Dashboard for time-series and alarm visualisation

uncover the underlying structure within the signal data by identifying orthogonal axes of maximal variance.

Correlation Matrix The correlation matrix is a square matrix that summarises the correlations between pairs of variables in a dataset. In essence, it quantifies the strength and direction of the linear relationship between variables. Each entry in the matrix represents the correlation coefficient between two variables, ranging from -1 to 1. A correlation coefficient close to 1 indicates a strong positive relationship, meaning that as one signal increases in value, the other tends to increase as well. Conversely, a value close to -1 signifies a strong negative relationship, indicating that as one signal increases, the other tends to decrease. A correlation coefficient near 0 suggests little to no linear correlation between the variables. By analysing the correlation matrix of the different signals of the cap-applicator, it is possible to gain a first insight into the interdependencies of signals.



	PVCA_A	PVCA_B	PVCF_A	PVCF_B	PVCPP_A	PVCPP_B	PVCP_A	PVCP_B	PVWE_A	PVWE_B	PVWP_A	PVWP_B	PVWT_A	PVWT_B	SCAC	SPWE_A	SPWE_B
PVCA_A	1.00	0.95	0.75	0.46	-0.02	-0.01	0.71	0.66	0.03	0.02	-0.03	-0.01	0.05	0.02	0.03	0.03	0.02
PVCA_B	0.95	1.00	0.72	0.49	-0.03	-0.02	0.65	0.72	0.04	0.05	-0.04	-0.02	0.05	0.04	0.02	0.04	0.05
PVCF_A	0.75	0.72	1.00	0.44	0.05	0.05	0.51	0.50	-0.02	-0.04	0.02	0.02	0.02	-0.00	-0.06	-0.02	-0.04
PVCF_B	0.46	0.49	0.44	1.00	-0.10	-0.22	0.35	0.38	-0.01	-0.01	-0.23	-0.36	0.17	0.25	0.04	-0.01	-0.01
PVCPP_A	-0.02	-0.03	0.05	-0.10	1.00	0.86	0.01	0.01	-0.11	-0.20	0.70	0.54	-0.20	-0.31	-0.15	-0.12	-0.20
PVCPP_B	-0.01	-0.02	0.05	-0.22	0.86	1.00	0.01	0.01	-0.29	-0.28	0.58	0.78	-0.36	-0.52	-0.13	-0.30	-0.28
PVCP_A	0.71	0.65	0.51	0.35	0.01	0.01	1.00	0.89	0.03	0.02	-0.00	-0.00	0.03	0.02	0.02	0.03	0.02
PVCP_B	0.66	0.72	0.50	0.38	0.01	0.01	0.89	1.00	0.03	0.04	-0.01	-0.00	0.03	0.03	0.01	0.03	0.04
PVWE_A	0.03	0.04	-0.02	-0.01	-0.11	-0.29	0.03	0.03	1.00	0.87	0.02	-0.25	0.76	0.67	-0.03	0.99	0.87
PVWE_B	0.02	0.05	-0.04	-0.01	-0.20	-0.28	0.02	0.04	0.87	1.00	-0.12	-0.24	0.67	0.75	-0.08	0.89	1.00
PVWP_A	-0.03	-0.04	0.02	-0.23	0.70	0.58	-0.00	-0.01	0.02	-0.12	1.00	0.74	-0.42	-0.50	0.02	0.01	-0.12
PVWP_B	-0.01	-0.02	0.02	-0.36	0.54	0.78	-0.00	-0.00	-0.25	-0.24	0.74	1.00	-0.57	-0.72	0.03	-0.25	-0.25
PVWT_A	0.05	0.05	0.02	0.17	-0.20	-0.36	0.03	0.03	0.76	0.67	-0.42	-0.57	1.00	0.86	-0.10	0.74	0.67
PVWT_B	0.02	0.04	-0.00	0.25	-0.31	-0.52	0.02	0.03	0.67	0.75	-0.50	-0.72	0.86	1.00	-0.14	0.68	0.75
SCAC	0.03	0.02	-0.06	0.04	-0.15	-0.13	0.02	0.01	-0.03	-0.08	0.02	0.03	-0.10	-0.14	1.00	-0.03	-0.08
SPWE_A	0.03	0.04	-0.02	-0.01	-0.12	-0.30	0.03	0.03	0.99	0.89	0.01	-0.25	0.74	0.68	-0.03	1.00	0.89
SPWE_B	0.02	0.05	-0.04	-0.01	-0.20	-0.28	0.02	0.04	0.87	1.00	-0.12	-0.25	0.67	0.75	-0.08	0.89	1.00

Figure 3.3: Correlation matrix of the cap applicator signals

From Fig.3.3 we can identify some notable information. In primis, it is notable that the majority of the variable pairs show almost no correlation. This suggests that all the variables are indeed meaningful and that there is no redundant information stored. There are, however, some notable correlations to underline. In particular, signals registered by the two different lines of production are usually highly correlated ($corr(\_A, \_B) > 0.85$), which is easily explained since the two lines work simultaneously and it is hence likely that they will register similar outputs.

Principal Component Analysis Principal Component Analysis (PCA) [71] is a dimensionality reduction technique widely used in data analysis. The fundamental idea behind this technique is to transform a dataset containing possibly correlated variables into a new set of uncorrelated variables called principal components, which are linear combinations of the original variables and are ordered in such a way that the first few capture the maximum variance present in the data. PCA achieves this transformation by finding the eigenvectors and eigenvalues of the covariance matrix of the original data, with the eigenvectors representing the directions of maximum variance and the eigenvalues indicating the magnitude of variance along these directions. Through this process, PCA allows for the extraction of meaningful patterns and structures from complex datasets. Therefore, by retaining only the most significant principal components, PCA helps to simplify the dataset while preserving as much of the original information as possible. This technique can be leveraged in multivariate time series analysis to estimate the independence of variables. In the context of such analysis, where each channel of the series represents a dimension, performing PCA facilitates the determination of the number of dimensions that would be needed if various channels were subjected to those linear combinations.

To establish the amount of information contained in each dimension extracted by PCA, the proportion of variance explained by each principal component is examined. This is typically achieved by analysing the eigenvalues of the covariance or correlation matrix, that represent the amount of variance captured by each principal component. A common practice is to calculate the proportion of variance explained (PVE) by each principal component, which is obtained by dividing the eigenvalue of each principal component by the sum of all eigenvalues. This provides a percentage that indicates the proportion of total variance captured by each dimension. Before performing PCA, it is common practice to scale the data using techniques such as min-max scaling. Scaling is essential because PCA is sensitive to the magnitude of the variables involved. When variables have different scales or units, those with larger scales can dominate the variance calculations, leading to biased results. Leveraging scikit-learn `MinMaxScaler`, we transform the data so that each variable is confined to a range between 0 and 1, ensuring that all variables contribute equally to the following analysis.

#	**PVE (%)**
PC1	43.70
PC2	22.32
PC3	11.72
PC4	7.57
PC5	4.77
PC6	3.32
PC7	2.30
PC8	1.79
PC9	1.24
PC10	0.71

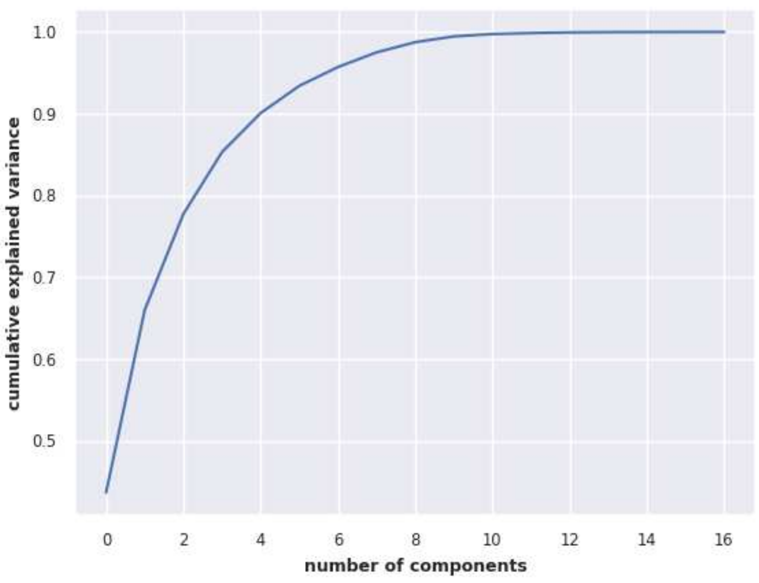


Figure 3.4: Proportion of variance explained by the 10 most relevant principal components (*left*) and cumulative explained variance (*right*)

The results of the PCA analysis for the entire dataset are visible in Fig.3.4. It can be noticed that the variance is well split among different principal components (PCs). In particular, the first nine PCs contribute at least to explain 1% of the variance each. This well-distributed variance across multiple PCs implies that no single component dominates the variability in the dataset, suggesting that the underlying structure of the data is complex and multidimensional and reinforcing the request for a multivariate approach.

Alarm analysis The preliminary analysis of signals in our study prompted the consideration of incorporating the alarm log information to further enrich our understanding of the anomalous events. To gain insights into alarm behaviour across different machines, we conducted initial analyses focusing on alarm patterns and occurrences. The proposed methodology combines DBSCAN clustering technique [21] and Apriori pattern detection [72] to identify common clusters of alarm logs within a time series dataset.

First, the DBSCAN algorithm is applied to cluster the alarm logs based on their distance in the temporal space. In particular, DBSCAN evaluates the clusters based on their spatial density, defining clusters as regions of high density separated by regions of low density. This allows DBSCAN to identify clusters of varying shapes and sizes, effectively distinguishing between core points and noise. Subsequently, the Apriori algorithm is employed for association rule mining on the clustered alarm logs. The Apriori algorithm [72] is a renowned method for frequent pattern discovery that helps identify associations among items in large datasets. The algorithm exploits the so-called apriori property, which asserts that any subset of a frequent itemset must also be frequent. The Apriori algorithm efficiently extracts frequent itemsets by iteratively discovering increasingly larger itemsets from the dataset. It begins by identifying all frequent individual items, then extends these sets by one item at a time, checking if the resulting set meets the minimum support threshold. If a set does not meet the threshold, its supersets are pruned. This process continues until no further extension of frequent itemsets is possible. This algorithm is thus utilised to identify patterns between different alarms occurring together, defining itemsets representing commonly occurring alarm combinations within each DBSCAN cluster. Association rules extracted from these itemsets specify relationships between alarms and their occurrences within clusters.

The analysis was deliberately conducted using basic approaches to ascertain the potential for integrating alarm signals with sensor time series data. By adopting a simple approach, we aimed to gain initial insights into the feasibility and effectiveness of combining these two types of data streams. This methodology, however, has shown no relevant results, and no meaningful recurrent set of error have been found. Moreover, in the literature previous researches have already explored advanced techniques for alarm forecasting using similar signals [73, 74]. Nevertheless, this study diverges in focus: rather than delving into complex forecasting models tailored specifically for alarm prediction, it is primarily interested in the exploration and analysis of time series derived from the signals themselves. This strategic decision stems from the understanding that alarm signals have already been extensively investigated in this context, and our aim is to complement this existing knowledge by investigating the temporal patterns and trends inherent in the sensor data.

3.4 Preprocessing

Before feeding the collected data into our model as a training dataset, it is important to preprocess the data to ensure its suitability for analysis. Preprocessing serves as a crucial preparatory step, involving various techniques aimed at cleaning, transforming, and structuring the raw data into a format conducive to subsequent analysis.

Initially, the data retrieved from online storage and stored locally is accessed by reading through multiple CSV files, each corresponding to a different machine. These files are then converted into Pandas dataframes. During this process, any accompanying metadata and alarm logs are automatically excluded, ensuring that only sensor data is preserved.

After loading the dataset, the initial step involves retaining only the segments of data during which the machine was actively engaged in production. This ensures that subsequent analyses focus solely on pertinent periods, particularly for anomaly detection within the operational phases of the cap applicator module. To achieve this, the status column undergoes conversion from integer to binary representation using the appropriate Numpy function, `binary_repr`, which returns the input number as a string of bits. Subsequently, the bit corresponding to the active production status is mapped to a new column, while the original status column, now redundant, is discarded.

Now that the periods of production have been identified, the next step involves filtering the dataset to exclude periods devoid of production activity. However, it is common for machines to briefly halt production, resulting in intermittent data entries reflecting these pauses, as visible in Fig.3.5. To address this, we have chosen to categorise instances where machines cease production for less than two data points (equivalent to two minutes) as periods of active production. Conversely, instances where there are gaps of at least 3 data points between two active production periods are treated as belonging to distinct time series. Furthermore, only time series containing more than 16 data points (Fig.3.6) are deemed suitable for model training, as smaller series may lead to errors due to insufficient input size. The selected time series are still retained within the same dataframe to expedite operations. However, each one is assigned a unique identifier to facilitate their separation at a later stage.

Given that the cloud storage is optimised to conserve memory, data points are only recorded when a change in value occurs. Consequently, gaps in the dataset emerge, with NaN (Not a Number) placeholders indicating missing values. To

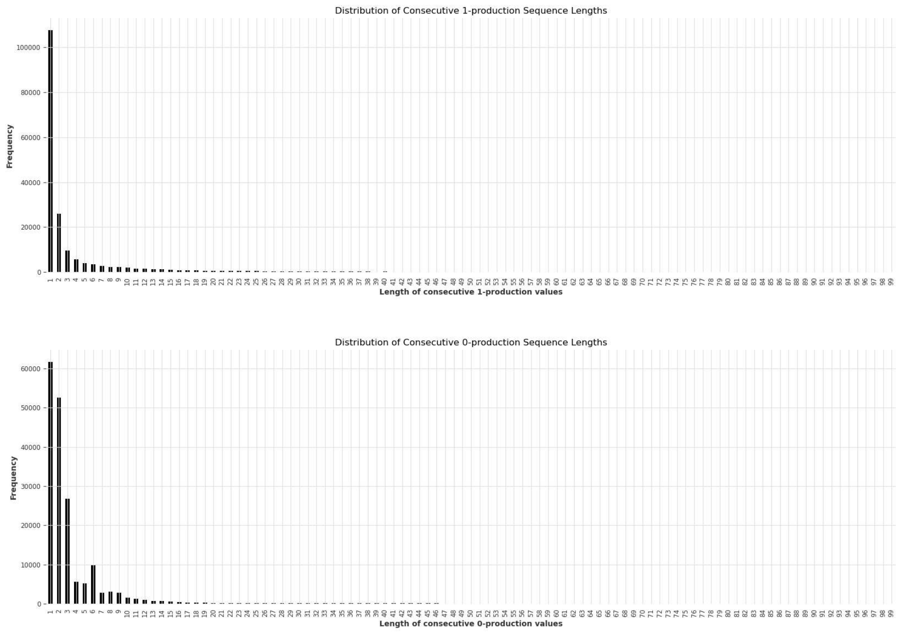


Figure 3.5: Distribution of consicutive session of production (*above*) or rest (*below*) before preprocessing divided by lenght

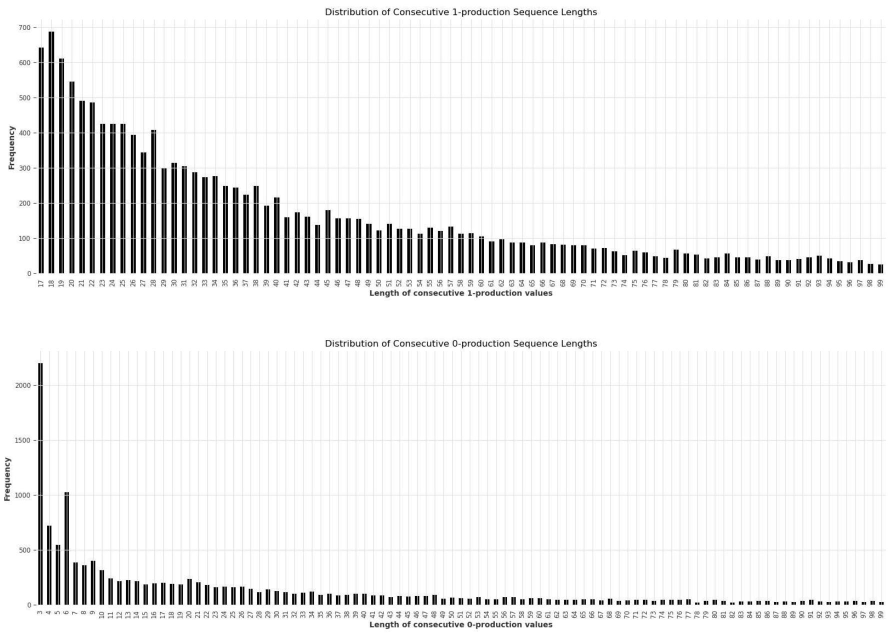


Figure 3.6: Distribution of consicutive session of production (*above*) or rest (*below*) after preprocessing divided by lenght

address this, a Pandas `fillna()` operation with forward fill option is employed to replace NaN entries with the last known value, thereby reconstructing the original data sequence. The sparsity of the dataset can be observed in Fig.3.7

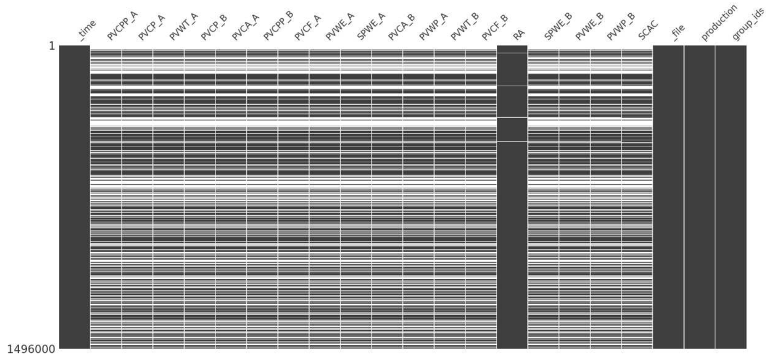


Figure 3.7: Distribution of data (black) and missing values (white) along the time series for different variables.

To ensure consistency, the datetime index of the dataframe is converted to type datetime64. Additionally, timezone information is stripped away for uniformity, as the location of production is not relevant to the analysis, and to mitigate potential errors in subsequent steps. Thereafter, irrelevant metadata, such as the CSV file source, is removed from the dataset. Furthermore, the recipe column is discarded since, as previously discussed, this signal is unreliable for forecasting as it strongly depends on the individual machine that produces it. The next step involves ensuring that all data points are uniformly spaced in time by resampling them at one-minute intervals. This time interval was chosen as it aligns with the standard sampling rate of the machinery for most of the signals. Therefore, it serves both to fill in missing minutes where the data remained unchanged and to standardise all values, as the data may have been sampled with millisecond errors, resulting in varying time intervals. Finally, the data is converted into `TimeSeries` format suitable for analysis using the Darts [75] library.

After performing common preprocessing, the entire dataset must be partitioned into three subsets: training, validation, and test datasets. The validation set serves the purpose of monitoring model performance during training to mitigate overfitting and tune hyperparameters, while the test set is reserved for evaluating performance upon completion of training. Specifically, 10% of the dataset has been allocated for both the test and validation subsets, leaving 80% of the dataset for the training set. The dataset is then rescaled using the Scikit-Learn [76] function `MinMaxScaler`, which updates the data by linearly transforming each feature to a $[0, 1]$ range, preserving the original distribution. Rescaling data before training a model is crucial for several reasons. Firstly, it ensures normalisation across features, preventing biases caused by features with larger scales from dominating those with smaller scales during training. Therefore, it allows all features to contribute equally to the model's predictions by bringing them to a similar scale, typically with decimal values in the $[0, 1]$ range. Secondly, rescaled data often leads to faster convergence during the training process, as gradient-based optimisation algorithms converge faster when features are on similar scales. Additionally, rescaling helps prevent numerical instability issues that may arise due to large differences in feature scales, thereby improving the stability of computations.

The model needs access to future covariates to function correctly. Future covariates are time series data that are already known for future time periods and do not need to be predicted. These series typically represent the passage of time, such as minutes, hours, or days of the week. They are often represented by periodic signals that rise and fall over time. For instance, minutes within an hour can be depicted by values that gradually increase from 0 to 60, forming a sawtooth pattern. This approach is valuable as it enables the model to capture time related patterns present in the data. In our dataset, there are no future covariates available for the model other than these synthetic time series. A fictional example of meaningful future covariates could be, for instance, the *expected number of filled cartons per hour*. However, the machinery does not autonomously generate any forecasted values that could aid in training the model.

3.5 Model Design and Implementation

The model selected for anomaly detection is the Temporal Fusion Transformer (TFT), discussed in Section 2.3.3.

This model has been retained as suitable for the following analysis due to some relevant properties:

- **State-of-the-Art Performance**: Given the complexity of the problem, models that have already demonstrated strong performance on time series forecasting benchmarks are preferred. As shown in the original paper, the TFT model has shown promising results in this regard, making it a suitable candidate for the tasks.

- **Multivariate Analysis**: The TFT model natively supports multivariate analysis, allowing for the incorporation of the multiple sensor signals of the examined dataset into the framework.

- **Handling Diverse Time Series**: The TFT model is capable of handling multiple time series of varying lengths. This flexibility accommodates the diverse nature of the preprocessed data, which includes different sizes of the time series due to the non-fixed duration of the production phase of each machine.

- **Contextual Understanding**: TFT leverages both Long Short-Term Memory (LSTM) networks and attention mechanisms to capture contextual information from both short and long sequences of data. This enables the model to effectively understand the temporal dependencies and contextual nuances present in the time series data, enhancing its ability to make accurate forecasts. The LSTM component is particularly useful for capturing context by focusing on nearby data points, such as understanding the phase of production at a given time. Meanwhile, the attention mechanism allows the model to detect differences in production over longer time periods; This can, for example, make the model learn how a signal behaves as the component it refers to heats up after prolonged operation.

- **Interpretability**: TFT provides built-in interpretability features that identify which data features are most indicative of machine behaviour over time. This insight aids in understanding the underlying patterns contributing to anomalies and enhances the interpretability of the model's predictions.

Limitations Although the selected architecture seems promising, it might not suit perfectly for the task, and some limitations must be taken into account. Firstly, the model is incapable of processing time series data featuring varying sets of variables. In other words, the dataset must maintain consistent time series structures with corresponding columns addressing relevant signal channels without the introduction or removal of any of them. Additionally, the model lacks the ability to manage columns entirely populated with `NaN` (not a number) values. While the model can automatically adapt to sparse missing points during the learning phase, it cannot guarantee this adaptability when confronted with portions of the datasets containing entirely missing columns. This limitation arises when the proportion of missing data becomes significant, rendering it no longer negligible. To the best of our knowledge, this problem is not addressed in literature. This limitation significantly restricts the scope of analysis, as it requires consistent series across all data points. The chosen dataset is limited to machines of a particular model and cannot be generalised to analyse the entire spectrum of machines produced by the industry under consideration. Specifically, the machinery choice is centred around a particular machine model that features a double production line, representing the dataset with the largest volume of available data.

Another limitation arises from the fact that the TFT model demands substantial computational resources, even when utilising specific parameter settings aimed at compressing the model as much as possible. Expanding the model and its capabilities would thus require an allocation of computational resources beyond current capacity, necessitating the introduction of additional resources. Additionally, there are limitations in evaluating global trends due to the possible variation of a certain signal for a certain machine due to wear. The model does not consider the individual context and temporal evolution of each machine but instead aggregates data from different machines with varying levels of wear. This hinders the model's ability to accurately assess overall trends and patterns.

3.5.1 Implementation

The TFT model has been implemented in Python using the Darts [75] package. Darts, short for "Differentiable Architecture for Time Series," is a library designed for time series forecasting and analysis that offers a comprehensive suite of tools and algorithms tailored for dealing with temporal data. Moreover, it offers numerous tools for anomaly detection within a simple framework. In particular, the TFT model is implemented adopting the homonymous sub-models from PyTorch-Forecasting [77]. The decision to utilise a pre-existing package appears natural given the circumstance that it pertains to an analysis of data that has not been previously examined. It is not prudent to allocate resources towards reconstructing an existing model, especially considering that this is an initial application of the model to this dataset. While a rationale for rebuilding the model from scratch could involve potential future customisation, such an undertaking is unnecessary at this juncture. Furthermore, the implementation provided by Darts is optimised for resource management.

Despite the advantageous feature of the TFT model in incorporating categorical covariates to contextualise various time series data, we are constrained from their utilisation in our study. Our primary objective is to develop a generalised predictive model that does not rely on machine-specific information but rather can accurately forecast signals, irrespective of machine identity. Available covariate options include machine metadata, encompassing parameters such as production type, hygiene standards, and location, among others. Furthermore, the signal `RA`, i.e. the type of recipe configured by each machine during a specific time series, is considered. However, the latter covariate, denoting a unique code assigned to the configuration of settings and controls for packaging a particular product, is rendered impractical due to its non-standardised assignment process across machines. Additionally, the absence of a universal encoding hampers the interpretation of these recipes, which are typically tailored to meet specific customer demands.

Parameters In Darts the TFT is implemented using the `TFTModel` class. In particular, the architecture is defined by specifying a set of parameters when initialising the object.

- `input_chunk_length`: Determines the size of the input window or the number of time steps considered as input to the model at once. When the length of the input series exceeds the specified length, it is divided into multiple inputs, each of which consists of a contiguous segment of the series with a length equal to the input length, ensuring that the entire series is processed in chunks of the specified size.

- `output_chunk_length`: Specifies the output size, i.e. the number of points forecasted into the future. Since the model is intrinsically probabilistic the real output size will be this quantity times the number of quantiles predicted for each point. This is also the length of the future covariate into the future.

- `hidden_size`: Defines the number of units (neurons) in the hidden layers of the neural network, determining the complexity and representational capacity of the model. It is the most relevant hyperparameter that regulates the trade-off between the dimension of the model and the computational cost.

- `lstm_layers`: Specifies the number of LSTM (Long Short-Term Memory) layers in the recurrent part of the network, both for the encoder and the decoder.

- `num_attention_heads`: Indicates the number of interpretable attention heads used in the attention mechanism, allowing the model to focus on different parts of the input sequence during processing.
- `full_attention`: A boolean value indicating whether to use full attention, that is, passing to the attention heads also the present and future time step, in addition to the past values, for computing the attention.
- `dropout`: Specifies the dropout rate, determining the proportion of units randomly dropped out during training to prevent overfitting by adding noise to the network [78].
- `batch_size`: Specifies the mini-batch size, indicating the number of samples are processed together in each training iteration. A smaller mini-batch size require less memory usage and compute the gradient using less data, resulting in faster computation but also slower and more noisy convergence.
- `n_epochs`: Specifies the number of training epochs, indicating how many times the entire dataset is passed forward and backward through the neural network during training.
- `likelihood`: Specifies the likelihood function used for probabilistic forecasts. By default, TFT uses quantile regression, determining which percentiles of the output distribution the model should predict. The set of percentiles that is used can be specified as an argument of the `QuantileRegression` function.
- `optimizer_kwargs`: A dictionary where the optimizer parameters can be specified. The most important of them is the learning rate, which controls how much the model weights are updated in response to the loss gradient.
- `pl_trainer_kwargs`: A dictionary where parameters passed to the PyTorch Lightning Trainer can be specified. Various settings can be configured here, such as enabling GPU usage and introducing an `EarlyStopping` mechanism. The latter is particularly useful for terminating training if the validation loss ceases to decrease, indicating that the model is no longer learning generalisable information.

Dataset The collected dataset, after preprocessing, has the following features:

- Contains two years of data, collected from February 2022 up to February 2024
- Data are fetched from 15 machines, which represent all available machines of the version in question, equipped with both double-line production and cap applicator modules
- Each time series contains 17 components, representing the different signals collected from the cap applicator sensors
- The total duration of the collected time series, i.e., the sum of all available data when machines are in production, lasts for 479 days and 12 hours, with a sampling frequency of one minute
- The whole dataset is divided into 13 594 series, of which 11 010 for the training set, 1 360 for the validation set and 1 224 for the test set
- The average duration of each time series is 50 minutes and 47 seconds.

Machine id	**N. samples**	**Proportion (%)**	**Tot. Duration (h)**	**Avg Duration (min)**
1385	313	2.1	256.2	36.0
1448	1579	10.4	728.1	59.0
1452	359	2.4	675.7	53.3
1456	1169	7.7	474.9	74.1
1465	502	3.3	533.1	28.1
1468	2637	17.3	446.1	44.9
1477	965	6.3	368.4	24.6
1483	358	2.4	339.6	55.9
1484	1788	11.8	389.8	55.0
1488	1016	6.7	249.9	63.6
1491	1817	11.9	390.7	36.6
1498	132	0.9	301.6	61.8
1499	386	2.5	257.2	40.2
1503	1471	9.7	193.2	29.9
1505	723	4.8	203.2	42.0

Table 3.2: Dataset attribute per machine

For training, only the initial 64 minutes of each time series were utilised. This decision is justified by the potential bias introduced by analysing unbalanced time series, which could result in the predominance of longer time series, possibly originating from the same machine. Additionally, the selection of 64 data points aligns with a power of two approximation to the dataset's mean duration of approximately 50 minutes. This is achieved by setting the `max_samples_per_ts` parameter in the `.fit` method of the TFT model.

Metrics Two different metrics have been used for evaluating the performance of the model, namely Mean Square Error and Symmetric Mean Absolute Percentage Error. The mean square error (MSE), denoted mathematically as:

$$\mathrm{MSE} = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2$$

is a fundamental metric in performance evaluation, particularly in regression analysis and predictive modeling. It quantifies the average squared difference between the actual values (y_t) and the predicted values ($\hat{y}_t$) produced by a model across n observations. Given that the model was trained using the Quantile Loss rather than the MSE, it follows that the MSE can be regarded as a metric unaffected by the training process, thus providing an independent evaluation of performance.

The symmetric Mean Absolute Percentage Error (sMAPE) is instead a commonly used metric for evaluating the accuracy of forecasting models. The formula for calculating sMAPE is as follows:

$$\mathrm{sMAPE} = \frac{100\%}{n}\sum_{t=1}^{n}\frac{|y_t - \hat{y}_t|}{\frac{1}{2}\left(|y_t| + |\hat{y}_t|\right)}$$

where y_t represents the actual value, $\hat{y}_t$ denotes the predicted value, and n indicates the total number of observations. Unlike the Mean Square Error (MSE), which measures the average squared difference between actual and predicted values, sMAPE calculates the percentage difference between them. This characteristic makes sMAPE particularly useful in scenarios where the scale of the data varies significantly or when the emphasis is on relative errors rather than absolute differences. Additionally, sMAPE is symmetric in nature, meaning that it treats overestimation and underestimation of values equally, providing a balanced measure of forecasting accuracy.

Chapter 4

Result

4.1 Training

The deep learning model is trained on a MSI Laptop equipped with an NVIDIA GeForce RTX 3050 GPU (4 GB GDDR6), an Intel Core i7-1280P CPU (14 cores, 3.60 GHz), and 16 GB of LPDDR4x-SDRAM. The system run on Windows Linux Subsystem and utilises TensorFlow 2.3.0 for model training. Additionally, the training process relies on CUDA Toolkit 12.1 GPU acceleration.

The hyperparameters of the model, visible in Tab.4.1 have been carefully selected to strike a balance between model effectiveness and computational efficiency. Given the computational demands of the model and our limited computational resources, we aimed to design a model that is both reasonable and effective while minimising the number of parameters requiring tuning during training. This approach not only ensures that the model is tractable within our resource constraints but also optimises the training process by reducing the burden of hyperparameter tuning. Moreover, an `EarlyStopping` callback, which halts the training process when the validation loss fails to improve for a specified number of epochs, is implemented to prevent overfitting and optimise model generalisation. With this selection of hyperparameters, the architecture includes about 15 000 trainable parameters.

The model was trained for a total of 9 epochs before being stopped early due to achieving validation stationarity. This process, which encompassed training and evaluation, lasted approximately 3 hours to complete. Detailed plots depicting the evolution of the quantile loss for both the training and validation sets can be found in Fig.4.1.

Hyperparameter	Value
Epochs	100
Input Size	8
Output Size	4
Hidden Layers	8
Recurrent Layers	1
Attention Heads	4
Batch Size	32
Learning Rate	1×10^{-4}
Dropout Rate	0.3

Table 4.1: Hyper-parameter Information for TFT Training

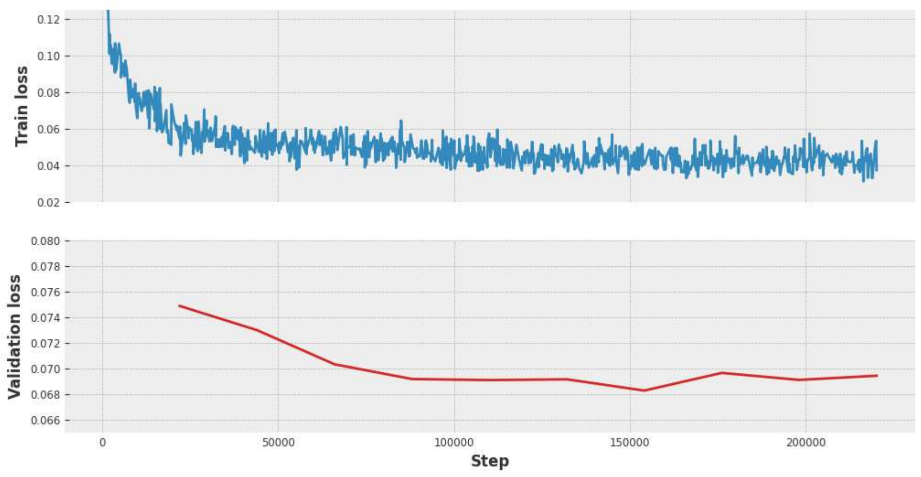


Figure 4.1: Evolution of the Quantile Loss during the training of the TFT

4.2 Performances evaluation

To evaluate the performance of the model, we utilised time-series data and employed the trained model to forecast future points. Specifically, all series were forecasted from their 8th point onwards, corresponding to the input size of the model. Subsequently, we evaluated the model's performance using two metrics: Mean Squared Error (MSE) and symmetric Mean Absolute Percentage Error (sMAPE). These metrics were computed pointwise for each time series, treating each timestep as a point in a multidimensional space, where different variables represent different dimensions. This approach was facilitated by the prior rescaling of our time-series data, ensuring that no individual feature unduly influenced the distance between true and predicted values. The outcomes of the two metrics are visible in Tab.4.2. It can be observed that the values of the test set are comparable to those of the training set for both metrics; hence, it can be claimed that a good degree of generalisation of the model has been reached. However, it is worth noting that the sMAPE exceeds 50% for both the training and test sets. This implies that, on average, the difference between the true value and the forecasted one is approximately half of the average absolute value of the two.

	Training Set	Test Set
MSE	0.018	0.020
sMAPE	52.3%	51.6%

Table 4.2: Performance Metrics for the TFT

4.2.1 Comparison with Other Models

To assess the efficacy of implementing the TFT, we evaluated its performance against a simpler model. Specifically, we considered a recurrent neural network with long short-term memory units. The chosen architecture features an input size window of 4 time steps, a hidden dimension comprising 32 neurons, and a single hidden layer. Additionally, we set the learning rate to 1×10^{-3} and the batch size to 100. Training was limited to a maximum of epochs, with early stopping based on validation set performance.

The model consists of approximately 7 700 trainable parameters. Training continued for 12 epochs until it reached the minimum validation loss, which occurred after approximately 2:30 hours. The results of the latter model are visible in Tab.4.3.

Upon evaluating the performance of the two models, it becomes apparent that while TFT demonstrates similar behaviour, its performance falls slightly short when compared to the LSTM. Despite the promising outcomes in the original study, our findings suggest a marginal deviation from the performance achieved by the alternative model. Although our model exhibits promising capabilities and holds potential for further refinement, these findings highlight how optimisation of the TFT model is necessary to achieve optimal performance levels.

	Training Set	Test Set
MSE	0.008	0.014
sMAPE	44.32%	43.98%

Table 4.3: Performance Metrics for the LSTM

4.3 Anomaly Detection Model

Anomaly evaluation is conducted on the test set, where anomaly scores are assigned based on the 2-norm between the true and predicted values. This approach allows for the identification of both point anomalies and anomalous time series. Point anomalies are identified as points with a high anomaly score, whereas for detecting anomalous time series, the anomaly score is averaged over each time series, resulting in fact in the root MSE. To assign a label to anomalous points or time series, a threshold must be set. To determine a reasonable threshold value, histograms are computed for both point anomalies and time series anomalies, as depicted in Fig.4.2 and Fig.4.3, respectively. The threshold is then determined by calculating the anomaly score value for the 99$^{\text{th}}$ percentile, which separates the top 1% of points with the highest scores. It can be observed that the distribution of the anomaly score per data point exhibits a densely populated region of points with low scores, alongside a long tail of points with significantly higher values. This results in over 6% of the points having an anomaly score greater than 1. Conversely, the distribution related to the time series scores shows a bell-shaped distribution, with only a few outliers having higher values. In this case, the threshold value falls approximately at the end of the more densely populated region, indicating that only a small percentage of the time series are fully

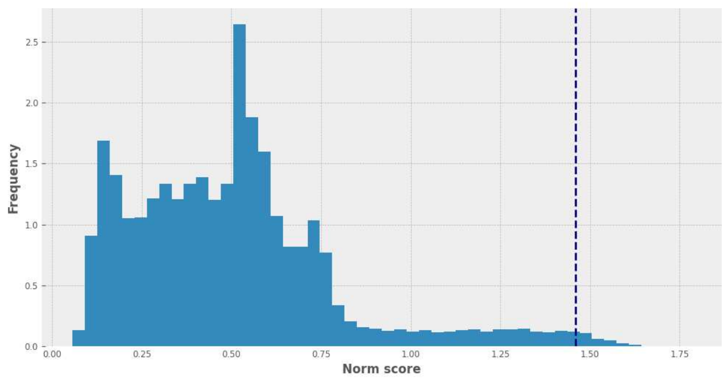


Figure 4.2: Distribution of the anomaly score per data point. Threshold value: 0.81

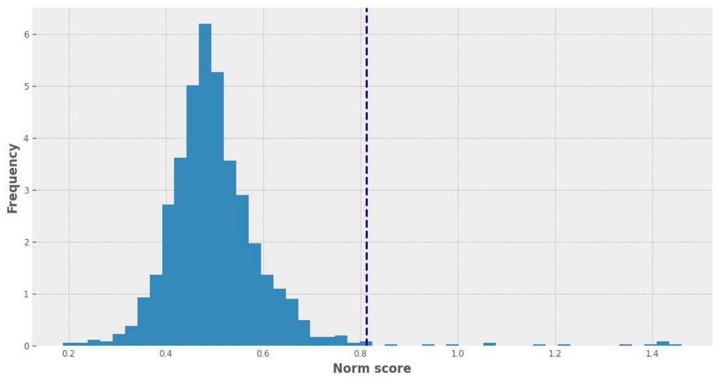


Figure 4.3: Distribution of the average anomaly score per time series. Threshold value: 1.46

anomalous.

Below, we analyse relevant examples of time series compared with the model's prediction. We begin by considering a time series marked as completely anomalous. In Fig.4.4, the signals of Current Welding Energy for two different production lines are depicted. We can observe how the model expects both lines to be active, while one of the lines is currently inactive, resulting in anomalous behaviour. However, upon examining time series in the region with lower anomaly scores, severe discrepancies between the prediction and actual data are also evident. Particularly, the set of time series with low anomaly scores is typically populated by short-duration series where, at times, as seen in Fig.4.5, the machine either does not output signals or produces constant signals, while the model predicts a variable signal over time. Finally, we also analyse the region of the time series with intermediate anomaly scores. This region is indeed populated by standard-length time series that exhibit a better degree of approximation, although they are not always able to predict simple and repetitive patterns, as shown in Fig.4.6.

4.4 Model Interpretability

Despite the Temporal Fusion Transformer's performance being found to be slightly inferior to that of other models, this architecture remains valuable due to its emphasis on promoting the interpretability of results that allows for insights into the relevance of different variables in the dataset. In particular, the TFT employs two different mechanisms to aid in interpreting the results. The first one utilises the interpretable multi-head attention mechanism. Through this architecture, it

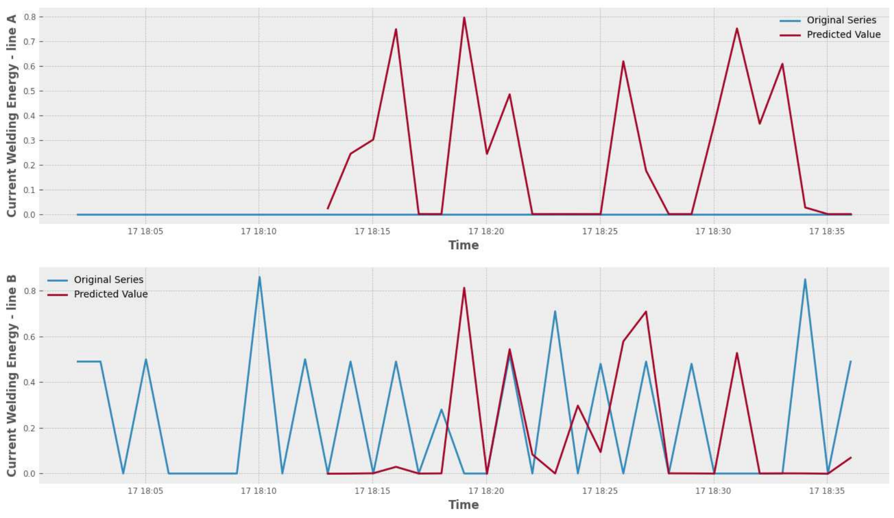


Figure 4.4: True and forecasted time series of an **outlier** (Anomaly score: 1.46). Here are represented the current welding energies of the two lines of production over time

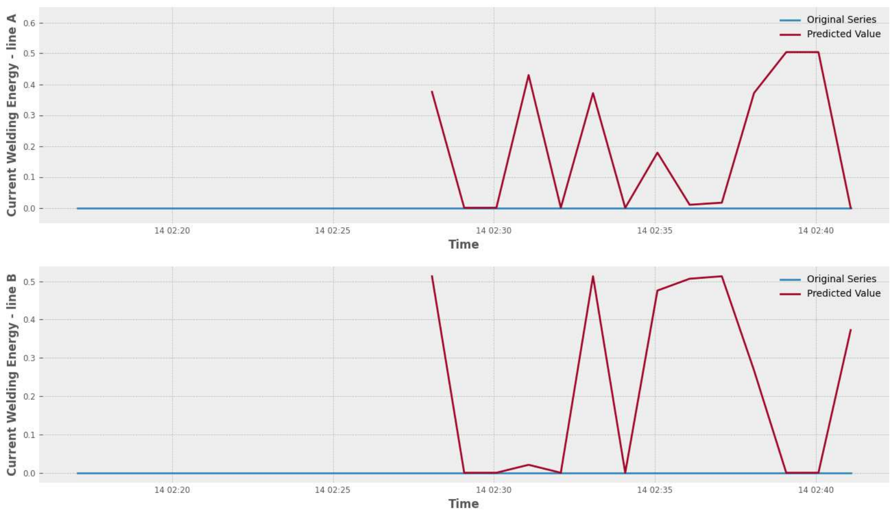


Figure 4.5: True and forecasted time series of a sample with a **low** anomaly score (0.29)

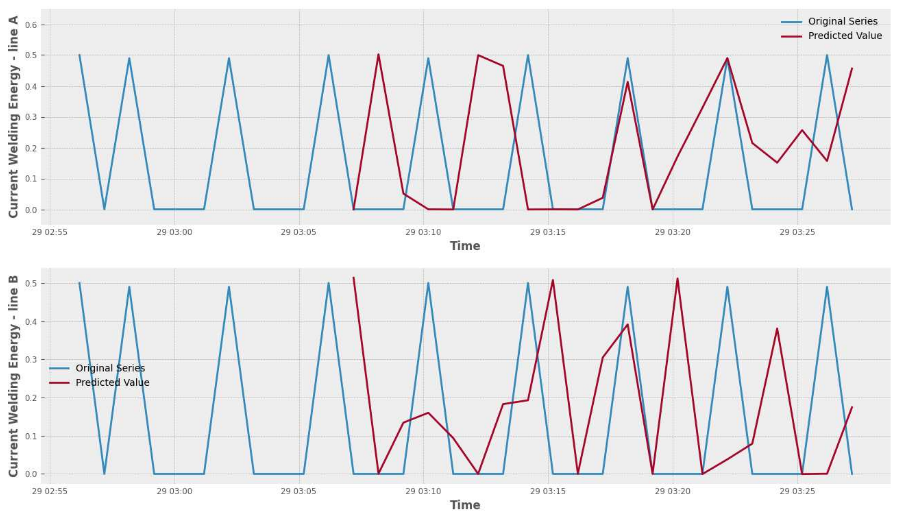


Figure 4.6: True and forecasted time series of a sample with **average** anomaly score (0.41)

becomes possible to extract information regarding the behaviour of the different attention heads by observing their attention levels over time. Unfortunately, during our training, the attention heads all converged on the same attention profile. Consequently, in Fig.4.7, the mean attention value is depicted, representing the value of each head (as they coincide). It is notable that the model assigns higher attention to points closer in time, progressively decreasing for further points.

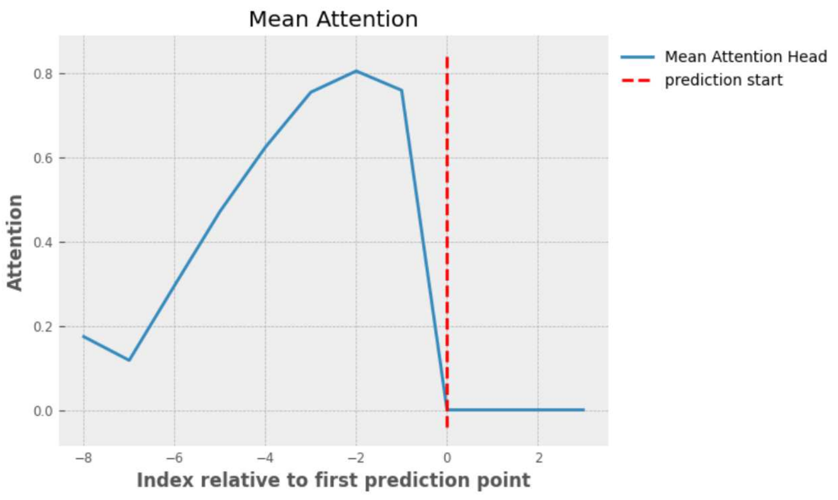


Figure 4.7: Mean attention value of the TFT for predicting a time series

The second mechanism that enhances the interpretability of the network involves leveraging the weights of the variable selection network. This enables the observation of which variables have been given more consideration in the data forecasting process, encompassing both the time series variables and the covariates. The results of this analysis are visible in Fig.4.8. We can notice that there are no abrupt changes in the variable importance, indicating that the model considers all variables in the prediction. It is interesting that the *welding energy set points* (SPWE) of the two lines are found at the extremes of the graph. This could signify that the two signals usually behave in a similar manner. However, the fact that only a variable per line is mainly considered is not a trend for all the signals. For example, the *applied energy at last sealed cap* (PVWE) signals are both high in the rankings, as is the *applied power at last sealed cap* (PVWP). It should be noted, however, that these results may be influenced by the unsatisfactory forecasting performance. It is uncertain whether the variables are weighted based on their actual ability to provide information about the data or if they are assessed using almost random values that do not contribute to a deeper understanding. This last statement is reinforced by the significance of future covariates. It can be observed that the temporal covariate representing the year is given the highest importance, despite temporal series typically lasting only a few hours. Conversely, the hour covariate, which could theoretically be more useful, is assigned a lower degree of importance.

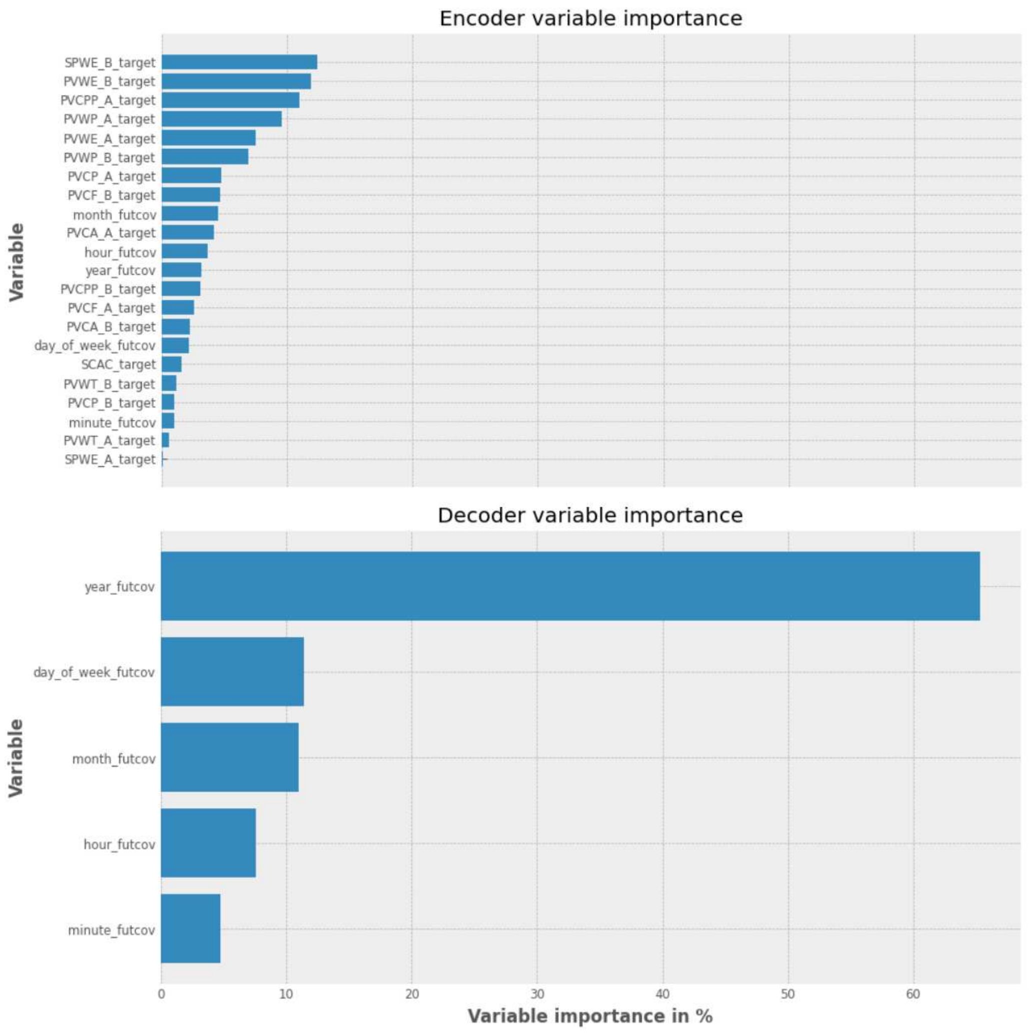


Figure 4.8: Variable-selection feature importance of the TFTModel. In the encoder are present all the input variables, while in the decoder are represented the future covariates

Chapter 5

Conclusions

The objective of this study was to develop a robust anomaly detection method applicable to real-world scenarios involving dairy packaging machinery. Specifically, our focus was on detecting anomalies within the cap-applicator module, which stands as a delicate component of the machinery undergoing examination. The task has been achieved by training an advanced deep learning model designed for multivariate time series forecasting, namely the Temporal Fusion Transformer.

It's worth noting that this analysis stands out as a pioneering endeavour in the examination of the machine-generated dataset, thus holding significant value for the company, as it marks one of the initial instances of data analysis for this specific dataset. Consequently, the retrieval and preprocessing of the data was an important step of this study, serving as a foundation for potential future investigations across various fields that could benefit from utilising the same dataset. In particular, through the analysis of the sensor signals from the cap applicator module, it was revealed that these time series typically lack correlation. This observation implies that each signal offers distinct and independent insights, enhancing our comprehension of the observed system from diverse perspectives. Furthermore, a dashboard has been created for data visualisation objectives, empowering users to compare various datasets and their distributions. Moreover, this dashboard allows the visualisation of the occurrence of alarm signals concerning sensor data, facilitating the straightforward identification of when these alarms arise in relation to the dataset.

In conclusion, the TFT model has been successfully implemented within our framework, and the training conducted using our dataset of interest has resulted in reaching a minimum for the validation loss. Leveraging this properly trained TFT model, it was possible to identify various anomalies within the dataset, including both point anomalies and anomalous time series instances. Despite these achievements, it is noteworthy that the TFT model currently trails behind other established models of time-series forecasting, such as LSTM, in terms of predictive performance. This discrepancy may stem from several factors, including hyperparameter tuning, implementation choices, and the complexity of the dataset. Nonetheless, the TFT model demonstrates competence in identifying certain plausibly anomalous time series instances, suggesting its potential utility in our real-world application. Finally, we have capitalised on the interpretability of the TFT by analysing the interpretable multi-head attention mechanism and the variable selection weights. This analysis has yielded valuable insights into the model's inner workings and decision-making process, as well as the underlying data patterns, and the relative importance of different sensors in predicting future data points.

5.1 Future Works and Final Suggestions

Building upon the insights gained from the present study, several promising directions emerge for future research. Firstly, there is potential for fine-tuning the model's hyperparameters to optimise its efficiency. This involves using packages for hyperparameter optimisations, such as `Optuna`, to achieve the best possible performance. Moreover, exploring the effects of increased computational power and larger model sizes on performance could yield valuable insights. By leveraging more computational resources and scaling up the model, we may uncover improvements in predictive accuracy and anomaly detection capabilities. Additionally, refining the anomaly detection system beyond forecasting is crucial. Various methods can be explored in order to improve the accuracy of the anomaly identification model, including evaluating different metrics for the anomaly score or implementing techniques that utilise metric ensembles. Furthermore, exploring additional practices for determining a good threshold to distinguish anomalous data from normal behaviour is warranted. Another avenue worth exploring is bidirectional time series forecasting. Traditional forecasting methods typically operate in a unidirectional manner, predicting future values based on past observations. However, bidirectional forecasting considers both past and future data points, allowing for a more comprehensive understanding of temporal patterns. This approach would enable us to detect anomalies in the early stages of time series, providing a deeper understanding of machine operations and rare events during initial production phases.

A natural extension of the model entails its expansion to encompass all signals emanating from the machinery, rather than solely focusing on the cap applicator module. However, such a generalisation cannot be achieved through a straightforward expansion of the existing model. This limitation arises due to the heterogeneous nature of machinery, where certain units are equipped with the cap applicator, thereby generating signals specific to this module, while others lack this component, resulting in a notable absence of corresponding signals. As evidenced previously, the model encounters challenges in effectively learning from multivariate time series characterised by incomplete feature sets. Consequently, a pragmatic approach involves the implementation of an ensemble model comprising two distinct components: one tailored to the studied module and another trained on the remaining signals emanating from diverse machinery configurations.

Incorporating alarm signals as temporal covariates could further improve anomaly detection capabilities, providing valuable contextual information on critical events or abnormal conditions in industrial processes. By integrating alarm signals as additional features in the time series data, we can enhance the model's ability to identify and respond to anomalies in real-time. However, effective encoding of alarm signals within one or more time series has to be carefully studied in order to ensure compatibility with existing anomaly detection frameworks.

Finally, investing resources in constructing an expert-certified anomaly dataset could significantly advance anomaly detection methodologies. This dataset would consist of labelled examples of known anomalies, verified by domain experts based on their impact on production, safety, or quality. Moreover, this approach could facilitate the development of semi-supervised frameworks, leveraging both labelled and unlabelled data to enhance anomaly detection accuracy and adaptability.

All these proposed advancements hold the potential to greatly benefit the company. Through a comprehensive analysis of the anomalies identified by this model, the door is opened to optimising machinery performance. By leveraging data-driven solutions tailored to address specific issues, productivity can be enhanced, operational inefficiencies mitigated, and overall equipment effectiveness maximised, helping the continuous improvement and innovation of the company's processes.

References

[1] S. Nousias, E.-V. Pikoulis, C. Mavrokefalidis, and A. S. Lalos, "Accelerating deep neural networks for efficient scene understanding in multi-modal automotive applications," *IEEE Access*, vol. 11, pp. 28208–28221, 2023.

[2] S. Kaushik, A. Choudhury, P. K. Sheron, N. Dasgupta, S. Natarajan, L. A. Pickett, and V. Dutt, "Ai in healthcare: time-series forecasting using statistical, neural, and ensemble architectures," *Frontiers in big data*, vol. 3, p. 4, 2020.

[3] E. Ghaderpour, H. Dadkhah, H. Dabiri, F. Bozzano, G. Scarascia Mugnozza, and P. Mazzanti, "Precipitation time series analysis and forecasting for italian regions," *Engineering Proceedings*, vol. 39, no. 1, p. 23, 2023.

[4] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine learning*, vol. 23, pp. 69–101, 1996.

[5] D. M. Hawkins, *Identification of outliers*, vol. 11. Springer, 1980.

[6] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

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

[8] J. Liu, G. Xie, J. Wang, S. Li, C. Wang, F. Zheng, and Y. Jin, "Deep industrial image anomaly detection: A survey," *Machine Intelligence Research*, vol. 21, no. 1, pp. 104–135, 2024.

[9] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed networks," in *Proceedings of the 2019 SIAM International Conference on Data Mining*, pp. 594–602, SIAM, 2019.

[10] S. Vaidya, P. Ambad, and S. Bhosle, "Industry 4.0–a glimpse," *Procedia manufacturing*, vol. 20, pp. 233–238, 2018.

[11] R.-J. Hsieh, J. Chou, and C.-H. Ho, "Unsupervised online anomaly detection on multivariate sensing time series data for smart manufacturing," in *2019 IEEE 12th conference on service-oriented computing and applications (SOCA)*, pp. 90–97, IEEE, 2019.

[12] Y. Weng and L. Liu, "A collective anomaly detection approach for multidimensional streams in mobile service security," *IEEE Access*, vol. 7, pp. 49157–49168, 2019.

[13] Y. Zhang, P. Peng, C. Liu, and H. Zhang, "Anomaly detection for industry product quality inspection based on gaussian restricted boltzmann machine," in *2019 IEEE international conference on systems, man and cybernetics (SMC)*, pp. 1–6, IEEE, 2019.

[14] G. Li and J. J. Jung, "Deep learning for anomaly detection in multivariate time series: Approaches, applications, and challenges," *Information Fusion*, vol. 91, pp. 93–102, 2023.

[15] K. Choi, J. Yi, C. Park, and S. Yoon, "Deep learning for anomaly detection in time-series data: Review, analysis, and guidelines," *IEEE access*, vol. 9, pp. 120043–120065, 2021.

[16] Y. Bao, Z. Tang, H. Li, and Y. Zhang, "Computer vision and deep learning–based data anomaly detection method for structural health monitoring," *Structural Health Monitoring*, vol. 18, no. 2, pp. 401–421, 2019.

[17] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao, "Adbench: Anomaly detection benchmark," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32142–32159, 2022.

[18] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[19] Z.-H. Zhou, "A brief introduction to weakly supervised learning," *National science review*, vol. 5, no. 1, pp. 44–53, 2018.

[20] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia,*

December 2–6, 2018, Revised Selected Papers, Part III 14, pp. 622–637, Springer, 2019.

[21] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, pp. 226–231, 1996.

[22] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[23] S. Chandrakala and C. C. Sekhar, "A density based method for multivariate time series clustering in kernel feature space," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 1885–1890, IEEE, 2008.

[24] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*, pp. 413–422, IEEE, 2008.

[25] Y. Qin and Y. Lou, "Hydrological time series anomaly pattern detection based on isolation forest," in *2019 IEEE 3rd information technology, networking, electronic and automation control conference (ITNEC)*, pp. 1706–1710, IEEE, 2019.

[26] G. E. Box and D. A. Pierce, "Distribution of residual autocorrelations in autoregressive-integrated moving average time series models," *Journal of the American statistical Association*, vol. 65, no. 332, pp. 1509–1526, 1970.

[27] C. A. Sims, "Macroeconomics and reality," *Econometrica: journal of the Econometric Society*, pp. 1–48, 1980.

[28] M. Kropf, D. Hayn, D. Morris, A.-K. Radhakrishnan, E. Belyavskiy, A. Frydas, E. Pieske-Kraigher, B. Pieske, and G. Schreier, "Cardiac anomaly detection based on time and frequency domain features using tree-based classifiers," *Physiological measurement*, vol. 39, no. 11, p. 114001, 2018.

[29] J. Wang, S. Shao, Y. Bai, J. Deng, and Y. Lin, "Multiscale wavelet graph autoencoder for multivariate time-series anomaly detection," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1–11, 2023.

[30] X. Hou and L. Zhang, "Saliency detection: A spectral residual approach," in *2007 IEEE Conference on computer vision and pattern recognition*, pp. 1–8, Ieee, 2007.

[31] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at microsoft," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3009–3017, 2019.

[32] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Muller, "A unifying review of deep and shallow anomaly detection," *Proceedings of the IEEE*, vol. 109, pp. 756–795, 2020.

[33] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM computing surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.

[34] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[37] D. Xu, E. Ricci, Y. Yan, J. Song, and N. Sebe, "Learning deep representations of appearance and motion for anomalous event detection," *arXiv preprint arXiv:1510.01553*, 2015.

[38] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," *Advances in neural information processing systems*, vol. 12, 1999.

[39] R. T. Ionescu, F. S. Khan, M.-I. Georgescu, and L. Shao, "Object-centric auto-encoders and dummy anomalies for abnormal event detection in video,"

in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7842–7851, 2019.

[40] A. Makhzani and B. Frey, “K-sparse autoencoders,” *arXiv preprint arXiv:1312.5663*, 2013.

[41] C. Doersch, “Tutorial on variational autoencoders,” *arXiv preprint arXiv:1606.05908*, 2016.

[42] N. Gugulothu, P. Malhotra, L. Vig, G. Shroff, *et al.*, “Sparse neural networks for anomaly detection in high-dimensional time series,” in *AI4IOT workshop in conjunction with ICML, IJCAI and ECAI*, pp. 1551–3203, 2018.

[43] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, 2014.

[44] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery,” in *International conference on information processing in medical imaging*, pp. 146–157, Springer, 2017.

[45] Y. Choi, H. Lim, H. Choi, and I.-J. Kim, “Gan-based anomaly detection and localization of multivariate time series data for power plant,” in *2020 IEEE international conference on big data and smart computing (BigComp)*, pp. 71–74, IEEE, 2020.

[46] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, “Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks,” in *International conference on artificial neural networks*, pp. 703–716, Springer, 2019.

[47] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, “Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding,” in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 387–395, 2018.

[48] N. Ding, H. Ma, H. Gao, Y. Ma, and G. Tan, “Real-time anomaly detection based on long short-term memory and gaussian mixture model,” *Computers & Electrical Engineering*, vol. 79, p. 106458, 2019.

[49] I. Golan and R. El-Yaniv, “Deep anomaly detection using geometric transformations,” *Advances in neural information processing systems*, vol. 31, 2018.

[50] G. Pang, L. Cao, L. Chen, and H. Liu, “Learning representations of ultrahigh-dimensional data for random distance-based outlier detection,” in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2041–2050, 2018.

[51] M. M. Moya, M. W. Koch, and L. D. Hostetler, “One-class classifier networks for target recognition applications,” *NASA Sti/Recon technical report N*, vol. 93, p. 24043, 1993.

[52] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[53] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, “Deep clustering for unsupervised learning of visual features,” in *Proceedings of the European conference on computer vision (ECCV)*, pp. 132–149, 2018.

[54] G. Pang, C. Yan, C. Shen, A. v. d. Hengel, and X. Bai, “Self-trained deep ordinal regression for end-to-end video anomaly detection,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12173–12182, 2020.

[55] G. Pang, C. Shen, and A. Van Den Hengel, “Deep anomaly detection with deviation networks,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 353–362, 2019.

[56] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986,” *Biometrika*, vol. 71, pp. 599–607, 1986.

[57] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, “Recurrent neural network based language model,” pp. 1045–1048, 2010.

[58] L. Shen, Z. Li, and J. Kwok, “Timeseries anomaly detection using temporal hierarchical one-class network,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 13016–13026, 2020.

[59] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[60] J. Zhang, P. Wang, R. Yan, and R. X. Gao, “Long short-term memory for machine remaining life prediction,” *Journal of manufacturing systems*, vol. 48, pp. 78–86, 2018.

[61] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.

[62] A. Riaz, M. Nabeel, M. Khan, and H. Jamil, “Sbag: a hybrid deep learning model for large scale traffic speed prediction,” *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 1, 2020.

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[64] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, “Temporal fusion transformers for interpretable multi-horizon time series forecasting,” *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021.

[65] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *arXiv preprint arXiv:1511.07289*, 2015.

[66] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, “Language modeling with gated convolutional networks,” in *International conference on machine learning*, pp. 933–941, PMLR, 2017.

[67] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.

[68] Y. Gal and Z. Ghahramani, “A theoretically grounded application of dropout in recurrent neural networks,” *Advances in neural information processing systems*, vol. 29, 2016.

[69] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka, “A multi-horizon quantile recurrent forecaster,” Nov. 2017.

[70] Bokeh Development Team, *Bokeh: Python library for interactive visualization*, 2018.

[71] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[72] R. Agrawal, R. Srikant, *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, pp. 487–499, Santiago, 1994.

[73] D. Dalle Pezze, C. Masiero, D. Tosato, A. Beghi, and G. A. Susto, "Formula: A deep learning approach for rare alarms predictions in industrial equipment," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 1491–1502, 2021.

[74] D. Tosato, D. Dalle Pezze, C. Masiero, G. A. Susto, and A. Beghi, "Alarm logs of industrial packaging machines," 2022.

[75] J. Herzen, F. Lässig, S. G. Piazzetta, T. Neuer, L. Tafti, G. Raille, T. Van Pottelbergh, M. Pasieka, A. Skrodzki, N. Huguenin, *et al.*, "Darts: User-friendly modern machine learning for time series," *Journal of Machine Learning Research*, vol. 23, no. 124, pp. 1–6, 2022.

[76] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[77] J. Beitner, "Introducing PyTorch Forecasting state-of-the-art forecasting with neural networks made simple," 2020.

[78] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.