Unsupervised Anomaly Detection for Humidity Sensor Data

Authors
Dirk Willeboord Baars
Pinyapat Manasboonpermpool

Supervisor
Krzysztof Podgórski

Lund University School of Economics and Management
Master's Thesis of Data Analytics and Business Economics, 15 credits
May, 2022

Abstract

Detecting anomalies in loT sensor devices for humidity is crucial for the maintenance and safety of households and buildings. An Auto-Encoder (AE) can detect anomalies in sensor data. However, the reality is that most sensor data is unlabelled, while it is recommended to use labelled data when training the model for anomaly detection with AE. Therefore together with Sensative AB we developed a Long-Short-Term Memory (LSTM)-AE model that is tested with labelled data and later applied to unlabelled humidity series. We found a detection threshold of 99.5% in the experiment with labelled data is optimal when using LSTM-AE on sensor data. Based on the experiment, a 99.5% threshold is applied to unlabelled humidity series, detecting four collective anomalies in humidity series 1217 and two collective anomalies in humidity series 2595 with LSTM-AE.

Keywords: LSTM, Auto-Encoder, Anomaly Detection, Time series, Deep Learning

Acknowledgements

We thank our supervisor, Krzysztof Podgórski, for his valuable time and constructive feedback. His guidance and contribution were crucial to our process. His inputs and ideas allowed us to explore beyond our prior knowledge from the deep learning module. Furthermore, his kindness and approachableness were truly appreciated. We have learned significantly from him throughout this thesis period.

We also want to thank Sara Moricz for allowing us to do this thesis at Sensative AB. We sincerely appreciate her contribution, guidance, continuous encouragement, and motivation while writing this thesis.

Thank you!

Content

Abbreviations

AE	Auto-Encoder
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
LSTM	Long Short-Term Memory
LSTM-AE	Long Short-Term Memory Auto-Encoder
MAE	Mean Absolute Error
NN	Neural Network
RNN	Recurrent Neural Network

1 Introduction

Organisations leveraging mechanisms in place to detect anomalies fast and quickly take actions gain a competitive edge.

Halstead, Vikraman, and Prasad (2022)

Every day terabytes of data stream into various databases, and millions of services are built on top of it. However, can we trust the data? We might be in the next round of digitalisation where the Internet of Things (IoT) technologies connect the physical world to the digital one via various sensors. For example, it holds the promise that the elderly can be guarded against fall injuries via motion sensors. Now more than ever, the data has to be trustworthy and recent research on how best to screen incoming data for anomalous events is very active. An anomaly might be that a sensor is tampered with and starts reporting very high values. Finding good solutions to detect anomalies unlock tremendous potential values (Peña et al. 2016).

Humidity Importance

In this thesis, we help a company, Sensative AB, to develop a method detecting anomalies. Sensative AB sells an integration platform, meaning that Sensative ABs customers put the data originating from sensors from various manufacturers in one place. We choose to focus on humidity data as a test case. We do this because humidity impacts air quality, human health, and work efficiency (Kim, Chu & Shin, 2014). Previously, constant air measurement was challenging and expensive, but nowadays, humidity can be measured with the help of sensors (Shi et al. 2013). Besides human health, insights into humidity levels can prevent undesirable outcomes or catastrophic breaks, e.g., pipeline leakages. Sensors are located close to pipes and heat insulation where the humidity level can detect the leak rate (Saha & Sengupta, 2007). The potential adverse outcomes of the failure to report anomalous behaviour are unmeasurable. High humidity can cause mould forming (Gravesen et al. 1999; Chen et al. 2004), air leakage (Li et al. 2020), and malfunctioning of building materials (Zhou & Chen, 2016), all having effects on the lifespan of a building.

Anomaly Definition

Anomalous events need to be flagged to the user requiring immediate action, preventing further damage. Flagging an anomaly is usually done via anomaly detection (Munir et al. 2018; Al-amri et al. 2021), and there are multiple definitions for anomalies. An anomaly can be: “An observation which deviates so significantly from other observations to arouse suspicion that it was generated by a different mechanism” Hawkins (1980). According to Barnett & Lewis (1984), “An outlier is an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data.” Cook, Mısırlı and Fan (2020) defined an anomaly for IoT as: “The measurable consequences of an unexpected change in the state of a system which is outside of its local or global norm.” Stating the majority of data is non-anomalous and can change over time. Chandola, Banerjee, and Kumar (2009) present an overview of anomalies detected in data: First, an anomaly can be a single outlier, called a point anomaly. A point anomaly in a time series returns to its ‘non-anomalous state’ within a few observations. These point anomalies can be statistical noise, fault in the sensor system, or an unexplained change in the operating system. Second, a contextual anomaly depends on the context.

For example, a high temperature that is non-anomalous in summer can be anomalous in winter. Finally, a collective anomaly is when data points are anomalous regarding the rest of the dataset. An essential quality of a collective anomaly is that the data points must be related. The individual points are not an anomaly themselves, but together they form a collective anomaly. Take an electrocardiogram (ECG) depicting a heartbeat with sensors, whereby arrhythmias are a collective anomaly (Thil et al. 2021). Likewise, a rapid variation in temperature caused by a fire is a collective anomaly instead of a multiple-point anomaly (Egilmez & Ortega, 2014). Since humidity is sequential data, this paper will find anomalies on a point and collective basis.

Categories of Anomaly Detection Methods

Subsequently, there are three categories of anomaly detection methods. First, supervised anomaly detection applies to labelled data, whereby the data classifies as regular or irregular. Previous work applied Support Vector Machine (SVM), discriminating classifiers, e.g., to predict suspicious patterns of attacks in an IoT system. Another method is Naive Bayes, applied to intrusion detection and classifying anomalous patterns in, e.g., smart homes. Also, K Nearest Neighbour (KNN) is used to construct similarities and detect unusual attacks, such as cybercrimes in an industrial system (Al-amri et al. 2021). Second, semi-supervised anomaly detection trains a model with non-anomalous samples, providing scores to classify anomalies (Yao et al. 2019). Third, unsupervised anomaly detection applies to unlabelled data. This type of detection can be complex due to the lack of labelled data to train a model. To use unlabeled data, one assumes few anomalies, allowing the model to learn information (Yao et al. 2019). Real-world sensor data is usually large and unlabeled, and convolutional supervised anomaly detection methods struggle to handle high amounts of unlabeled data (Ergen and Kozat, 2019). Deep learning (DL) methods can work with unsupervised-based anomaly detection methods and train them at a low cost (Kwon et al. 2019).

Deep Learning Models

Simply putting a fixed-level threshold does not apply to time-series data to detect anomalies. First, based on the context, an anomaly can change over time. Second, time series considers the previous data point due to its sequential order, which can be applied in DL methods (Chandola, Banerjee, & Kumar, 2009). DL models can have a generative (e.g., Auto-Encoder (AE)), a descriptive (e.g., Recurrent Neural Network (RNN)), or a hybrid approach (Kingma & Welling, 2019). A hybrid approach can be a Convolutional Neural Network (CNN) or an RNN combined with an AE. However, RNN performs better in real-world applications than conventional anomaly detection methods (Pang et al. 2021). An AE is one of the unsupervised neural networks (NN)s that perform well in anomaly detection. The main goal of AE is to reconstruct input data into symmetrical output data by learning to narrow the dimension of the data. In addition, the reconstruction allows the measurement of the error in the gap between the input and the output component (Chollet, 2016). RNN-AE has three approaches, the Gated Recurrent Units (GRU)-AE, the Long Short-Term Memory (LSTM)-AE, and the Basic RNN-AE, whereby the last AE is invalid for applying to time series data due to the vanishing gradient problem. On the other hand, the GRU-AE and LSTM-AE solve the vanishing gradient problem. When comparing the latter, the LSTM-AE has higher accuracy than GRU-AE when using long sequences, because LSTM-AE can select more parameters (Jun et al. 2020). Since IoT sensors generate a vast amount of time series data; LSTM-AE applies in this paper.

Importance

Previous research focuses on LSTM-AE for labelled data (Li & He, 2019; Kang et al. 2021; Lui et al. 2022). However, no previous paper discusses the detection of anomalies in training unlabeled data. This is surprising because, as mentioned before, sensor data is often unlabeled. Therefore in an attempt to fill this theoretic void, the objective of this study is to apply an LSTM-AE to unlabeled data. Furthermore, LSTM-AE will be applied to humidity data, an important concept affecting human health and also the life expectancy of a building. Therefore, this thesis will provide insights into detecting anomalies in humidity data. So far, no previous work has applied LSTM-AE to humidity data, making this study the first one to present.

Contribution

This paper will apply LSTM-AE to unlabelled humidity series provided by Sensative AB. This method is applied because LSTM-AE allows the processing of sequential unlabelled time series data. Also, the LSMT-AE model will be tested with labelled data. This data, retrieved from Yahoo, is constructed for anomaly detection. After testing the model, an optimal threshold, based on the reconstruction error, will be chosen to see which sensitivity level fits best for anomaly detection of sensor data. Eventually, the model tested on the labelled benchmark data will be applied to the unlabelled humidity series. Based on this, the main research question is:

"How can LSTM-AE apply to unlabelled humidity series?"

Next, this paper will look into an optimal level of a reconstruction loss threshold to check anomalies within unlabelled data. Therefore the sub-question is:

"What is a proper threshold for anomaly detection in unlabelled sensor data?"

Results

This paper tested an LSTM-AE against benchmark labelled data. In this testing, the optimal Mean Absolute Error (MAE) loss threshold came down to 99.5% for the highest efficiency of an early warning system. In addition, the model detected four collective anomalies for humidity series 1217 and two collective anomalies for humidity series 2595, whereby these anomaly points had a substantial deviation in the test MAE loss from the train MAE loss.

Limitations

First of all, a limitation of this study is applying an unlabelled dataset to train an LSTM-AE. Beforehand we could only assume that the training data was non-anomalous, limiting the conclusions of the results. Second, it was not known what type of anomalies were present in the humidity series, e.g. point anomaly or collective anomalies, which is crucial for knowing the amount to be detected in the anomaly detection method.

Takeaways

This thesis applies an LSTM NN combined with a "plain" AE. There are more autoencoders available, whereby a variational AE (VAE) is an extension whereby the output space has a more regular shape based on the mean and the data variation. In this way, anomalous data can also be detected in how much they deviate in oscillations (Han et al. 2021). This thesis focuses on a "plain" AE because a VAE requires more programming than is feasible within a thesis project. The version we have tried detects approximately 99 percent of the anomalies in the benchmark labelled dataset, leading us to suggest that although it might be good to try the variational type, it might not be strictly necessary.

Organisation of the Paper

The remainder of this paper is organised as follows. The following section presents anomaly detection methods, reviewing prominent literature on anomaly detection, specifically on LSTM-AE. After that, the methodological foundations explain NN, AE and LSTM followed by an explanation of our applied model LSTM-AE. The fourth section applies the LSTM-AE model to the labelled benchmark data and afterwards to the unlabelled humidity series in section five. Finally, the conclusion and suggestions for future research are discussed.

2 Anomaly Detection Methods

2.1 Anomaly Detection and Deep Learning

Previous research from the earliest development covered the area of classical Machine Learning (ML). These methods include Clustering-Based, Isolation Forests, Support Vector Machines, and Gaussian Distribution. However, the challenge occurred when outlier points did not interact with the prior points of the sequential nature (Provotar, Yaroslav & Maksym, 2020). Previous research showed an attempt to construct standard patterns to detect anomalies with unlabelled data. For example, K Nearest-Neighbours (KNNs) were used to find anomaly scores based on each distance point to its kth-position neighbour (Angiulli & Pizzuti, 2002). From the Machine Learning perspective, Principal Component Analysis (PCA) was introduced as the tool to reform low-dimensional data that can assist in capturing the variance in the data to detect the reconstruction error of such projection (Shyu et al. 2003). Feature Bagging (FB) demonstrated aggregated scores to identify anomalies in small dataset samples (Lazarevic & Kumar, 2005).

However, in recent years, the methodology's focus has been shifted toward the Deep Learning (DL)-based implications and has been successfully applied to detecting anomalies. Mohammadi et al. (2018) conducted a Google trend research, showing that DL is more popular than the topics mentioned above, with a significant rise since 2006. The DL models render capabilities to tackle challenges of undefined patterns, noises from the input data, the increasing length in time series, and the time series' property of dynamic involvement in the form of convex or non-linear data (Chalapathy & Chawla, 2019).

DL algorithms reveal better achievements in anomaly detection. Most techniques are related to lowering the original data's dimension to separate the anomalies from the non-anomalous data. The squeezed dimensional information is closely the same as the original data. The reconstruction of the data in which we expect the selective nature without unimportant attributes and noise. Thus, reconstruction error or loss is the subject of defining the anomalies in scores in general. Reconstruction errors are applied in AE. AE's property of encoding and decoding to reconstruct the output data samples rendering the same as the input allowing to form the reconstruction errors as anomaly scores when exceeding the threshold (Aggarwal, 2015). Due to the continuous development of AI Technology, extracting data features can be easily performed using an AE (Xie et al. 2020). An efficient improvement of some soft sensors has appeared after the implication of AE to extract the data features and reduce the data dimensionality (Yao & Ge, 2017). Deep Autoencoder Gaussian Model (DAGM), the combination of Deep Autoencoders and Gaussian Mixture, was introduced to lower the data dimension and construct the reconstruction errors in late 2018 (Zong et al. 2018). Research has begun to explore new areas of DL methods in hybridising the two models to detect time series data more efficiently.

Further exploration of other algorithms, such as CNN, appears to be popular for data detection. CNN is another type of AE widely applied in various computer science and natural language processing domains, due to its parameter efficiency (Munir et al. 2019). In anomaly detection, CNN is used to predict the upcoming timestamp in time series on the same horizon of the temporal frame (Munir et al. 2019). Therefore, the window time-series application is the primary context for predicting the new values passed on to the anomaly detector. In the simplest form, convolution is a mathematical operation on two functions of real-valued arguments to produce a third function.

2.2 LSTM-AE and Anomaly Detection

Previous research showed that LSTM is a fit for data with long-term dependencies. Wang et al. (2020) applied LSTM on time-series sensor data of a maintenance system. They were successful in analysing the non-linear relationship. The LSTM model applied for anomaly detection on MIT-BIH arrhythmia database renders relatively high accuracy, precision and recall at 82.10%, 78.40% and 84.79% respectively (Gómez et al. 2020). However, this method is mostly used in data clustering and can be insufficient in extracting temporal features, which can be achieved by using an AE (Chen et al. 2021). Throughout the years, several attempts have been made to apply LSTM-AE on detecting anomalies.

Li and He (2019) introduced an LSTM-AE to detect video content anomalies. The video content contains temporal and spatial information, whereby an LSTM network in the network processes the temporal information. The spatial information was processed by breaking the video fragment into multiple parts and extracting multiple images from each segment. The LSTM-AE created a latent representation of the sequential data and minimised the distance between the sequential input data and the representation data. The proposed method is superior to previous methods, having an accuracy of 94.16% and a precision of 90.31%. Kang et al. (2021) also successfully applied an LSTM-AE to detect anomalies in the Brake Operating System of metro trains. They trained the model using training that did not contain any anomalies. Afterward, they defined a threshold in the training data set using the maximum function of MAE loss. The model was successfully implemented, outperforming other AE implementations, having an accuracy of 94.44% and a precision of 97.94%. Likewise, Lui et al. (2022) researched time-series Electrocardiogram (ECG) data to diagnose heart diseases. Instead of only using a plain AE, Lui et al. (2022) introduced an LSTM-AE that cooperates with ECG data more efficiently, making better classifications for the arrhythmia effects. The method made it possible to input ECG signals without preprocessing directly. The model is adequate, having an accuracy of 98.57% and precision of 97.55%, having higher accuracy and precision than the previously mentioned research by Li and He (2019) and Kang et al. (2021). Furthermore, Chen et al. (2021) showed that LSTM can be applied to uncertain, volatile data. In their research, an LSTM-AE was applied to wind-turbine data, which are used in a harsh and changing environment, e.g., severe storms. They also showed that LSTM-AE, the hybrid model, outperforms regular ANN, AE, or LSTM models in accuracy measures. Next, methodological foundations for the NN and AE are discussed.

3 Methodological Foundations

This section will discuss several NN models and their features to clarify the papers' models and introduce our proposed methodology.

3.1 Introduction to Neural Network Models

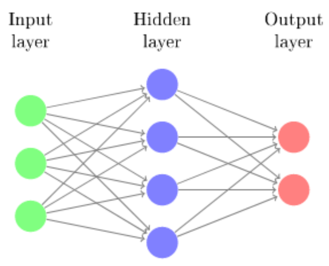


Figure 1: An ANN with One Hidden Layer

Figure 1 shows a visualisation of an Artificial Neural Network (ANN). An ANN is a ‘neural network’ because the neurons are a network of interconnected perceptrons or nodes. The input of the previous layer activates the nodes, similar to human neurons firing when reaching the threshold by activation (Goodfellow et al. 2016). In-depth information about human neurons is available in Appendix A. A node is written as follows:

$$\varphi(w_o + \sum_{i=1}^{m} x_i w_i) \tag{1}$$

In the node, x_i resembles the ith input, φ the activation function, w_o the bias term and $(\sum_{i=1}^{m} x_i w_i)$ the weighted sum of inputs. The node computes the weighted sum of the input vector, adds the bias term and applies a function, getting a number as an output. The NN output is the model's prediction, and the weights W are the trained parts (Goodfellow et al. 2016). Figure 1 shows that ANN has three layers: An input layer, an output layer, and one or more hidden layers. The nodes in the layer are visualised by circles, whereby the number of nodes matches the number of input values. First, the input layer is fed with external data, then the hidden layer can change the number of nodes, and then final output layer is the result of the algorithm. (Goodfellow et al. 2016). For example, Figure 1 depicts an ANN whereby the blue second layer is the hidden layer and has four nodes. The last layer is the output layer, containing two nodes, similar to the number of dependent variables of the model (Jain, Mao, & Mohuiddin, 1996).

Feedforward models are the most common NN architectures. Tabular data can be applied to feedforward NN to understand non-linear relationships between input and output. The name feedforward comes from the feedforward propagation generation, whereby the input layer passes the data to the next layer, giving it to the next until it reaches the output (Goodfellow et al. 2016). Furthermore, NN applies several concepts, namely activation functions, backpropagation and epochs which are explained in Appendix B, C and D.

Loss Functions and Optimization Algorithms

A minimised loss function is used to find the weights. For example, a loss function *J(W)* of a NN computed on *n* data points can be written as:

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \lambda(f(x_i, W), y_i) \tag{2}$$

Here, W, is the matrix of weights, x_i is a vector of the ith data point, y_i is the dependent variable, $f(x_i, W)$ is the formula notation and λ is the loss function (Friedman, Hastie & Tibshrani, 2001). The loss function is based on the problem of the NN. A popular loss function is the Mean Absolute Error (MAE) loss, written as:

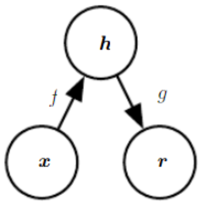
$$Loss(MAE) = \frac{\sum_{i=1}^{n} |x_i - \hat{x}_i|}{n} \tag{3}$$

In the MAE loss function, n is the number of samples, x_i represents the original input, while $\hat{x}_i$ is the output. Other popular loss functions are Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) (Wei et al. 2022).

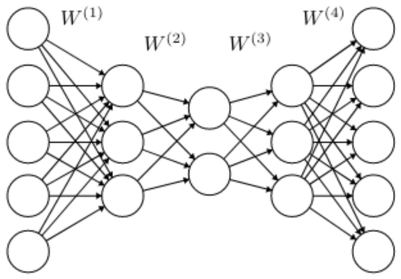
As stated in Appendix C, backpropagation updates the weights in a backward matter. The magnitude of the updates depend on the Learning Rate (LR), η. Several optimizers have a fixed LR, such as Gradient Descent (GD) and Batch Gradient Descent (BGD). However, optimizers also apply an LR that can change, such as Adam and RSMprop. According to Ruder (2016), Adam is an advanced optimizer, quickly finding the optimal weights. Starting at an initial learning rate, Adam samples a subset of batches (Appendix D), estimating the first and second moments of gradients through backpropagation and updates the weights accordingly. Then Adam adapts the learning rate, computing new learning rates for each parameter at each step, considering the results of the previous weight updates when computing new ones. Through this process, Adam avoids local minima and converges fast. Adam only needs an initial learning rate, which will be tuned. Furthermore, Adam applies for training LSTM NNs since it is computationally efficient and requires little memory, making it suitable for large datasets (Braei & Wagner, 2020; Chen et al. 2021; Kingma & Ba, 2014). Since Adam is one of the best choices to use as an optimization algorithm, it will be applied in the model of this paper.

3.2 Introduction to Auto-Encoder

An Autoencoder (AE) is an ANN, categorised as generative unsupervised DL. It is based on the compression of the input values and then reconstructs the input values (decompress), so the error between the input vector, x, and the network output, $\hat{x}$, is small. These operations of compressing and decompressing are depicted as encoding and decoding (Chollet, 2016). Figure 2 shows an AE of three layers, also known as a multilayered AE. Figure 2 shows that the encoding part exists of the first and second layers, depicted with the function $h = f(x)$. It compresses representations such as bottleneck features or latent input variables into low-dimensional data. The decoding layer exists out of the second and third layer, generating the construction, depicted as $r = g(h)$. The decoder network artificially reconstructs targets from the compressed representations into its own decoder's output with the same encoder input size. The second layer is part of both the encoder and decoder. This structure is a feed forward network because the objective is to reproduce the input data on the output layer (Charte et al. 2018; Goodfellow et al. 2016).



I. A simple AE with an encoder (f) and decoder (g)



II. AE with a NN structure: W denotes weight matrices

Figure 2: View of I. Simple AE and II. AE with a NN structure retrieved from Charte el al. (2018)

Autoencoders have symmetrical structures, whereby the encoder and the decoder have similar layers. However, there is a distinction depending on the dimension of the encoding layer. An AE copies the input to output, producing results through the hidden layer. The dimensions in the hidden layer influence the output. For instance, the hidden layer can change the level of dimensions, forcing the model to choose the important features to decode later. The learning process for the hidden layer via the minimization of the loss function is depicted as:

$$L(x,\ g(f(x))). \tag{4}$$

The loss function penalises for the dissimilarities of x. One must choose the dimensions carefully, because when the dimensions in the hidden layer are lower than x, the hidden layer can force the model to select too few features, leading to an under-complete AE. On the other hand, when allowing too much capacity, the decoder will copy the data without knowing extra information about the distribution, leading to overfitting and failing to learn from the data (Goodfellow et al. 2016; Charte el al. 2018).

3.3 Neural Networks for Time-Dependent Data

Recurrent Neural Networks

When using feedforward NNs or other machine learning models, the data is often assumed to be independent and identically distributed in time (Koekkoek & Booltink, 1999). However, the data in this paper concerns sequential, time-series data; therefore, this assumption does not hold.

To deal with sequential data, RNN introduces memory into a NN, allowing time-series analysis (Mohajerin & Waslander, 2017). As depicted in Figure 3, RNN takes d_x-dimensional vectors $x = (x_1,\ x_1,\ x_t)$, and processes them sequentially. A hidden state, $h_t \in R^{dh}$, is created with every memory update for each input vector x. Unlike feedforward models such as CNN, the RNN has a feedback mechanism using the information of the previous output as the next input for the sequence (Braei & Wagner, 2020). The main feature of recurrent networks is the feedback mechanism between the hidden units across time that stabilises a recurrent mechanism. In its simplest form, the recurrent network defines at each step t how h_t updates, based on the current vector x_t and the hidden state at the previous step, shown in equation 5 (Goodfellow et al. 2016).

$$h_t = b + f(Ux_t + Wh_{(t-1)}) \tag{5}$$

For RNN, f has often a *tanh* or *logistic* activation function (Appendix B). The weight matrices $U \in R^{dh \times dx}$ and $W \in R^{dh \times dh}$ describe the input-to-hidden and hidden-to-hidden connections, whereby $b \in$

R^{dh} is the bias vector. The first hidden state h_0, is ought to be set to 0. After each timestep, an output vector is created, written as:

$$\hat{yt} = g(Vh_t + c) = g(o_t) \tag{6}$$

The weight matrix $V \in R^{dy \times dh}$ describes the hidden-to-output vector, whereby $c \in R^{dy}$ is the bias vector. For RNN, all parameters are the same in each time step. When the application is a classification problem, the function g is often a *softmax* function, and for regression, it can be a fully connected NN. The training loss of an RNN measures how far each output o is from training target y. Loss L computes $\hat{y}$ and compares this to target y. The total loss over an x values sequence paired with a y value sequence would summate all losses over all time steps. Total loss is a classic example of RNN, mapping an input sequence to an output sequence of the same length. (Goodfellow et al. 2016)

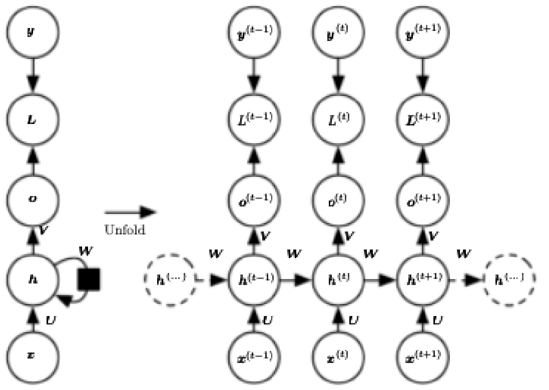


Figure 3: Visualisation of an RNN, retrieved from Goodfellow et al. (2016).

A downside of the RNN is the vanishing gradient problem. The signal flows 'back in time' over previous inputs' feedback connections with gradient-based learning, building adequate input storage. However, this will lead to a long learning time for the traditional backpropagation method because the output of t depends on earlier sequence input, leading to a gradient that will vanish and a NN to stop training (Hochreiter, 1998). LSTM overcomes this problem.

Long Short-Term Memory

LSTM is an application of RNN, engaging sequence learning tasks such as handwriting recognition or speech recognition sentiment analysis (Hochreiter & Schmidhuber, 1997). LSTM introduces three new gates and a cell state to deal with the vanishing gradient problem. The cell state, c_t, is multiplied with new data, creating a long-term memory and storing the previous information. The cell state lets the model work further on information or forgets it after transferring a cell. In addition, the cell state contains information on the input of the x sequences across timesteps. The previous hidden state, h_t, refers to the output of the previous point in time, also known as the short-term memory. It is added with the new data, x_t (Wei et al. 2022). The gates of an LSTM allow the NN to work on time series with long term timescales and model dependencies. Also, most LSTM models do not require timestamping in sequence as an input, like CNN, because the memory keeps track of the information of recent timestamps (Braei & Wagner, 2020).

LSTM allows cells to use their parameters for every cell, which is considered the main difference compared with RNN. As a result, the model can store information on a pattern based on characteristics in a small time frame. After completing the pattern, detailed information from the previous parts is discarded and moves to the following similar pattern. The ability to discard and go to the following pattern allows for solving tasks requiring sequence modelling with temporal dependencies.

Figure 4 shows an LSTM cell. Several gated mechanisms are implemented within the cell through weighted connections, using a logistic sigmoid activation. The logistic sigmoid activation checks the connection, converging to 1 (true) or 0 (false). Sigmoid activations can constitute smooth curves in this range of 0 to 1, while the model remains differentiable, telling the gate what action to undertake. For example, the forget gate can forget information by setting it to 0 through the sigmoid layer when evaluating the previous information at the current *t*. Equation 7 shows the output of the forget gate, noted as f_t . The forget gate, in particular, prevents the vanishing gradient problem. During training, the model learns the parameters of the matrices (*W* and *U*) in the forget gate.

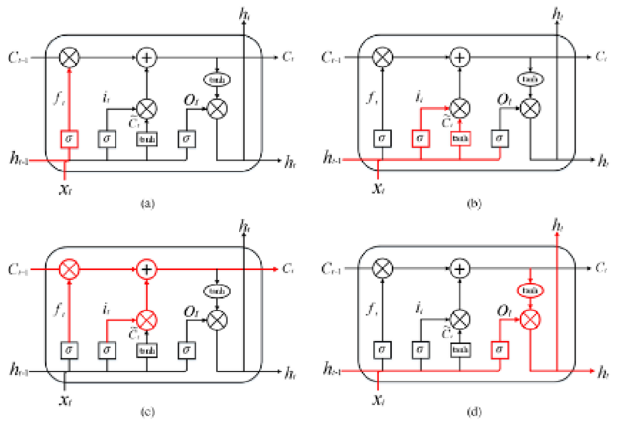


Figure 4: LSTM cell structure. (a) Forget gate. (b) Update Gate. (c) Cell State. (d) Output gate, retrieved from Xie et al. (2020).

In Equation 9 one can see that with j_t, the LSTM model can accumulate information from the current time step, taking into account the previous step. On the other hand, Equation 8 shows how the update gate, i_t, decides which information will be added to the cell state from the current time step. Finally, in Equation 10, one can see that the updated cell state, c_t, is calculated based on the previous cell state, c_{t-1}, the output of the forget gate, f_t, and the update gate. The $\odot$ indicates an element-wise vector product. In Figure 4 this is depicted as an *x*. As shown in Equation 10, subsequently, the calculation of c_t, the output of h_t (equation 12) is calculated. A *tanh* function distributes the gradients, allowing the cell state estimations to flow longer without vanishing or exploding.

$$f_t = sigmoid(W_f * x_t + U_f * h_{t-1} + b_f) \tag{7}$$

$$i_t = sigmoid(W_i x_t + \mathrm{Ui}_{ht-1} + b_i) \tag{8}$$

$$j_t = tanh(W_j x_t + U_j h_{t-1} + b_j) \tag{9}$$

$$c_t = \mathrm{f}_t \odot c_{t-1} + i_t \odot j_t \tag{10}$$

$$o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o) \tag{11}$$

$$h_t = o_t \odot tanh(c_t) \tag{12}$$

LSTM has performed well in recent publications, becoming the primary method when processing sequential, temporal data (Jozefowicz, Zaremba & Sutskever, 2015).

3.4 Overview of Proposed Model LSTM-AE

This paper proposes a combination of LSTM and an AE to analyse the unlabeled humidity series. The ANN-AE with LSTM can learn a representation of unlabelled data through self-supervised learning. In the encoding process, a fixed size vector from the set of input data sequences is put into the LSTM-AE. As mentioned before, LSTM allows an NN to apply memory in the encoder, providing a memory cell to retain the dependencies. The AE also reduces the dimensions of the representation. Afterward, the LSTM decoder regenerates its fixed vector size deriving from the detected representation of the input data. Training the network neglects the noises that generate the reconstruction output that can be compared in terms of the reconstruction error rates used to detect anomalies. An overview of this process is shown in Figure 5 and is further elaborated in this chapter.

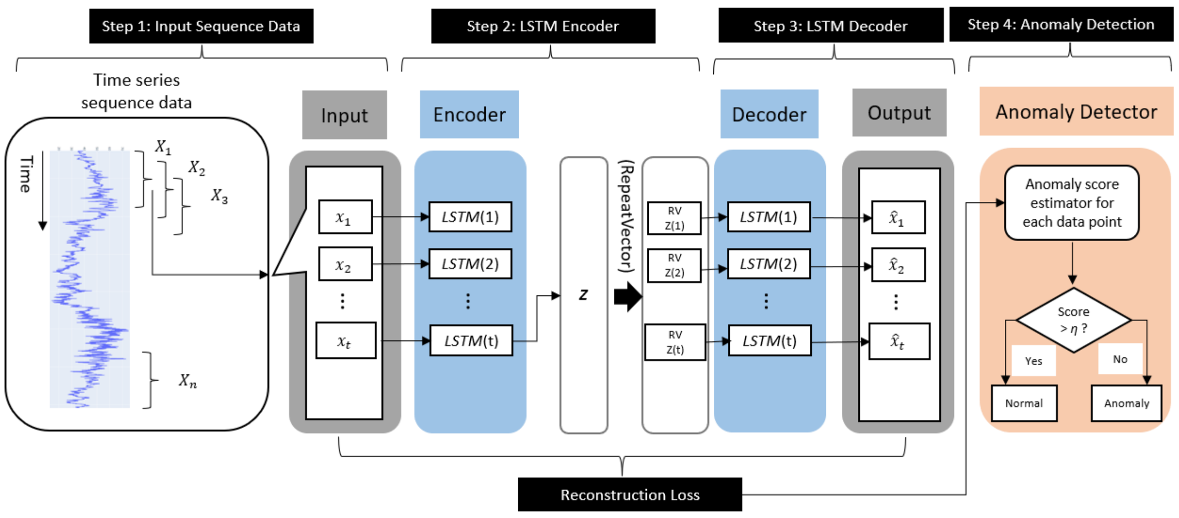


Figure 5: Overview of LSTM-AE, guided figure from (Wei et al. 2022)

A fitting language to apply AE is Python. Python is a programming language which can be used in collaboration with Google Colab as the main notebook for coding and training the model. Keras, a DL application programming interface built on top of Tensorflow, consists of the proposed model's core functions. This paper applies Python Version 3.7.13.

In Keras, the data is stored in multi-dimensional matrices, called tensors. One uses a 3D tensor when data has a time component. The 3D tensor is shaped as batch, timesteps, and features. Whereby timesteps represent the specific time axis, as shown in Figure 6. A rule of thumb is that the timesteps are on the second axis. For example, Humidity 1217 series records data every five minutes, creating 178,727 values/batches. The timesteps indicate the size of the memory of the LSTM, so when setting the timesteps at 10, the neural network assigns a sequence of 10 values to interact with the 10 LSTM cell units. The third value, feature, shows the number of features an observation has. In the case of humidity, every value has the same feature, the measurement of humidity (Chollet, 2017). In Keras, the LSTM layer uses hyperparameters, such as the units, return_sequences, return_state, dropout and batches, discussed in Appendix D.

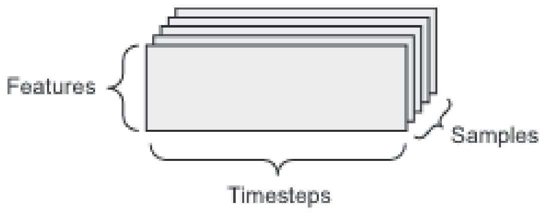


Figure 6: A Three Dimensional Time Series Data Sensor, retrieved from Chollet (2017).

Reconstruction Loss

Reconstruction loss is one of the most popular methods to detect anomalies (Xu et al. 2021). Engaging with an autoencoder model renders the input and output data to be compared. It allows detecting the difference between the output and the input data for anomaly detection. The goal for anomaly detection is to minimise the reconstruction loss between the output and input, using each sample x_i, depicted as:

$$x_i = \frac{1}{n} \sum_{n=1}^{n} |\hat{x}_i - x_i| \tag{13}$$

In Equation 13, x and $\hat{x}$ represent input and output data, respectively. Whileas, n indicates the number of samples in the training set. The following formula calculating the representation loss for the total samples (*N)*:

$$loss = \frac{1}{N} \sum_{N=i}^{1} x_i \tag{14}$$

Training Parameters

Table 1 presents the summary of the selected hyperparameters to train our model. More details can be viewed in Appendix D. The size of timestamps is set at 10 due to its best performance on the evaluation metrics after comparing with the other sizes at 15 and 30. Dropout 0.2 is applied to the LSTM layers to prevent overfitting, outperforming the setting of Dropout 0.1. After testing with different sizes of batch and epoch, the batch size at 10 and epoch at 64 showed us the best results throughout all the experiments.

Table 1: LSTM-AE Training Parameters

Hyperparameters	**Values**	**Description**
Timestamps	10	Input sequence (samples)
Dropout	0.2	Number of neurons ignored
Batch size	10	Number of samples in one fwd/bwd pass
Epoch	64	Number of one fwd/bwd pass of all samples

Data Pre-Processing

Before applying machine learning techniques, one must consider data cleaning. A time series exists primarily of recordings of a constant recording, e.g., a recording every five minutes. When time-series data is not collected in a constant interval, data preprocessing is applied to create equally time-spaced datasets by filling in empty/missing values in the time-steps (Singh & Olinsky, 2017). Regarding anomaly detection, no further requirements for data preprocessing are needed. Filtering out data could lead to filtering out anomalies, which should be detected by the anomaly detection method (Zhang et al. 2017).

The data normalisation technique assists in eliminating the inconsistent scales to reduce the time consumption and the complication of the model training (Wei et al. 2022). In this paper, the normalised data supports the construction of reconstruction error which defines anomalies. The standard scalar normalisation is written down as follows:

$$Z_i = \frac{X_i - \mu}{S} \tag{15}$$

Z_i represents all the numeric values in the range of scaled data, X_i represents a data point, μ refers to the mean value from the feature, and S refers to the feature standard deviation. The data normalisation is used for each dataset, with both training and testing values at the same scale by shifting mean to zero, and the standard deviation to one. This process aligns with further LSTM modelling, carrying the potential changes in magnitude using tanh input and sigmoidal gates (Farzad, Mashayekhi & Hassanpour, 2019).

Before training the model, the dataset is split into two main sets. The training set (80%) is used to train the model, and the test set (20%) is used for testing the model. According to Trinh et al. (2019), most of the training has to be non-anomalous. This allows the architecture to learn more efficiently for the reconstructed representation of common examples with low reconstruction errors, which can be used to compare to anomalous samples in the testing set more accurately. Furthermore, during the model fitting, the training data is further reserved for validation holdout set at 10% to evaluate validation metrics.

3.5 LSTM-AE Steps

The following part illustrates in four steps how the LSTM-AE is implemented and how the parameters are set. Figure 5 shows a graphical representation of the four steps. The first step is putting in the input sequence data. Second, the data is processed in the encoder, and in the third step, it will be processed in the decoder. In the final step, it is explained how anomalies will be detected.

Step 1: Input Sequence Data

A time series can be split into a series of sequences: $[\boldsymbol{X}_1, \boldsymbol{X}_2, ...\boldsymbol{X}_N]$. In the time series, each $\boldsymbol{X}$ sequence consists of sequential data points x_t, representing the input feature at timestep t: $[x_1, x_2, x_3, ...x_t]$. The length of the windows, W, are fixed. The main concept of data windowing is that each window keeps moving one step at a time in a sequential way, demonstrated as follows:

$$\begin{aligned} \boldsymbol{X}_1 &= [x_1, x_2, ..., x_W] \\ \boldsymbol{X}_2 &= [x_2, ..., x_{W+1}] \\ \boldsymbol{X}_N &= [x_N, ..., x_T] \end{aligned} \tag{16}$$

This process generates a data sequence into a 2D array in which each dimension presents the samples at the assigned timesteps. In this paper, 10 timestamps are set for each dataset. Thus, the list of samples is obtained per 10 timesteps: $[x_1, x_2, x_3, ...x_{10}]$ for the sequences of $\boldsymbol{X}_1$ *to* $\boldsymbol{X}_N$.

Step 2: LSTM Encoder

The LSTM Encoder engages with the LSTM layer, and learns the most relevant features from the input sequences.

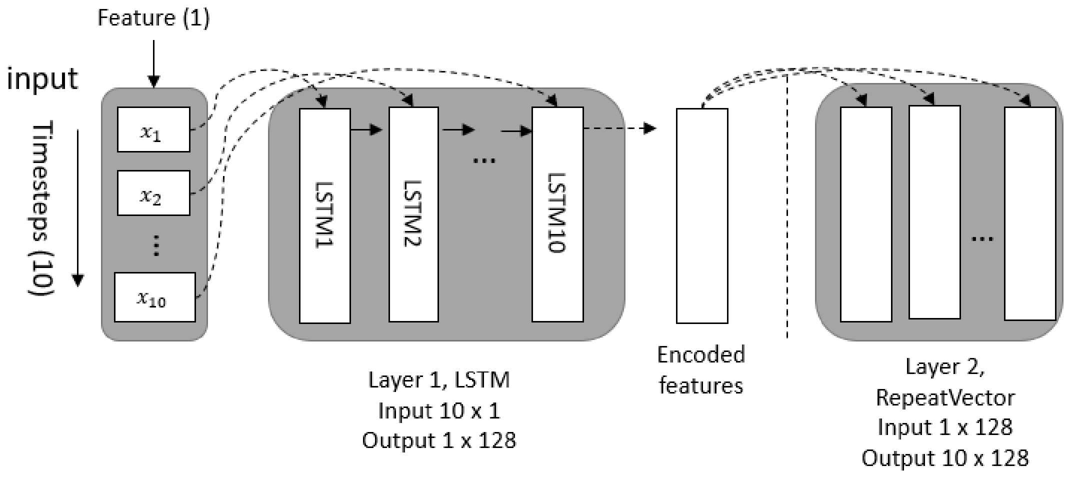


Figure 7: Layout of LSTM Encoder

Figure 7 demonstrates that each $\boldsymbol{X}_i$ contains 10 samples recorded on 10 timesteps and one feature, taking a 10x1 form. Next, these 10x1 samples are put into the first layer of the encoder, generating layer 1, consisting of 10 LSTM cells based on the number of assigned timesteps.

Each of the 10 LSTM cells can memorise or ignore the information of the previous samples. The 10 LSTM cells sequentially analyse the data. For instance, the second LSTM cell maintains information from the first LSTM cell, which is passed on to the third cell. The cells learn from their previous cells and select the information deemed necessary to memorise. The last LSTM cell in the sequence contains the necessary samples and features crucial to maintaining long-term memory. Then it transforms the output to a 1x128 vector, also known as the encoded features. In the transformation, the data is standardised, taking into account the different magnitudes of the data. Then the standardised data is put into the second layer. The second layer is the bridge between the encoder and the decoder. As mentioned in section 3.2.1, it compresses the dimensions into a representation. The second layer uses RepeatVectors to generate duplicates of the input 1x128 vector in the second layer into the number of timesteps equal to the output 10x128 (Appendix D). To illustrate, with 10 timesteps, 10 feature copies are created and presented in the form of 10x128 as the output.

Step 3: LSTM Decoder

Figure 8 shows the next step of the LSTM-AE, the LSTM decoder. The RepeatVectors encoded output (i.e., 10x128) from the second layer is processed as an input to the LSTM decoder in the third layer. After interacting with the encoded features from the RepeatVectors, the 10 LSTM cell units create an output of the same dimensional form of 10x128, noted as *L*, which represents the result of all learnings. Next, to obtain the final output with the same size as the original input (i.e., 10x1), TimeDistributed is used in order to obtain the final output as the same size as the input. The output *L* of the LSTM decoding process is multiplied with another layer, TimeDistributed carrying a 1x128 vector, noted as *T*. The interaction of the two components *L* x *T* is the final output results in the form of 10x1, which have the same size as the input.

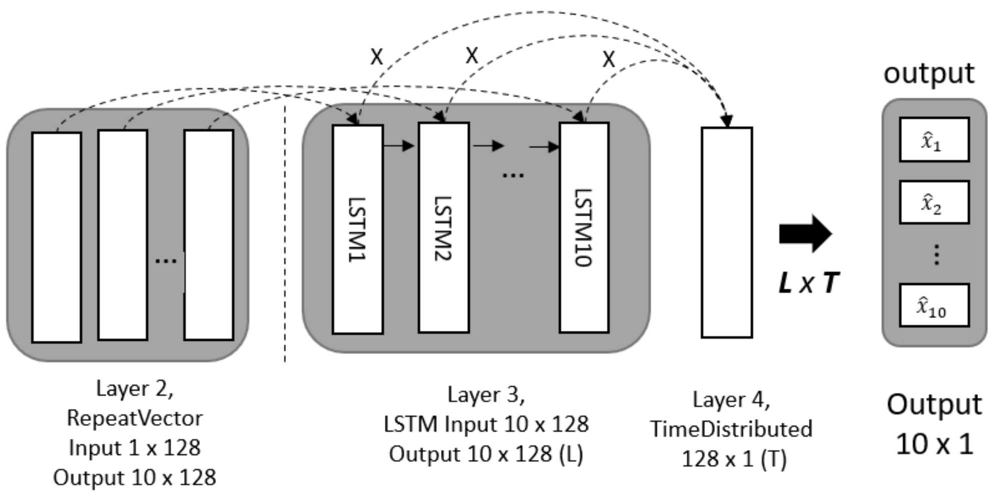

X
X
X
output
LSTM1
LSTM2
LSTM10
...
$\hat{x}_1$
$\hat{x}_2$
$\hat{x}_{10}$
L x T
Layer 2, RepeatVector Input 1 x 128 Output 10 x 128
Layer 3, LSTM Input 10 x 128 Output 10 x 128 (L)
Layer 4, TimeDistributed 128 x 1 (T)
Output 10 x 1

Figure 8: Layout of LSTM Decoder

Step 4: Anomaly Detection

Training Phase

The data is split into a training and a test set. Both sets are used in separate parts of the autoencoder. First, the training data applies up until the LSTM decoder. Then, the output from the training set is a reconstructed set in the form of a 1D vector 10x1. As seen in Figure 5, the reconstructed vector is compared to the input vector, calculating the reconstruction loss. To measure the reconstruction error, a loss function is used. Previously, Wei et al. (2022) and Kang et al. (2021) used the MAE to calculate the error for an LSTM autoencoder. They stated that MAE is preferred instead of MSE and RMSE. For both MSE and RMSE, the errors are squared before being averaged. This can lead to higher weights because of the more significant errors, making the model more sensitive to noise.

The MAE will be high for anomalous data but lower for non-anomalous data. According to Dau et al. (2014), the standard approach to defining the anomaly threshold is to approximate the maximum error after the training. However, this appears to be an extreme scenario to capture the anomalies in the sensor data. According to Provotar et al. (2020), a more suitable method for determining a threshold in anomaly detection is creating a decision boundary in the form of a threshold. Provotar et al. (2020) applied a 99.9% percentile to detect audio sample anomalies. This thesis will apply a similar threshold and test a 99.5% threshold to review thresholds based on different data sets.

Testing Phase

Similar to the training phase, an input sequence of time series in the test dataset is assigned for 10 samples or data points at the given 10 timesteps. Next, all sequences of the test dataset are fed into the trained LSTM encoder and reduced the feature representation as input sequences to the LSTM decoder. Finally, the LSTM decoder generates a time series output to be detected for anomalies. As shown in equation 17, the sequence with a value higher than the threshold is classified as an anomaly and the values with a reconstruction error lower than the threshold are classified as non-anomalous.

$$X' = \begin{cases} X_i' \text{ is anomalies, } \text{if } test_{MAE\ Loss[i]} > \eta \\ X_i' \text{ is normal, } \text{otherwise} \end{cases} \tag{17}$$

4 Analysis of Labelled Benchmark Data

This section validates the proposed methodology by applying the LSTM-AE to the ‘Yahoo S5 data set’ as a benchmark data series.

4.1 Yahoo Data Description

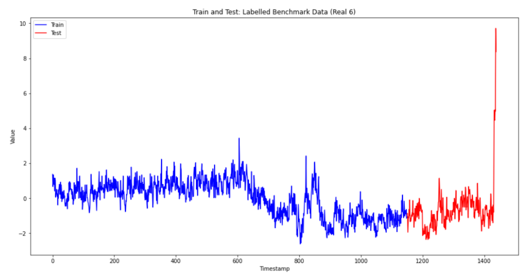
The Yahoo S5 dataset is part of the Yahoo Webscope program and is freely accessible for academic purposes (Yahoo, n.d.). The dataset is extensively used to evaluate methods for anomaly detection (Braei & Wagner, 2020; Thill, Konen, & Bäck, 2017; Yoshihara & Takahashi, 2022). The dataset is deemed effective to test an anomaly detection algorithm. Goa et al. (2021) state that the *“Yahoo data set has a good coverage of different varieties of anomalies in time series, such as seasonality, level change, variance change, and their combinations.”* The Yahoo S5 dataset contains 367 real-time series with labelled anomalies. They are divided into sets of A1, A2, A3, and A4 benchmarks. The A1 benchmark series are real production traffic data of Yahoo services, while the other benchmark series are synthetic time series. This paper applies the real-life A1 series, to validate the methodology. The A1 series contain 67 univariate time series, whereby humans labelled the anomalies.

For the analysis of the humidity series, two time series will be used. Therefore, we randomly picked two series from the A1 benchmark series to test the model. The selected series were the 6th and 10th of the A1 benchmark dataset and named ‘Real 6’ and ‘Real 10’. Both Real series are three-dimensional, measuring ‘*value*,’ ‘*timestamp*,’ and ‘*is_anomaly*’ each hour. Out of the three dimensions, *value* is the main feature, capturing the values of the timestamps. Then the *is_anomaly* column indicates if the data is anomalous (1), or non-anomalous (0). Further details of the datasets are found in Table 2.

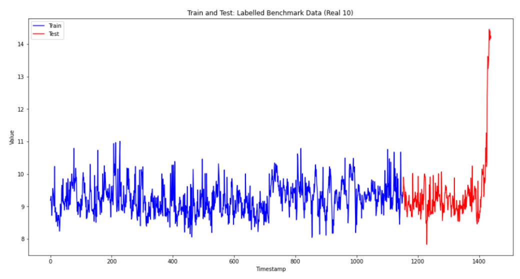
Table 2: Yahoo A1 Benchmark Datasets for Anomaly Detection

Dataset	**Size**	**Anomaly Proportion**	**Anomaly**	**Normal**
Real 6	1,439	0.56%	8	1,431
Real 10	1,439	0.91%	13	1426

As Trinh et al. (2019) mentioned, most of the training data have to be non-anomalous. Therefore, before we trained the model, we traced the location of the anomalies found in Appendix E. Figure 9 shows the split between train and test data to confirm the training data did not contain anomalies.



I. Real 6 Series



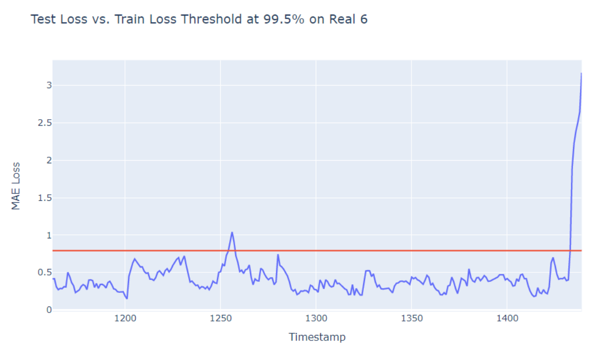
II. Real 10 Series

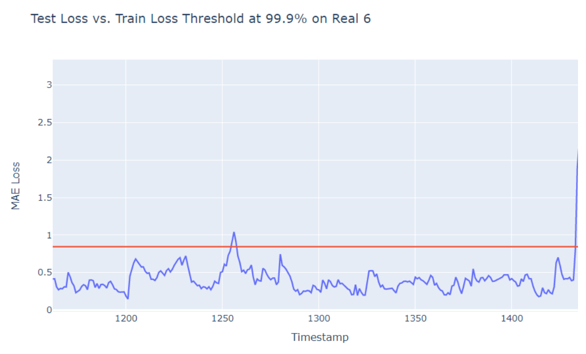
Figure 9: Overview of Train and Test Sets on I. Real 6 Series and II. Real 10 Series

4.2 Application of LSTM-AE to Labelled Benchmark Data

Real 6 Series

Following the steps of LSTM-AE, the data for Real 6 is put into the LSTM-AE accordingly. The training parameters are found in Table 1. Two MAE loss thresholds apply, one with a level of 99.5% and the other with 99.9% (as previously recommended by Provotar et al. (2019)). One can find an overview of the parameters in Appendix F. Figure 10 shows the threshold obtained by the training MAE loss and the test MAE loss. The figure shows that the 99.5% MAE loss threshold is at 0.79, while the 99.9% MAE loss is at 0.85.



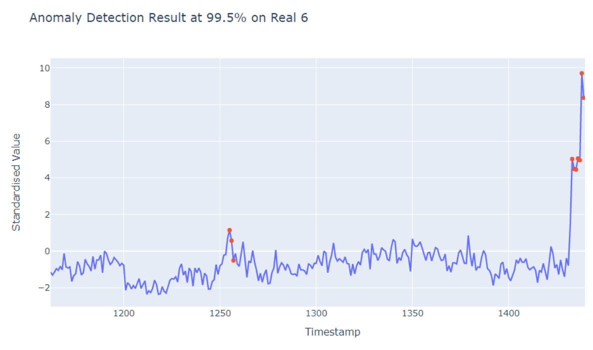


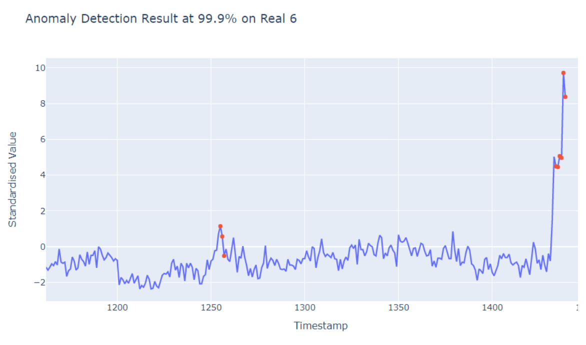
I. 99.5% Threshold

II. 99.9% Threshold

Figure 10: Test Loss vs. Train Loss Threshold I. 99.5% and II. 99.9% on Real 6 Series

The final anomaly detection at a threshold of 99.5% and 99.9% can be found in Figure 11. The evaluation of the model is done via a confusion matrix, further discussed in section 4.3.





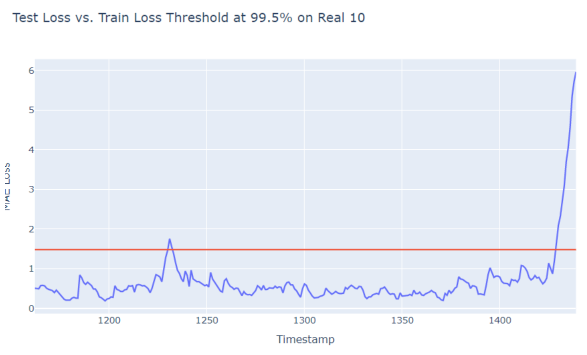
I. 99.5% Threshold

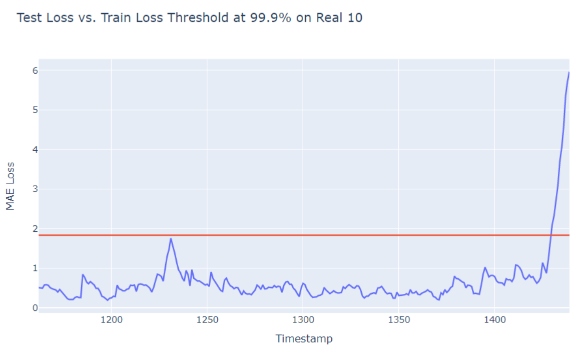
II. 99.9% Threshold

Figure 11: Anomaly Detection a. 99.5% and b. 99.9% on Real 6 Series

Real 10 Series

Real 10 series applies the same method as Real 6 series. Since Real 10 series uses the same architecture, the parameters after building the model are the same as the parameters of Real 6 series (Appendix F). Figure 12 shows that the 99.5% threshold of training MAE loss is at 1.48, while the 99.9% training MAE loss threshold is at 1.83.

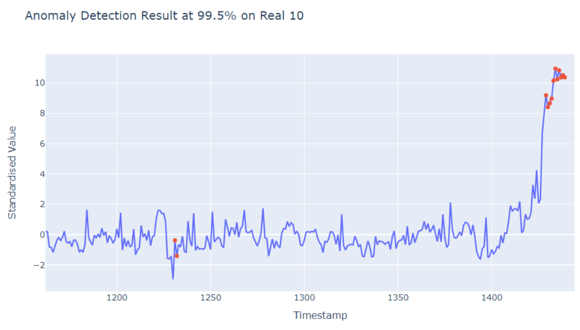


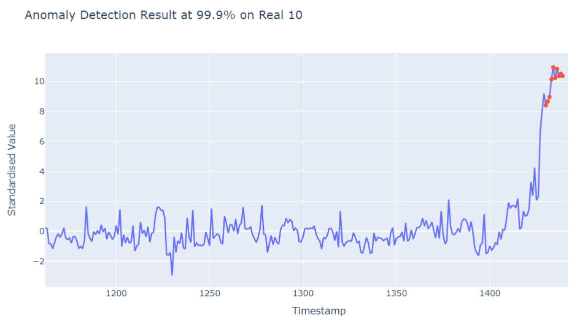


I. 99.5% Threshold II. 99.9% Threshold

Figure 12: Test Loss vs. Train Loss Threshold I. 99.5% and II. 99.9% on Real 10 Series

The final anomaly detection at a threshold of 99.5% and 99.9% can be found in figure 13. Likewise, with the Yahoo Real 6 data, the evaluation of the model is done via a confusion matrix, discussed in section 4.3.





I. 99.5% Threshold II. 99.9% Threshold

Figure 13: Anomaly Detection I. 99.5% and II. 99.9% on Real 10 Series

4.3 Analysis of Labelled Benchmark Data

Confusion Matrix and Evaluation Metrics

A confusion matrix is used to evaluate classified models. The matrix can compare accurate labels versus predicted labels in the matrix. This paper concerns actual anomalies vs. predicted anomalies. Table 3 shows a confusion matrix. The rows represent the actual condition, and the columns represent the predicted condition. The confusion matrix contains four values. The True positive (TP) value correctly identifies anomalous data points as anomalous. The True Negative (TN) value correctly identifies non-anomalous data points as non-anomalous. The False Positive (FP) value incorrectly identifies non-anomalous data points as anomalous. Finally, the False Negative (FN) value incorrectly identifies anomalous data points as non-anomalous. The anomalies would all be TP in a perfect setting, or the non-anomalous would be correctly predicted as non-anomalous (TN). In an imperfect setting, all anomalies would be incorrectly predicted as non-anomalous data (FN), or all non-anomalous data would be incorrectly predicted as anomalous (FP). The latter values are type II and type I errors, respectively (Géron, 2019).

Table 3: Confusion Matrix

		Predicted Condition	
		Normal	**Anomaly**
Actual Condition	**Normal**	True Negative (TN)	False Positive (FP)
	Anomaly	False Negative (FN)	True Positive (TP)

The confusion matrix can give in-depth information about several metrics. One of the metrics is accuracy. Accuracy is the proportion of total correct predictions (Equation 18). However, accuracy does not discriminate how the labels are distributed. For instance, if the confusion matrix has 95% non-anomalous data with 5% anomalies and all the non-anomalous data is correctly classified while the anomalies are not, the accuracy would still be 95%. Therefore, one must consider precision, recall, and the F1 score. Precision is the proportion of the predicted positive data points (Equation 19) and recall is the proportion of actual positive data points that are correctly predicted (Equation 20). Precision and recall are interchangeable; increasing the threshold for recall will reduce the precision, vice versa. To combine both metrics, one can calculate the F1-score (Equation 21). A F1 score ranges from 0 to 1, whereby an F1 score of 1 indicates that the model predicts the actual values in the correct class (Géron, 2019).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{18}$$

$$Precision = \frac{TP}{TP + FP} \tag{19}$$

$$Recall = \frac{TP}{TP + FN} \tag{20}$$

$$F1\text{-}score = 2\left(\frac{Precision \cdot Recall}{Precision + Recall}\right) \tag{21}$$

Real 6 Series Analysis

As mentioned, the benchmark model on Real 6 series is applied to two MAE loss threshold scenarios. The goal is to see which option of the thresholds renders a better model, evaluated with the evaluation metrics. In Table 4 one sees that the 99.5% threshold outperforms the 99.9% level in all evaluation metrics. Accordingly, in the confusion matrix, the model with a threshold level of 99.5% can more accurately capture true positive values (7 out of 8) showing only one false negative value. However, compared with the model at the threshold level of 99.9%, as shown in Table 5, the model captures fewer actual anomalies (6 out of 8) and further incorrectly indicates two false negative values.

Table 4: Evaluation Metrics on Real 6

Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Real 6 (99.5%)	98.56%	70%	87.50%	77.77%
Real 6 (99.9%)	98.20%	66.67%	75%	70.58%

Table 5: Detection Results Based on Confusion Matrix Real 6 with 99.5% and 99.9% Threshold

			Predicted Condition	
			Normal	**Anomaly**
99.5% Threshold	*Actual Condition*	**Normal**	267	3
		Anomaly	1	7
99.9% Threshold	*Actual Condition*	**Normal**	267	3
		Anomaly	2	6

Real 10 Series Analysis

The same concept of determining the model's performance applies to the Real 10 Series. The evaluation metrics and confusion matrix remain the leading indicators of finding a better detection anomaly model. As shown in Table 6, for Real 10, the threshold level of 99.9% outperforms the threshold at level 99.5% in all evaluation metrics, except for recall. However, as shown in Table 7, the 99.5% threshold model predicts more actual anomalies (TP) but less actual non-anomalous data (TN) than the 99.9% threshold model.

Table 6: Evaluation Metrics of Real 10 Series

Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Real 10 (99.5%)	98.56%	84.62%	84.62%	84.62%
Real 10 (99.9%)	98.92%	100%	76.92%	86.96%

Table 7: Detection Results Based on Confusion Matrix Real 10 with 99.5% and 99.9% Threshold

			Predicted Condition	
			Normal	**Anomaly**
99.5% Threshold	*Actual Condition*	**Normal**	263	2
		Anomaly	2	11
99.9% Threshold	*Actual Condition*	**Normal**	265	0
		Anomaly	3	10

Threshold Model Selection

Based on accuracy for Real 6 Series, the 99.5% threshold model outperforms the 99.9% threshold model and vice versa for Real 10 Series. However, accuracy is higher due to correctly identifying non-anomalous data, considering all four values of the confusion matrix (Davis & Goadrich, 2006). In the case of this thesis, Sensative AB rather gets a warning when there is no anomaly (FP) than not getting any warning while there is an anomaly (FN). Therefore, detecting an FP is more beneficial than an FN, which aligns with Sensative ABs' initiative to generate an early warning system for the user to mitigate any anomalies as quickly as possible. Therefore recall is prioritised when comparing threshold levels. Based on the recall, the 99.5% threshold model outperforms the 99.9% threshold model for both Real 6 and Real 10 series. Therefore, the 95.5% threshold model is applied to the analysis of humidity series data in section 5.

4.4 Comparison to Similar LSTM-AE Models

Table 8 presents the performance comparison of this thesis' benchmark data sets to three similar LSTM-AE models, previously discussed in the anomaly detection methods section.

Table 8: LSTM-AE Results and References of Anomaly Detection

Dataset	**Ref**	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Real 6 (99.5%)	Our proposal	98.56%	70.00%	87.50%	77.77%
Real 6 (99.9%)	Our proposal	98.20%	66.67%	75.00%	70.58%
Real 10 (99.5%)	Our proposal	98.56%	84.62%	84.62%	84.62%
Real 10 (99.9%)	Our proposal	98.92%	100.00%	76.92%	86.96%
ECG	Lui et al. (2022)	98.57%	97.55%	97.98%	-
e-VDS	Li & He (2019)	94.16%	90.31%	88.45%	89.37%
Break Operating Unit (BOU) data	Kang et al. (2018)	94.44%	97.94%	85.77%	91.45%

Based on all of the results, the LSTM-AE models are reputable in achieving at least 94% accuracy, which is in line with our results of achieving at least 98% accuracy for all models. Our highest 99.5% MAE threshold level accuracy is similar to the highest accuracy of the compared articles, measured in an ECG dataset (Lui et al. 2022). We emphasise recall in line with the generation of an early warning system. Regarding this data view, the goal is similar to detecting arrhythmia classification in an ECG dataset. Compared to Lui et al. (2022), their LSTM-AE has higher precision and recall. However, our LSTM-AE, tested with the benchmark series, still obtains a recall of approximately 85%, deemed feasible for further application of unlabelled data series.

An explanation for an inferior recall can be the adjustment of the threshold set at 99.5%, which is not applied in other models. The LSTM-AE models from Lui et al. (2022), Li and He (2019), and Kang et al. (2018) apply a threshold based on the maximum MAE training loss, declaring anomalies considered extreme to compare. Setting a threshold of 99.5% and 99.9% affects the evaluation metrics of the confusion matrix to a less conservative method. However, we keep the threshold because, in line with the early warning system, it is more important to find an anomaly while there is none instead of finding no anomaly while there is one. Based on this discussion, we will further apply the LSTM-AE with a 99.5% MAE threshold level to the humidity series in the next section.

5 Analysis of Humidity Series

5.1 Description of Humidity Series Data

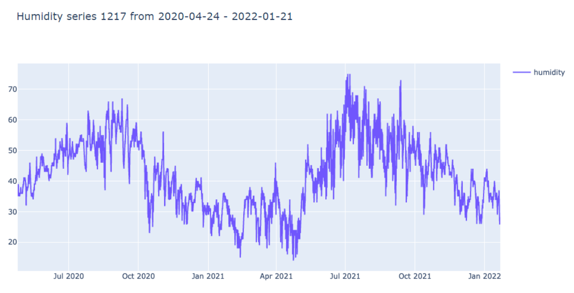
Sensative AB provided the humidity series, part of a compact meta dataset consisting of 4,335 sensor datasets. The data starts in 2020 and ends in 2022, whereby the exact dates and times differ per series. The sensors were multifunctional, measuring more features simultaneously (e.g., humidity, light, pressure, illuminance). Based on the importance of humidity, this paper will apply the methodology to the humidity series. According to Benjamin Fowelin (personal communication, 5 May 2022), Software Tester at Sensative AB, anomalous occurrences in general appear between 4 to 10%. Therefore, the assumption is that most data are non-anomalous, and few anomalies are present to detect.

Furthermore, this paper will research two humidity series. The first series, humidity series 1217, has the most available data points, and the second series, humidity series 2595, has the least data points in the meta dataset. Using different data points gives insights into the methodological architecture's effect for different sizes of humidity series. The humidity series have one feature, '*humidity value,'* so the data is univariate. In addition, every humidity value is timestamped, indicating the recording time. According to Sara Moricz, Senior Data Scientist at Sensative AB, the humidity is measured relative to temperature and pressure, in percent (personal communication, 5 May 2022). Table 9 shows further details about both humidity series.

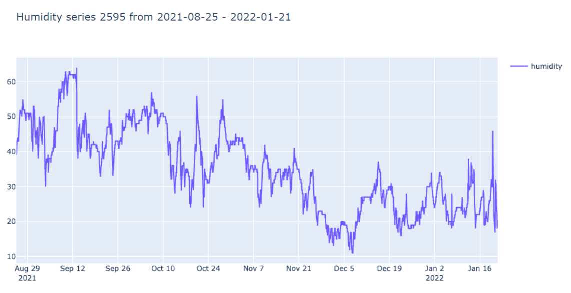
Table 9: Sensative AB Humidity Series for Anomaly Detection

Dataset	Series	Size	Begin Time	End Time
Humidity	1217	178,727	24-04-2020 (13:44:15)	21-01-2022 (12:03:58)
Humidity	2595	20,681	25-08-2021 (15:12:12)	21-01-2022 (11:58:47)

In both humidity series 1217 and 2595, no empty or Not a Number (NaN) values are present. Since it is unlabelled data, it is unclear whether a data point is an anomaly. Figure 14 shows a graphical representation of humidity series 1217 and 2595.



I. Series 1217

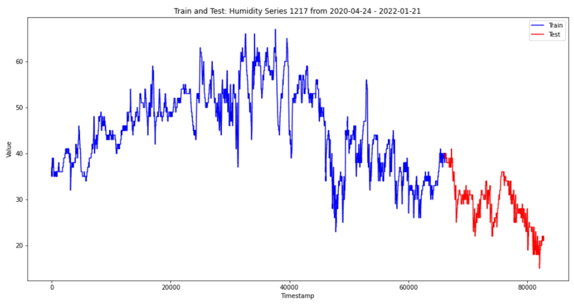


II. Series 2595

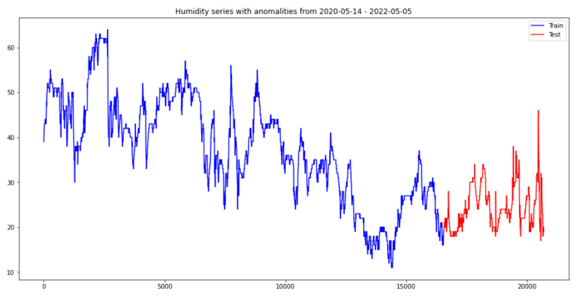
Figure 14: Graphical Representation of Humidity Series I. 1217 and II. 2595

Train and Test Splitting

First, the humidity series data splits into train and test data. Assuming that both humidity series 1217 and 2595 have a majority of non-anomalous data, the training set will be the first 80% of the sequence, and the last 20% of the sequence is test data. The training and test set cannot exist of randomly assigned data points of the whole set since the humidity series is sequential data. Figure 15 shows a graphical representation of the training and test data for humidity series 1217 and 2595.



I. Series 1217

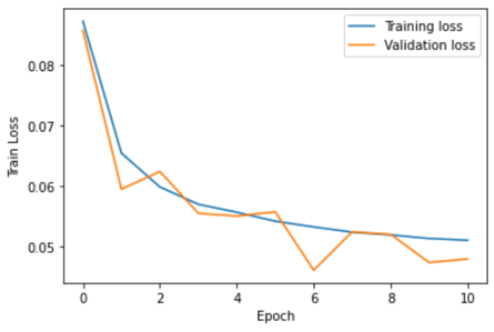


II. Series 2595

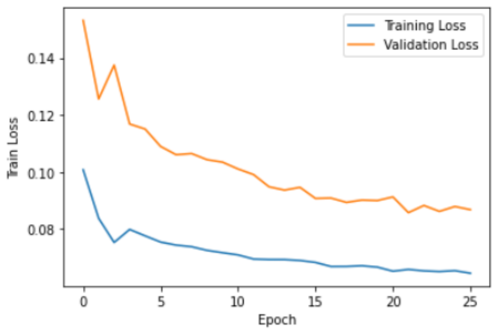
Figure 15: Representation of Train and Test Sets of Humidity Series I. 1217 and II. 2595

5.2 Application of LSTM-AE to Humidity Series

To build the model, one needs to set hyperparameters, found in Table 1 and explained in Appendix D. After building the model, there are 198,273 parameters (Appendix F). Then the model is fit, calculating the training and validation loss, shown in Figure 16. For the humidity series 1217, the training loss minimises smoothly, while the validation loss oscillates. The model appears to fit well; however, there are parts when the model seems to underfit when the training and validation loss are similar. For humidity series 2595, the training and validation loss are both minimised in a comparable matter, showing no signs of underfitting. However, the model might be slightly overfitting because the training loss is much lower than the validation loss.



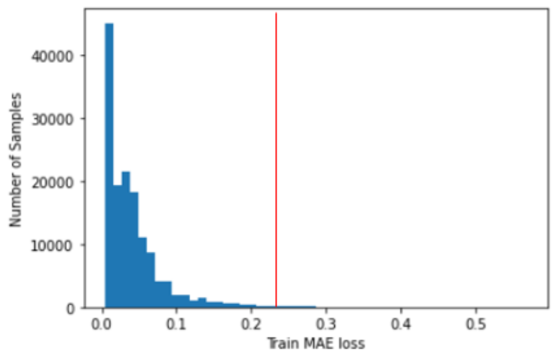
I. Series 1217



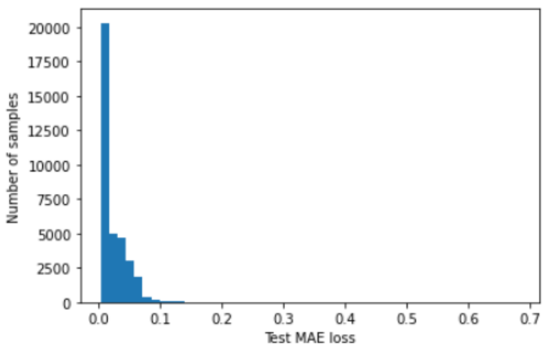
II. Series 2595

Figure 16: Model Fit of Humidity Series I. 1217 and II. 2595

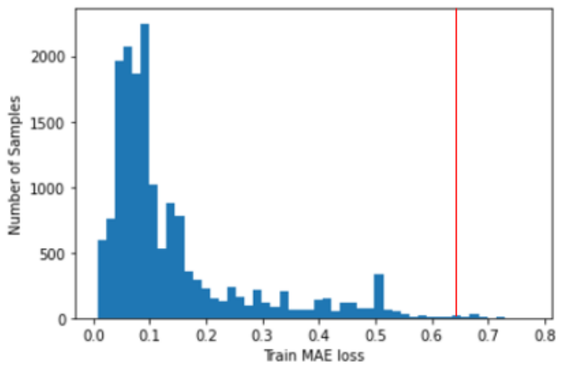
Figure 17 shows the application of the MAE-loss to retrieve the reconstruction error based on the training data. A threshold of 99.5% applies to the train MAE loss, used to find the anomalies based on the test MAE loss. Humidity series 1217 and 2595 have a threshold of 0.231 and 0.636, respectively.



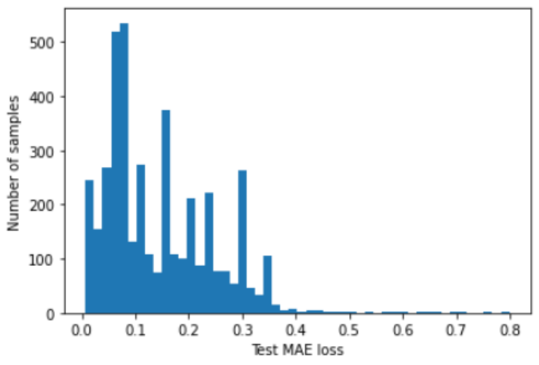
I. Series 1217 Train MAE Loss



II. Series 1217 Test MAE Loss



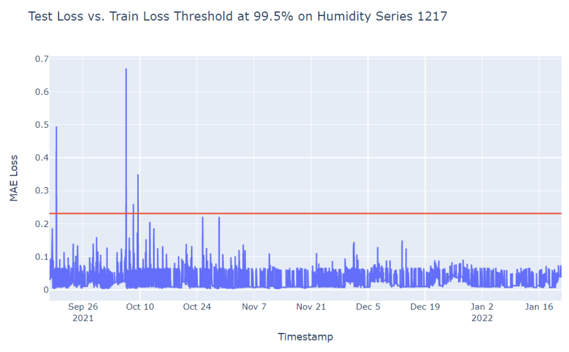
III. Series 2595 Train MAE Loss



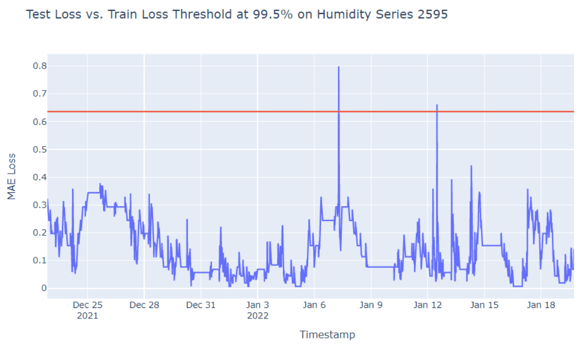
IV. Series 2595 Test MAE Loss

Figure 17: Representation of MAE Loss for Humidity Time Series I. 1217 Train Data II. 1217 Test Data and III. 2595 Train Data IV. 2595 Test Data

Figure 18 shows the threshold in combination with the standardised MAE loss of the test data. For humidity series 1217, at the beginning of the data sequence, four instances cross the threshold, meaning at least four anomalies. For humidity series 2595, there are at least four anomalies in the second half, where the MAE loss of the test data crosses the MAE loss training threshold.



I. Series 1217

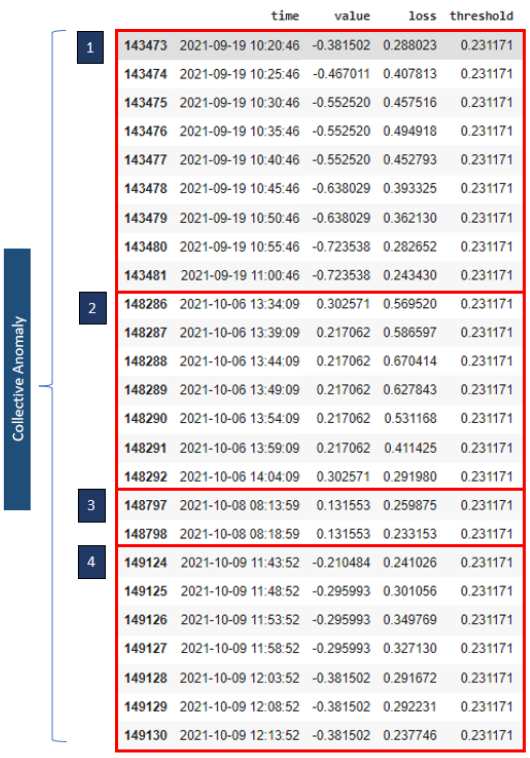


II. Series 2595

Figure 18: Test MAE Loss and 99.5% Train MAE Loss for Humidity Series I. 1217 and II. 2595

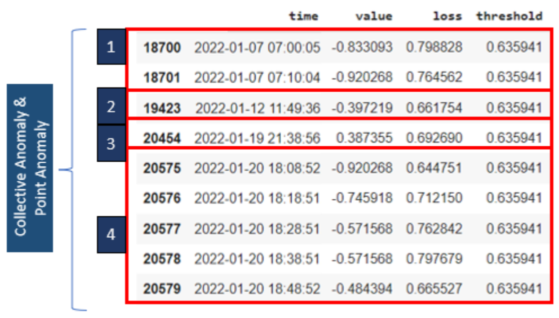
Table 10 and Figure 19 show the results of the anomaly detection at a threshold of 99.5% for the humidity series. For humidity series 1217 and 2595, 25 and 9 point anomalies were detected, respectively. However, detecting the sensor anomaly should capture collective anomalies rather than point anomalies. Based on the results, humidity series 1217 has four collective anomalies and humidity series 2595 has two collective anomalies and two points of anomaly. Table 10 shows the standardised anomaly values and the test loss per value. Furthermore, Appendix G shows a zoom-in on both humidity series 1217 and 2595, where it is clear that either a sudden rise or drop in value precedes the anomaly.

Table 10: Overview of Anomalies on Standardised Values with a 99.5% Threshold for Humidity Series I. 1217 and II. 2595.



Collective Anomaly

		time	value	loss	threshold
1	**143473**	2021-09-19 10:20:46	-0.381502	0.288023	0.231171
	143474	2021-09-19 10:25:46	-0.467011	0.407813	0.231171
	143475	2021-09-19 10:30:46	-0.552520	0.457516	0.231171
	143476	2021-09-19 10:35:46	-0.552520	0.494918	0.231171
	143477	2021-09-19 10:40:46	-0.552520	0.452793	0.231171
	143478	2021-09-19 10:45:46	-0.638029	0.393325	0.231171
	143479	2021-09-19 10:50:46	-0.638029	0.362130	0.231171
	143480	2021-09-19 10:55:46	-0.723538	0.282652	0.231171
	143481	2021-09-19 11:00:46	-0.723538	0.243430	0.231171
2	**148286**	2021-10-06 13:34:09	0.302571	0.569520	0.231171
	148287	2021-10-06 13:39:09	0.217062	0.586597	0.231171
	148288	2021-10-06 13:44:09	0.217062	0.670414	0.231171
	148289	2021-10-06 13:49:09	0.217062	0.627843	0.231171
	148290	2021-10-06 13:54:09	0.217062	0.531168	0.231171
	148291	2021-10-06 13:59:09	0.217062	0.411425	0.231171
	148292	2021-10-06 14:04:09	0.302571	0.291980	0.231171
3	**148797**	2021-10-08 08:13:59	0.131553	0.259875	0.231171
	148798	2021-10-08 08:18:59	0.131553	0.233153	0.231171
4	**149124**	2021-10-09 11:43:52	-0.210484	0.241026	0.231171
	149125	2021-10-09 11:48:52	-0.295993	0.301056	0.231171
	149126	2021-10-09 11:53:52	-0.295993	0.349769	0.231171
	149127	2021-10-09 11:58:52	-0.295993	0.327130	0.231171
	149128	2021-10-09 12:03:52	-0.381502	0.291672	0.231171
	149129	2021-10-09 12:08:52	-0.381502	0.292231	0.231171
	149130	2021-10-09 12:13:52	-0.381502	0.237746	0.231171

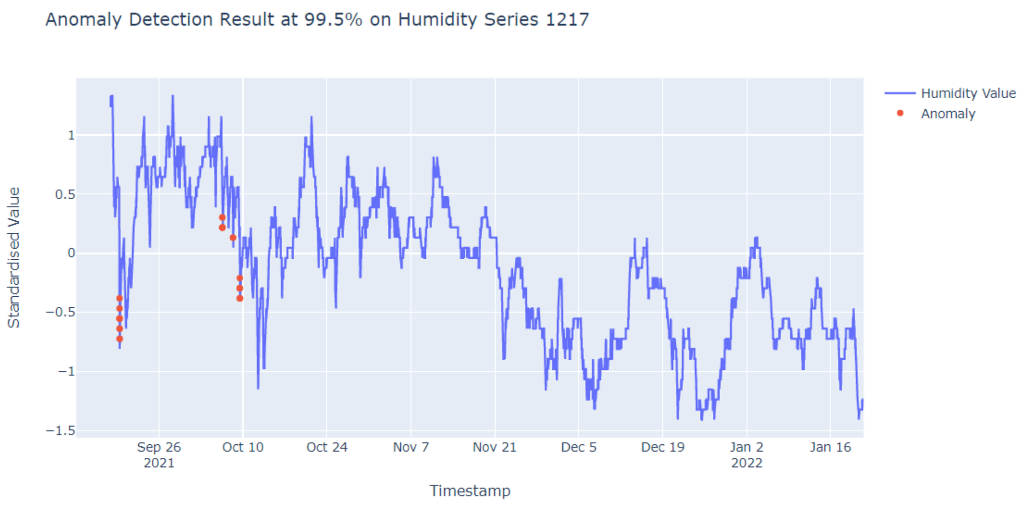


Collective Anomaly & Point Anomaly

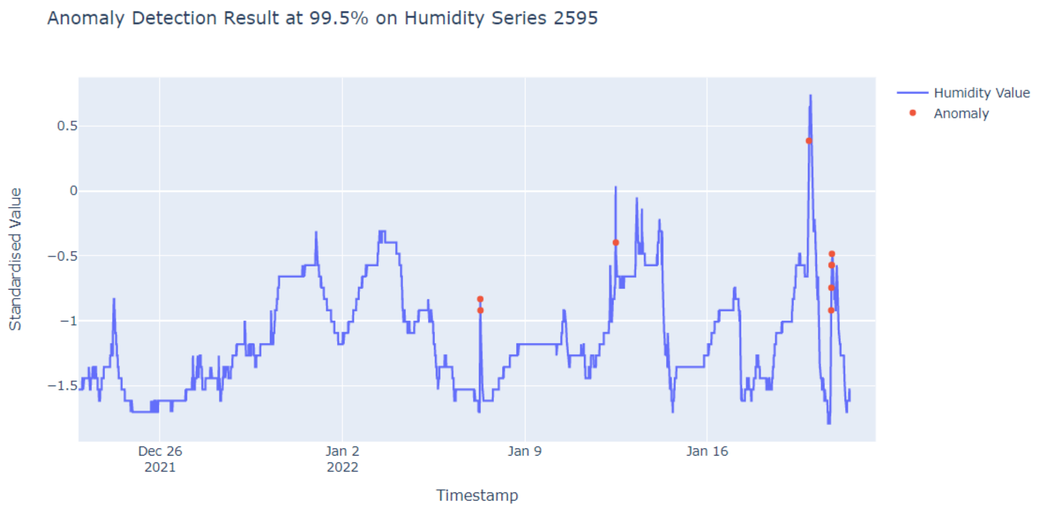
		time	value	loss	threshold
1	**18700**	2022-01-07 07:00:05	-0.833093	0.798828	0.635941
	18701	2022-01-07 07:10:04	-0.920268	0.764562	0.635941
2	**19423**	2022-01-12 11:49:36	-0.397219	0.661754	0.635941
3	**20454**	2022-01-19 21:38:56	0.387355	0.692690	0.635941
4	**20575**	2022-01-20 18:08:52	-0.920268	0.644751	0.635941
	20576	2022-01-20 18:18:51	-0.745918	0.712150	0.635941
	20577	2022-01-20 18:28:51	-0.571568	0.762842	0.635941
	20578	2022-01-20 18:38:51	-0.571568	0.797679	0.635941
	20579	2022-01-20 18:48:52	-0.484394	0.665527	0.635941

I. Series 1217

II. Series 2595



I. Series 1217



II. Series 2595

Figure 19: Anomaly Detection on Standardised Values with a 99.5% Threshold for Humidity Series I. 1217 and II. 2595

6 Conclusion

This thesis proposed the methodology of LSTM-AE for anomaly detection in unlabelled humidity series. Due to the limitation of validating the model's performance using unlabelled data, we selected our developed model based on the benchmark labelled data with the threshold of 99.5% (i.e., 98.56% accuracy, 87.50% recall) applied to our unlabeled humidity series. The MAE between the input and output from the LSTM-AE network allowed us to obtain non-anomaly and anomaly patterns. The training MAE loss at 99.5% was set as a threshold. The data values in the test set are fed into the network to obtain the reconstruction errors. These errors are compared against the threshold and report anomalies if exceeding the threshold. Eventually, our LSTM-AE detected 25 point anomalies or four collective anomalies in humidity series 1217, and 9 point anomalies or two collective anomalies for humidity series 2595. Therefore, based on our experiment LSTM-AE can detect anomalies in unlabelled data series. However, one must take into account that evaluating an anomaly detection model with unlabelled trained data remains challenging.

6.1 Future Work

For further development on anomaly detection at Sensative AB, we recommend having labelled data for training a model so that one can assess the model's practicality and reliability for anomaly detection. In the same pattern, with labelled data, other types of sensor data can be trained to find their own models by adjusting parameters that are suitable for their certain type.

Furthermore, we recommend Sensative AB: When using data on an already trained model, one should consider that test data is taken from a similar context. For instance, a sensor placed in a household generates different humidity values than a sensor placed in a garage. Using similar contexts can prevent misleading results, such as false positive results.

Finally, when having labelled data, other anomaly algorithms can apply. For instance, a LSTM variational autoencoder (LSTM-VAE) can give more insights. An LSTM-VAE can reconstruct a probabilistic distribution for each window-based thresholding which is more efficient for eliminating false alarms (Park, Hoshi, & Kemp, 2018). Lastly, this thesis includes solely univariate data to experiment. However, we suggest applying multivariate sensor data, enhancing the performance of detecting data values in different features.

References

Aggarwal, C. C. (2015). Outlier Analysis. *In Data mining*, *Springer*, pp. 237–263.

Al-amri, R., Murugesan, R. K., Man, M., Abdulateef, A. F., Al-Sharafi, M. A., & Alkahtani, A. A. (2021). A Review of Machine Learning and Deep Learning Techniques for Anomaly Detection in IoT Data, *Applied Sciences*, vol. 11, no. 12, pp. 5320

Angiulli, F., & Pizzuti, C. (2002). Fast Outlier Detection in High Dimensional Spaces. Principles of Data Mining and Knowledge Discovery, *Springer*, vol. 2431, Available online: https://doi.org/10.1007/3-540-45681-3_2 [Accessed 20 April 2022]

Barnett, V., & Lewis, T. (1984). Outliers in Statistical Data, New York: Wiley

Braei, M., & Wagner, S. (2020). Anomaly Detection in Univariate Time-Series: A Survey on the State-of-the-Art, preprint, Available online: https://arXiv:2004.00433 [Accessed 18 April 2022]

Chalapathy, R., & Chawla, S. (2019). Deep Learning for Anomaly Detection: A Survey, preprint, Available online: https://doi.org/10.48550/arXiv.1901.03407 [Accessed 24 April 2022]

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly Detection: A Survey, *ACM Computing Surveys*, vol. 41, no. 3, pp.1-58

Charte, D., Charte, F., García, S., del Jesus, M. J., & Herrera, F. (2018). A Practical Tutorial on Autoencoders for Nonlinear Feature Fusion: Taxonomy, Models, Software and Guidelines. *Information Fusion*, vol. 44, pp. 78-96

Chen, H., Deng, S., Bruner, H., & Garcia, J. (2004). Roots of Mold Problems and Humidity Control Measures in Institutional Buildings with Pre-Existing Mold Condition, *Texas A&M University Libraries*, Available online: https : / /hdl .handle .net /1969 .1 /4605 [Accessed 10 May 2022]

Chen, H., Liu, H., Chu, X., Liu, Q., & Xue, D. (2021). Anomaly Detection and Critical SCADA Parameters Identification for Wind Turbines Based on LSTM-AE Neural Network. Renewable Energy, vol.172, pp. 829-840

Chollet, F. (2016). Building Autoencoders in Keras, The Keras Blog, Available at: https://blog.keras.io/building-autoencoders-in-keras.html [Accessed 20 April 2022]

Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, IEEE, pp. 1800-1807

Cook, A. A., Mısırlı, G., & Fan, Z. (2020). Anomaly Detection for IoT Time-Series Data: A Survey, *IEEE*, vol. 7, no. 7, pp. 6481-6494

Dau, A. H., Ciesielski, V. & Song A. (2014). Anomaly Detection Using Replicator Neural Networks Trained on Examples of One Class, SEAL 2014: Proceedings of the 10th International Conference on

Simulated Evolution and Learning, vol. 8886, pp. 311-322, Available online: https://doi.org/10.1007/978-3-319-13563-2_27 [Accessed 3 May 2022]

Davis, J., & Goadrich, M. (2006). The Relationship between Precision-Recall and ROC curves, In Proceedings of the 23rd International Conference on Machine learning (ICML '06), *Association for Computing Machinery,* pp. 233–240, Available Online: https://doi.org/10.1145/1143844.1143874 [Accessed 17 May 2022]

Devarakonda, A., Naumov, M., & Garland, M. (2017). Adabatch: Adaptive Batch Sizes for Training Deep Neural Networks, preprint, Available online: https://arXiv:1712.02029 [Accessed 20 May 2022]

Egilmez, H. E., & Ortega, A. (2014). Spectral Anomaly Detection Using Graph-Based Filtering for Wireless Sensor Networks, 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), *IEEE*, pp. 1085-1089

Farzad, A., Mashayekhi, H., & Hassanpour, H. (2019). A Comparative Performance Analysis of Different Activation Functions in LSTM Networks for Classification, *Neural Computing and Applications*, 31(7), pp.2507-2521

Friedman, J., Hastie, T., & Tibshirani, R. (2001). The Elements of Statistical Learning, *Springer*, vol. 1, no. 10.

Gao, J., Song, X., Wen, Q., Wang, P., Sun, L., & Xu, H. (2021). RobustTAD: Robust Time Series Anomaly Detection via Decomposition and Convolutional Neural Networks, preprint, Available online: https://arxiv.org/abs/2002.09545v2 [Accessed date May 17 2022]

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, *O'Reilly Media, Inc*

Gómez, J., Quispe, A. & Kemper, G., (2020). A Comparative Study of Deep Learning Techniques Aimed at Detection of Arrhythmias from ECG Signals. In Brazilian Technology Symposium, *Springer*, vol. 233, pp. 385-395

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y., (2014). Generative Adversarial Nets, *Advances in neural information processing systems*, vol. *27*, pp. 2672-2680

Goodfellow, I., Bengio Y., & Courville A. (2016). Deep Learning. [e-book] MIT Press, Available at: http://www.deeplearningbook.org [Accessed 11 April 2022]

Gravesen, S., Nielsen, P. A., Iversen, R., & Nielsen, K. F. (1999). Microfungal Contamination of Damp Buildings-Examples of Risk Constructions and Risk Materials, *Environmental Health Perspectives*, vol. 107, no. 3, pp. 505-508

Halstead, J., Vikraman, R., & Prasad, R. (2022). Business Anomaly Detector: A Novel Approach to Identify Anomalies in Time Series Semi-Supervised Transactional Data [pdf], Available at: https://www.researchgate.net/profile/John-Halstead-3/publication/360503008_Business_Anomaly_Detector_A_Novel_Approach_to_Identify_Anomalies_in_Time_Series_Semi-Supervised_Transactional

_Data/links/627a9cd7973bbb29cc7228d2/Business-Anomaly-Detector-A-Novel-Approach-to-Identify-Anomalies-in-Time-Series-Semi-Supervised-Transactional-Data.pdf [Accessed 23 May 2022]

Han, P., Ellefsen, A. L., Li, G., Holmeset, F. T., & Zhang, H. (2021). Fault Detection With LSTM-Based Variational Autoencoder for Maritime Components. *IEEE Sensors Journal*, vol. 21, no.19, pp. 21903-21912

Hawkins, D.M. (1980). Identification of Outliers, London: Chapman and Hall

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, Available online: https://doi.org/10.1162/neco.1997.9.8.1735 [Accessed 14 April 2022]

Hochreiter, S. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107-116

Jain, A. K., Mao, J., & Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *Computer*, vol. 29, no. 3, pp. 31-44

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An Empirical Exploration of Recurrent Network Architectures. In International Conference on Machine Learning, *PMLR*, pp. 2342-2350

Jun, K., Lee, D. W., Lee, K., Lee, S., & Kim, M. S. (2020). Feature Extraction Using an RNN Autoencoder for Skeleton-Based Abnormal Gait Recognition, *IEEE* , vol. 8, pp. 19196-19207

Kang, J., Kim, C. S., Kang, J. W., & Gwak, J. (2021). Anomaly Detection of the Brake Operating Unit on Metro Vehicles Using a One-Class LSTM Autoencoder. *Applied Sciences*, vol. 11, no. 19, pp. 9290, Available online: https://doi.org/10.3390/app11199290 [Accessed 13 May 2022]

Kim, J.-Y., Chu C.-H., & Shin S.-M. (2014). ISSAQ: An Integrated Sensing Systems for Real-Time Indoor Air Quality Monitoring, *IEEE Sensors Journal*, vol. 14, no. 12, pp. 4230–4244

Kingma, D. P., & Welling, M. (2019). An Introduction to Variational Autoencoders, *Foundations and Trends in Machine Learning*, vol. 12, no. 4, pp. 307-392, Available online: https://doi.org/10.48550/arXiv.1906.02691 [Accessed 10 May 2022]

Kingma, D. P. & Ba, J. (2014). Adam: A Method for Stochastic Optimization, preprint, Available online: https://arXiv.1412.6980 [Accessed 5 May 2022]

Koekkoek, E. J. W., & Booltink, H. (1999). Neural Network Models to Predict Soil Water Retention. *European Journal of Soil Science*, vol. 50, no. 3, pp. 489-495

Kwon, D., Kim, H., Kim, J., Shu, S. C., Kim, I., & Kim, K. J. (2019). A Survey of Deep Learning-Based Network Anomaly Detection, *Cluster Computing*, vol. 22, no. 1, pp. 949–961

Lazarevic, A. & Kumar, V. (2005). Feature Bagging for Outlier Detection, Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, *ResearchGate GmbH*, vol. 21, pp.157-166

LeCun, Y., Bottou, L., Orr, B. G., & Müller, R. K. (1998). Efficient BackProp, *Lecture Notes in Computer Science*, vol. 1524, pp. 9–50

Li, S., & He, W. (2019). VidAnomaly: LSTM-Autoencoder-Based Adversarial Learning for One-Class Video Classification With Multiple Dynamic Images. In 2019 IEEE International Conference on Big Data (Big Data), *IEEE*, pp. 2881-2890.

Li, Y., Dang, X., Xia, C., Ma, Y., Ogura, D., & Hokoi, S. (2020). The Effect of Air Leakage Through the Air Cavities of Building Walls on Mold Growth Risks, *Energies*, vol. 13, no. 5, pp. 1177

Liu, P., Sun, X., Han, Y., He, Z., Zhang, W., & Wu, C.. (2022). Arrhythmia Classification of LSTM Autoencoder based on Time Series Anomaly Detection. *Biomedical Signal Processing and Control*, vol. 71, Available online: https://doi.org/10.1016/j.bspc.2021.103228 [Accessed 10 May 2022]

Martens, J., & Sutskever, I. (2012). Training Deep and Recurrent Networks with Hessianfree Optimization. Neural Network: Tricks of the Trade, *Lecture Notes in Computer Science*, vol. 7700, pp. 479–535

Mohajerin, N., & Waslander, S. L. (2017). State Initialization for Recurrent Neural Network Modeling of Time-Series Data. In 2017 International Joint Conference on Neural Networks (IJCNN), *IEEE*, pp. 2330-2337

Mohammadi, M., Al-Fuqaha, A., Sorour, S., & Guizani, M. (2018). Deep Learning for IoT Big Data and Streaming Analytics: A Survey. IEEE Communications Surveys & Tutorials, *IEEE*, vol. 20, no. 4, pp. 2923-2960

Munir, M., Siddiqui, S. A., Dengel A., & Ahmed, S. (2019). DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series, *IEEE*, vol. 7, pp. 1991-2005

Pang, G., Shen, C., Cao, L., & Hengel, A. V. D. (2021). Deep Learning for Anomaly Detection: A Review, *ACM Computing Surveys*, vol. 54, no. 2, pp. 1-38, Available online: https://doi.org/10.1145/3439950 [Accessed 10 May 2022]

Park, D., Hoshi, Y., & Kemp, C. C. (2018). A Multimodal Anomaly Detector for Robot-Assisted Feeding Using an LSTM-Based Variational Autoencoder, *IEEE*, vol. 3, no. 3, pp. 1544-1551

Peña, M., Biscarri, F., Guerrero, J. I., Monedero, I., & León, C. (2016). Rule-Based System to Detect Energy Efficiency Anomalies in Smart Buildings, a Data Mining Approach, *Expert Systems with Applications*, vol. 56, pp. 242-255

Provotar, I. O., Yaroslav M. L., & Maksym M. V. (2020). Unsupervised Anomaly Detection in Time Series Using LSTM-Based Autoencoders, In 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, IEEE,* pp. 513-517

Ruder, S. (2016). An Overview of Gradient Descent Optimization Algorithms, preprint, https://arXiv:1609.04747 [Accessed 18 May 2022]

Shi, Y., Luo, Y., Zhao, W., Shang, C., Wang, Y., & Chen, Y. (2013). A Radiosonde Using a Humidity Sensor Array with a Platinum Resistance Heater and Multi-Sensor Data Fusion, *Sensors 2013*, vol. 13, no. 7, pp. 8977-8996

Shyu, M., Chen, S., Sarinnapakorn, K., & Chang, L. (2003). A Novel Anomaly Detection Scheme Based on Principal Component Classifier [pdf], Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.299&rep=rep1&type=pdf [Accessed 20 April 2022]

Singh, N., & Olinsky, C. (2017). Demystifying Numenta Anomaly Benchmark. 2017 International Joint Conference on Neural Networks (IJCNN), *IEEE*, pp. 1570-1577

Thill, M., Konen, W., & Bäck, T. (2017). Online Anomaly Detection on the Webscope S5 Dataset: A Comparative Study. 2017 Evolving and Adaptive Intelligent Systems (EAIS), *IEEE*, pp. 1-8

Thill, M., Konen, W., Wang, H., & Bäck, T. (2021). Temporal Convolutional Autoencoder for Unsupervised Anomaly Detection in Time Series, Applied Soft Computing, vol. 112, Available Online: https://doi.org/10.1016/j.asoc.2021.107751 [Accessed 7 April 2022]

Trinh, H. D., Zeydan, E., Giupponi, L., & Dini, P. (2019). Detecting Mobile Traffic Anomalies Through Physical Control Channel Fingerprinting: A Deep Semi-Supervised Approach, *IEEE*, vol. 7, pp. 152187-152201, doi: 10.1109/ACCESS.2019.2947742

Wang, Z., Ma, H., Chen, H., Yan, B., & Chu, X. (2020). Performance Degradation Assessment of Rolling Bearing Based on Convolutional Neural Network and Deep Long-Short Term Memory Network, International Journal of Production Research, vol. 58, no. 13, pp. 3931-3943

Wei, Y., Jang-Jaccard, J., Xu, W., Sabrina, F., Camtepe, S., & Boulic, M. (2022). LSTM-Autoencoder based Anomaly Detection for Indoor Air Quality Time Series Data, Available online: https://doi.org/10.48550/arXiv.2204.06701 [Accessed 4 May 2022]

Williams, R. W., & Herrup, K. (1988). The Control of Neuron Number, *Annual Review of Neuroscience,* vol. 11, no. 1, pp. 423-453, Available online: https://doi.org/10.1146/annurev.ne.11.030188.002231 [Accessed 8 April 2022]

Xie, W., Wang, J., Xing, C., Guo, S., Guo, M., & Zhu, L. (2020). Variational Autoencoder Bidirectional Long and Short-Term Memory Neural Network Soft-Sensor Model Based on Batch Training Strategy, *IEEE Transactions on Industrial Informatics*, *IEEE*, vol. 17, no. 8, pp. 5325-5334

Xu, W., Jang-Jaccard, J., Singh, A., Wei, Y., & Sabrina, F. (2021). Improving Performance of Autoencoder-Based Network Anomaly Detection on NSL-KDD Dataset, *IEEE*, vol. 9, pp. 140136-140146

Yahoo (n.d.). Computing Systems Data: S5 - A Labeled Anomaly Detection Dataset, version 1.0(16M), Available online: https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70 [Accessed 10 May 2022]

Yao, L. & Ge, Z., (2017). Deep Learning of Semisupervised Process Data with Hierarchical Extreme Learning Machine and Soft Sensor Application, *IEEE Transactions on Industrial Electronics*, *IEEE*, vol. 65, no. 2, pp. 1490–1498

Yoshihara, K., & Takahashi, K. (2022). A Simple Method for Unsupervised Anomaly Detection: An Application to Web Time Series Data, *PLoS One*, vol. 17, no. 1, Available online: https://doi.org/10. 1371/journal.pone.0262463 [Accessed 17 May 2022]

Yuste, R. (2015). From the Neuron Doctrine to Neural Networks, *Nature Reviews Neuroscience*, vol. 16, no. 8, pp.487-497

Zhang, A., Song, S., Wang, J., & Yu, P. S. (2017). Time Series Data Cleaning: From Anomaly Detection to Anomaly Repairing, Proceedings of the VLDB Endowment, Available online: https://doi.org/10.14778/3115404.3115410 [Accessed 16 May 2022]

Zhang, C., Li, S., Zhang, H., & Chen, Y. (2020). Velc: A New Variational Autoencoder based Model for Time Series Anomaly Detection, Available online: https://doi.org/10.48550/arXiv.1907.01702 [Accessed 14 April 2022]

Zhou, B., & Chen, Z. (2016). Experimental Study on the Hygrothermal Performance of Zeolite-Based Humidity Control Building Materials, *International Journal of Heat and Technology*, vol. 34, no. 3, pp. 407-414

Zong, B., Song, Q., Min, M.R., Cheng, W., Lumezanu, C., Cho, D., & Chen, H. (2018). Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection, *In International Conference on Learning Representations (ICLR)*

Appendices

Appendix A: Human Neuron

NNs are traced back to the nervous system of a human. Figure A.1 shows a biological neuron found in the nervous system. The nervous system of the human brain consists of around 85 billion neurons, which communicate through neurotransmitters (Williams and Herrup, 1985). Neurons are connected axons to dendrite, whereby a neuron receives electrotechnical stimuli once surpassing a certain threshold. Then, the signal fires along the axon. This (conditional firing) is also called the activation, which resembles the activation in ANN (Yuste, 2015).

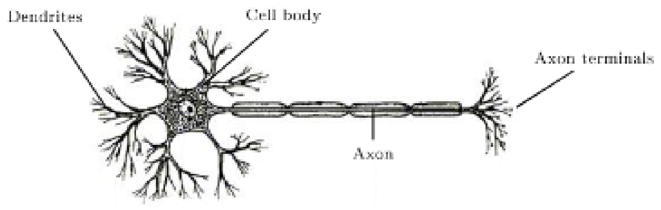
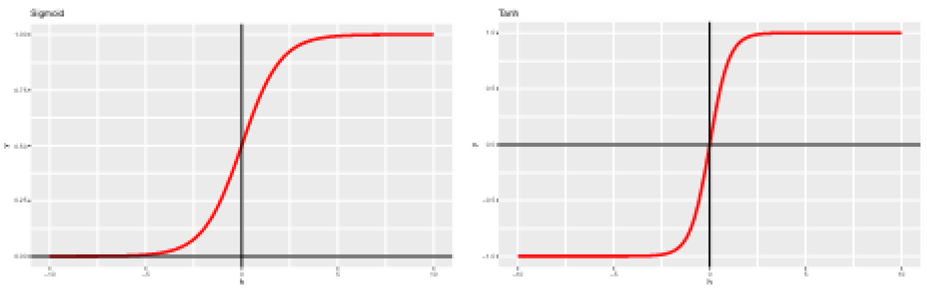


Figure A.1: A neuron.

Appendix B: Activation Functions

An activation function weights the input of the previous unit into the next one. Usually, activation functions are nonlinear, allowing NN to learn nonlinear problems. Many activation functions are available, such as the Rectified Linear Units (ReLU) or the binary activations. However, the most common functions are sigmoid functions, depicted in Figure B.1 (Charte et al. 2018).



Logistic Tanh

Figure B.1: Two Sigmoid Activation Functions, Logistic and Tanh. Figure retrieved from Charte et al. (2018).

The logistic function is a sigmoid function, shown in Equation B.1. Tanh is also a sigmoid function but more symmetrical and with a steeper slope than the logistic function. Equation B.2 shows the tanh formula, which creates stronger gradients and is therefore preferred to the logistic function (LeCun et al. 1998; Charte et al. 2018).

$$s_{sigm}(x) = \sigma(x) = \frac{1}{1 + e^{x}} \tag{B.1}$$

$$s_{tanh}(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{B.2}$$

Appendix C: Backpropagation

The first step to find the weights in a trained part, is to initialise the weights at random, drawing from a uniform distribution on an interval around 0. Afterwards the weights are updated by an optimization algorithm, finding the configuration of the optimal weights. At the core of this process is the backpropagation, choosing the direction of the weight update. Then, it computes the loss function gradient taking into account the weights via: $\eta \frac{\delta J(W)}{\delta W}$. The backpropagation first calculates the gradient of the output layer weights and iterates backward through the layers, applying the chain rule of calculus for gradient computations; hence it is called backpropagation (Goodfellow et al. 2016). In equation C.1, the weight-matrix W^m results from the mth iteration, updated from the previous weight matrix W^{m-1}. Backpropagation is written as:

$$W^{m} \leftarrow W^{m-1} - \eta \frac{\delta J(W)}{\delta W}. \tag{C.1}$$

Appendix D: LSTM Hyperparameter Settings in Keras

Units

The units in the LSTM layer of Keras depict the hidden units. According to the Keras documentation, it determines the dimensionality of the output space, setting the hidden- and cell-state matrix sizes. The bigger the units are, the more capacity is allowed, allowing for more learning. As mentioned in the chapter on the under complete autoencoder, this parameter can be set and prevent overfitting.

Timestamps

Timestamps indicate a time sliding window which is set to create sequences onto the input data before feeding into the LSTM-AE network. This is one of the requirements when engaging with the LSTM layers for time series. The number of timesteps can significantly impact the computation of the reconstruction errors. In general, time series data set time steps at 10, 15, 20, 25, 30, 35, and 40 (Wei et al. 2022). This thesis selects sizes of 10, 15, and 30. The time sliding window at 10 has the best results with the highest accuracy.

Dropout

Dropout regularisation prevents overfitting during the training of the network. During training, the dropout will randomly set outputs of hidden layers to zero. Keras distinguishes between *dropout* and *recurrent_dropout*. Whereby *dropout* is used for the dropout of the input units of the layer. *Recurrent_dropout* is used for the dropout between the recurrent units of the layer.

Batches, epochs, and early stopping

In the training process, a NN engages with much data; this paper has, e.g., 10,000+ data points. Therefore the training data set is divided into a fixed-sized batch series. Each batch is sequentially processed in one epoch; the individual training samples within a batch can be parallelly processed (Goodfellow et al. 2016). A small batch size produces convergence in fewer epochs, while large batch sizes create more data that are processed in parallel, improving efficiency (Devarakonda, Naumov & Garland, 2017). However, mini-batches present a better result in building a model related to the NN, especially in recurrent types, where high sensitivity to minor parameter changes is common (Martens & Sustskever, 2012). Concluded from our experiment, a batch size of 10 rendered the best results compared to other trials.

The optimised algorithm runs according to a predefined amount of epochs. A downside of having many epochs is overfitting. Overfitting leads to a solution that fits the training data but does not generalise on test data. To avoid overfitting, early stopping is applied. Early stopping is a regularisation method that stops the process when no improvement in validation loss is made after a certain amount of epochs, also called patience (Goodfellow et al. 2016).

Repeat vector, Return_sequences and return_state

First, one should understand that the hidden states are LSTM layer outputs. Keras then uses *return sequences*, whereby *false* is the default, stating the output results from the last LSTM hidden state or the last timestep from the current sequence. When setting *return sequences* to *true*, all hidden states from all previous LSTM layer timesteps will be returned. *Repeatvector* is a similar option to *return_sequences = false*, because it repeats the last output of the encoder. See Figure D.1 for a visualisation of repeatvector and return_sequences. Finally, the output of the LSTM layer is defined by '*return_state,'* which returns the h_t output with a false setting and returns the C_t output when it is set to true. When set to true, the output will return h_t of the previous last step, h_t of the previous step (again), and C_t.

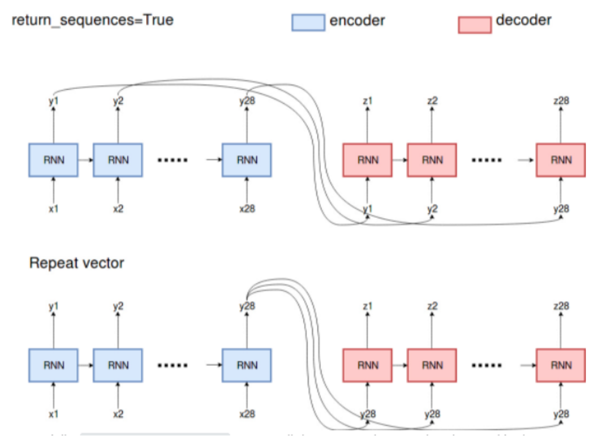
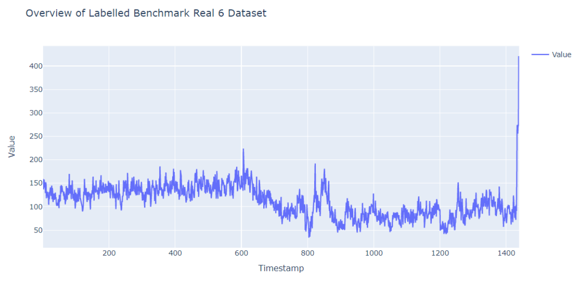
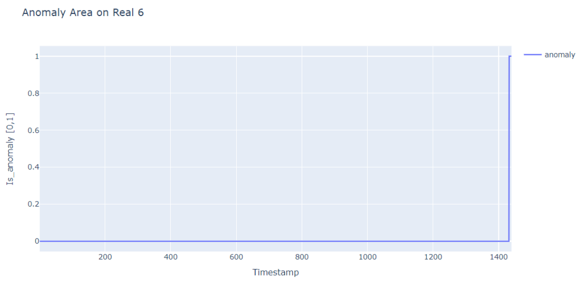


Figure D.1: *Return_sequences=true* and *Repeat vector*

Appendix E: Overview of Yahoo Real Series and Anomalies

Real 6 Series



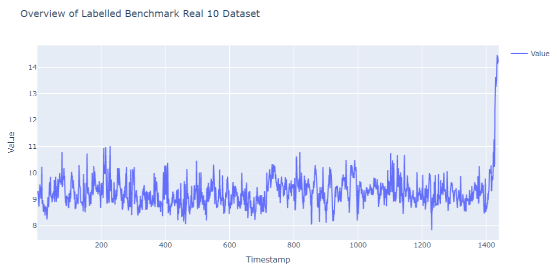


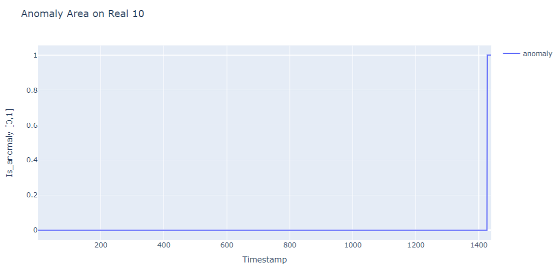
I. Overview of Real 6 Series

II. Overview of anomaly location in Real 6 Series

Figure E.1: A Graphical Representation of Real 6 Series' I. Overview of Series II. Anomaly Overview

Real 10 Series





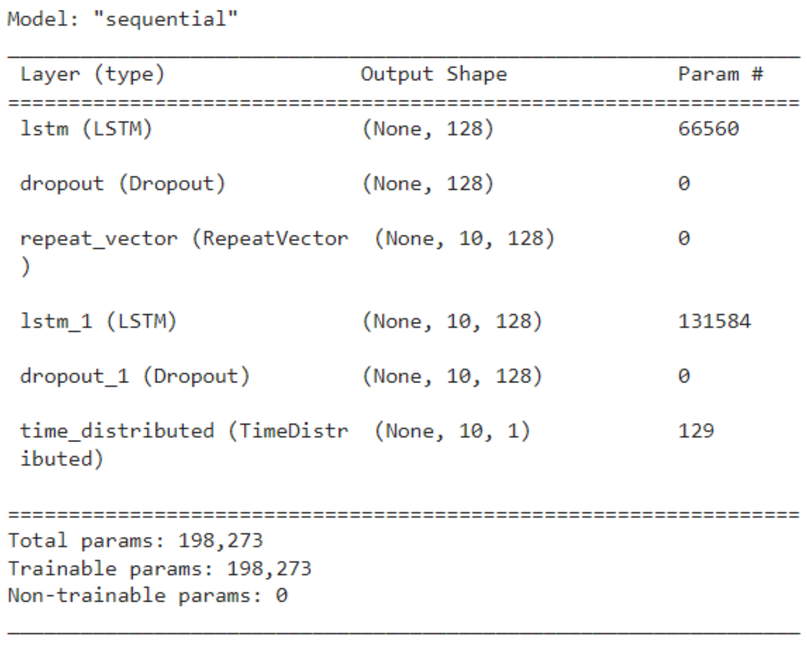
I. Overview of Real 10 Series

II. Overview of anomaly location in Real 10 Series

Figure E.2: A Graphical Representation of Real 10 Series' I. Overview of Series II. Anomaly Overview

Appendix F: Parameters, Model Fit and Reconstruction Error

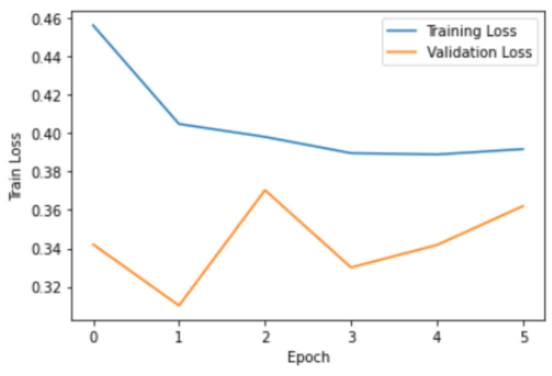
Parameters



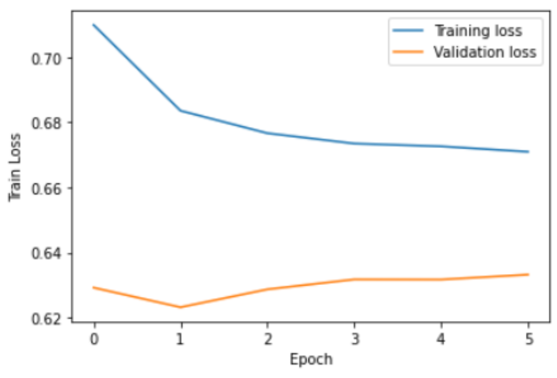
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 128)               66560

 dropout (Dropout)           (None, 128)               0

 repeat_vector (RepeatVector  (None, 10, 128)          0
 )

 lstm_1 (LSTM)               (None, 10, 128)           131584

 dropout_1 (Dropout)         (None, 10, 128)           0

 time_distributed (TimeDistr  (None, 10, 1)            129
 ibuted)

=================================================================
Total params: 198,273
Trainable params: 198,273
Non-trainable params: 0
_________________________________________________________________
```

Figure F.1: Yahoo Series Parameters for LSTM-AE

Model Fit



I. Model Fit Real 6 Series

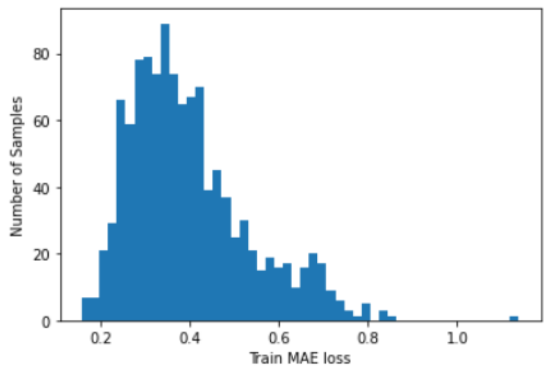


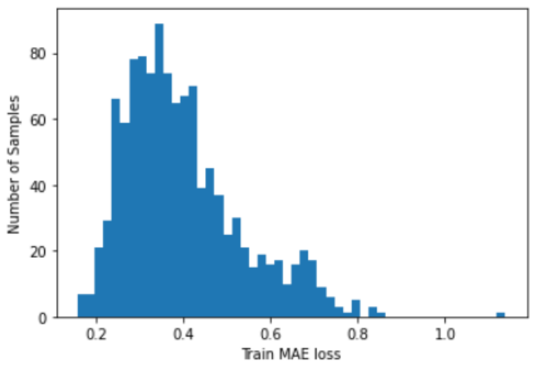
II. Model Fit Real 10 Series

Figure F.2: Model Fit for Yahoo Series I. Real 6 II. Real 10

Reconstruction Error

Real 6 Series

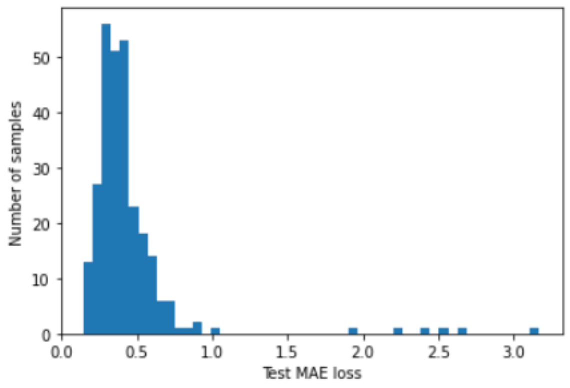


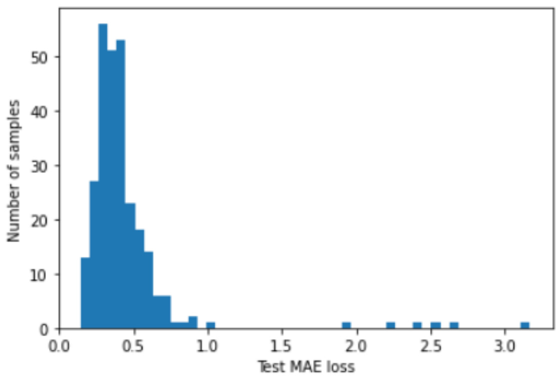


I. 99.5% threshold

II. 99.9% threshold

Figure F.3: A Graphical Representation of Real 6 Series' Reconstruction Error Train MAE Loss for I. 99.5% Threshold II. 99.9% Threshold





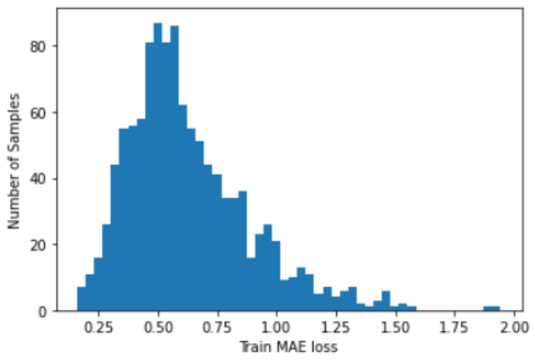
I. 99.5% Threshold

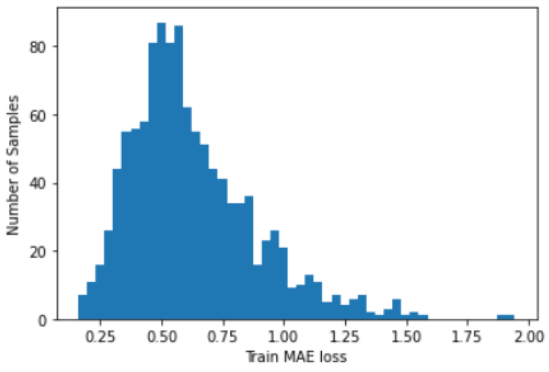
II. 99.9% Threshold

Figure F.4: A Graphical Representation of Real 6 Series' Reconstruction Error Test MAE Loss for I. 99.5% Threshold II. 99.9% Threshold

Reconstruction Error

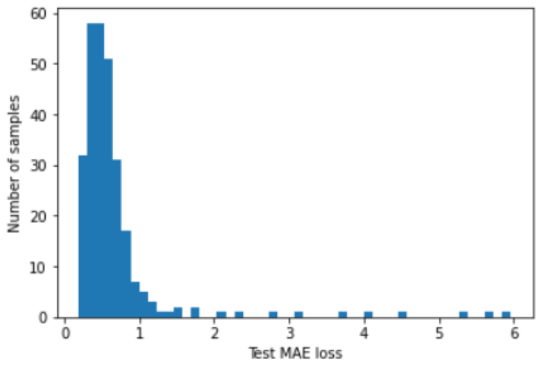
Real 10 Series

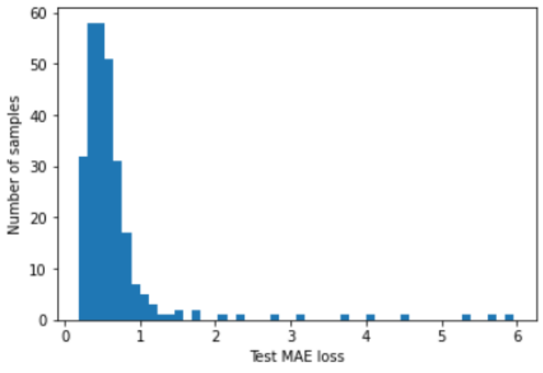




I. 99.5% threshold   II. 99.9% threshold

Figure F.5: A graphical representation of Real 10 Series' Reconstruction error Train MAE loss for I. 99.5% threshold II. 99.9% threshold





I. 99.5% threshold   II. 99.9% threshold

Figure F.6: A graphical representation of Real 10 Series' Reconstruction error Test MAE loss for I. 99.5% threshold II. 99.9% threshold

Appendix G: Zoom-In Results Humidity Series 1217 & 2595

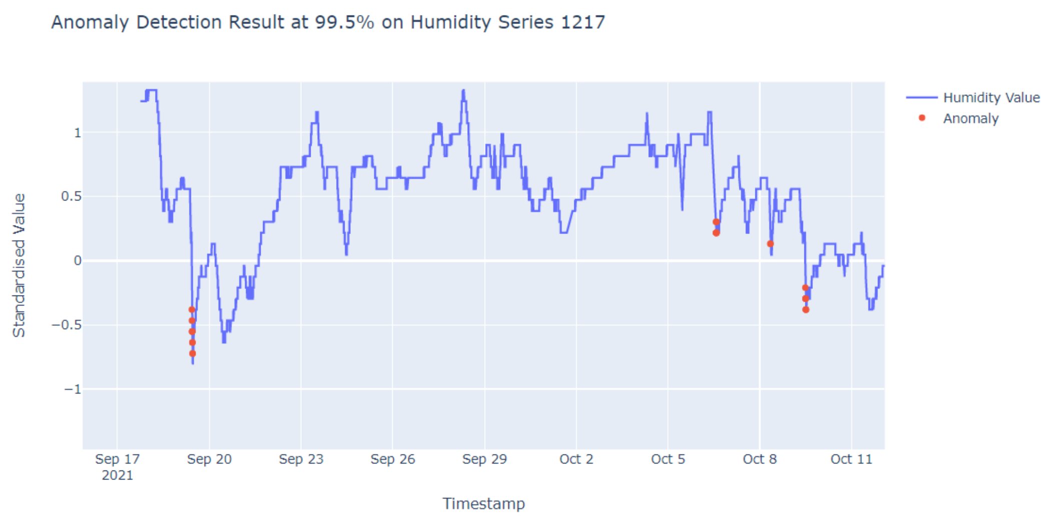


Figure G.1: Zoom-In on Collective Anomalies of Humidity Series 1217

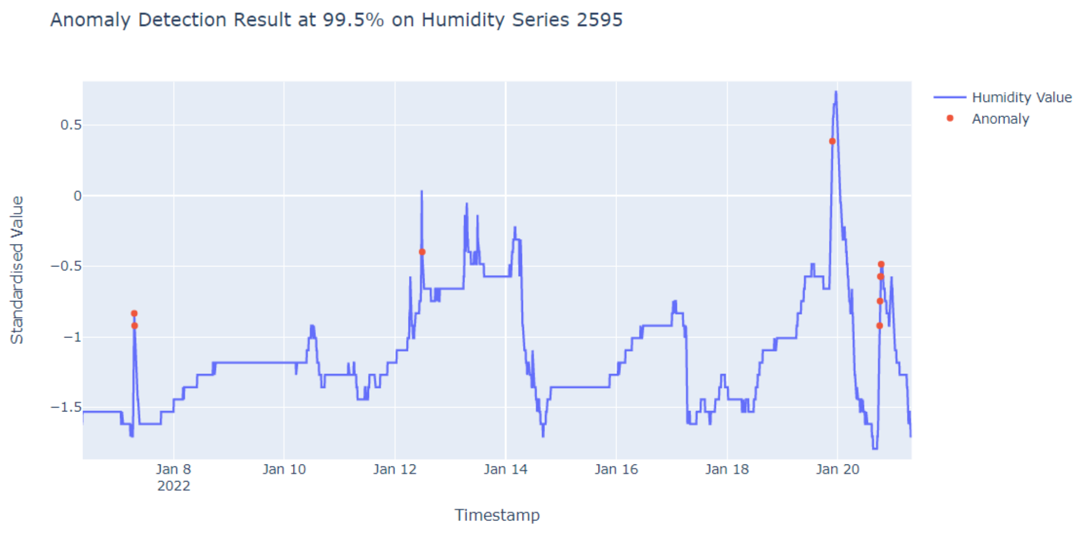


Figure G.2: Zoom-In on Collective Anomalies of Humidity Series 2595