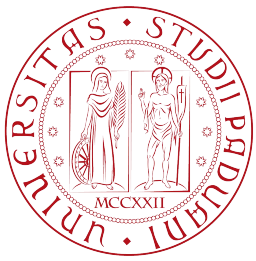


Università degli Studi di Padova

Physics and Astronomy Department "Galileo Galilei"

Master Degree in Physics of Data

Bayesian Autoencoder for Trigger-less Anomaly Detection

Supervisor:
Prof. Marco Zanetti

Candidate:
Giacomo Di Prima

Accademic Year 2023/2024

Contents

Abstract

In the realm of time series analysis, accurate anomaly detection is crucial for a variety of different applications, ranging from industrial process monitoring to financial fraud detection. Traditional methods often struggle with the complexity and high dimensionality inherent in time series data. This work explores the integration of an Autoencoder with Long Short Term Memory (LSTM) layers and a Convolutional Neural Network (CNN) used as a predictor within the realm of Bayesian inference, to predict anomalies in time series data, aiming to enhance detection accuracy and robustness by identifying confidence intervals that allows to classify normal behavior in the data over anomalies.

The proposed model leverages the strengths of both LSTMs and CNNs to capture temporal dependencies and extract complex features from time series data, respectively. The Autoencoder architecture, designed to learn efficient representations of the input data, consists of an encoder that compresses the input into a lower-dimensional space and a decoder that reconstructs the input from this compressed representation. By incorporating LSTM layers, the model effectively retains long-term dependencies within the sequential data, while the CNN layers facilitate the extraction of localized patterns.

This thesis contributes to the field of time series analysis by presenting a novel hybrid model that combines the temporal learning capabilities of LSTMs with the spatial feature extraction prowess of CNNs, encapsulated within an Autoencoder architecture. The findings highlight the model's effectiveness in identifying anomalies, paving the way for future research and applications in various domains requiring reliable anomaly detection.

An extensive experimental evaluation is conducted on benchmark datasets, where the model's performance is compared against state-of-the-art anomaly detection techniques. The results demonstrate that the Autoencoder with LSTM and CNN layers improves the precision and recall of anomaly detection. Furthermore, the model exhibits robustness in handling noise and variability in time series data, showcasing its potential for real-world applications.

Chapter 1

Time Series Anomaly Detection

1.1 Introduction

Recent advances in technology allow us to collect a large amount of data over time in diverse research areas [1]. Observations that have been recorded in an orderly fashion and that are correlated in time constitute a time series. Time series data mining aims to extract all the meaningful knowledge from these data, and several mining tasks (e.g., classification, clustering, forecasting, and outlier detection) have been considered in the literature [2, 3, 4]. Outlier detection has become more and more relevant for many researchers and practitioners and is now one of the main tasks of time series data mining. The field has been studied in a variety of application domains such as credit card fraud detection, intrusion detection in cyber-security, or fault diagnosis in industrial machinery. In all examples cited above, outliers can be thought as observations that do not follow the expected behaviour. In time series, the word can have two different meanings, and the semantic distinction between them is mainly based on the interest of the analyst or the particular scenario considered. These observations have been related to noise, erroneous, or unwanted data, which by themselves are not interesting to the analyst [5]. In these cases, outliers should be deleted or corrected to improve data quality and generate a cleaner dataset that can be used by other data mining algorithms. For example, sensor transmission errors are eliminated to obtain more accurate predictions, because the main objective is to make predictions. Nevertheless, in recent years and especially in the area of time series data, many researchers have aimed to detect and analyse unusual but interesting phenomena. Fraud detection is an example of this because the main objective is to detect and analyse the outlier itself. These observations are often referred to as anomalies [5], hence the name anomaly detection (AD) to describe the family of practices and algorithms whose goal is to find and study these outliers among the general data structure.

1.2 Anomaly Detection Taxonomy

Outlier detection techniques in time series data vary depending on the input data type, the outlier behaviour and the nature of the method. Therefore, a comprehensive taxonomy is proposed that encompasses these three aspects [1]. Figure 1.1 shows an overview of the resulting structure that will be described in detail below.

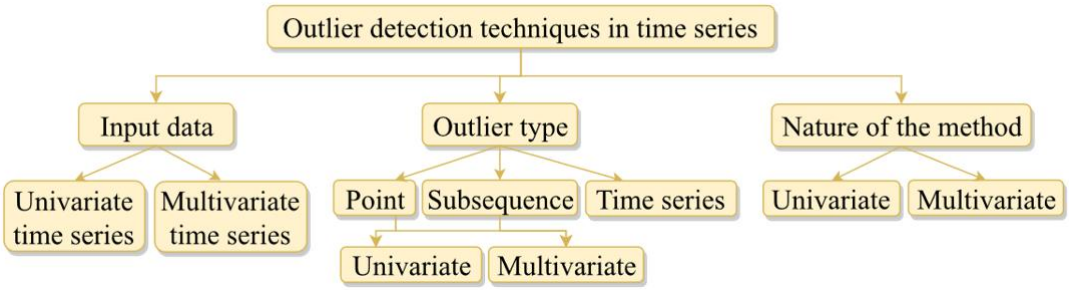


Figure 1.1: Anomaly Detection Taxonomy

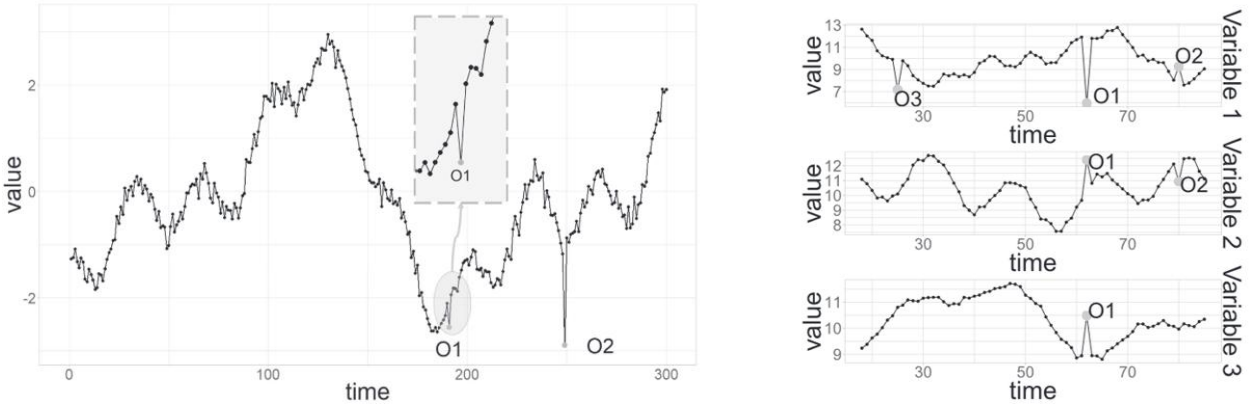
1.2.1 Input Data

The first axis represents the type of input data that the detection method is able to deal with (that is, univariate or multivariate time series).

- **Univariate time series** - A univariate time series $X = \{x_t\}_{t \in T}$ is an ordered set of real-valued observations, where each observation is recorded at a specific time $t \in T \subseteq \mathbb{Z}^+$.
  Then, x_t is the point or observation recorded at time t and $S = x_p, x_{p+1}, ..., X_{p+n-1}$ the **subsequence** of length $n \leq T$ starting at position $p, t \in T$ and $p \leq T - n + 1$. It is assumed that each observation x_t is a realised value of a certain random variable X_t. In figure 1.2a and 1.3a are reported examples of univariate time series.

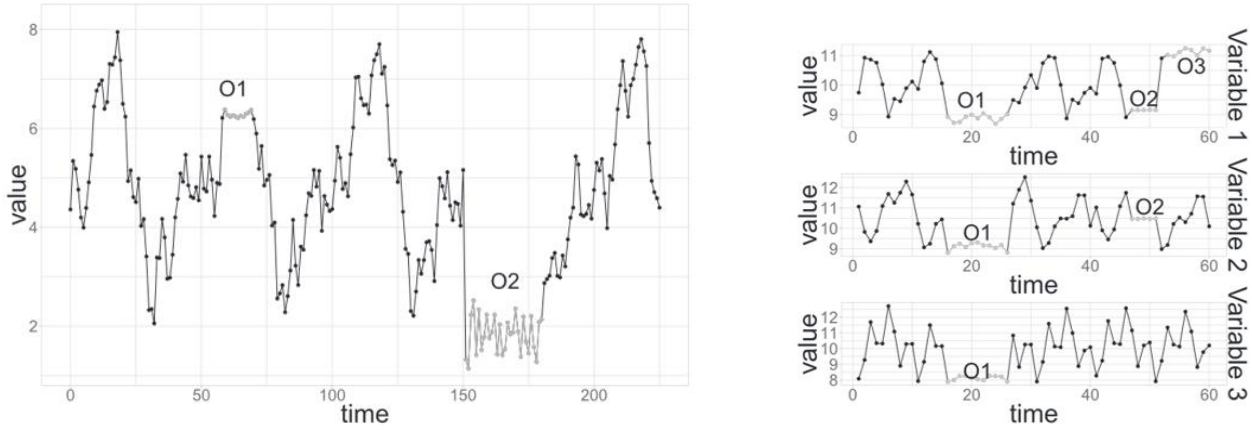
- **Multivariate time series** - A multivariate time series $X = \{x_t\}_{t \in T}$ is defined as an ordered set of k-dimensional vectors, each of which is recorded at a specific time $t \in T \subseteq \mathbb{Z}^+$ and consists of k real-valued observations, $x_t = x_{1t}, x_{2t}, ..., x_{kt}$. Note that this definition could include irregularly sampled time series or dimensions (variables) with different temporal granularities, assuming that some of the x_{it} values might be missing.
  Then, x_t is said to be a point and $S = x_p, x_{p+1}, ..., x_{p+n-1}$ a subsequence of length $n \leq T$ of the multivariate time series X, for $pt \in T$ and $p \leq T - n + 1$. For each dimension $j \in \{1, ..., k\}$, $X_j = \{x_j t\}_t \in T$ is a univariate time series, and each observation x_{jt} in the vector x_t is a realised value of a random time-dependent variable X_{jt} in $X_t = (X_{1t}, ..., X_{kt})$. In this case, each variable could depend not only on its past

values, but also on the other variables dependent on time. In figure 1.2b and 1.3b are reported examples of multivariate time series.



(a) (b)

Figure 1.2: Time series examples - (**a**) Univariate time series with two point anomalies: $O1$ and $O2$. (**b**) Multivariate time series with different point anomalies: $O1$, $O2$ and $O3$.



(a) (b)

Figure 1.3: Time series examples - (**a**) Univariate time series with two subsequence anomalies: $O1$ and $O2$. (**b**) Multivariate time series with different subsequence anomalies: $O1$, $O2$ and $O3$.

1.2.2 Anomaly Type

The second axis describes the type of outlier that the method aims to detect (that is, a point, a subsequence or a time series).

- **Point outliers** - A point outlier is a datum that behaves unusually in a specific time instant when compared to the other values in the time series (global outlier) or to its neighbouring points (local outlier). Point outliers can be univariate or multivariate depending on whether they affect one or more time-dependent variables, respectively. Figures 1.2a and 1.2b represent a univariate and a multivariate time series, respectively, with point anomalies.

- **Subsequence outliers** - This term refers to consecutive points in time whose joint behaviour is unusual, although each observation individually is not necessarily a point outlier. Subsequence outliers can also be global or local and can affect one (univariate subsequence outlier) or more (multivariate subsequence outlier) time-dependent variables. In figure 1.3a and 1.3b are shown a univariate and a multivariate time series, respectively, with subsequence anomalies.

- **Outlier time series** - Complete time series can also be outliers, but can only be detected when the input data are a multivariate time series. Figure 1.4 depicts an example of an outlier time series.

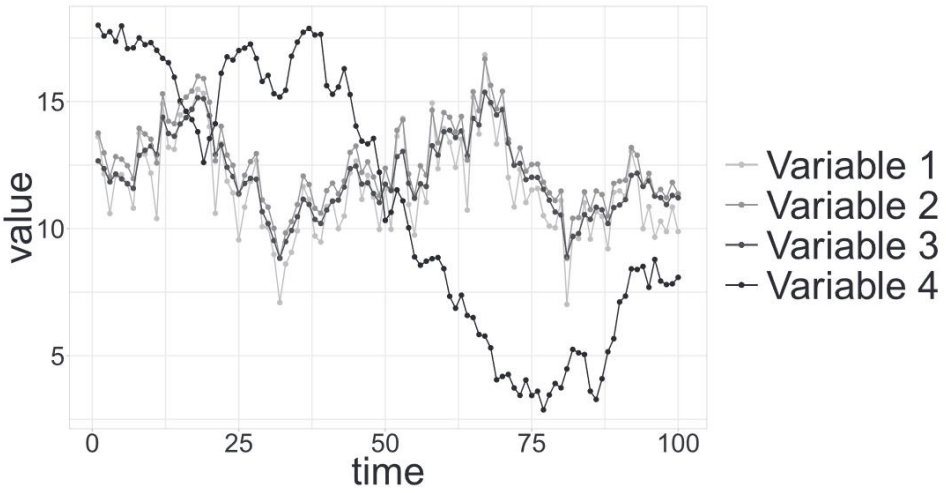


Figure 1.4: Outlier time series example: the behaviour of Variable 4 significantly differs from the ones of the other time series.

Observe that this axis is closely related to the input data type. If the method only allows univariate time series as input, then no multivariate point or subsequence outliers can be identified. In addition, outlier time series can only be found in multivariate time series. Finally, it should be noted that outliers depend on the context. Thus, if the detection method uses the entire time series as contextual information, then the detected outliers are global. Otherwise, if the method only uses a segment of the series (a time window), then the detected outliers are local, because they are outliers within their neighbourhood.

1.2.3 Nature of the Method

The third axis analyses the nature of the detection method used (that is, if the detection method is univariate or multivariate). A univariate detection method only considers a single time-dependent variable, whereas a multivariate detection method is able to simultaneously work with more than one time-dependent variable. Note that the detection method can be univariate, even if the input data are a multivariate time series, because an individual analysis

can be performed on each time-dependent variable without considering the dependencies that may exist between the variables. In contrast, a multivariate technique cannot be used if the input data are a univariate time series. Thus, this axis will only be mentioned for multivariate time series data.

1.3 Point Anomalies

Point outlier detection is the most common anomaly detection task in the field of time series. This section presents techniques for detecting this kind of outlier and, in particular, the focus will be on two key characteristics that these methods may present:

- **Temporality** - whether or not the order of the time series is considered. The main difference between these two approaches is that the latter yields the same results when fed ordered and shuffled data. Within the methods that take into account temporality, a subgroup of these makes use of time windows.

- **Streaming data** - whether or not the methoEvend is able to detect if a new incoming datum is an outlier as soon as it arrives without the need of waiting for new data. Within this group, some methods use a fixed model throughout the stream evolution, whereas others update the models used for detection with the new information received: either by retraining the whole model or by learning in an incremental manner.

In the following sub-section will be discussed some techniques developed to deal with univariate and multivariate time series anomaly detection.

1.3.1 Univariate Time Series

The most popular and intuitive definition for the concept of point anomaly is a point that significantly deviates from its expected value. Therefore, given a univariate time series, a point at time t can be declared an outlier if the distance to its expected value is greater than a predefined threshold τ :

$$|x_t - \hat{x}_t| > \tau, \tag{1.1}$$

where x_t is the observed data point, and $\hat{x}_t$ is its expected value. The outlier detection methods based on the strategy described in equation 1.1 are denominated ***model-based*** [1] and are the most common approaches in the literature. Although each technique computes the expected value $\hat{x}_t$ and the threshold τ differently, they are all based on fitting a model (either explicitly or implicitly). If $\hat{x}_t$ is obtained using previous and subsequent observations to x_t (past, current, and future data), then the technique is within the **estimation** model-based methods. In contrast, if x_t is obtained relying only on previous observations to x_t (past

data), then the technique is within the **prediction** model-based methods. In practice, the main difference between using estimation or prediction methods is that techniques within this latter category can be used in streaming time series, because they can determine whether or not a new datum is an outlier as soon as it arrives.

Some other univariate outlier detection methods analyse all of the residuals obtained from different models to identify the outliers. For example, Hochenbaum et al. [6] use STL decomposition, and Akouemo and Povinelli [7, 8, 9] use ARIMA models with exogenous inputs, linear regression, or artificial neural networks (ANNs). Although most of these models can also be used in prediction, in this case, outliers are detected in the residual set using past and future data. Specifically, once the selected model is learnt, hypothesis testing is applied over the residuals to detect the outliers.

In contrast to estimation models, prediction models-based techniques fit a model to time series and obtain $\hat{x}_t$ using only past data; that is, without using the current point x_t or any posterior observations. Points that are very different from their predicted values are identified as outliers. All of the techniques within this category can deal with streaming time series. Within prediction-based methods, some use a fixed model and thus are not able to adapt to the changes that occur in the data over time. For example, the DeepAnT outlier detection approach presented by Munir et al. [10] applies a fixed Convolutional Neural Networks (CNNs) to predict values in the future. Other methods use an autoregressive model [11] or an ARIMA model [12], which obtain confidence intervals for the predictions instead of only point estimates. Consequently, these methods implicitly define the value of τ.

1.3.2 Multivariate Time Series

The input time series is sometimes a multivariate time series with possibly correlated variables rather than a univariate time series. In contrast to this case, the detection method used to identify point outliers in multivariate time series can deal not only with a single variable but also with more than one variable simultaneously. Additionally, a point outlier in a multivariate time series can affect one (univariate point) or more than one (multivariate point, a vector at time t) variable.

Univariate Techniques

Given that a multivariate time series is composed of more than one time-dependent variable, a univariate analysis can be performed for each variable to detect univariate point outliers, without considering dependencies that may exist between the variables. Although the literature barely provides examples of this type of approach, in essence, all of the univariate techniques could be applied to each time-dependent variable of the input multivariate time

series. As one of the few examples, Hundman et al. [13] propose using the Long-Short-Term Memory (LSTM) prediction model-based method to predict spacecraft telemetry and find point outliers within each variable in a multivariate time series, following the idea of equation 1.1.

Correlation dependencies between the variables are not considered when applying univariate techniques to each time-dependent variable, leading to a loss of information. To overcome this problem and at the same time to leverage that univariate detection techniques are highly developed, some researchers apply a preprocessing method to the multivariate time series to find a new set of uncorrelated variables where univariate techniques can be applied. These methods are based on dimensionality reduction techniques and, as depicted in figure 1.5, the multivariate series is simplified to a lower dimension representation before applying univariate detection techniques. Since the new series are combinations of the initial input variables, the identified outliers are multivariate; that is, they affect more than one variable. Some of those dimensionality reduction techniques are based on finding the new set of uncorrelated variables by calculating linear combinations of the initial variables: Papadimitriou et al. [14] propose an incremental Principal Component Analysis (PCA) algorithm to determine the new independent variables; Galeano et al. [15] suggest reducing the dimensionality with projection pursuit, which aims to find the best projections to identify outliers; Baragona and Battaglia [16] propose using Independent Component Analysis (ICA) to obtain a set of unobservable independent non-Gaussian variables.

Other techniques reduce the input multivariate time series into a single time-dependent variable rather than into a set of uncorrelated variables. For example, Lu et al. [17] define the transformed univariate series using the cross-correlation function between adjacent vectors in time.

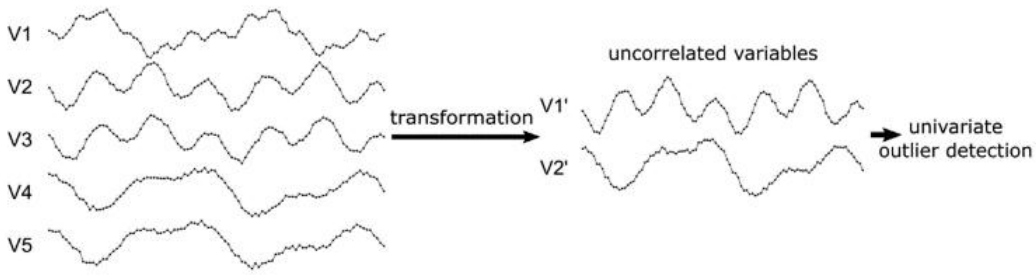


Figure 1.5: Example of multivariate time series dimensionality reduction.

Multivariate Techniques

In contrast to the univariate techniques, the multivariate methods deal simultaneously with multiple time-dependent variables, without applying previous transformations. These approaches perform the detection directly using all the original input data variables and can

be divided into two groups:

- **Model-based** - As in univariate time series, these techniques can also be used to detect point outliers in multivariate time series. The methods within this group are based on fitting a model that captures the dynamics of the series to obtain expected values in the original input time series. Then, for a predefined threshold τ , outliers are identified if

$$||x_t - \hat{x}_t|| > \tau, \tag{1.2}$$

  where x_t is the actual k-dimensional data point, and $\hat{x}_t$ its expected value. Note that this is a generalization of the definition given for the model-based techniques in univariate time series. Also in this context can be found estimation and prediction models. Within the former category, one of the most commonly used method is Autoencoders (AEs), which are a type of neural network that learns only the most significant features of a training set used as the reference of normality. Since outliers often correspond to nonrepresentative features, autoencoders fail to reconstruct them, providing large errors in equation 1.2 [18, 19].
  Prediction model-based techniques also fit a model to a multivariate time series, but the expected values are the predictions for the future made on the basis of past values: for example, the DeepAnt algorithm [10] is capable of detecting point outliers in multivariate time series using the CNN prediction model.

- **Dissimilarity-based** - These techniques are based on computing the pairwise dissimilarity between multivariate points or their representations, without the need for fitting a model. Therefore, for a predefined threshold τ , τ is a point outlier if

$$s(x_t, \hat{x}_t) > \tau, \tag{1.3}$$

  where x_t is the actual k-dimensional points. These methods do not usually use the raw data directly, but instead use different representation methods. For example, Cheng et al. [20, 21] represent the data using a graph where nodes are the multivariate points of the series and the edges the similarity values between them computed with the Radial Basis Function.

Chapter 2

Neural Networks for Time Series Forecasting

In the realm of model-based methods for time series anomaly detection, this chapter will focus its attention on some key features of ANN. In particular, it will be presented a brief overview of the architectures that will be used in order to develop the main network that will be presented in the following chapters.

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision and have become the go-to model for a wide range of visual recognition tasks. Their unique architecture, inspired by the human visual system, allows CNNs to effectively capture spatial hierarchies in images, making them highly effective for tasks like image classification, object detection, and segmentation. This review aims to provide a comprehensive overview of CNNs, discussing their architecture, key concepts, advancements, and applications.

2.1.1 Architecture of CNNs

CNNs are characterized by their use of convolutional layers, which apply convolutional filters to the input data. This process involves sliding filters across the input to produce feature maps, capturing local patterns such as edges, textures, and shapes [22]. The typical architecture of a CNN includes several types of layers:

- **Convolutional Layers** - These layers are the core building blocks of CNNs. They apply a set of learnable filters to the input, producing feature maps that highlight various aspects of the input image. The convolution operation is mathematically represented

as

$$(I * K)((x, y)) = \sum_m \sum_n I(x + m, y + m) \cdot K(m, n) \tag{2.1}$$

where I is the input image and K is the convolutional kernel. x and y represent the pixel locations in the image, and inset m and n are indexes over the kernel entries.

- **Pooling Layers** - Pooling layers reduce the spatial dimensions of the feature maps by sub-sampling the output from a convolutional layer: max-pooling or average-pooling are two of the most popular kind. This down-sampling process helps to reduce the computational load and make the representation more invariant to small translations of the input.

- **Fully Connected Layers** - These layers are usually placed at the end of the network and are responsible for making the final classification. They connect every neuron in one layer to every neuron in the next layer, similar to traditional neural networks.

- **Activation Functions**: function applied to the output of different layers in order to introduce non-linearity within the model, allowing to learn more complex patterns. One example could be the ReLU function, defined as

$$ReLU(x) = \max(0, x) \tag{2.2}$$

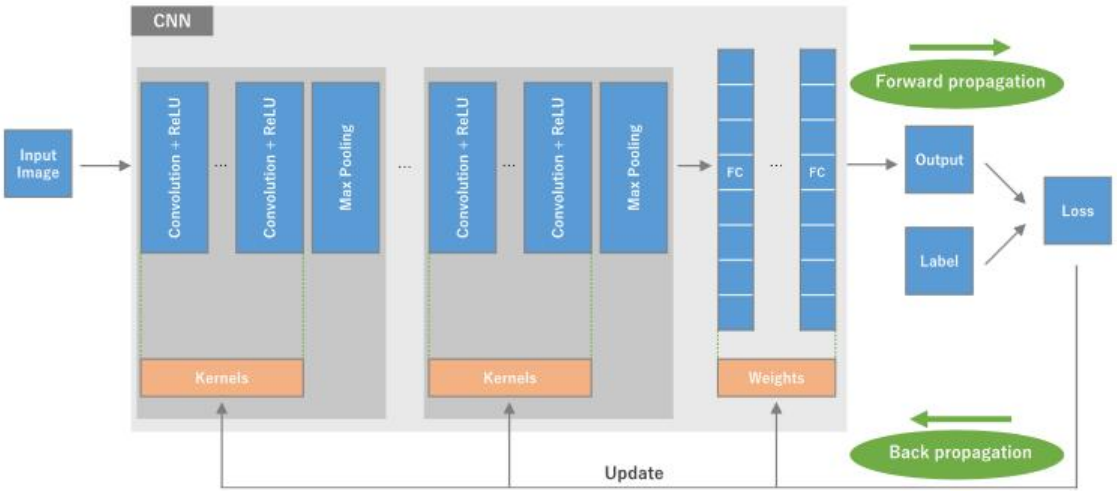


Figure 2.1: Example of the general structure of a CNN architecture: here is presented how data flow into the architecture and the model gets updated via back propagation.

2.1.2 Time Series Adaptation

Convolutional Neural Networks, originally designed for image processing tasks, have recently been adapted for time series forecasting due to their ability to capture local patterns and hierarchical features in the data. Using this framework, there are a few key concepts that apply to CNNs which helps them in dealing with time series forecasting:

- **Temporal Convolutions** - 1D convolutions are specifically designed to handle the sequential nature of time series data, capturing temporal dependencies and patterns [23].

- **Receptive Field** - The receptive field in a time series CNN determines how many time steps in the past influence the prediction. Increasing the receptive field can capture long-term dependencies.

- **Dilated Convolutions** - To capture larger temporal contexts without increasing the computational burden, dilated convolutions introduce gaps between filter elements, allowing for an exponentially growing receptive field with linear filter size increase.

- **Residual Connections** - Inspired by ResNet [24], residual connections help mitigate the vanishing gradient problem, allowing for deeper networks and improving performance on complex time series [25].

2.2 Long-Short-Term Neural Networks

Long-Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) architecture [26] designed to model sequential data and capture long-range dependencies. Introduced by Hochreiter and Schmidhuber in 1997 [27], LSTMs have addressed many of the limitations faced by traditional RNNs, such as the problem of vanishing gradients. This chapter explores the structure and application of LSTMs, particularly in the domain of time series forecasting.

2.2.1 Historical Background

Traditional RNNs were developed to handle sequential data by maintaining a hidden state that captures information from previous time steps. However, RNNs struggle with learning long-term dependencies due to issues such as vanishing and exploding gradients, which hinder their effectiveness in tasks that require memory over extended periods. LSTMs were developed to overcome these challenges by introducing a memory cell capable of maintaining its state for long periods of time.
The innovative architecture of LSTMs includes gating mechanisms that regulate the flow of information, allowing the network to learn what to keep, discard, and output from the cell state. This capability makes LSTMs particularly suitable for time series forecasting, where understanding temporal dependencies is crucial.

2.2.2 LSTM Architecture and Cell Structure

LSTM networks is composed of several units called *cells*. An LSTM cell comprises three main gates: the input gate, the forget gate, and the output gate. These gates control the cell state C_t and the hidden state h_t, allowing the network to manage information over long sequences. In the following, the three kinds of gate are briefly described and represented, figure 2.2:

- **Forget Gate** - Decides what information to discard from the cell state.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.3}$$

  where σ is the sigmoid function, W_f is the weight matrix, h_{t-1} is the previous hidden state, x_t is the current input, and b_f is the bias term.

- **Input Gate** - Determines what new information to store in the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_f)\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_f) \tag{2.4}$$

- **Cell State Update** - Updates the cell state with the new information.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.5}$$

- **Output Gate** - Controls what information to output from the cell state.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)h_t = o_t * \tanh(C_t) \tag{2.6}$$

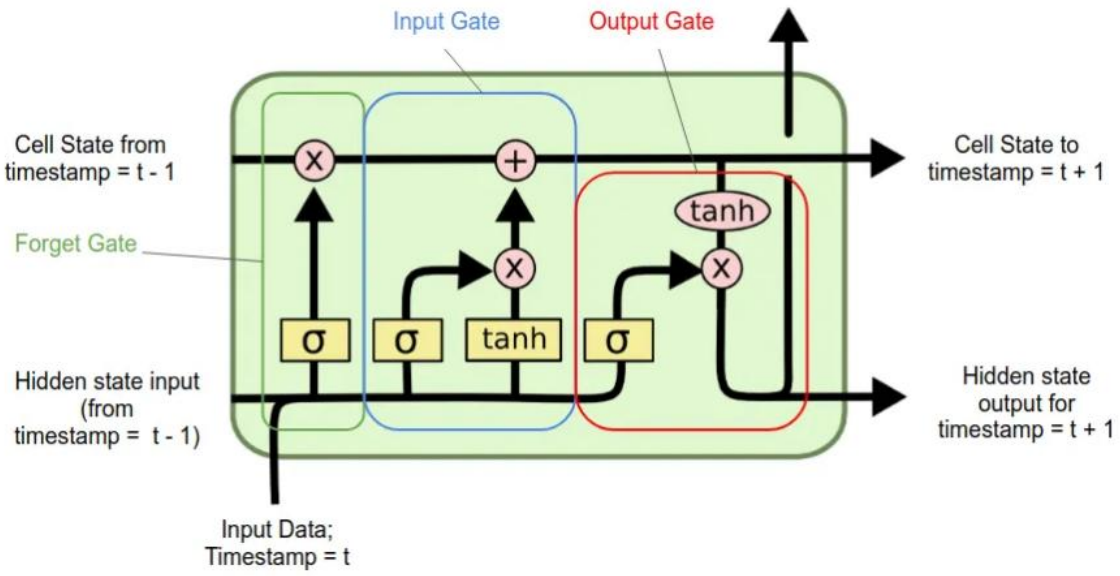


Figure 2.2: LSTM Cell Structure.

2.2.3 Time Series Applications

Here are presented some of the most popular fields in which LSTM architecture has been applied.

Financial Time Series

LSTMs are widely used in financial time series forecasting, where they predict stock prices, commodity prices, and market indices [28]. The ability of LSTMs to capture temporal dependencies and learn from past trends makes them valuable tools for financial analysts and traders. Studies have shown that LSTMs outperform traditional statistical methods in predicting complex financial time series.

Weather Prediction

Weather forecasting is highly dependent on understanding temporal patterns and dependencies in meteorological data. LSTMs have been successfully applied to predict various weather parameters, such as temperature, precipitation, and wind speed [29]. Their capability to handle long-range dependencies allows them to model seasonal patterns and trends effectively.

Energy Load Forecasting

Accurate forecasting of energy demand is critical for efficient management and planning of the power grid. LSTMs are used to predict energy consumption patterns by learning from historical load data [30]. Their application helps to optimise energy distribution, reduce costs, and improve the reliability of the power supply.

Chapter 3

Bayesian Inference and Neural Networks

3.1 Bayesian Inference

The Bayesian paradigm in statistics contrasts with the frequentist paradigm, with a major area of distinction in hypothesis testing [31]. It is based on two simple ideas. The first is that probability is a measure of belief in the occurrence of events, rather than the limit in the frequency of occurrence when the number of samples goes toward infinity, as assumed in the frequentist paradigm. The second idea is that prior beliefs influence posterior beliefs. Bayes' theorem, which states that:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} = \frac{P(D,H)}{\int_H P(D,H')dH'} \tag{3.1}$$

summarizes this interpretation. Formula 3.1 is still true in the frequentist interpretation, where H and D are considered as sets of outcomes.
The Bayesian interpretation considers H to be a hypothesis about which one holds some prior belief, and D to be some data that will update one's belief about H. The probability distribution $P(D|H)$ is called the likelihood. It encodes the aleatoric uncertainty in the model, i.e., the uncertainty due to the noise in the process.
$P(H)$ is the prior and $P(D) = \int_H P(D,H')dH'$ the evidence. $P(H|D)$ is called the posterior. It encodes the epistemic uncertainty, i.e., the uncertainty due to the lack of data. $P(D|H)P(H) = P(D,H)$ is the joint probability of D and H. Using Bayes' formula to train a predictor can be understood as learning from the data D. In other words, the Bayesian paradigm not only offers a solid approach for the quantification of uncertainty in deep learning models but also provides a mathematical framework to understand many regularization techniques and learning strategies that are already used in classic deep learning [32] .

3.2 Bayesian Neural Networks

A Bayesian Neural Network (BNN) is defined slightly differently across the literature, but a commonly agreed definition is that a BNN is a stochastic artificial neural network trained using Bayesian inference [33]. The goal of artificial neural networks (ANNs) is to represent an arbitrary function $y = \phi(x)$. Traditional ANNs such as feedforward networks and recurrent networks are built using one input layer l_0, a succession of hidden layers $l_i, i = 1, ..., n-1$, and one output layer l_n. (Here, $n+1$ is the total number of layers.) In the simplest architecture of feedforward networks, each layer l is represented as a linear transformation, followed by a nonlinear operation s, also known as an activation function:

$$l_0 = x, \tag{3.2}$$

$$l_i = s_i(W_i l_{i-1} + b_i) \quad \forall i \in [1, n], \tag{3.3}$$

$$y = l_n. \tag{3.4}$$

Here, $\theta = (W, b)$ are the parameters of the network, where W are the weights of the network connections and b the biases. A given ANN architecture represents a set of functions isomorphic to the set of possible parameters θ.

Deep learning is the process of regressing the parameters θ from the training data D, where D is composed of a series of input x and their corresponding labels y. The standard approach is to approximate a minimal cost point estimate of the network parameters θ, i.e., a single value for each parameter, using the backpropagation algorithm, with all other possible parametrizations of the network discarded.

The cost function is often defined as the log likelihood of the training set, sometimes with a regularization term included. From a statistician's point of view, this is a maximum likelihood estimation (MLE), or a maximum a posteriori (MAP) estimation when regularization is used. The point estimate approach, which is the traditional approach in deep learning, is relatively easy to deploy with modern algorithms and software packages, but tends to lack explainability [34].

The final model might also generalize in unforeseen and overconfident ways on out-of-training distribution data points [35, 36]. This property, in addition to the inability of ANNs to say "I don't know", is problematic for many critical applications. Of all the techniques that exist to mitigate this [37], stochastic neural networks have proven to be one of the most generic and flexible. Stochastic neural networks are a type of ANN built by introducing stochastic components into the network. This is performed by giving the network either a stochastic activation or stochastic weights to simulate multiple possible models θ with their associated probability distribution $p(\theta)$. Thus, BNNs can be considered a special case of ensemble learning [38].

The main goal of using a stochastic neural network architecture is to obtain a better idea of the uncertainty associated with the underlying processes. This is accomplished by comparing the predictions of multiple sampled model parametrizations θ. If the different models agree, then the uncertainty is low. If they disagree, then the uncertainty is high. This process can be summarized as follows:

$$\theta \sim p(\theta), \tag{3.5}$$

$$y = \phi_\theta(x) + \epsilon \tag{3.6}$$

where ϵ represents random noise to account for the fact that the function ϕ is only an approximation.

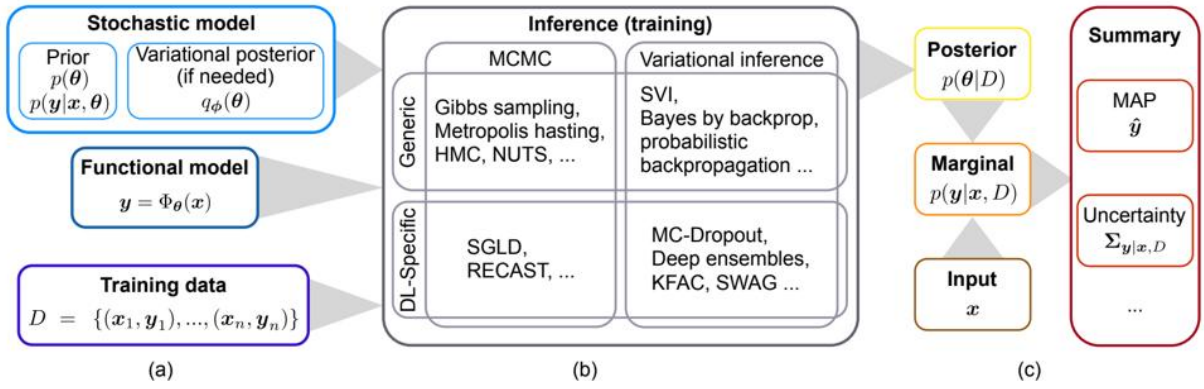


Figure 3.1: Workflow to design (a), train (b) and use a BNN for predictions (c).

A BNN can then be defined as any stochastic artificial neural network trained using Bayesian inference [39]. To design a BNN, the first step is the choice of a deep neural network architecture, i.e., a functional model. Then, one has to choose a stochastic model, i.e., a prior distribution over the possible model parametrization $p(\theta)$ and a prior confidence in the predictive power of the model $p(y|x, \theta)$ (Figure 3.1a). The model parametrization can be considered to be the hypothesis H and the training set is the data D. The choice of a BNN's stochastic model is somehow equivalent to the choice of a loss function when training a point estimate neural network. The model parameters will be denoted by θ, the training set by D, the training inputs by D_x, and the training labels by D_y. By applying Bayes' theorem, and enforcing independence between the model parameters and the input, the Bayesian posterior can be written as:

$$p(\theta|D) = \frac{p(D_y|D_x, \theta)p(\theta)}{\int_\theta p(D_y|D_x, \theta')p(\theta')d\theta'} \propto p(D_y|D_x, \theta)p(\theta) \tag{3.7}$$

The Bayesian posterior for complex models such as artificial neural networks is a high dimensional and highly non-convex probability distribution [40]. This complexity makes com-

puting and sampling it using standard methods an intractable problem, especially because computing the evidence $\int_\theta p(D_y|D_x,\theta')p(\theta')d\theta'$ is difficult. To address this problem, two broad approaches have been introduced: (1) Markov chain Monte Carlo and (2) variational inference.

When using a BNN for prediction, the probability distribution $p(y|x,D)$ [12], called the **marginal distribution** and which quantifies the model's uncertainty on its prediction, is of particular interest. Given $p(\theta|D)$, $p(y|x,D)$ can be computed as:

$$p(y|x,D) = \int_\theta p(y|x,\theta')p(\theta'|D)d\theta' \tag{3.8}$$

In practice, $p(y|x,D)$ is sampled indirectly using Equation 3.6. The final prediction can be summarized by statistics computed using a Monte Carlo approach (3.1c). A large set of weights θ_i is sampled from the posterior and used to compute a series of possible outputs y_i , as shown in Algorithm 1, which corresponds to samples from the marginal distribution.

In Algorithm 1, Y is a set of samples from $p(y|x,D)$ and θ a collection of samples from $p(\theta|D)$. Usually, aggregates are computed on those samples to summarize the uncertainty of the BNN and obtain an estimator for the output y. This estimator is denoted by $\hat{y}$. When performing regression, the procedure that is usually used to summarize the predictions of a BNN is model averaging [23]:

$$\hat{y} = \frac{1}{|\Theta|}\sum_{\theta_i\in\Theta}\phi_{\theta_i}(x) \tag{3.9}$$

This approach is so common in ensemble learning that it is sometimes called **ensembling**. To quantify uncertainty, the covariance matrix can be computed as follows:

$$\sum_{y|x,D} = \frac{1}{|\Theta|-1}\sum_{\theta_i\in\Theta}(\phi_{\theta_i}(x)-\hat{y})(\phi_{\theta_i}(x)-\hat{y})^T \tag{3.10}$$

Algorithm 1 Inference procedure for a BNN.

Define $p(\theta|D) = \frac{p(D_y|D_x,\theta)p(\theta)}{\int_\theta p(D_y|D_x,\theta')p(\theta')d\theta'}$
for $i = 0$ to N **do**
  Draw $\theta_i \sim p(\theta|D)$;
  $y_i = \phi_{\theta_i}(x)$
end forreturn $Y = \{y_i|i\in[0,N)\}, \Theta = \{\theta_i|i\in[0,N)\}$

3.3 Bayesian Learning Advantages

One of the major critiques of Bayesian methods is that they rely on prior knowledge. This is especially true in deep learning, as deriving any insight about plausible parametrization for a given model before training is very challenging. Thus, why use Bayesian methods for deep learning? Discriminative models implicitly represent the conditional probability $p(y|x, \theta)$, and Bayes' formula is an appropriate tool to invert conditional probabilities, even if one has little insight about $p(\theta)$ a priori.
While there are strong theoretical principles and schema upon which this Bayes' formula can be based [41], here are presetend some of the practical benefits granted by using BNN.

- First, Bayesian methods provide a natural approach to quantify uncertainty in deep learning since BNNs have better calibration than classical neural networks [42, 43, 44], i.e., their uncertainty is more consistent with the observed errors. They are less often overconfident or underconfident.

- Second, a BNN allows distinguishing between the epistemic uncertainty $p(\theta|D)$ and the aleatoric uncertainty $p(y|x, \theta)$ [45]. This makes BNNs very data-efficient since they can learn from a small dataset without overfitting [46]. At prediction time, out-of-training distribution points will have high epistemic uncertainty instead of blindly giving a wrong prediction.

- Third, the no-free-lunch theorem for machine learning [47] can be interpreted as stating that any supervised learning algorithm includes some implicit prior. Bayesian methods, when used correctly, will at least make the prior explicit. Integrating prior knowledge into ANNs, which work as black boxes, is difficult but not impossible. In Bayesian deep learning, priors are often considered as soft constraints, analogous to regularization, or data transformations such as data augmentation in traditional deep learning. Most regularization methods used for point estimate neural networks can be understood from a Bayesian perspective as setting a prior.

Chapter 4

Bayesian Autoencoder for Confidence Interval Forecasting

4.1 Introduction

As stated in the first chapter of this work, accurate time series forecasting and reliable estimation of the prediction uncertainty are critical for anomaly detection, optimal resource allocation, budget planning, and other related tasks [48]. This problem is challenging, especially during high variance segments (e.g., holidays, sporting events), because extreme event prediction depends on numerous external factors that can include weather, city population growth, or marketing changes (e.g., driver incentives) [49] that all contribute to the uncertainty of the forecast. LSTM model [27] has gained popularity due to its end-to-end modeling, ease of incorporating exogenous variables, and automatic feature extraction abilities [50]. By providing a large amount of data across numerous dimensions, it has been shown that an LSTM network can model complex nonlinear feature interactions [51], which is critical for modeling complex extreme events. A recent paper from 2017 [52] has shown that a neural network forecasting model is able to outperform classical time series methods in cases with long, interdependent time series. However the problem of estimating the uncertainty in time-series predictions using neural networks remains an open question. The prediction uncertainty is important for assessing how much to trust the forecast produced by the model, and has profound impact in anomaly detection.

In this work is proposed a novel end-to-end model architecture for time series prediction, and quantify the prediction uncertainty using Bayesian Neural Network, which is further used for large-scale anomaly detection.

Under this framework, the prediction uncertainty can be decomposed into three types: *model uncertainty*, *inherent noise*, and *model misspecification*. **Model uncertainty**, also referred to as epistemic uncertainty, captures the ignorance of the model parameters, and can be

reduced as more samples being collected. **Inherent noise**, on the other hand, captures the uncertainty in the data generation process and is irreducible. These two sources have been previously recognized with successful application in computer visions [53]. The third uncertainty from **model misspecification**, however, has been long-overlooked [48]. This captures the scenario where the testing samples come from a different population than the training set, which is often the case in time series anomaly detection. Similar ideas have gained attention in deep learning under the concept of adversarial examples in computer vision [54], but its implication in prediction uncertainty remains unexplored.
This thesis presents a novel hybrid model that combines the temporal learning capabilities of LSTMs with the spatial feature extraction prowess of CNNs, encapsulated within an Autoencoder architecture. Taking advantage of bayesian learning techniques the model is able to compute confidence interval for its prediction allowing to make anomaly detection without the need of training a trigger.

4.2 Bayesian Approximation

In order to compute confidence intervals for the prediction made by the architecture which will be presented in the next section, the network developed is inspired by the Monte Carlo dropout (MC dropout) framework proposed in [55] and [56], which requires no change of the existing model architecture and provides uncertainty estimation almost for free. Specifically, stochastic dropouts are applied after each hidden layer, and the model output can be approximately viewed as a random sample generated from the posterior predictive distribution [57]. As a result, the model uncertainty can be estimated by the sample variance of the model predictions in a few repetitions, as explained in the previous chapter, 3.

4.2.1 Method

Given a trained neural network $f^{\hat{W}}(\cdot)$ where $\hat{W}$ represents the fitted parameters, as well as a new sample x^*, the goal is to evaluate the uncertainty of the model prediction, $\hat{y}^* = f^{\hat{W}}(x^*)$. Specifically, to quantify the prediction standard error, η so that an approximate α-level prediction interval can be constructed by

$$[\hat{y}^* - z_{\alpha/2}\eta, \hat{y}^* + z_{\alpha/2}\eta] \tag{4.1}$$

where $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of a standard Normal. This prediction interval is critical for various tasks. For example, in anomaly detection, anomaly alerts will be fired when the observed value falls outside the constructed 95% interval. As a result, underestimating η will lead to high false positive rates. In the rest of this section, we will present our uncertainty

estimation algorithm in Section 3.1, which accounts for three different sources of prediction uncertainties. This framework can be generalized to any neural network architectures.

4.2.2 Prediction Uncertainty

Let $f^W(\cdot)$ the functional neural network representation, where f captures the network architecture, and W is the collection of model parameters. In a Bayesian neural network, a prior is introduced for the weight parameters, and the model aims to fit the optimal posterior distribution. For example, in regression, a Gaussian prior is commonly assumed:

$$W \sim N(0, \mathbb{I}) \tag{4.2}$$

Let us further specify the data generating distribution $p(y|f^W(x))$. In regression, is often assume

$$y|W \sim N(f^W(x), \sigma^2) \tag{4.3}$$

with some noise level σ. In classification, the softmax likelihood is often used. For time series prediction, the focus will be on the regression setting. Given a set of N observations $X = \{x_1, ..., x_N\}$ and $Y = \{y_1, ..., y_N\}$, Bayesian inference aims at finding the posterior distribution over model parameters $p(W|X, Y)$. With a new data point $x*$, the prediction distribution is obtained by marginalizing out the posterior distribution:

$$p(y^*|x^*) = \int_W p(y^*|f^W(x^*))p(W|X, Y)dW \tag{4.4}$$

In particular, the variance of the prediction distribution quantifies the prediction uncertainty, which can be further decomposed using law of total variance:

$$\begin{aligned} \mathrm{Var}(y^*|x^*) &= \mathrm{Var}[\mathbb{E}(y^*|W, x^*)] + \mathbb{E}[\mathrm{Var}(y^*|W, x^*)] && (4.5)\\ &= \mathrm{Var}(f^W(x^*)) + \sigma^2 && (4.6) \end{aligned}$$

It can immediately be seen that the variance is decomposed into two terms:

(i) $\mathrm{Var}(f^W(x^*))$, which reflects our ignorance over model parameter W, referred to as the *model uncertainty*;

(ii) σ^2 which is the noise level during data generating process, referred to as the *inherent noise*.

However, this is not always the case in practice. In anomaly detection, in particular, it is expected that certain time series will have unusual patterns, which can be very different from the trained model. Therefore, here is proposed that a complete measurement of prediction

uncertainty should be a combination from three sources: (i) model uncertainty, (ii) model misspecification, and (iii) inherent noise level [48].

Model uncertainty

The key to estimating model uncertainty is the posterior distribution $p(W|X,Y)$, also referred to as Bayesian inference. This is particularly challenging in neural networks because the non-conjugacy due to nonlinearities. There have been various research efforts on approximate inference in deep learning. Here, following the idea in [55] and [56] to approximate model uncertainty using Monte Carlo dropout (MC dropout). The algorithm proceeds as follows: given a new input x^*, we compute the neural network output with stochastic dropouts at each layer. That is, randomly dropout each hidden unit with certain probability p. This stochastic feed-forward is repeated B times, and we obtain $\{\hat{y}^*_{(1)}, ..., \hat{y}^*_{(B)}\}$. Then the model uncertainty can be approximated by the sample variance:

$$\hat{\text{Var}}(f^W(x^*)) = \frac{1}{B}\sum_{b=1}^{B}(\hat{y}^*_{(b)} - \bar{\hat{y}}^*)^2 \tag{4.7}$$

where $\bar{\hat{y}}^* = \frac{1}{B}\sum_{b=1}^{B}\hat{y}^*_{(b)}$ [55].

Model misspecification

Next, the problem of capturing potential model misspecification has to be adressed. In particular, the goal is to capture the uncertainty when predicting unseen samples with very different patterns from the training data set. In [48] is proposed to account for this source of uncertainty by introducing an encoder-decoder to the model framework. The idea is to train an encoder that extracts the representative features from a time series, in the sense that a decoder can reconstruct the time series from the encoded space. At test time, the quality of encoding of each sample will provide insight on how close it is to the training set. Another way to think of this approach is to first fitting a latent embedding space for all training time series using an encoder-decoder framework. Then, measuring the distance between test cases and training samples in the embedded space.

Inherent noise

Finally, the inherent noise level σ^2 needs to be estimated. In the original MC dropout algorithm [55], this parameter is implicitly determined by a prior over the smoothness of W . As a result, the model could end up with drastically different estimations of the uncertainty level depending on this pre-specified smoothness [57]. This dependency is undesirable in anomaly detection, because would be desirable for the uncertainty estimation to also have

robust frequentist coverage, but it is rarely the case that the correct noise level is known a priori. In [48] is proposed a simple and adaptive approach that estimates the noise level via the residual sum of squares, evaluated on an independent held-out validation set. Following the steps of the original paper [48] it can be shown that:

$$\hat{\sigma^2} = \frac{1}{V}\sum_{v=1}^{V}(y'_v - f^W(x'_v))^2 \tag{4.8}$$

where V is the size of the validation set, provides an asymptotically unbiased estimation on the inherent noise level. In the finite sample scenario, it always overestimates the noise level and tends to be more conservative.

At last, here are presented the MC dropout algorithm and the one needed to compute the finel boundaries of the prediction confidence intervals.

Algorithm 2 MC Dropout.

Input: data x^*, encoder $g(\cdot)$, prediction network $h(\cdot)$, dropout probability p, number of iterations B
Output: prediction $\hat{y}^*_{mc}$, uncertainty η_1
for $b = 1$ to B **do**
  $e^*_{(b)} \leftarrow$ *VariationalDropout*$(g(x^*), p)$
  $z^*_{(b)} \leftarrow$ *Concatenate*$(e^*_{(b)}$, external feature$)$
  $\hat{y}^*_{(b)} \leftarrow$ *Dropout*$(h(z^*_{(b)}), p)$
end for
$\hat{y}^*_{mc} \leftarrow \frac{1}{B}\sum_{b=1}^{B}\hat{y}^*_b$
$\eta_1^2 \leftarrow \frac{1}{B}\sum_{b=1}^{B}(\hat{y}^*_b - \hat{y}^*)^2$ **return** $\hat{y}^*_{mc}$, η_1

Algorithm 3 Inference.

Input: data x^*, encoder $g(\cdot)$, prediction network $h(\cdot)$, dropout probability p, number of iterations B
Output: prediction $\hat{y}^*$, uncertainty η
$\hat{y}^*_{mc}, \eta_1 \leftarrow$ *MCDropout*(x^*, g, h, p, B)
for x'_v in validation set $\{x'_1, ...x'_V\}$ **do**
  $\hat{y}'_v \leftarrow h(g(x'_v))$
end for
$\eta_2^2 \leftarrow \frac{1}{V}\sum_{v=1}^{V}(\hat{y}^*_v - y^*)^2$
$\eta \leftarrow \sqrt{\eta_1^2 + \eta_2^2}$ **return** $\hat{y}^*$, η

4.3 Model Design

In this section is presented BeaConF (Bayesian Autoencoder for Confidence interval Forcasting).
The complete architecture of the neural network is shown in Figure 4.1. The network contains two major components: (i) an encoder-decoder framework that captures the inherent pattern in the time series, which is learned during pre-training step, and (ii) a prediction network that takes input from both the learned embedding from encoder-decoder, as well as any potential external features to guide the prediction.

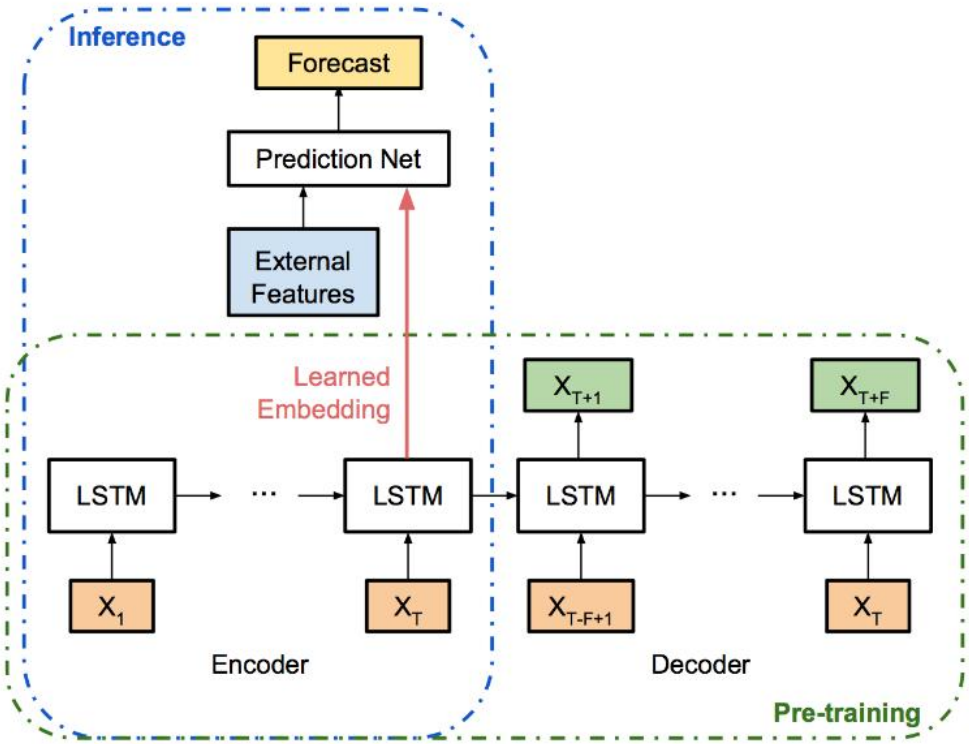


Figure 4.1: Neural network architecture, with a pre-training phase using a LSTM encoder-decoder, followed by a prediction network, with input being the learned embedding concatenated with external features.

4.3.1 Autoencoder

Prior to fitting the prediction model, we first conduct a pre-training step to fit an encoder that can extract useful and representative embeddings from a time series. The goals are to ensure that:

(i) the learned embedding provides useful features for prediction;

(ii) unusual input can be captured in the embedded space, which will get further propagated to the prediction network in the next step.

Here, an encoder-decoder framework with two-layer LSTM cells is used. Specifically, given a univariate time series $\{x_t\}_t$, the encoder reads in the first T timestamps $\{x_1, ..., x_T\}$, and

constructs a fixed-dimensional embedding state. After then, from this embedding state, the decoder constructs the following F timestamps $\{x_T + 1, ..., x_T + F\}$ with guidance from $\{x_T - F + 1, ..., x_T\}$ (Figure 4.2). The intuition is that in order to construct the next few timestamps, the embedding state must extract representative and meaningful features from the input time series. This design is inspired from the success of video representation learning using a similar architecture [58].

4.3.2 Prediction Network

After the encoder-decoder is pre-trained, it is treated as an intelligent feature-extraction blackbox. Specifically, the last LSTM cell states of the encoder are extracted as learned embedding. Then, a prediction network is trained to forecast the next one or more timestamps using the learned embedding as features. In the scenario where external features are available, these can be concatenated to the embedding vector and passed together to the final prediction network.

Inspired by DeepAnt [10], here a CNN is used as prediction network: two convolutional layers, each followed by a max-pooling layer, are used in this architecture as shown in Figure 4.2. The input layer has w input nodes as we have converted the data into w latent vectors. Each convolution layer is composed of 32 filters (kernels) followed by an element-wise activation function, ReLU. Last layer of the network is a fully connected (FC) layer in which each neuron is connected to all the neurons in the previous layer. This layer represents the network prediction for the next time stamp. The number of nodes used in the output layer are equal to F. In our case, we are predicting only the next time stamp, so the number of output node is 1.

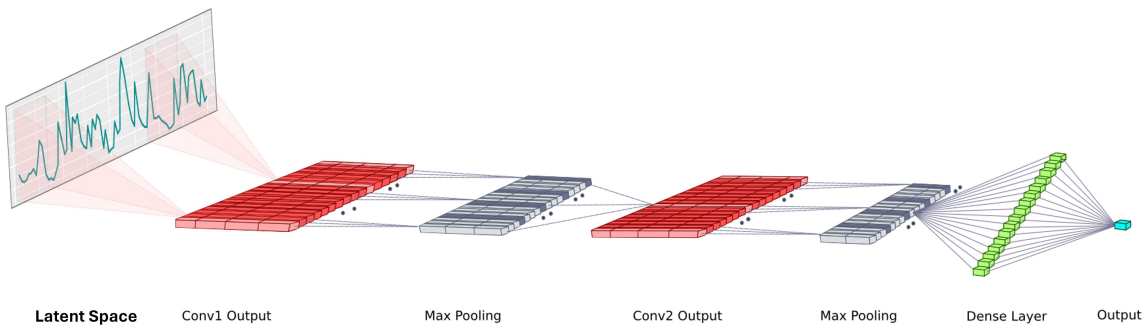


Figure 4.2: Predictor architecture for time series forecasting: A convolutional neural network with two convolutional layers, two max pooling, and a fully connected layer.

Chapter 5

Benchmark Datasets

Before proceeding with the experiments conducted in order to determine BAeCon performances, let us review the main dataset that was used in order to test it and compare it with respect to other already existing algorithms.

In this short chapter will be presented NAB benchmark [59], its data and scoring systems with its merits and limitations.

5.1 Numenta Anomaly Benchmark

Benchmarks designed for static datasets do not adequately capture the requirements of real-time applications. For example, scoring with standard classification metrics such as precision and recall do not suffice because they fail to reflect the value of early detection. An artificial separation into training and test sets does not properly capture a streaming scenario nor does it properly evaluate a continuously learning algorithm. The NAB methodology and scoring rules (described below) are designed with such criteria in mind. Following insights from industry experts [59] it is beneficial for the industry to include real-world labeled data from multiple domains. Such data is rare and valuable, and NAB attempts to incorporate such a dataset as part of the benchmark.

5.1.1 Dataset

NAB aims to represent the variety of anomalous data and the associated challenges detectors face in real-world streaming applications. We define anomalies in a data stream to be patterns that do not conform to past patterns of behavior for the stream. This definition encompasses both point anomalies (or spatial anomalies) as well as temporal anomalies. For example, a spiking point anomaly occurs when a single data point extends well above or below the expected range. Streaming data commonly also contains temporal anomalies,

such as a change in the frequency, sudden erratic behavior of a metric, or other temporal deviations. Anomalies are defined with respect to past behavior. This means a new behavior can be anomalous at first but ceases to be anomalous if it persists; i.e. a new normal pattern is established. Fig. 1 shows a few representative anomalies taken from the NAB dataset.
The data currently in the NAB corpus represents a variety of metrics ranging from IT metrics such as network utilization to sensors on industrial machines to social media chatter. We also include some artificially-generated data files that test anomalous behaviors not yet represented in the corpus's real data, as well as several data files without any anomalies. The current NAB dataset contains 58 data files, each with 1000-22,000 data instances, for a total of 365,558 data points. The NAB dataset is labeled by hand, following a meticulous, documented procedure. Labelers must adhere to a set of rules when inspecting data files for anomalies, and a label-combining algorithm formalizes agreement into ground truth labels. The process is designed to mitigate human error as much as possible[1]. In addition a smooth scoring function (described below) ensures that small labeling errors will not cause large changes in reported scores.
It is often prohibitively expensive to collect an accurately labeled set of anomalous data instances that covers all types of anomalous behavior [60]. A key element of the NAB dataset is the inclusion of real-world data with anomalies for which the causes are known.

5.1.2 NAB Scoring System

In NAB an anomaly detector accepts data input and outputs instances which it deems to be anomalous. The NAB scoring system formalizes a set of rules to determine the overall quality of anomaly detection. We define the requirements of the ideal, real-world anomaly detector as follows:

1. Detects all anomalies present in the streaming data.
2. Detects anomalies as soon as possible, ideally before the anomaly becomes visible to a human.
3. Triggers no false alarms (no false positives).
4. Works with real time data (no look ahead).
5. Is fully automated across all datasets (any data specific parameter tuning must be done online without human intervention).

There are three key aspects of scoring in NAB:

[1]The full labeling process and rules can be found in the NAB wiki, along with the label-combining source code, in the NAB repository.

- **anomaly windows** - An anomaly window consists of a sequence of data points centered around one or more true anomalies in a dataset. These windows are used by the NAB scoring function to compute weights of individual anomaly detections for a given dataset. It is of note that, in the NAB scoring system, if there are multiple detections within a particular anomaly window, then only the earliest detection is considered as a true positive, and all subsequent anomaly detections within the window are ignored.

- **scoring function** - NAB defines three different application profiles: standard, reward low false positives, and reward low false negatives. In the standard profile, relative weights are assigned to true positives, false positives and false negatives, based on the window size, whereas in the reward low false positives profile and the reward low false negatives profile, greater penalties are assigned for false positives and false negatives respectively. NAB provides anomaly scores for each of these application profiles for every dataset.

- **application profiles** - Given an anomaly window and an application profile, NAB uses the following sigmoidal scoring function to compute weight of each anomaly detection:

$$\sigma^A(y) = (A_{TP} - A_{FP}) \left(\frac{1}{1 + e^{5y}} \right) - 1 \tag{5.1}$$

where A is the given application profile, y is the relative position of the detection in the given anomaly window,A_{TP} is the corresponding weight of true positives in profile A, and A_FP is the corresponding weight of false positives in profile A. This scoring function is designed so as to give higher positive scores to true positive detections earlier in a window and negative scores to detections outside the window. Using the weights of individual detections, the 'raw score' of a dataset, denoted by S_d^A is calculated as follows:

$$S_d^A = \left(\sum_{y \in Y_d} \sigma^A(y) \right) + A_{FN} f_d \tag{5.2}$$

In this equation, the sum of weighted scores of true positives and false positives is computed, which is then discounted by the weighted sum of missed detections computed as $A_{FN} f_d$, where A_{FN} denotes the weight of false negatives in profile A, and f_d denotes the total number of false negatives (or missed detections) in dataset d. Using Eq. 5.2, the final normalized, reported score over all 58 datasets is computed as follows:

$$S_{NAB}^A = 100 \cdot \frac{S^A S_{null}^A}{S_{perfect}^A S_{null}^A} \tag{5.3}$$

where S_A is the sum of raw scores over all datasets, (i.e. $\sum_d S_d^A$), S_A null is the sum of raw scores achieved by a null detector (i.e., the one that outputs no anomaly

detections), and , $S^A_{perfect}$ is the sum of raw scores achieved by a (hypothetical) perfect detector (i.e., the one that outputs all true positives and no false positives/negatives).

5.2 NAB Limitations

Following the work of Singh et al. [61] in this section are presented some limitation that are inherit of the NAB benchmark.

5.3 Dataset

1. **Missing values in datasets** - A time-series is generally composed of data points that are collected at a constant interval or frequency (e.g., every 1 hour, every 10 minutes). In other words, the pacing of observation times is constant in a general time series. In scenarios where observations are not collected at a constant interval, one or more forms of data preprocessing is done to create equally spaced time series before fitting a model to the dataset, failing which a time series model may yield inaccurate results. NAB provides time series datasets that are mostly composed of observations gathered at a constant interval, but some of the datasets do not have this desirable property due to which time series models fail to perform well on these datasets.

2. **Difference in data distribution** - In many NAB datasets (like occupancy6005, speed6005), the distribution of data for the first few days (e.g., 4 or 5 days) happens to be quite different from the distribution for rest of the time series. This characteristic of certain NAB datasets negatively impacts with the training of machine learning or time series models because the first few days of data is typically used for training these models while rest of the dataset is used for the testing purpose. This difference in training and test distributions is one of the potential causes of low performance of different models that have been so far evaluated on NAB datasets.

5.4 Scoring System

1. **Determining anomaly window's size** - The NAB scoring system is based on anomaly windows, but there is no systematic way of determining the optimal size of anomaly windows. Lavin and Ahmad in [59] chose the window size to be 10% the number of instances in a dataset, divided by the number of anomalies in the given dataset. But, window size cannot be chosen in this way in many real-world problem settings, especially in streaming environments, because the number of instances in a

dataset may not be known in advance (e.g., in streaming environment) and more importantly, it is impossible to know the number of anomalies in a streaming dataset before an anomaly detection algorithm is executed on the dataset. This leaves little room for applicability of the proposed way of selecting window size in real-life problem scenarios.

2. **Gaps in scoring function** - The scoring function of the NAB framework is primarily based on Eq. 5.1, which is used to compute the weight of each detection given an anomaly window and an application profile. But, this equation is not well-defined and has the following gaps. First, it is unclear as to how y (which denotes the relative position of a detection in the given anomaly window) should be computed. Hence, the range of values that y can take on remains unknown. Second, in Eq. 5.1, value of $5y$ is chosen as the power of e (in the second term). But the reason for choosing the value of 5 is not clear, due to which it remains unknown as to what impact can other values, like 5.1 or 50, have on the weights of detections and how will it change the overall NAB score of an algorithm. Third, the scoring example provided in [59] does not match with Eq. 5.1. For convenience sake, we replicate that figure in Fig. 5.1a, where we see that when the relative position of the detection is outside the anomaly window, i.e., at y (approximately) equal to -5, the scaled sigmoid value (i.e., the second term of Eq. 5.1) is -1. This is incorrect as, in this case, the second term of Eq. 5.1 would yield a value close to 0, i.e., -0.0000000000139 (calculated as $\frac{1}{1+e^{5(-5)}} - 1$). This is shown pictorially in Fig. 5.1b. Similarly, when the relative position of the detection is -2 in the anomaly window, then the scaled sigmoid value is $+0.9999$. This is incorrect as well as, the second term of Eq. 5.1 would still yield a value close to 0, i.e., -0.0000453, (calculated as $\frac{1}{1+e^{5(-2)}} - 1$).

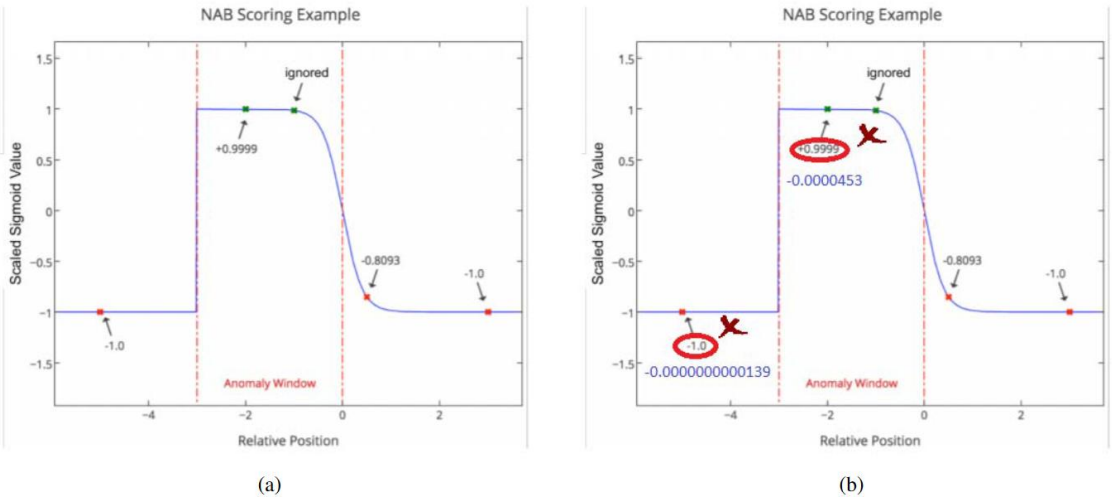


Figure 5.1: Example of NAB scoring system: (a) Example as provided originally in the NAB framework description [59] (b) Example after analysis and corrections

5.4.1 Alternative Metrics

Given the limitation this scoring system present, the algorithm presented in this thesis is going to be tested using a different metric.
Let us call **precision** the measure of how often the algorithm correctly predicts an anomaly:

$$precision = \frac{TP}{TP + FP} \tag{5.4}$$

and **recall** the measure of how often a a machine learning model correctly identifies anomalies from all the actual ones:

$$recall = \frac{TP}{TP + FN} \tag{5.5}$$

Notice the fact that a low value for the recall indicates that the algorithm in question is classifying incorrectly a lot of anomalies as ordinary data, while a low precision indicates the fact that a lot of ordinary points are misclassified as anomalies. These metrics are widely spread across the literature and already used to benchmark a variety of different algorithms. The results that will be presented in the following chapters are going to me measured using the **F-score** which consists in the harmonic mean between precision and recall:

$$F - score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.6}$$

The highest possible value of an F-score is 1.0, indicating perfect precision and recall, and the lowest possible value is 0, if precision and recall are zero.

Chapter 6

Experiments

In this chapter is described the methodology implemented in order to train, validate and test BAeCon using the NAB dataset, taking into account its limitations, and confronting the results with some of the most successful results in the literature.
The entire implementation has been carried using Python as programming language and in particular the PyTorch package [62].

6.1 Data Preparation

In order to get the algorithm work at its full capabilities and taking into consideration the fact that data coming from the NAB datasets do differ a lot just think about their origin, the data preprocessing needs to be standardised across each one of the time-series. Besides, the data need to be arranged and structured based on the width of the window used for making the prediction and splitted into **training**, **validation** and **test** sets.

6.1.1 Scaling

All data have been rescaled projecting all data points from the time series in a range between 0 and 1. This was operated by using the function *MinMaxScaler* from the Scikit-learn Python package [63].

6.1.2 Window Size

Following the guide lines from the original paper from Numenta [59], the windows size has been set to be equal to 10% of the time-series length, meaning the BAeCon can use an array of that length as input in order to cast a prediction about the future points.

6.1.3 Data Split

Each time-series has been split into three sub-series in order to train, validate and test the algorithm, respectively 70%, 15% and 15% of the time-series length. Notice that the training and validation sets are not allowed to contain any anomaly. That is because the goal is to make the network learn a clean representation of the time-series so that it will be able to recognise variation when it will deal with anomalies.

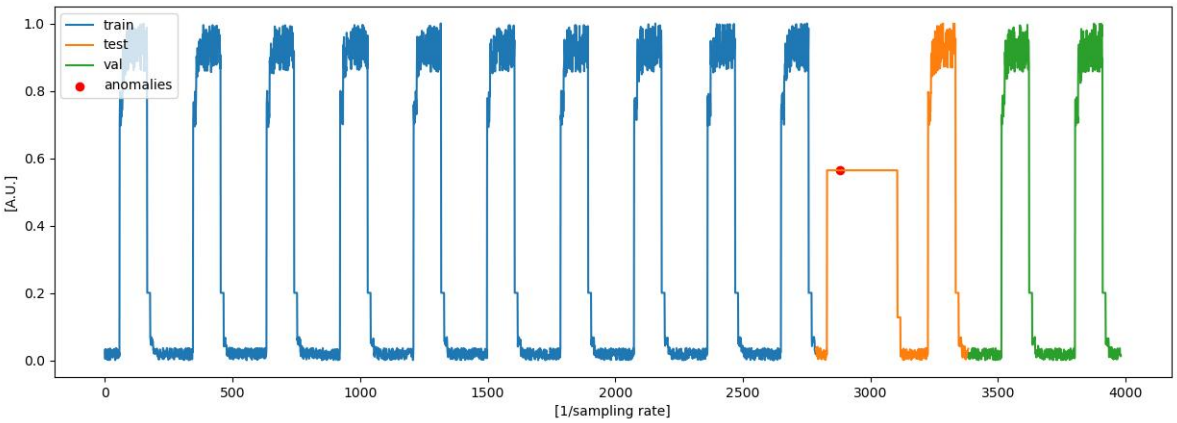


Figure 6.1: Time series split into training, validation and testing sets. Notice the fact that the anomaly is contained only within the test subset of data.

6.2 Autoencoder Training

In order to train the model, the data have been arranged into batch of 128 sets (input points within the window, and target output) and the procedure is set to last up to 150 epochs: if the training loss function were to reach convergence, with a tolerance of $1e-6$, the procedure is stopped. At the end of each epoch the model is tested on the validation set and the loss function is evaluated on it in order to check the training convergence and to allow to save the model version that better performed on never seen data. In Fig. 6.3 can be seen 20 randomly sampled training procedures for the Autoencoder: to be noted that for each time-series in the NAB datasets the loss functions reach convergence.
In order to train the Autoencoder the following parameters have been set:

- **Loss function** - Mean Squared Error (MSE).
- **Optimizer** - Adam [64] with learning rate equal to $1e-3$.
- **MC dropout probability p** - has originally been set to 0.5. The algorithm converge for a wide range o values, in the 0 to 1 interval, but as will be demonstrated in the following sections, it role is to allow the user to adapts the final confidence interval for the final prediction.

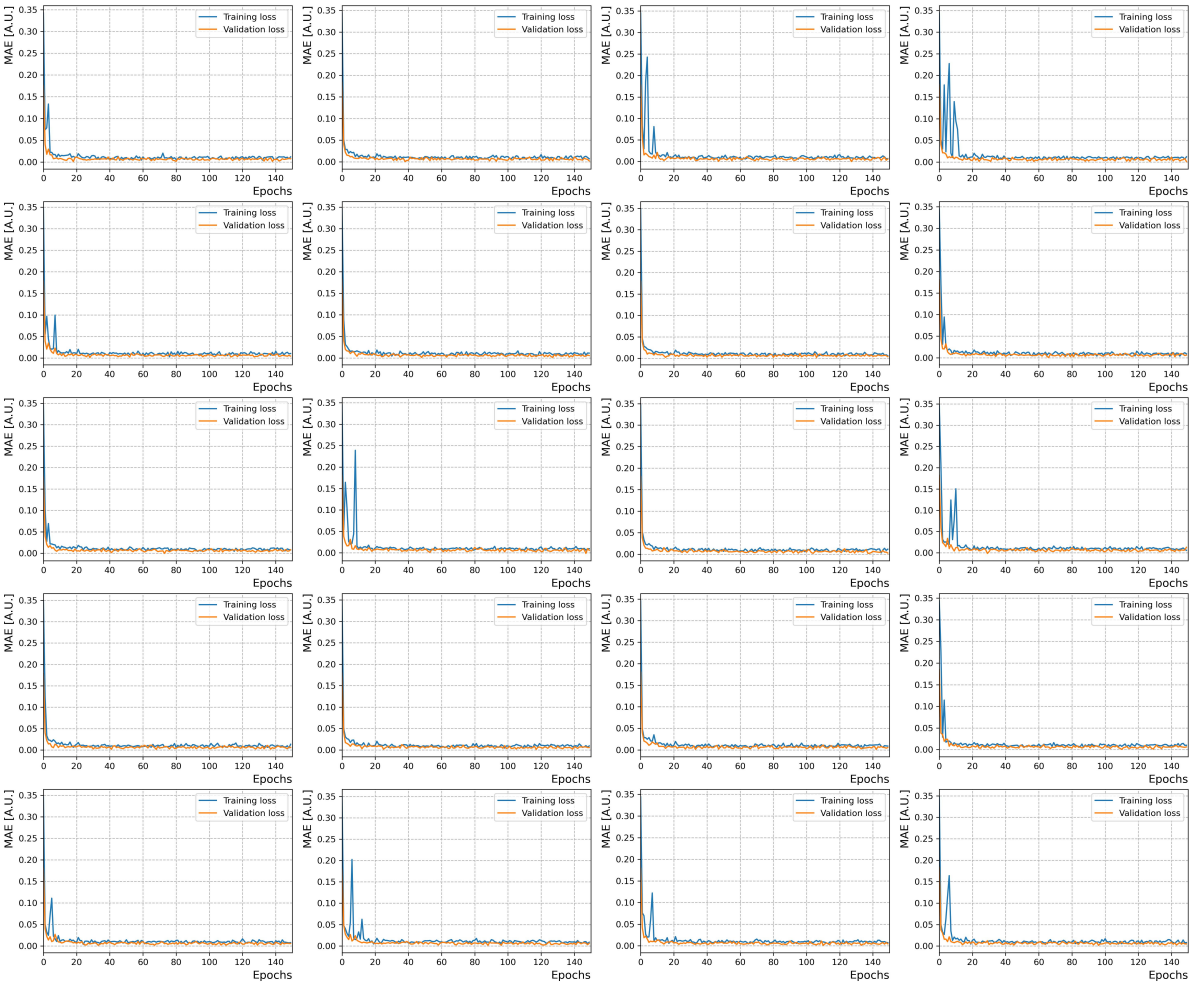


Figure 6.2: Training procedure for 20 randomly sampled NAB time-series

6.3 Predictor Training

Once the Autoencoder has been trained, its decoder part can be detached and connect the encoder to the CNN which will act as a predictor. It is important to note that MC dropout is applied at this new part of the network.
The procedure remains the same as the one for the autoencoder.

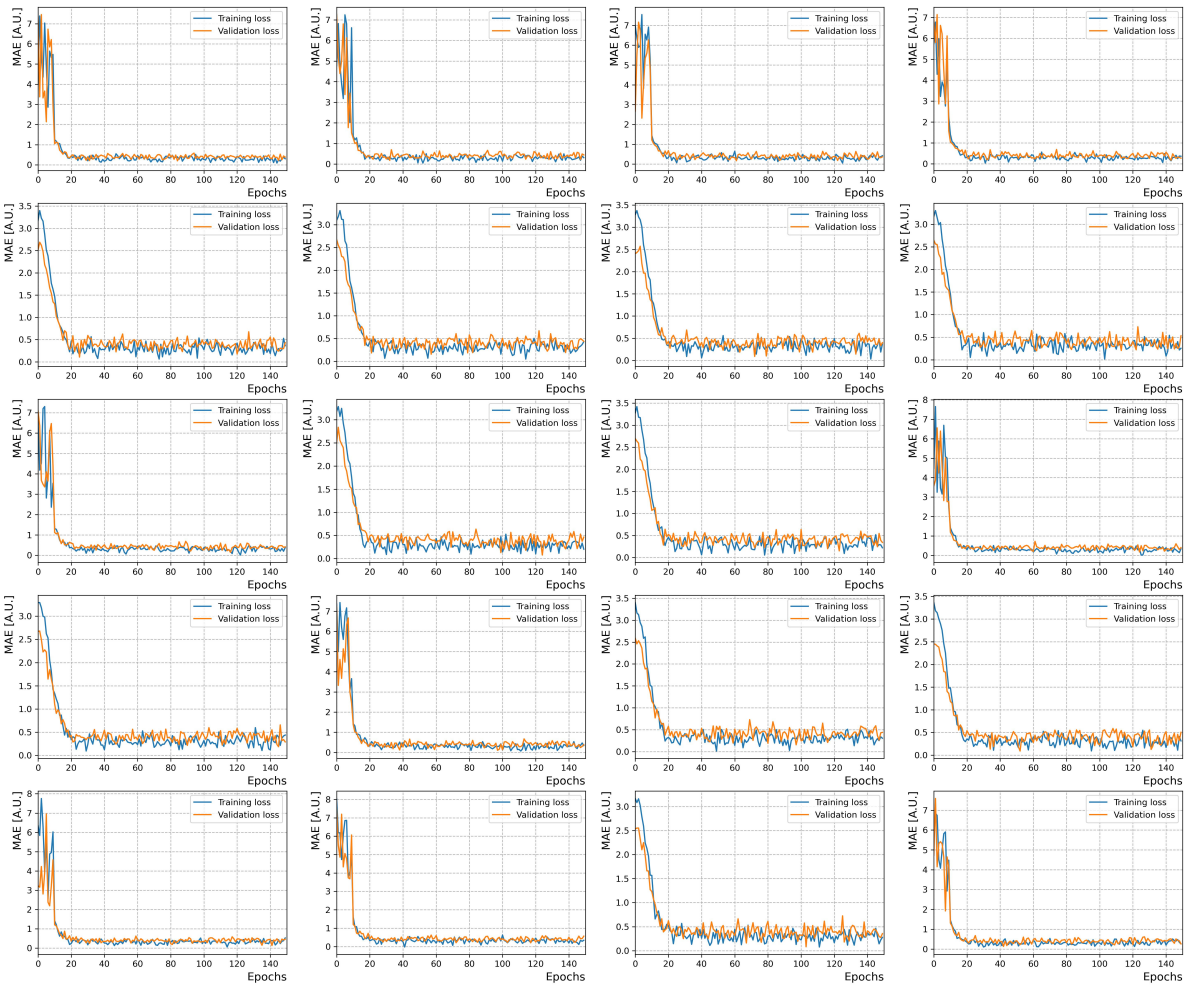


Figure 6.3: Training procedure for 20 randomly sampled NAB time-series with the CNN attached to the Encoder.

6.4 MC Dropout and p parameter

Has already discussed on in chapter 3, MC dropout is used in order to introduce stochasticity into the model prediction so that it is possible to build a statistic for the prediction hence defining a confidence interval for anomaly detection. Intuitively, as the value of p increase the forecast made for a given data window become more and more variable, broadening its possible value distribution.

This as important repercussion over the computation as the confidence intervals: since they are calculated from the standard variation of the forecast set, the wider is the prediction distribution the more coverage the uncertainty will have over the data broadening the confidence intervals.

This behaviour has been tested for 20 randomly sampled time series coming from the NAB dataset: fixing the input data feed to the network the forecast has been repeated 100 times for different values of p, in particular 0.10, 0.25, 0.50, 0.75. Has expected, upon inspection of the prediction distribution it can be verified that the tails do in fact become more and

more prominent as p increases, as shown in Fig. 6.4.

The final score for the algorithm will be computed using the value of p yielding the best results. In order to look for such value, the average precision and recall of the algorithm as been computed over the different domains of the NAB time series for different values of p: the best results were achieved by $p = 0.3$ as can be seen in Fig. 6.5.

Construction the confidence intervals in this way, when computing the average data coverage across all entire time series for all the ones in the data set, the result is around 94%, meaning that the algorithm is accepting data as not an anomaly with the confidence probability.

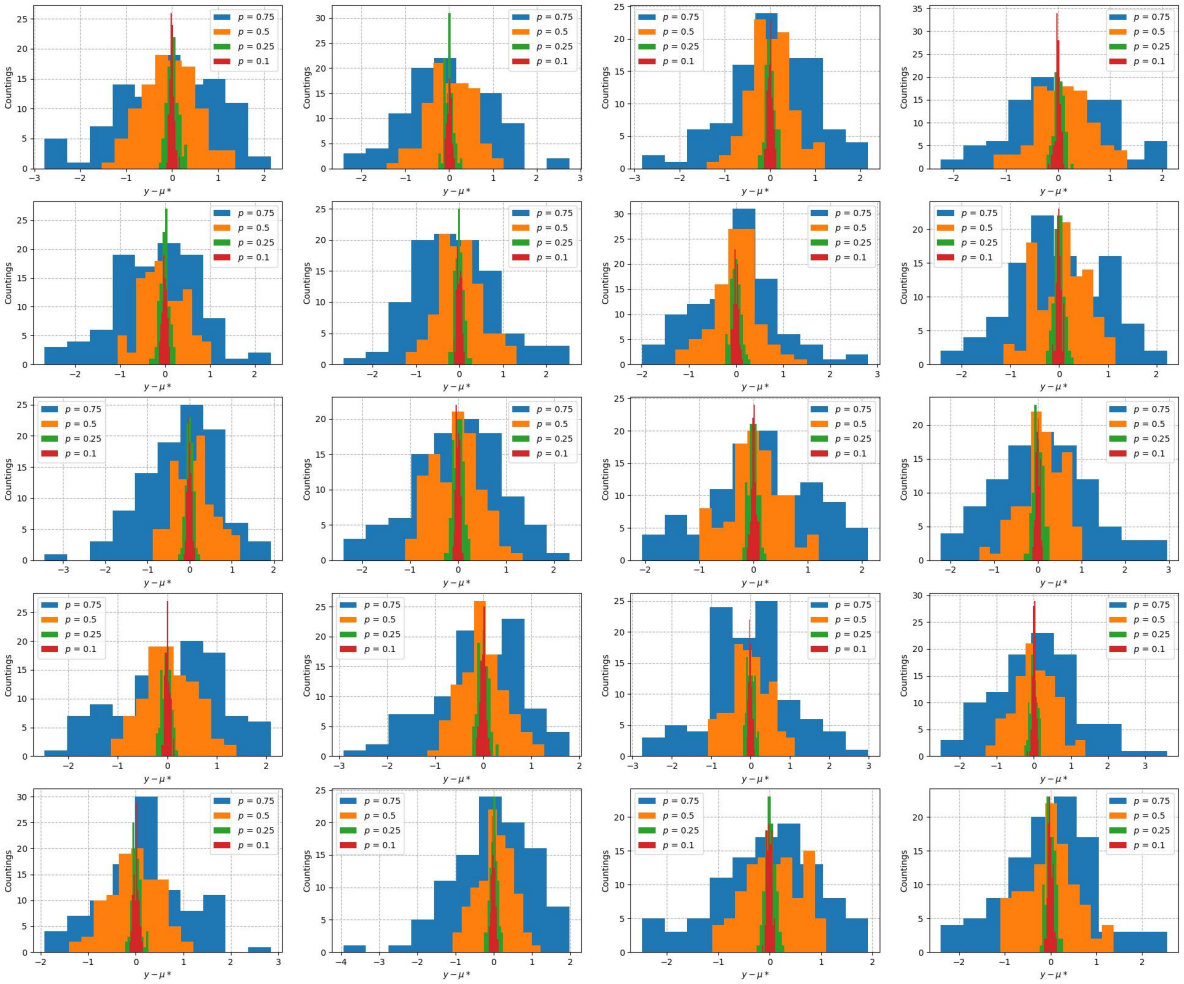


Figure 6.4: For 20 randomly sampled time series from the NAB dataset BAeCon has been trained and given a fixed input from the respective test dataset the forecast has been performed 100 times with different setting for p. On the y axis are represented the counting for the histograms while on the x axis is represented each prediction shifted by the set average value.

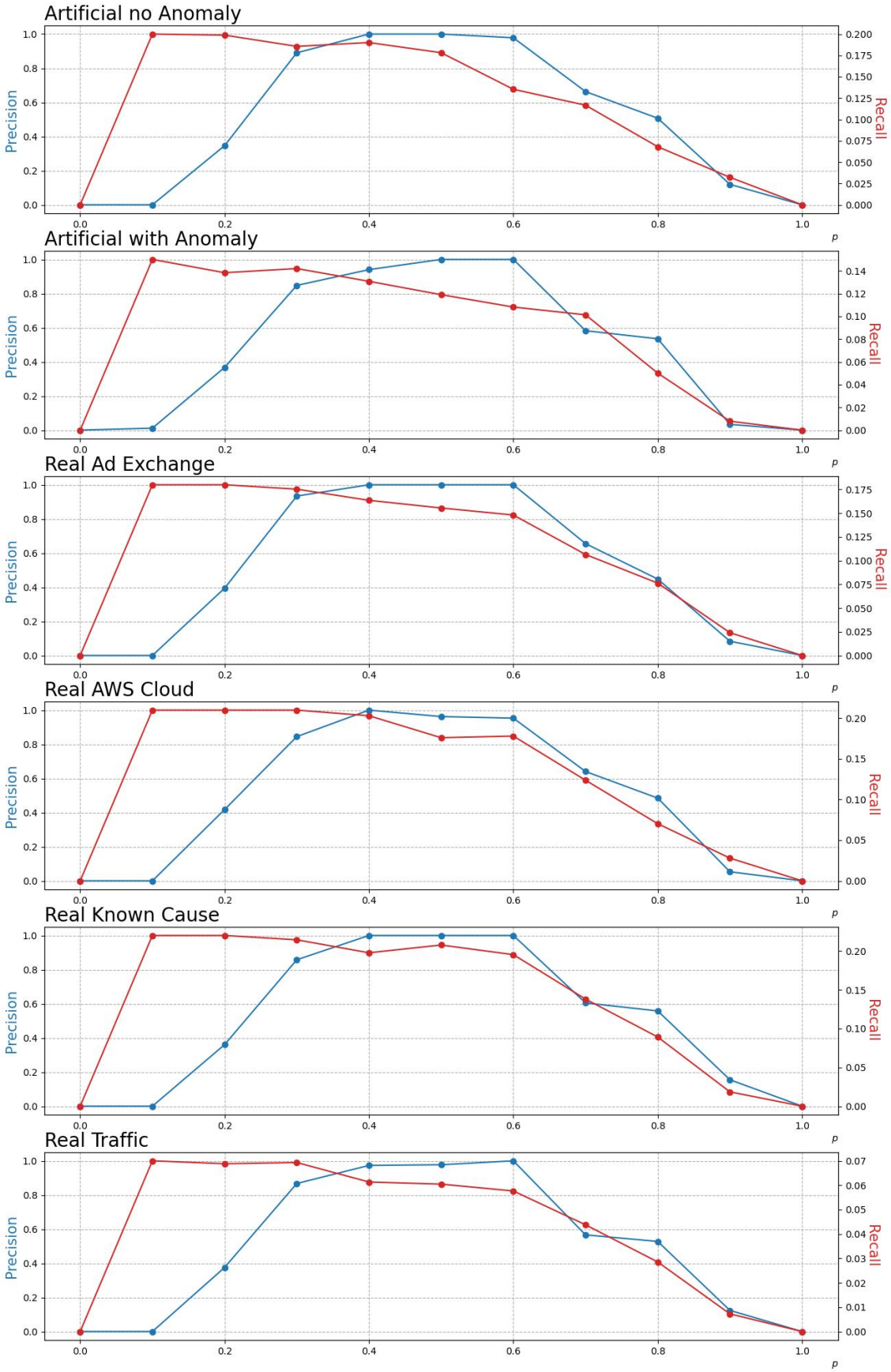


Figure 6.5: Precision - Recall trade-off: in blue, on the left y axis, there is the precision computed as averages over the different domains with different values of p. On the right y axis in red, can be found the recall.

6.5 Results

In this section are presented the results scored by BAeCon on the entire NAB dataset clustered by the different domain to which belongs the time series. The F-score is compared with the one coming from different anomaly detection algorithms such as: Twitter's Anomaly Detection (Twitter ADVec), context OSE, Skyline, Numenta, Multinomial Relative Entropy [65], Bayes changepoint detection [66], EXPoSE [67], simple sliding threshold and DeepAnt [10].

As can be seen from table 6.1, BAeCon is the algorithm that yield the best average F-score across all the domain of the NAB dataset. That is probably due to the average low recalls that the other algorithms yield, as shown in [10], meaning that in general all of this models misclassify anomalies as normal behaving data thus resulting in a high number of false negatives. Even when considering the precision metric BAeCon score on average better better than its competitor, meaning that most of the registered anomalies are in fact abnormal data and the number of false positives is low.

	Bayes Changepoint	Context OSE	EXPoSE	HTM JAVA	KNN CAD	Numenta	NumentaTM	Relative Entropy	Skyline	Twitter ADVec	WindowedGaussian	DeepAnt	BAeCon
Artificial no Anomaly	0	0	0	0	0	0	0	0	0	0	0	0	0
Artificial with Anomaly	0.009	0.004	0.004	0.007	0.003	0.012	0.007	0.021	0.043	0.017	0.013	0.156	**0.307**
Real Ad Exchange	0.018	0.022	0.005	0.034	0.024	0.040	0.035	0.024	0.005	0.018	0.026	0.132	**0.243**
Real AWS Cloud Watch	0.006	0.007	0.015	0.018	0.017	0.018	0.018	0.053	0.013	0.053	0.013	0.146	**0.295**
Real Known Cause	0.007	0.005	0.005	0.013	0.008	0.018	0.015	0.048	0.008	0.017	0.006	0.200	**0.336**
Real Traffic	0.012	0.02	0.011	0.032	0.013	0.033	0.036	0.033	0.091	0.020	0.045	0.223	**0.343**
Real Tweets	0.003	0.003	0.003	0.010	0.004	0.009	0.010	0.006	0.035	0.018	0.026	0.075	**0.128**

Table 6.1: Here are reported the average F-scores for each domain of the NAB dataset, achieved by the reviewed algorithms. The score in bold characters are the best of the category.

Chapter 7

Conclusion and Future Works

Given the experiments conducted on widely spread data used across the literature, this works successfully show the effectiveness of BAeCon which is able to leverages the strengths of both LSTMs and CNNs to capture temporal dependencies and extract complex features from time series data. The results demonstrate that the Autoencoder with LSTM and CNN layers improves the precision and recall of anomaly detection. Furthermore, the model exhibits robustness in handling noise and variability in time series data showcasing its potential for real-world applications given that most of the time series used in the proposed experiments are not artificial. Furthermore, this work demonstrate the effectiveness of Bayesian inference in the context of anomaly detection, allowing to construct a way of discriminating anomalies in a scalable and adaptable way.

In the future would be crucial to expand the experimentation to wider and more divers datasets, tackling the world of multidimensional time series in order to take full advantage of the autoencorder structure and the convolutional layers.

Furthermore, a promising direction to follow would be testing different dimensions and scales for the model, in order to gauge if it is possible to further imporve the results by adjusting the architecture.

Bibliography

[1] Ane Blázquez-García et al. "A review on outlier/anomaly detection in time series data". In: *ACM Computing Surveys (CSUR)* 54.3 (2021), pp. 1–33.

[2] Eamonn J Keogh. "Mining time series data". In: *Wiley StatsRef: Statistics Reference Online* (2014).

[3] Philippe Esling and Carlos Agon. "Time-series data mining". In: *ACM Computing Surveys (CSUR)* 45.1 (2012), pp. 1–34.

[4] Tak-chung Fu. "A review on time series data mining". In: *Engineering Applications of Artificial Intelligence* 24.1 (2011), pp. 164–181.

[5] Charu C Aggarwal. *Outlier analysis second edition*. 2016.

[6] Jordan Hochenbaum, Owen S Vallis, and Arun Kejariwal. "Automatic anomaly detection in the cloud via statistical learning". In: *arXiv preprint arXiv:1704.07706* (2017).

[7] Hermine N Akouemo and Richard J Povinelli. "Time series outlier detection and imputation". In: *2014 IEEE PES General Meeting— Conference & Exposition*. IEEE. 2014, pp. 1–5.

[8] Hermine N Akouemo and Richard J Povinelli. "Probabilistic anomaly detection in natural gas time series data". In: *International Journal of Forecasting* 32.3 (2016), pp. 948–956.

[9] Hermine N Akouemo and Richard J Povinelli. "Data improving in time series using ARX and ANN models". In: *IEEE Transactions on Power Systems* 32.5 (2017), pp. 3352–3359.

[10] Mohsin Munir et al. "DeepAnT: A deep learning approach for unsupervised anomaly detection in time series". In: *Ieee Access* 7 (2018), pp. 1991–2005.

[11] David J Hill and Barbara S Minsker. "Anomaly detection in streaming environmental sensor data: A data-driven modeling approach". In: *Environmental Modelling & Software* 25.9 (2010), pp. 1014–1022.

[12] Yang Zhang et al. "Statistics-based outlier detection for wireless sensor networks". In: *International Journal of Geographical Information Science* 26.8 (2012), pp. 1373–1392.

[13] Kyle Hundman et al. “Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 387–395.

[14] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. “Streaming pattern discovery in multiple time-series”. In: (2005).

[15] Pedro Galeano, Daniel Peña, and Ruey S Tsay. “Outlier detection in multivariate time series by projection pursuit”. In: *Journal of the American Statistical Association* 101.474 (2006), pp. 654–669.

[16] Roberto Baragona and Francesco Battaglia. “Outliers detection in multivariate time series by independent component analysis”. In: *Neural computation* 19.7 (2007), pp. 1962–1984.

[17] Hui Lu et al. “An outlier detection algorithm based on cross-correlation analysis for time series dataset”. In: *Ieee Access* 6 (2018), pp. 53593–53610.

[18] Mayu Sakurada and Takehisa Yairi. “Anomaly detection using autoencoders with nonlinear dimensionality reduction”. In: *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*. 2014, pp. 4–11.

[19] Tung Kieu, Bin Yang, and Christian S Jensen. “Outlier detection for multidimensional time series using deep neural networks”. In: *2018 19th IEEE international conference on mobile data management (MDM)*. IEEE. 2018, pp. 125–134.

[20] Haibin Cheng et al. “A robust graph-based algorithm for detection and characterization of anomalies in noisy multivariate time series”. In: *2008 IEEE international conference on data mining workshops*. IEEE. 2008, pp. 349–358.

[21] Haibin Cheng et al. “Detection and characterization of anomalies in multivariate time series”. In: *Proceedings of the 2009 SIAM international conference on data mining*. SIAM. 2009, pp. 413–424.

[22] Zewen Li et al. “A survey of convolutional neural networks: analysis, applications, and prospects”. In: *IEEE transactions on neural networks and learning systems* 33.12 (2021), pp. 6999–7019.

[23] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling”. In: *arXiv preprint arXiv:1803.01271* (2018).

[24] Brett Koonce and Brett Koonce. “ResNet 50”. In: *Convolutional neural networks with swift for tensorflow: image recognition and dataset categorization* (2021), pp. 63–72.

[25] Weicong Kong et al. “Short-term residential load forecasting based on resident behaviour learning”. In: *IEEE Transactions on power systems* 33.1 (2017), pp. 1087–1088.

[26] Stephen Grossberg. “Recurrent neural networks”. In: *Scholarpedia* 8.2 (2013), p. 1888.

[27] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[28] Thomas Fischer and Christopher Krauss. “Deep learning with long short-term memory networks for financial market predictions”. In: *European journal of operational research* 270.2 (2018), pp. 654–669.

[29] Xingjian Shi et al. “Convolutional LSTM network: A machine learning approach for precipitation nowcasting”. In: *Advances in neural information processing systems* 28 (2015).

[30] Xin Wang et al. “LSTM-based short-term load forecasting for building electricity consumption”. In: *2019 IEEE 28th international symposium on industrial electronics (ISIE)*. IEEE. 2019, pp. 1418–1423.

[31] Alexander Etz et al. “How to become a Bayesian in eight easy steps: An annotated reading list”. In: *Psychonomic bulletin & review* 25.1 (2018), pp. 219–234.

[32] Nicholas G Polson and Vadim Sokolov. “Deep learning: A Bayesian perspective”. In: (2017).

[33] Laurent Valentin Jospin et al. “Hands-on Bayesian Neural Networks–a Tutorial for Deep Learning Users”. In: *arXiv e-prints* (2020), arXiv–2007.

[34] Scott Cheng-Hsin Yang et al. “Mitigating belief projection in explainable artificial intelligence via Bayesian teaching”. In: *Scientific reports* 11.1 (2021), p. 9863.

[35] Chuan Guo et al. “On calibration of modern neural networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 1321–1330.

[36] Jeremy Nixon et al. “Measuring calibration in deep learning.” In: *CVPR workshops*. Vol. 2. 7. 2019.

[37] Dan Hendrycks and Kevin Gimpel. “A baseline for detecting misclassified and out-of-distribution examples in neural networks”. In: *arXiv preprint arXiv:1610.02136* (2016).

[38] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[39] David JC MacKay. “A practical Bayesian framework for backpropagation networks”. In: *Neural computation* 4.3 (1992), pp. 448–472.

[40] Pavel Izmailov et al. “What are Bayesian neural network posteriors really like?” In: *International conference on machine learning*. PMLR. 2021, pp. 4629–4640.

[41] Christian P Robert et al. *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Vol. 2. Springer, 2007.

[42] John Mitros and Brian Mac Namee. "On the validity of Bayesian neural networks for uncertainty estimation". In: *arXiv preprint arXiv:1912.01530* (2019).

[43] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. "Being bayesian, even just a bit, fixes overconfidence in relu networks". In: *International conference on machine learning*. PMLR. 2020, pp. 5436–5446.

[44] Yaniv Ovadia et al. "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift". In: *Advances in neural information processing systems* 32 (2019).

[45] Armen Der Kiureghian and Ove Ditlevsen. "Aleatory or epistemic? Does it matter?" In: *Structural safety* 31.2 (2009), pp. 105–112.

[46] Stefan Depeweg et al. "Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning". In: *International conference on machine learning*. PMLR. 2018, pp. 1184–1193.

[47] David H Wolpert. "The lack of a priori distinctions between learning algorithms". In: *Neural computation* 8.7 (1996), pp. 1341–1390.

[48] Lingxue Zhu and Nikolay Laptev. "Deep and confident prediction for time series at uber". In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2017, pp. 103–110.

[49] Wolfram Manzenreiter. "ACCOUNTING FOR MEGA-EVENTS". In: *INTERNATIONAL REVIEW FOR THE SOCIOLOGY OF SPORT* 39.2 (2004), pp. 187–203.

[50] Mohammad Assaad, Romuald Boné, and Hubert Cardot. "A new boosting algorithm for improved time-series forecasting with recurrent neural networks". In: *Information Fusion* 9.1 (2008), pp. 41–55.

[51] Olalekan Ogunmolu et al. "Nonlinear systems identification using deep dynamic neural networks". In: *arXiv preprint arXiv:1610.01439* (2016).

[52] Nikolay Laptev et al. "Time-series extreme event forecasting with neural networks at uber". In: *International conference on machine learning*. Vol. 34. sn. 2017, pp. 1–5.

[53] Alex Kendall and Yarin Gal. "What uncertainties do we need in bayesian deep learning for computer vision?" In: *Advances in neural information processing systems* 30 (2017).

[54] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

[55] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.

[56] Yarin Gal and Zoubin Ghahramani. “A theoretically grounded application of dropout in recurrent neural networks”. In: *Advances in neural information processing systems* 29 (2016).

[57] Yarin Gal et al. “Uncertainty in deep learning”. In: (2016).

[58] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. “Unsupervised learning of video representations using lstms”. In: *International conference on machine learning*. PMLR. 2015, pp. 843–852.

[59] Alexander Lavin and Subutai Ahmad. “Evaluating real-time anomaly detection algorithms–the Numenta anomaly benchmark”. In: *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*. IEEE. 2015, pp. 38–44.

[60] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[61] Nidhi Singh and Craig Olinsky. “Demystifying Numenta anomaly benchmark”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 1570–1577.

[62] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[63] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[64] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[65] Chengwei Wang et al. “Statistical techniques for online anomaly detection in data centers”. In: *12th IFIP/IEEE international symposium on integrated network management (IM 2011) and workshops*. IEEE. 2011, pp. 385–392.

[66] Ryan Prescott Adams and David JC MacKay. “Bayesian online changepoint detection”. In: *arXiv preprint arXiv:0710.3742* (2007).

[67] Markus Schneider, Wolfgang Ertel, and Fabio Ramos. “Expected similarity estimation for large-scale batch and streaming anomaly detection”. In: *Machine Learning* 105 (2016), pp. 305–333.