Tampere University of Applied Sciences

Mobile Machine Anomaly Detection in Container Handling Operations

Mikko Heikkilä

MASTER'S THESIS
December 2023

Master's Degree Programme in Data Expertise and Artificial Intelligence

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Master’s Degree Programme in Data Expertise and Artificial Intelligence

HEIKKILÄ, MIKKO
Mobile Machine Anomaly Detection in Container Handling Operations

Master's thesis 50 pages, appendices 2 pages
December 2023

The objective of this Master’s thesis is to research Deep Learning (DL) based anomaly detection methods for unlabeled time series data in container handling operations. Detected anomalies can be the sign of a defect in the container handling equipment. Predictive maintenance aims to detect and prevent failures in industrial equipment by analyzing Key Performance Indicator (KPI) data and identifying anomalies that indicate potential issues or malfunctions.

This Master’s thesis examines forecasting based deep learning methods with convolutional and long short-term memory layers for detecting anomalies from onboard control system data. The final goal of the work is to obtain a method to identify signs of abnormal steering behavior.

The thesis presents the background of the problem in the introduction with the research questions. The thesis also covers the test setup of the experiments along with the training and testing of the selected deep learning method.

The results show that it is possible to detect anomalies by using the selected method.

Key words: deep learning, predictive maintenance, anomaly detection

CONTENTS

SPECIAL VOCABULARY

ANN	Artificial Neural Network
CBM	Condition Based Maintenance
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
DOF	Degrees of Freedom
KPI	Key Performance Indicator
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
MTSAD	Multivariate Time Series Anomaly Detection
RNN	Recurrent Neural Network
PDM	Predictive Maintenance
PVM	Preventive Maintenance
SAE	Sparse Auto Encoder
TCN	Temporal Convolutional Network
TFT	Temporal Fusion Transformers

1 INTRODUCTION

1.1 Motivation

Container terminals are critical hubs of the global logistics system and they play a crucial role in facilitating international trade and commerce. Containers are transferred from ships to storage areas, trucks and trains with heavy machinery in daily operations. In recent years, there has been an increasing trend to automate the container handling fully or partially. Automated container handling equipment can accurately move and stack containers in marine terminals in a safe manner.

Automated equipment, however, requires regular maintenance in order to function properly and effectively. Equipment failure while serving a vessel can be costly and might even partially halt the operations. Therefore, regular or preventive maintenance of automated container handling equipment is essential for ensuring the reliable and efficient operation of container terminals and minimizing disruption.

Predictive maintenance techniques such as Machine Learning (ML) can help to identify potential issues in the onboard sub-systems that may not be detected through routine inspections. This technique allows targeted maintenance activities between normal scheduled ones. The anomalies are a sign of abnormal behavior, indicating a need for a system inspection before becoming an issue for the operations.

1.2 State of Art

The field of anomaly detection for time-series data has been a significant area of research for long time. Early studies on anomaly detection methods primarily focused on statistical approaches. However, in recent years, there has been a surge in the development of machine learning algorithms specifically designed for detecting anomalies in time-series data.

Abnormal system behavior detection using ML based methods for time series data have gained significant attention due to their ability to capture complex patterns and temporal dependencies. These methods have also shown great promise in the field of predictive maintenance of different industrial equipment.

Anomalies in the data can indicate either a past or potential future fault. There are different approaches to identifying anomalies, but in this thesis, focus is on the DL forecasting approach. This method involves using control system and sensor data to predict subsequent values of lateral control point error and comparing the deviation against a threshold. Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) can be utilized for this purpose. In this thesis, both models are tested in the experiments section.

1.3 **Objectives and Research Questions**

The main task in this thesis is to evaluate two different DL models for sequential data from container handling equipment and their ability to distinguish abnormal data points. After training the models, a suitable threshold limit was chosen for the anomaly detection. There are two research questions for the thesis based on the objectives:

First, main question: “*Can a Deep Neural Network detect anomalies from unlabeled control system and sensor data of a container handling equipment?”*

Second, subquestion: “*Could Machine Learning be utilized for predictive maintenance in a container handling equipment?*”

2 INTRODUCTION TO MAINTENANCE STRATEGIES

Maintenance strategies can be classified into four distinct categories, each differing in terms of complexity and effectiveness (Susto, G et al. 2012).

Run-to-failure (R2F) maintenance approach involves conducting repairs or restorative actions only after a failure has occurred. It is the simplest form of maintenance management but often leads to increased costs due to the large number of defective products resulting from the failure.

Preventive maintenance (PVM) or scheduled maintenance strategy, maintenance activities are performed periodically on a planned schedule, aiming to anticipate process or equipment failures. This approach aims to prevent failures, but it may also result in unnecessary maintenance actions being carried out at times.

In Condition-based maintenance (CBM) the maintenance actions are initiated after observing specific conditions indicating a degradation of the process or equipment. This approach relies on continuous monitoring of the machine or process health, enabling maintenance actions to be performed only when they are genuinely required. The drawback of CBM is that maintenance activities cannot be planned in advance.

Predictive maintenance (PDM) or statistical-based maintenance is similar to CBM, where maintenance actions are carried out only when necessary. However, PDM incorporates prediction tools to determine when such actions are likely to be required, allowing for the implementation of planning and scheduling schemes. PDM systems often employ custom-defined health factors or statistical inference methods to facilitate decision-making.

Among statistical inference-based methods, ML and DL approaches have been shown to provide increasingly effective solutions for predictive maintenance (Zhang, W et al. 2019)

3 ANOMALY DEFINITION AND DETECTION METHODS

In the field of data-analysis , anomaly detection, known also as outlier detection, is usually understood as identification of rare items, events or observations which are deviating significantly from the majority of the data and do not conform to a well-defined notion of normal. Hence, anomaly detection is a task of identifying the rare items, events, or observations that raise suspicions by differing significantly from the majority of the data. There are three broad categories of anomaly detection techniques that exist (Chandola, V et al.2009).

Unsupervised anomaly detection techniques identify anomalies in an unlabeled test dataset by assuming that the majority of instances in the dataset are normal. These methods seek out instances that appear to deviate the most from the rest of the data.

Supervised anomaly detection methods requires a dataset that has been labeled as “normal” and “abnormal” and involves training a classifier.

Semi-supervised anomaly detection methods constructs a model representing normal behavior from a given normal training dataset, and then tests the likelihood of a test instance to be generated by the learned model.

The reporting of anomalies is also an important aspect of any anomaly detection technique, and it typically falls into two main categories (Chandola, V et al.2009)

Scoring techniques assign an anomaly score to each instance in the test data, indicating the extent to which that instance is considered anomalous. Consequently, the output of such techniques is a ranked list of anomalies. Domain expert can choose to focus on analyzing the top few anomalies or apply a predefined cutoff threshold to select the anomalies deemed most significant.

In labeling technique a specific label is assigned, either "normal" or "anomalous," to each test instance. While scoring-based anomaly detection techniques allow analysts to use a domain-specific threshold to select relevant anomalies, techniques that provide binary labels do not offer direct control over this selection process. However, the expert can indirectly influence this choice through parameter settings within each technique.

3.1 **Time Series**

A time series represents a sequence of observations ordered chronologically, with data points typically collected at uniform time intervals (Yufeng, Yu et al. 2014). This type of data can be categorized as either univariate or multivariate time series. In the context of univariate time series, only one variable changes over time. For instance, a straightforward illustration of this is a sensor recording temperature every second. Conversely, a multivariate time series involves the inclusion of multiple time-dependent variables. These variables, beyond their temporal dependence, can exhibit varying levels of correlation with each other and in many cases encompass nonlinearity. Alternatively viewed, a multivariate time series is a composite of distinct univariate time series, interrelated to varying extents. An instance of such a multivariate time series might arise from a tri-axial accelerometer generating three-dimensional observations for each axis (x, y, z) every second. Another example of multivariate time series data is the data recorded from mobile machine onboard control system, which forms the basis for the experiments in this thesis.

A time series is deemed stationary when its statistical characteristics remain consistent over time. This means that key attributes like the mean, variance, and autocorrelation structure of the series don't change as time progresses. Stationarity holds significance because various traditional statistical techniques designed for time series analysis rely on this property. However, in real-world scenarios, time series data is usually non-stationary, non-linear and dynamically evolving. Hence, deep learning models should be able to detect anomalies in real time (Chalapathy, R et al. 2019).

After general description of time series a more formal presentation of is provided. A time series $X = \{(t) \mid 1 \leqslant t \leqslant m\}$, unfolds as a series of d-dimensional observation vectors $(t) = (x1(t), x2(t), \ldots xd(t))$ arranged in chronological order. These observations are often gathered at uniform, discrete time intervals. When $d = 1$, it's referred to as a univariate time series, and when $d > 1$, it becomes a multivariate time series (Yufeng, Y et al. 2014).

3.2 Type of Anomaly

In literature, types of anomalies in univariate and multivariate time series are generally categorized into 3 different groups, which are (1) Point anomalies, (2) Contextual Anomalies and (3) Collective Anomalies (Chalapathy, R et al. 2019)
A point anomaly arises when a single sensor reading significantly diverges from the rest of the data points. This can happen for example due to a measurement beyond the expected range. These anomalies are often caused by unreliable or faulty sensors, data recording errors, or localized operational irregularities.
Contextual anomalies occur when a sensor reading stands out as anomalous only within a specific context. In isolation, the measured value might seem normal, but it becomes anomalous when compared to preceding or subsequent readings.
Collective anomalies involve a subset of sensor measurements behaving distinctively compared to other subsets. This category is characterized by patterns where a portion of the data follows different behavior than the larger set (Belay, M et al.2023) A visual representation of these anomaly categories are depicted in Figure 1.

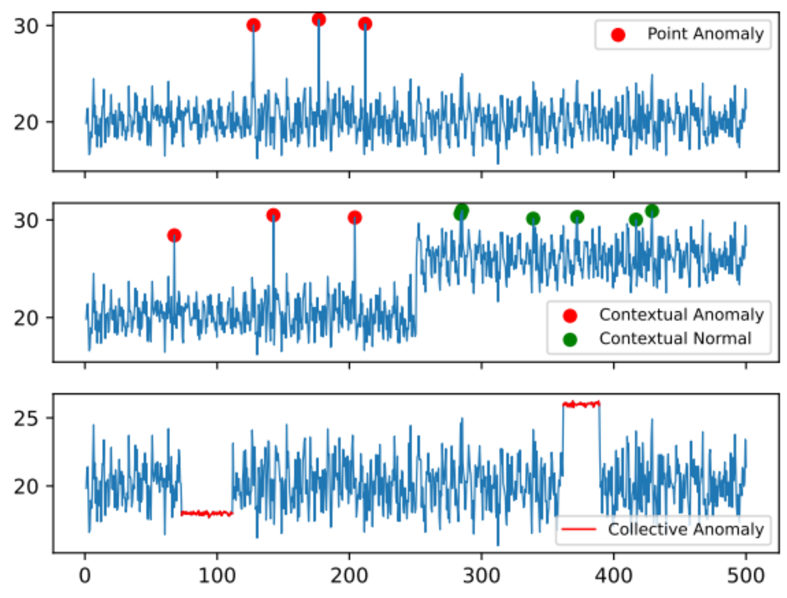


FIGURE 1. Anomalies in time series data (Belay, M et al.2023)

4 ANOMALY DETECTION AND DEEP LEARNING

Unsupervised Multivariate Time Series Anomaly Detection (MTSAD) methods can be classified based on the foundational approach they adopt to identify anomalies within the data. A majority of these methods can be categorized within one of the three fundamental approaches as depicted in Figure 2. These approaches include the reconstruction approach, prediction approach, and compression approach.

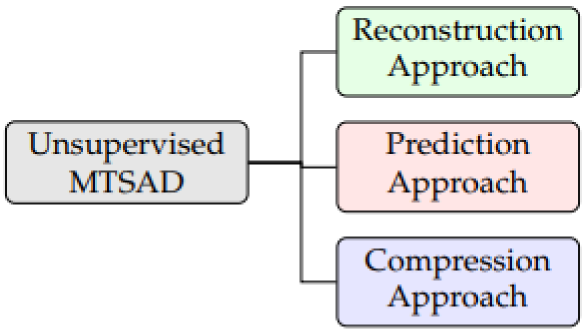


FIGURE 2. Categories of unsupervised MTSAD (Belay, M et al.2023)

Within reconstruction-based methodologies, the procedure involves compressing a training set of multidimensional time series data into a lower-dimensional latent space, followed by its subsequent reconstruction back to its original dimensions. These approaches rest on the assumption that anomalies exhibit suboptimal reconstruction. Hence, the degree of reconstruction error or the probability of reconstruction is utilized as an indicator of anomalousness. Autoencoders are often utilized for anomaly detection by learning to reconstruct a given input and the model is trained exclusively on normal data (Zhang, Y et al. 2023)

Compression methods, similar to reconstruction techniques, encode time series segments into a low-dimensional latent space. However, unlike reconstruction, anomaly scores are directly computed in the latent space. Dimensionality reduction methods decrease computation time and can also serve to reduce model complexity and prevent overfitting. If the dimensions of the latent space are uncorrelated, compression methods allow for the utilization of univariate analysis techniques without neglecting dependencies among variables in the original data (Belay, M et al.2023).

Prediction-based approach revolves around leveraging both present and past values (typically gathered from a finite sliding window) to predict forthcoming single or multiple time steps. By comparing the predicted data points against the actual observed values, an anomaly score is formulated based on the extent of disparity and notable deviations between the projected and real data signify an anomaly. RNN, LSTM and CNN models are typically used for sequence prediction (Zhang, Y et al. 2023).

Next, this chapter explores how different deep learning networks have been utilized for the task of anomaly detection in time series data. Finally, proposed method for the experiments is introduced based on the literature review.

4.1 **Literature Review**

DeepAnT, introduced by Munir, M et all, is an anomaly detection model that uses CNN for predicting future values in timeseries. It stands out for being good at spotting irregularities in both simple and more complex time series data, doing better than older methods based on density and distance. According to the authors, DeepAnT displays the ability to identify various anomalies in time series data, including point anomalies, contextual anomalies, and discord in an unsupervised setting. Unlike other methods that directly learn anomalies, DeepAnT takes a different approach. It leverages unlabeled data to grasp and understand the overall data distribution, which then informs its predictions about the regular behavior of a time series.

Another interesting case presented by Bartosz, P et all utilized LSTM-based method for early failure detection in production process where clutch shaft misalignment is detected by using LSTM deep learning algorithm. The study involved an analysis of vibration acceleration measurements taken on a bearing node within a propulsion engine connected to a four-wheel clutch via a shaft. The shaft, which has three bearing nodes, experienced a fault when the middle bearing was intentionally shifted 5 mm away from the shaft axis, causing misalignment within the system. The main objective in the concept study was to model the time of failure based on fault data collected from the system.

In a research article by Davari, N et all. DL based sparse autoencoder was utilized for predictive maintenance of air compressor in a metro train. In the paper a data driven PDM framework was implemented to detect failures on air compression unit in timely manner while the train is in operation. During the training phase of the network, only a dataset consisting of normal data is used as input. The network is trained to understand and learn the patterns and characteristics of this normal data. Subsequently, in the testing phase, when the network encounters new data, it employs trained knowledge to distinguish between normal and abnormal instances.

Kiavash, F et al. tested a PDM method based on autoencoder and convolutional layers for a 3-degree-of-freedom (3 DoF) robot that is used for picking and placing parts in a factory. The proposed architecture incorporates the strengths of both semi-supervised training and feature extraction capabilities of AEs along with the CNNs to capture spatial information within the provided signal sequences. By combining these techniques, the architecture aims to enhance the overall performance of the system.

Finally, Safavi, S et all. proposed a fault detection, isolation, identification and prediction architecture for autonomous vehicles. In summary, the research introduced a novel method for predicting the health status of electronic sensors in autonomous vehicles. It utilized a CNN classifier for real-time monitoring and construction of a health index. The evolution of this index was then used to forecast sensor health using a Temporal Fusion Transformer network (TFT). Additionally, a feature extraction and Deep Neural Network (DNN) classification technique was proposed for identifying specific fault types. The method was evaluated using a real-world dataset from car manufacturer Audi, providing a practical validation of its effectiveness. The fault detection system achieved an impressive accuracy of 99.84% in detecting faults.

In summary, Auto-Encoders, CNNs, RNNs (including LSTMs), and their combinations have been extensively used for analyzing time series data and performing predictive maintenance or anomaly detection tasks. These neural network architectures have demonstrated their ability to capture spatial and temporal dependencies, making them valuable tools for a wide range of applications in these domains.

In this chapter, five different cases were introduced. Based on literature review, the proposed method for the thesis is based on DeepAnT, which seems like a good choice for multivariate timeseries forecasting and anomaly detection task by using unlabeled data. As a comparison, LSTM based model is also tested to see differences in the performance and training time.

4.2 **Methodological Basis**

This chapter provides an overview of various Neural Network models and their distinctive characteristics. The chapter begins with a presentation of the theoretical foundations of Artificial Neural Networks, activation functions and optimization algorithms, followed by CNN, RNN and then introducing LSTM, which is a type of RNN.

4.3 **Overview of Artificial Neural Network Models**

Artificial Neural Network (ANN) structures were created based on existing models of biological nervous systems and the human brain. The essential building blocks of these networks are artificial neurons, which serve as simplified versions of biological neurons. The design of artificial neurons is inspired by studying how a cell membrane in a biological neuron generates and transmits electrical impulses.

Artificial neurons are interconnected through numerous artificial synapses, which are represented by vectors and matrices of synaptic weights. These connections enable the flow of information and play an important role in the network's ability to learn, process data, and make predictions based on the acquired knowledge (Silva, I et al. 2017)

The simplest artificial neuron is called a perceptron. It outputs a binary value **y** (either 0 or 1), based on the dot product of a real-valued input vector **x** and a vector of weights **w**, summed with a bias **b** as visualized in Figure 3 (Nielsen, M. 2015)

In mathematical terms, the internal computations carried out by the perceptron can be presented using the following expressions:

$$Output\ y = \begin{cases} 0 \ if \ \ w \cdot x + b \ \leq 0 \\ 1 \ if \ \ w \cdot x + b \ > 0 \end{cases} \tag{1}$$

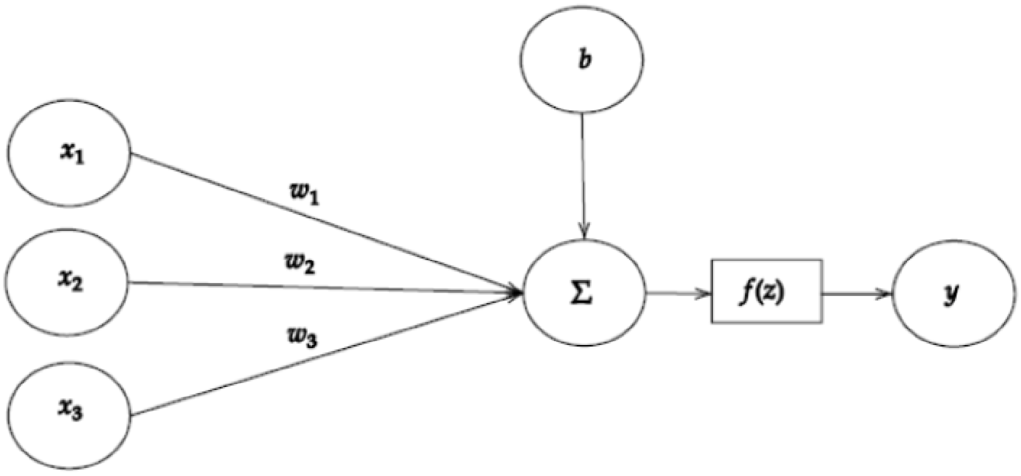


FIGURE 3. Perceptron model (Nielsen, M. 2015)

Feedforward neural networks or Multi Layer Perceptrons (MLPs) represent a fundamental class of deep learning models. The primary objective of a feedforward network is to approximate a specific function denoted as f*. The term "feedforward" is used because the information flows strictly in one direction, starting from the input x, passing through intermediate computations that define the function f, and culminating in the output y. In this type of network, there are no feedback connections, meaning the model does not have outputs that loop back into itself during the evaluation process (Goodfellow et al. 2016).

Figure 4 shows visual representation of an ANN. In this network, the nodes receive input signals from the previous layer, and when the accumulated input surpasses a certain threshold, the nodes are activated (Goodfellow et al. 2016)

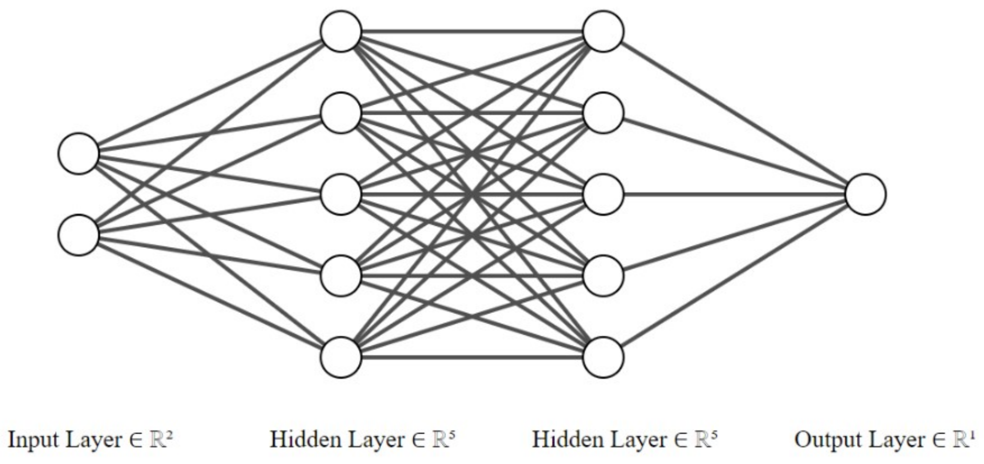

Input Layer $\in \mathbb{R}^2$
Hidden Layer $\in \mathbb{R}^5$
Hidden Layer $\in \mathbb{R}^5$
Output Layer $\in \mathbb{R}^1$

FIGURE 4. Deep Neural Network with 4 layers (Goodfellow et al. 2016)

In general, an artificial neural network can be divided into three parts, referred to as layers.

Input Layer:
The input layer plays a crucial role in receiving information such as data, signals, features or measurements from the external environment. These inputs, often represented as samples or patterns, are typically normalized within certain pre-defined limits using activation functions. This normalization enhances the numerical precision of the mathematical operations performed by the network.

Hidden Layers:
Hidden layers are the intermediate layers within the neural network, and they consist of neurons responsible for identifying and extracting patterns associated with the analyzed process or system. These layers perform the majority of the internal processing within the network, facilitating complex computations and feature extraction.

Output Layer:
The output layer, similar to the hidden layers, is composed of neurons and serves the critical function of producing and presenting the final network outputs.

These outputs result from the processing carried out by the neurons in the previous layers and represent the network's conclusions or predictions based on the input data.

In addition to these three primary layers, artificial neural networks can have various architectures, characterized by the arrangement and interconnections of neurons and layers (Silva et al. 2017)

4.4 Activation Functions

In an MLP network, the final output is determined through the use of various activation functions (AF), also referred to as transfer functions. Activation functions are functions used to compute the weighted sum of input and biases. Each activation function serves a specific purpose in processing the neuron's input and producing the desired output for different types of tasks in the network. Figure 5 illustrates several commonly used activation functions, among which are popular choices like ReLU (Rectified Linear Unit), Hyperbolic Tangent (Tanh), and Sigmoid/Logistic.

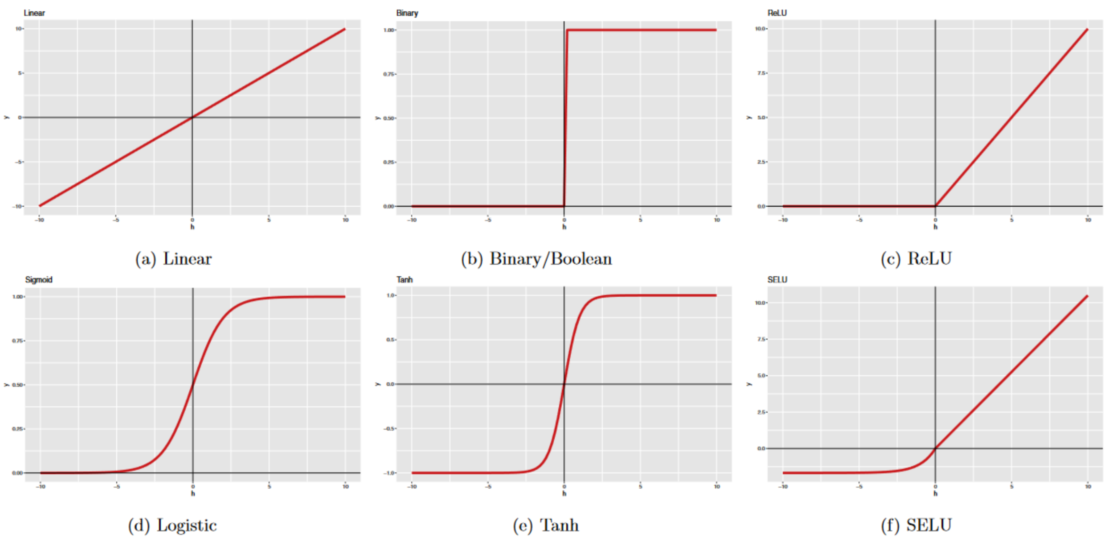


FIGURE 5. Common activation functions (Nwankpa et al. 2018)

Activation functions can be either linear or non-linear, depending on the function they represent. Linear activation functions simply pass the input to the next layer, while non-linear activation functions transform the input in a way that allows the network to learn more complex relationships. (Nwankpa et al. 2018)

Next, some activation functions are presented in more detail, along with typical use cases. It's worth noting that improved variants of the most common activation functions exist, but only one of them will be discussed here, namely the Leaky ReLU.

The Sigmoid activation function, also known as the logistic function, with output values range of 0 to 1. This non-linear activation function is primarily applied in feedforward neural networks. It is a differentiable real function with bounds, defined for real input values, featuring positive derivatives everywhere and a certain level of smoothness. The mathematical expression for the Sigmoid function is given by the relationship:

$$f\,(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

The sigmoid function is commonly employed in the output layers of deep learning architectures. Its main application lies in predicting probability-based outputs, making it particularly useful in binary classification problems and logistic regression tasks.

The hyperbolic tangent function (tanh), is another type of activation function widely used in deep learning. This function is known for its smoother and zero-centered nature, with output values ranging from -1 to 1. The mathematical expression for the tanh function is as follows:

$$f\,(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3}$$

The tanh function has been shown to give better training performance for multi-layer neural networks than the sigmoid function and are widely used inRNN's, particularly in tasks related to natural language processing and speech recognition. (Nwankpa et al. 2018)

The Rectified Linear Unit (ReLU) activation function was introduced by Nair and Hinton in 2010, and since then, it has become the most widely used activation function in deep learning applications, consistently delivering state-of-the-art results. ReLU is known for its faster learning capabilities and has proven to be highly successful and popular in various domains. When compared to the Sigmoid and tanh activation functions, ReLU outperforms them, providing better performance and generalization in deep learning tasks.

One of the key advantages of ReLU is that it maintains nearly linear behavior, which preserves the properties of linear models and simplifies the optimization process with gradient-descent methods.

The ReLU activation function operates as a threshold operation on each input element, setting values less than zero to zero. This mathematical expression for ReLU is given by:

$$f\,(x) = \max(0, x) = \begin{cases} x_i & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases} \tag{4}$$

The rectified linear unit (ReLU) function effectively rectifies input values that are less than zero, forcing them to zero and thereby addressing the vanishing gradient problem encountered in earlier activation functions. ReLU has found extensive use within the hidden units of deep neural networks, often combined with another activation function in the output layers for tasks like object classification and speech recognition.

One of the main advantages of using ReLU in computations is its faster performance, as it avoids expensive exponential and division calculations, resulting in enhanced overall computation speed. ReLU also introduces sparsity in the hidden units by compressing the values between zero and the maximum value.

However, ReLU has its limitations. It is prone to overfitting compared to the sigmoid function. Nonetheless, researchers have employed techniques like dropout function to mitigate the overfitting effects and improve the performance of deep neural networks.

Despite its advantages, ReLU has a significant drawback. During training, it can become fragile, causing some gradients to vanish and resulting in the death of certain neurons. These "dead" neurons lead to weight updates not activating in future data points, hampering the learning process as these neurons effectively give zero activation. To address this issue, the leaky ReLU was proposed.

Leaky ReLU (LReLU) was introduced in 2013 as an activation function designed to address the dead neuron problem of the standard ReLU. The primary purpose of LReLU is to keep the weight updates active throughout the entire propagation process during training.

To achieve this, LReLU introduces a small negative slope, represented by the parameter alpha (α), for input values less than zero. By incorporating this small negative gradient, LReLU ensures that the gradients are not completely zero at any point during training. The value of alpha is typically set to a very small constant, often around 0.01, which allows for a slight, non-zero gradient for negative inputs.

The mathematical expression for the LReLU activation function is given as follows:

$$f(x) = \alpha\mathrm{x} + \mathrm{x} = \begin{cases} x & if\ x > 0 \\ \alpha x, & if\ x \leq 0 \end{cases} \tag{5}$$

This small but essential modification enables LReLU to overcome the dead neuron issue and facilitate improved learning in deep neural networks.

Finally, the softmax function that is used in neural computing to compute a probability distribution from a vector of real numbers. The softmax function produces an output vector of probabilities, where each element of the vector represents the probability that the input vector belongs to a particular class. The sum of the probabilities in the output vector is always equal to 1.

The softmax formula is computed using the following relationship:

$$f(x)_i = \frac{e\ (x_i\ )}{\sum_j e\left(x_j\right)} \tag{6}$$

The softmax function is used in multi-class classification models to return a probability distribution over the different classes. The class with the highest probability is then the class that the model predicts the input belongs to. The main difference between the sigmoid and softmax activation functions is that the sigmoid function is used for binary classification, while the softmax function is used for multi-class classification (Nwankpa et al. 2018).

4.5 **Cost Functions and Optimization Algorithms**

The cost function J(θ) is a function in machine learning that is used to assess the overall accuracy of a model's estimations during the training process. It quantifies the "cost" or "error" arising from the disparity between the model's predicted outputs and the actual known outputs, requiring access to the true output values of the data.

During training, the model's estimated outputs and the corresponding known data are input into the cost function, which then computes the cost for the model by summing up the errors between each individual prediction and the true value. Essentially, the cost function evaluates how well the model is performing on the given dataset, and the ultimate goal is to minimize this cost to improve the model's predictive capabilities.

It's important to differentiate between cost functions and loss functions. While the term "cost function" is linked to the overall estimation error of the model with a specific dataset, "loss functions" focus on the error of a single output prediction. Both are related concepts but have distinct scopes and purposes

There are multiple ways to compute the errors between the model's estimates and the known data, and the choice of the appropriate cost function largely depends on the specific use case and the nature of the problem being solved. By selecting the right cost function tailored to the task at hand, the model's accuracy and performance can be significantly improved, leading to more reliable predictions and better outcomes (Bishop C. 2006), (Allwright S. 2023).

Mean Squared Error (MSE) is a widely used evaluation metric for regression models. It is calculated by taking the average of the squared differences between the predicted values and the actual target values.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2 \tag{7}$$

The Mean Absolute Error (MAE) is another commonly used metric in regression analysis. It quantifies the average absolute difference between the actual target values and the corresponding predicted values for each data point in the dataset.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}| \tag{8}$$

In the context of machine learning, the objective is to minimize the cost function, which serves as a measure of the model's performance. To achieve this, the weights and biases in the neural network need to be adjusted (Nielsen, M. 2015)

One approach to tackle this optimization problem is to compute the gradient of the cost function. This gradient consists of partial derivatives with respect to the weights and biases of the network, indicating how small changes in each unit of the network would impact the output. The gradient provides both the direction and magnitude of the steepest ascent in the cost function's landscape.

To move towards local minima of the cost function, an iterative process is adopted, where the weights and biases are updated by taking steps in the negative direction of the gradient as illustrated in Figure 6. The size of these steps is controlled by the learning rate, determining the magnitude of adjustments. Selecting an appropriate learning rate is essential as a large learning rate may overshoot the minimum, while a very small learning rate may result in slow convergence (Géron, A. 2019).

This weight and bias update algorithm, known as Gradient Descent GD, aids the model in iteratively refining its parameters. By repeatedly adjusting the parameters based on the gradient, the model progressively approaches a state where the cost function is minimized i.e. until the network converges, meaning that the error is no longer decreasing. The process of obtaining the model parameters in the direction of maximum variation involves adjusting the parameter values as follows:

$$\theta = \theta - \eta . \nabla_{\theta} J(\theta) \tag{9}$$

Where θ = model parameters, η = learning rate and $\nabla_{\theta} J$ gradient vector of the cost function (Madani. A, 2022).

There are a number of different optimization approaches besides GD that can be used with backpropagation, such as Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) (Sarker, 2021).

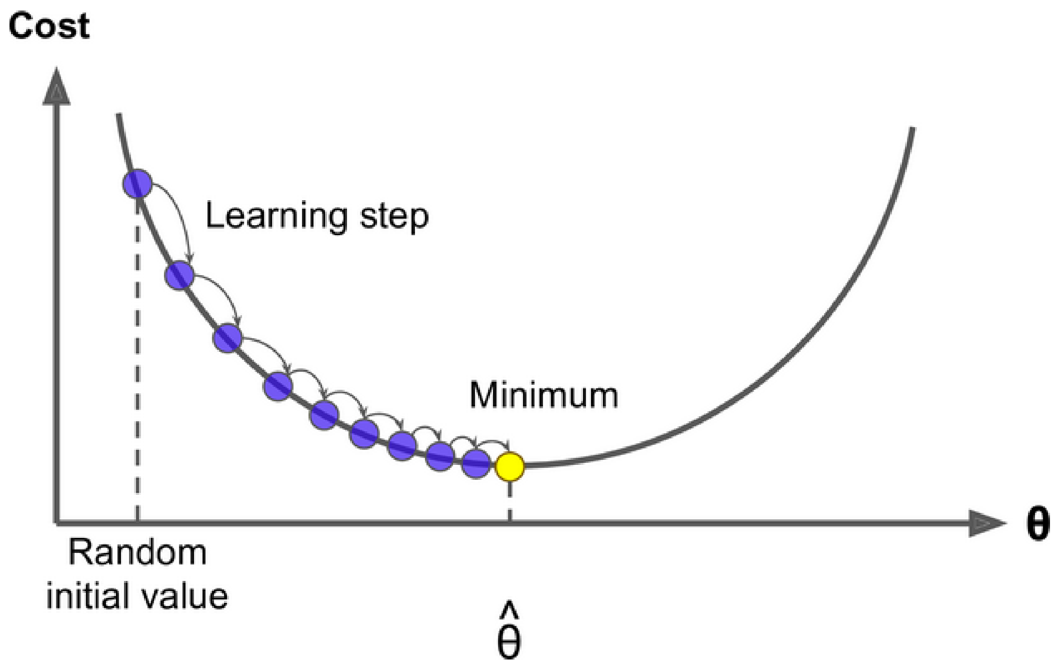


FIGURE 6. Gradient Descent algorithm (Géron, A. 2019)

The gradient, which indicates the direction and magnitude of the cost function's steepest decrease, is computed using a technique called Backpropagation. This algorithm involves moving backward in the neural network to calculate the partial derivatives of the cost function with respect to the network's parameters.

To apply Backpropagation, a training example is fed into the neural network, and its output is computed through forward propagation. After obtaining the output, the cost function is computed to measure the prediction error. Then, the Back-propagation algorithm is employed to calculate the gradients, which involves propagating backward through the neural network graph following the chain rule (Nielsen, M. 2015)

In regular GD, the weights are updated after forward propagating each individual sample and computing the cost, making the training process slow. To speed up training, Stochastic Gradient Descent (SGD) is often used. Instead of summing the costs of all training examples in the dataset, SGD randomly selects a batch of examples from the training set. The gradient descent algorithm is then applied using the average of the gradients calculated for that batch. This approach accelerates the weight updates and helps the neural network learn more efficiently (Zhang, A et al. 2021)
Adaptive Moment Estimation (Adam) is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. Unlike Gradient Descent (GD) and Stochastic Gradient Descent (SGD), which have a fixed learning rate throughout the training process, Adam uses adaptive learning rates that can change during training which helps to speed up the training process and enhances the model's ability to find an optimal solution effectively (Kingma & Ba, 2015)

4.6 **Convolutional Neural Networks**

Convolutional neural networks (CNNs) are a type of neural network that are specifically designed for processing data that has a grid-like topology. This includes data such as time series data, which can be thought of as a 1-D grid of data points, and image data, which can be thought of as a 2-D grid of pixels.

CNNs work by applying a mathematical operation called convolution to the input data (Goodfellow et al. 2016).

A common structure involves the repetition of a sequence comprising multiple convolutional layers and a pooling layer. This sequence is then followed by one or more fully connected layers. The process by which input data undergoes transformation into output by passing through these layers is referred to as forward propagation, as depicted in Figure 7 (Yamashita et al. 2018)

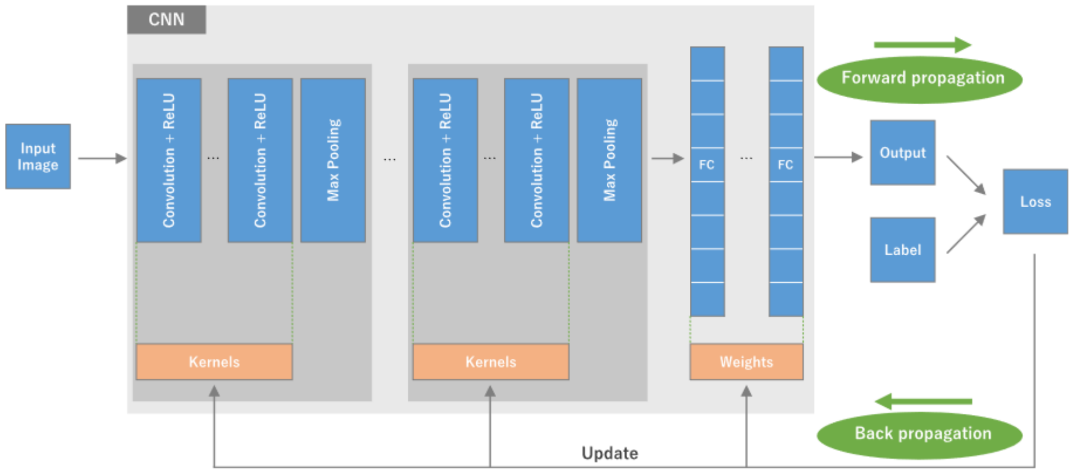


FIGURE 7. An overview of CNN architecture and training process (Yamashita et al. 2018)

Convolution is a specialized linear operation used for feature extraction. It involves applying a small array of numbers (kernel) across an input array (tensor). Each element of the kernel is multiplied with the corresponding element of the input tensor, and these products are summed to produce an output value in a feature map, Figure 8.

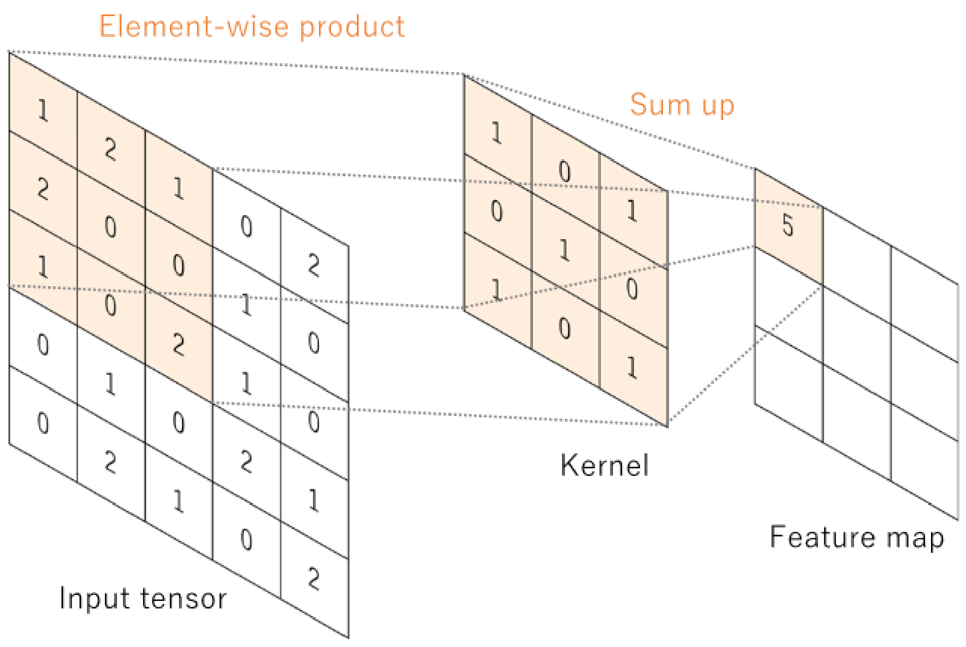


FIGURE 8. An example of convolution operation with a kernel size of 3 × 3, no padding and a stride of 1 (Yamashita et al. 2018)

Convolution operation has a limitation where the central point of each kernel cannot align with the outermost element of the input tensor. This constraint causes the output feature map's dimensions to become smaller than those of the input tensor. To overcome this, a technique called padding is employed, usually in the form of zero padding.

Padding involves adding rows and columns of zeros along the edges of the input tensor as depicted in Figure 9. This adjustment allows the center of the kernel to align with the outermost element, maintaining the same in-plane dimension throughout the convolution process. In essence, padding ensures that valuable information from the edges of the input is also considered during the convolution operation, helping to preserve the spatial size of the feature maps (Yamashita et al. 2018).

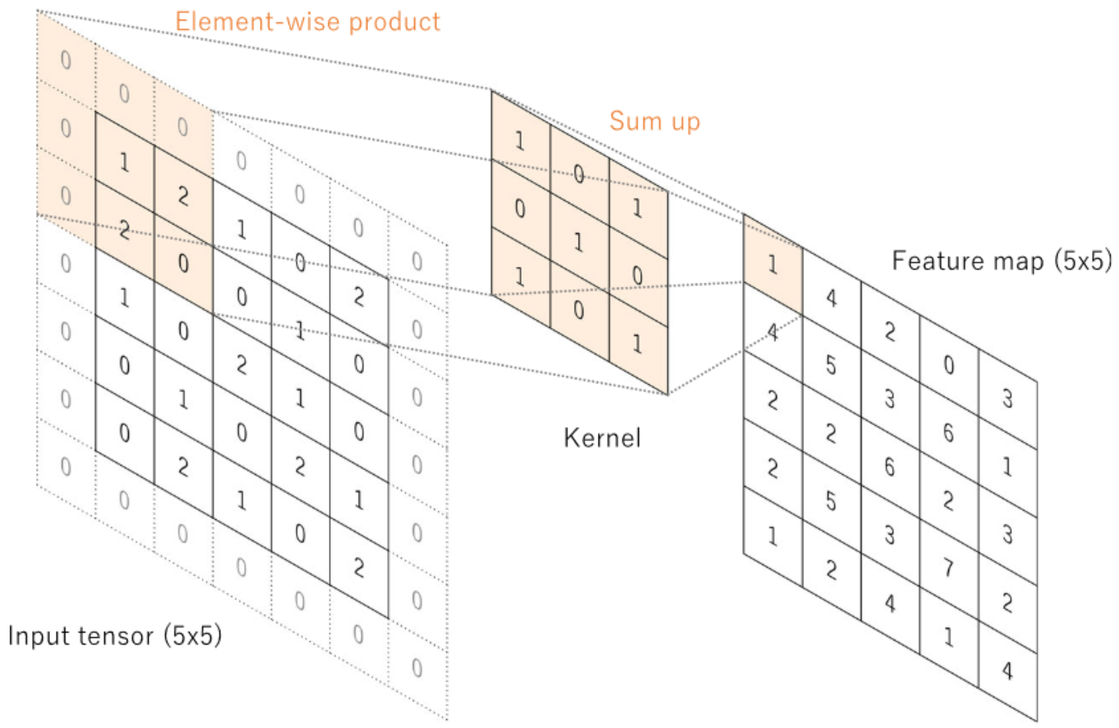


FIGURE 9. Convolution with zero padding (Yamashita et al. 2018)

Stride refers to the step size at which a kernel moves across the input data or image during the convolution process. Kernel slides over the data in discrete steps determined by the stride value. A larger stride value means that the filter skips more pixels or positions as it moves, resulting in a downsampled output. This downsampling reduces the spatial dimensions of the resulting feature maps, which can be beneficial for reducing computational complexity and memory usage in deeper layers of a neural network. Conversely, a smaller stride value, often set to 1, ensures that the kernel moves pixel by pixel across the input data, preserving more spatial information in the feature maps (Yamashita et al. 2018)

While the convolutional layer reduces the size of the input matrix, additional size reduction is often necessary. This is where the pooling operation proves useful. **Pooling** serves the purpose of eliminating noise and retaining the most vital and meaningful features within the data. Much like the convolutional operation, a specific-sized window slides across the input matrix, applying a mathematical operation to the covered section. The resulting output value from the pooling operation is then placed into a new matrix.

Figure 10 illustrates the concepts of max pooling and average pooling. In the upper part of the figure, a 2x2 max pooling operation scans a 4x4 input matrix and selects the maximum value from each quadrant of the matrix. Similarly, average pooling operates in the same manner, except it calculates the average value of the covered window (Esposito, D. 2020).

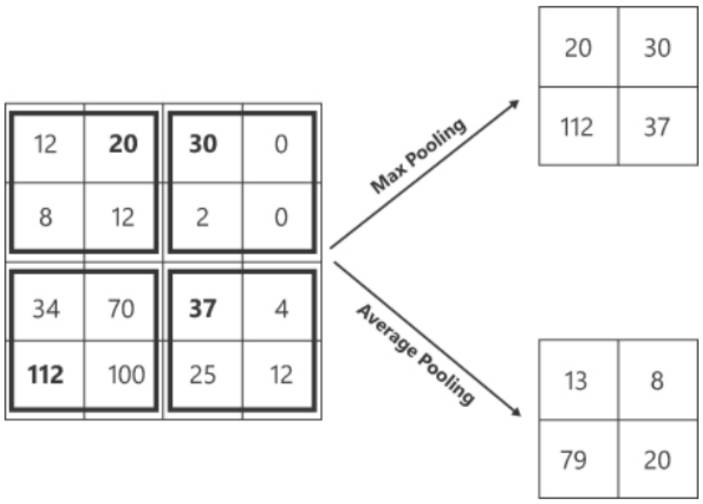


FIGURE 10. Pooling operation (Esposito, D. 2020)

These described layers form the foundational components of many typical convolutional neural network architectures. The outcomes yielded by these layers are then directed into a fully-connected layer which is accompanied by an activation function, typically ReLU. The fully-connected layer consists of multiple layers that connect all inbound feature maps to build a classification model (Esposito, D. 2020).

CNNs were originally designed for image analysis and have since been extended to process multidimensional time series data, effectively extracting correlations within them. CNNs utilize convolution operations, often in one layer, to capture patterns from the inherent (spatio)temporal structure of time series. This approach, when compared to fully connected networks, tends to yield more efficient training and improved performance for similar model complexity.
For multivariate time series, a window of data is transformed into a matrix, denoted as X. Multiple filters, with dimensions of width w and height h (equivalent to the number of channels), are applied to generate diverse feature maps. In the

context of 1D convolution, the *k*th filter traverses a one-directional path over the input matrix X, generating an output:

$$h_k = f_c(W_k * X + b_k) \tag{10}$$

where h_k is the kth output vector, $*$ represents the convolution operation, f_c is the activation function and W and b are weight and bias, respectively.

Temporal Convolutional Networks (TCNs) are a variant of CNNs developed for sequential data analysis. TCNs produce sequences by causal convolution, i.e., no information leakage from the future into the past. Modeling longer sequences with large receptive fields requires a deep network or a wide kernel, significantly increasing the computational cost. As a result, an effective TCN architecture employs dilated causal convolutions, Figure 11, rather than causal convolutions, resulting in an exponentially increasing receptive field. (Belay, M et al.2023).

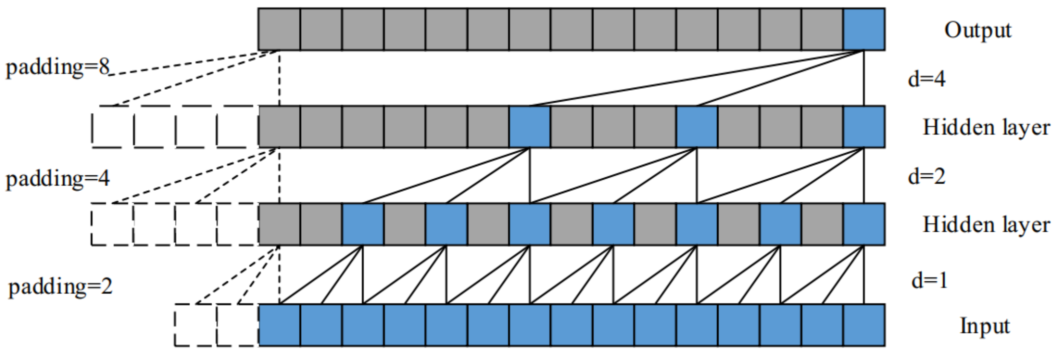


FIGURE 11. A dilated casual convolution with dilated factors d=1, 2, 4 and filter size k=3 (He, Y et al. 2019)

A dilated convolution is a specialized type of convolution operation where a filter is applied across a larger region than its own size by skipping input values based on a specified step. This mechanism is similar to pooling or stride convolutions, as it effectively expands the receptive field, but distinctively, it maintains the output size equal to the input size (He, Y et al. 2019)

4.7 **Recurrent Neural Networks**

Recurrent neural networks (RNNs) is another popular neural network for processing sequential data with temporal dependencies (Goodfellow et al. 2016). Hence, they are naturally well suited for managing time series data characterized by interconnected data points over time. RNN is much like a feedforward neural network, except it also has connections pointing backward allowing information from the past to persist, connecting it to the present. As depicted in Figure 12, RNN takes dx-dimensional vectors x = (x1, x1, xt), and processes them sequentially. While Recurrent Neural Networks (RNNs) come in diverse configurations, they share a common principle of using recurrent connections between hidden cells. In its most elementary form, an RNN's essence can be captured through the subsequent equations:

$$\begin{aligned} h_t &= \sigma_h(W_x x_t + W_h h_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned} \qquad 11$$

where h_t is the hidden state, σ_h and σ_y are the activation functions, W_x, W_h, W_y are the weight matrices, b_h and b_y are the bias vectors, x_t is the current input vector, h_{t-1} is the previous state, and y_t is the output vector (Belay, M et al.2023)

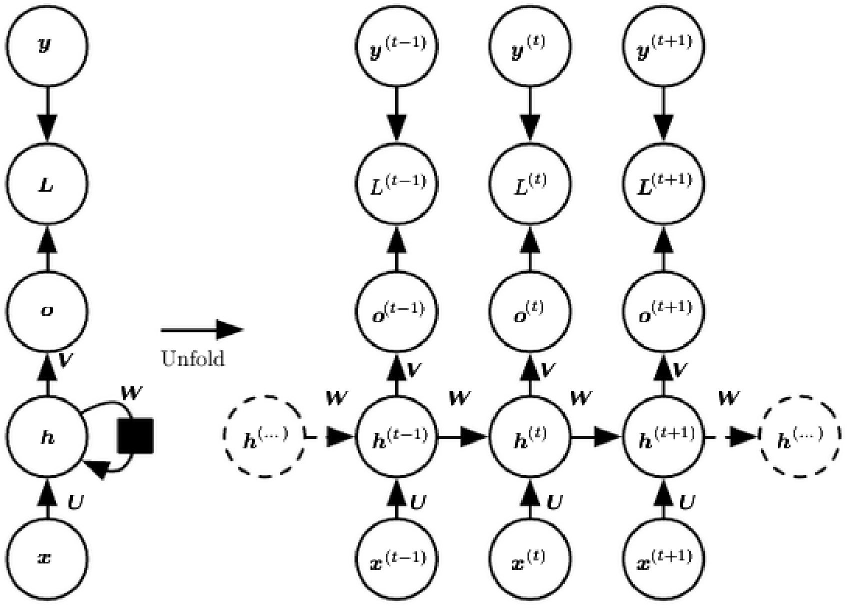


FIGURE 12. Time-unfolded recurrent neural network (Goodfellow et al. 2016)

RNNs undergo training through a gradient descent technique known as back-propagation through time (BPTT) algorithm. In simple terms, BPTT unfolds the RNN and backpropagates the error step by step throughout the entire input sequence, subsequently updating the network's weights based on the accumulated gradients. Nevertheless, the performance of RNNs tends to degrade significantly when tasked with modeling long-range temporal dependencies due to the exponential decay of gradients, a challenge commonly referred to as the "vanishing gradient problem”. To counter the challenges faced by conventional RNNs, novel recurrent architectures such as LSTM have emerged. These architectures mitigate the issues by employing diverse gate units to control the flow of new information, thereby facilitating storage and overwriting processes across each time step (Belay, M et al.2023).

4.8 **Long Short-Term Memory Networks**

The Long Short-Term Memory (LSTM) stands as a specialized type of Recurrent Neural Network (RNN) designed to address the prevalent vanishing gradient issue observed in traditional RNNs. This enhancement empowers LSTM to effectively capture extensive long-term dependencies, extending over 1000 time steps. Through the incorporation of an explicit memory mechanism (cell state) and specialized gating components, LSTM overcomes the challenge of vanishing gradients. These gating units precisely regulate the passage of information through the cell memory, sustaining a consistent error flow within the cell (Hochreiter, S 1997). A common LSTM unit, in Figure 13, is composed of a cell, an input gate, an output gate and a forget gate (Wei, Y et all. 2023)

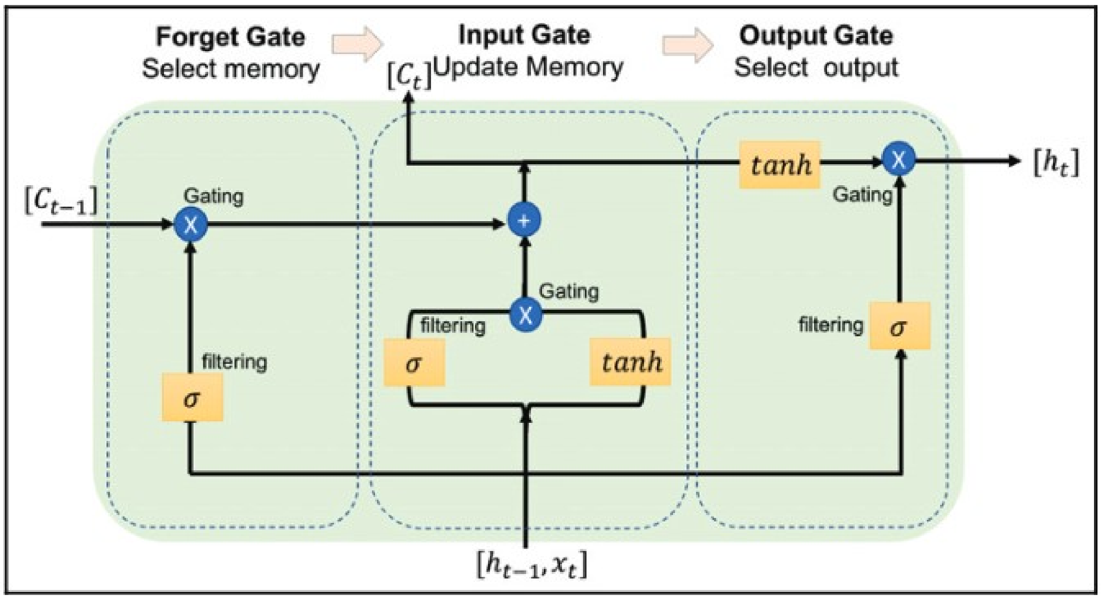


FIGURE 13. Basic LSTM architecture (Di, W et all. 2018)

Inside each cell, a series of three sequential operations are carried out on the incoming information, the current input denoted as X_t the previous output termed as h_{t-1} which comes from the previous cell cycle, and the cell state from the last iteration, referred to as C_{t-1}. In the Figure 13, both $h_{t-1}$and X_t are combined by concatenation.

The role of the **forget gate** is to determine which portions of the historical memory contained in the prior cell state C_{t-1} should be retained and which should be disregarded. This process is accomplished by passing the combined $[h_{t-1}, x_t]$ through an activation function, often sigmoid, resulting in an indicator vector. This vector is then employed to gate or modulate the previous cell state vector C_{t-1}.

The outcome, represented as f_t, embodies the memorized information that is carried forward from previous states and is anticipated to be valuable for the present context.

The **input gate** involves updating the cell state from $C_{t-1}$to C_t. The chosen memory obtained in the first step is added in a way that is conducive to the current input's filtered version. The filtration occurs via the input gate, which operates as a sigmoid layer determining which values should be updated. The outcome of this gating, multiplied by the tanh of the activation's result, is added to the previously selected memory vector from the forget gate. This cumulative result is harnessed to revise the cell state C_t.

Lastly, the **output gate** comes into play to decide the elements that should be produced as output. In essence, it selectively dictates which segment of the current cell state should be projected as the fresh hidden state, output, or prediction. Reiterating the usage of the sigmoid node, the $[h_{t-1}, x_t]$combination generates a filtering vector, which guides the selection of relevant parts within the current cell state. This updated cell state C_t undergoes a tanh operation, squashing the value to a range between -1 and 1. The outcome is then multiplied by the output of the sigmoid gate. This final product constitutes the ultimate output, denoted as h_t (Di, W et all. 2018).

5 EXPERIMENTS AND TEST SETUP

5.1 Background

Automated mobile machine steering system is part of the preventive maintenance program, including mechanical wheel alignment. In this process, wheels are first steered to the $0°$ position, which means steering cylinder stroke is at the middle. Next, both left and right angle transducer readings are saved to the steering computer. These readings are interpret as the $0°$ angle by the steering computer. Finally, the actual wheels are aligned straight by turning the steering rods while holding the middle position of the steering cylinder. Over time due to wear and tear, the alignment of the wheels may offset leading to unwanted steering behavior especially in higher velocities on straight paths. In extreme cases, the equipment might deviate from the pre-defined route causing unwanted stops in the operations. Poor alignment is also one of the top contributors for increased fuel / energy consumption.

Mobile machine steering system has 4 wheels on both sides which are interconnected via tie rods. The wheel deviations were measured while the steering controls where at $0°$, i.e straight position. Test unit corner wheels deviated 4-6 mm compared to the reference unit while the center wheels were inline. The measured deviations are in the same range compared to units in real operations that have been inspected within the preventive maintenance program.

Experiments were executed under supervised conditions with two mobile machines at a test site with ground based navigation system. The reference unit had gone through the process of mechanical wheel alignment and angle transducer $0°$ angle verification. The second unit, or test unit was also fully commissioned with the exclusion of the alignment and angle transducer check.

5.2 Data Collection and Analysis

The reference machine executed sequence of moves with no load, including straight driving back and forwards and turning to left and right. Total time of the sequence is approximately 10 minutes and the data produced during this time was stored in to a logfile. Then, same sequence was repeated with the test unit and logs were combined.

The KPI's from reference and test machine were compared by plotting and it was noticed that the different steering behavior is most evident while machine is travelling straight. One of the KPI's is a variable called "lateral control point tracking error", which indicates how much machine is laterally off from the path. The reasoning behind selecting this feature for the model is that if the steering is not working optimally, then the lateral control error starts to increase eventually in straight driving section as depicted in Figure 14. In this particular case the test unit is deviating more to the left side of the travelled path since the higher values are on the positive side.

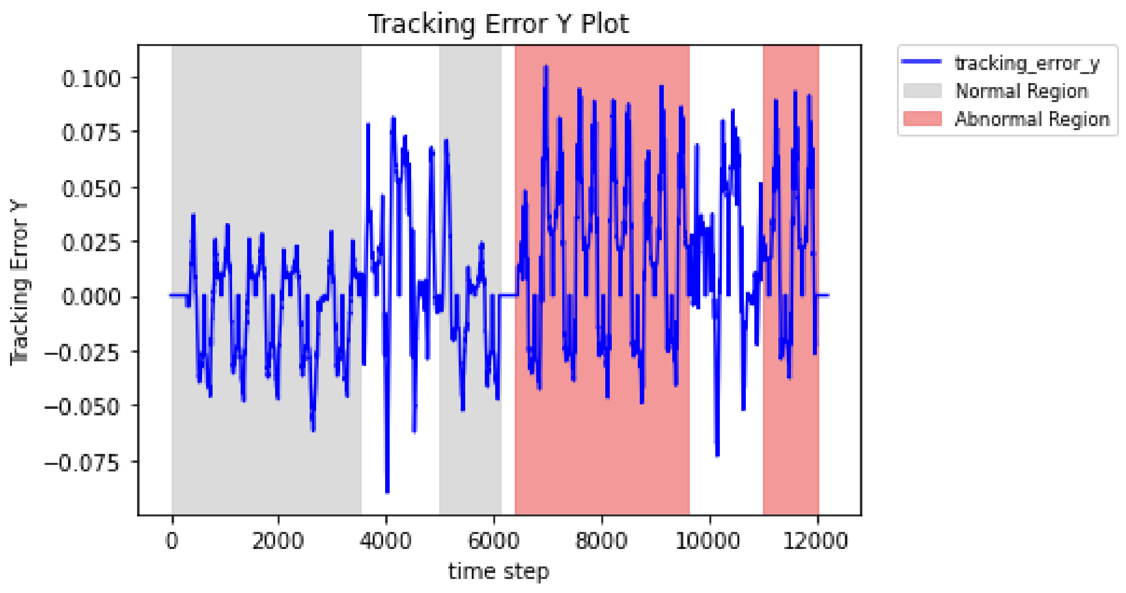


FIGURE 14. Normal and abnormal tracking error – straight driving sections highlighted.

5.3 Forecasting Approach

The goal is to construct a model that tries to establish connections between the given inputs (independent variables X) and the output (dependent variable y), aiming to minimize loss during the training process. Fundamentally, this task can be framed as Multivariate Time Series Forecasting. In this context, the objective is to predict future values of 'tracking_error_y' by leveraging the historical values of multiple features (X).

Selected features are illustrated in the model overview, Figure 15, and the architectural representation of this forecasting approach is depicted in Figure 16. The comprehensive model structure is described in Appendix 1.

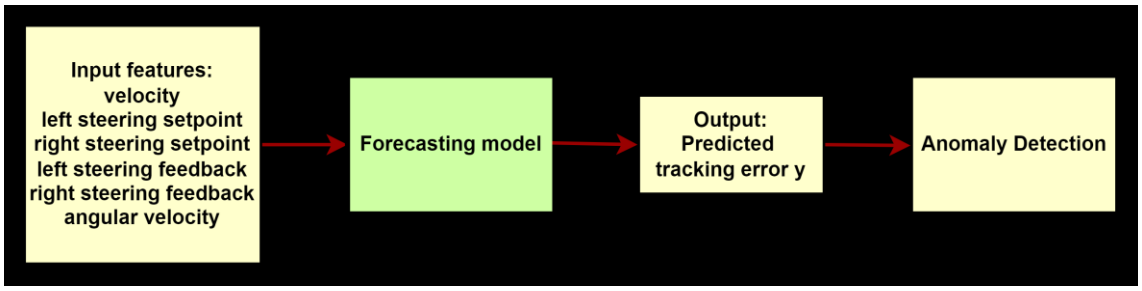


Figure 15. Model and anomaly detection overview

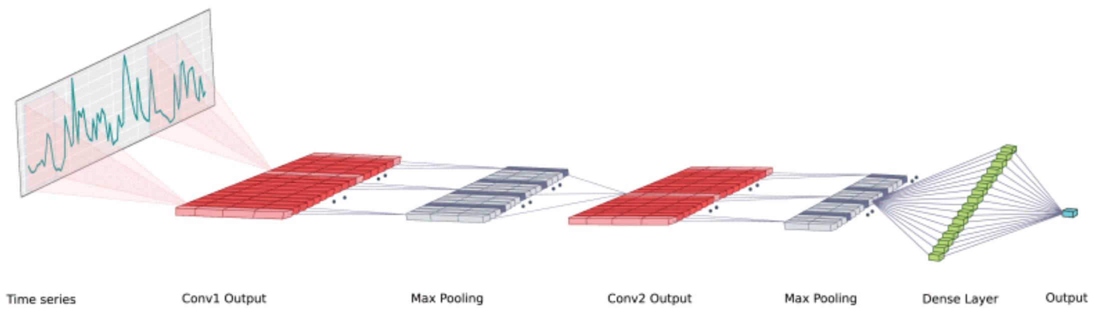


FIGURE 16. CNN based forecasting model with max pooling (Munir, M et all 2019)

5.3.1 Model Architecture

Two models were chosen for the experiments as mentioned in section 4.1. In the following chapters both CNN and LSTM architectures are described in more detail.

The CNN forecasting model is constructed with a sequential structure. It includes two Conv1D layers with a kernel size of 3 and 32 filters each, followed by LReLU activation. MaxPooling1D layers with a pool size of 2 and strides of 2 are employed after each convolutional layer. The model is flattened before a Dense layer with 32 units and LReLU activation, followed by a dropout layer for regularization. The output layer is a Dense layer with a single neuron for regression tasks. The Adam optimizer with a specified learning rate is used, and MAE is employed as the loss function.

LSTM applies also sequential structure, featuring two LSTM layers with 32 units each and LReLU activation. Dropouts are applied after each LSTM layer for regularization. The model is then flattened, followed by a Dense layer with 32 units and LReLU activation, and another Dropout layer. The output layer is a Dense layer with a single neuron for regression tasks. The Adam optimizer with a specified learning rate is used, and Mean Absolute Error (MAE) serves as the loss function, same as in CNN model. Detailed LSTM model structure is outlined in Appendix 2.

The key differences between the models: CNNs utilize convolutional and pooling layers for feature extraction from spatial data, whereas LSTMs leverage recurrent layers for capturing temporal dependencies in sequential data.

5.3.2 Data Pre-processing

In the data pre-processing phase, several steps were taken to prepare the dataset for model training:

- The data split into training and testing sets using a test size of 48% which is the data from test unit. This division ensures that only the “normal” data is used for training while preserving a separate set for model evaluation.

- The data is converted into sequences using a sliding window approach. Sequences are created with a length of 100 in order to capture temporal dependencies.

- MinMaxScaler is applied to normalize both input features and target variables. This scaling ensures that all values fall within the range [0, 1].

5.3.3 Model Training

Model training is an iterative process where different hyperparameter and sequence length values are experimented. Typically, the exploration begins with certain initial parameter values and are adjusted through successive iterations to refine the model's performance. Final parameters of the training process can be found from Table 1.

TABLE 1. Model parameters

learning rate	dropout	regularization	epochs	batch size
0.001	0.2	0.001	30	50

Both models underwent training using the prepared sequences and scaled features. The training process utilized 90% of the training dataset, with the remaining 10% reserved for validation. After each epoch, or complete pass through of the training data, the model's performance is evaluated on the validation data to prevent overfitting. Overfitting occurs when a model performs well on the training data but fails to generalize to new, unseen data. The Adam optimizer updates the model parameters to minimize MAE. Training and validation losses of CNN model are visualized in Figure 17.

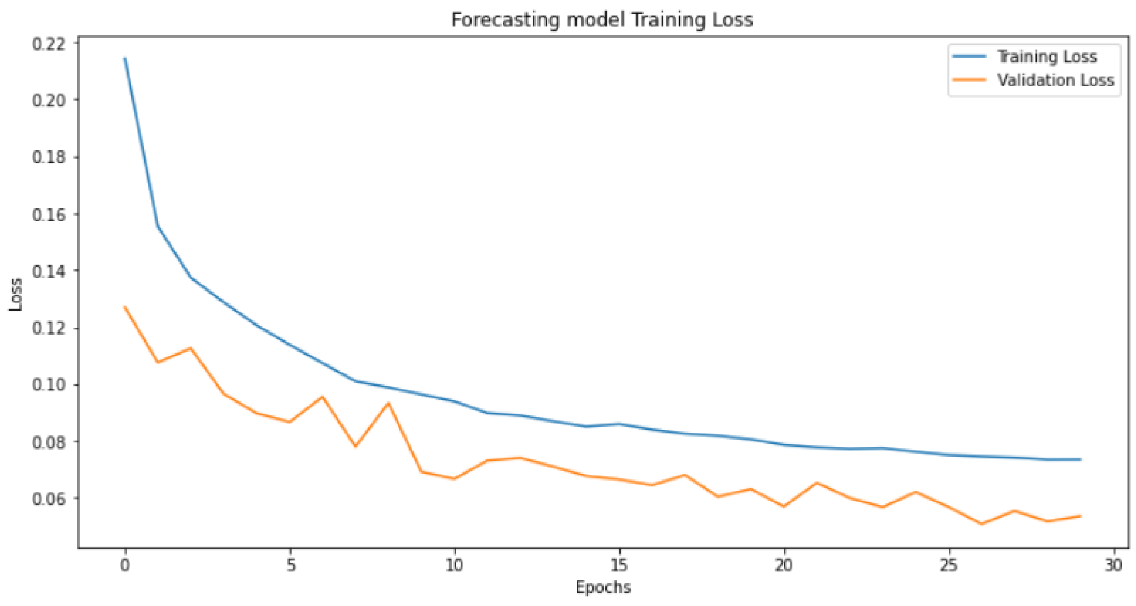


FIGURE 17. Line plots of training and validation loss values during the training phase of CNN forecasting model.

In Figure 18, the training and validation losses of the LSTM model are presented for comparison. Notably, in both instances, the validation loss is observed to be lower than the training loss. This discrepancy is likely attributed to the regularization techniques employed during training, which are not applied during the validation phase.

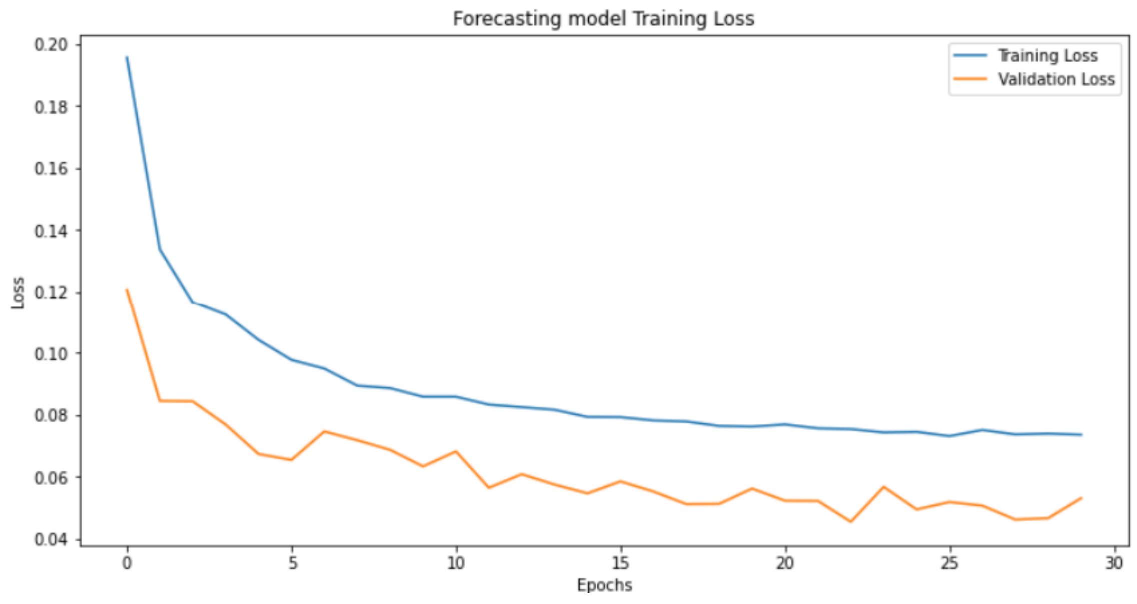


FIGURE 18. Line plots of training and validation loss values during the training phase of LSTM forecasting model.

In Figure 19, CNN model predictions on train and test sets are illustrated. It's evident that model is struggling more to get optimal predictions in sudden changes i.e. when machine is turning.

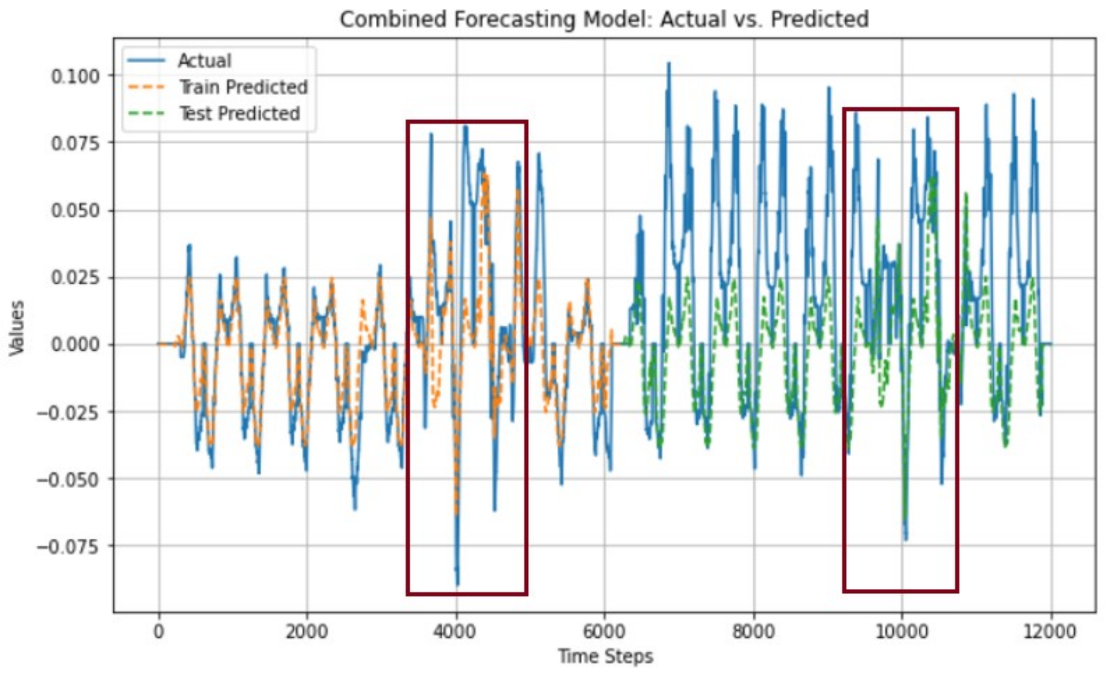


FIGURE 19.Line plot of predicted vs actual values – CNN.

Scatter plot, Figure 20, visually compares the actual values (X-axis) against the predicted values (Y-axis) generated by the forecasting model. Each point's color indicates the magnitude of the prediction error. A diagonal cluster suggests accurate predictions, while deviations highlight areas of outliers. The colorbar provides a reference for interpreting prediction errors.

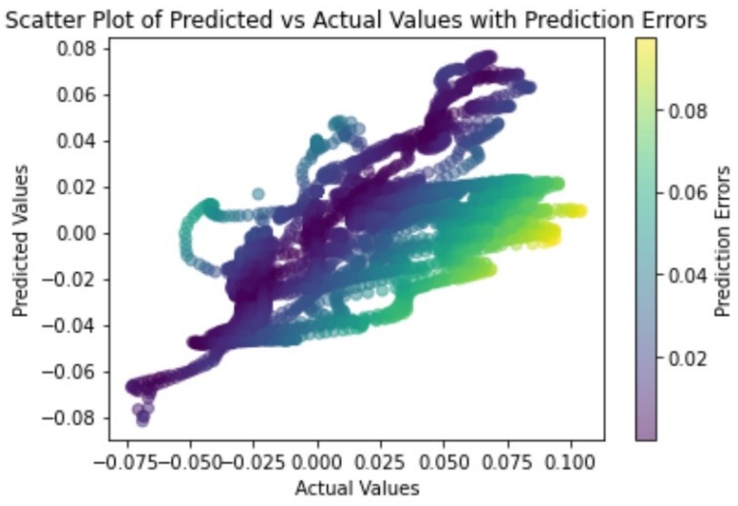


FIGURE 20. Scatter plot of prediction errors – CNN.

5.4 **Anomaly Detection**

Anomaly detection is based on predicting values on the test set and the prediction errors are calculated as absolute errors between actual and predicted values. MAE quantifies the average absolute difference between the actual target values and the corresponding predicted values for each data point in the dataset.

The threshold for anomaly detection is set at the value that corresponds to the 99th percentile of the prediction errors. Any prediction error above this threshold is considered an anomaly. Anomalies are visualized in Figures 21 and 22 together with the predictions and actual values from both models. The threshold limit essentially defines the sensitivity of the anomaly detection and should be assessed carefully in order to avoid false detections. In this particular case, values between -0.06 and 0.06 would be considered normal in the straight driving sections and values above would be suspicious.

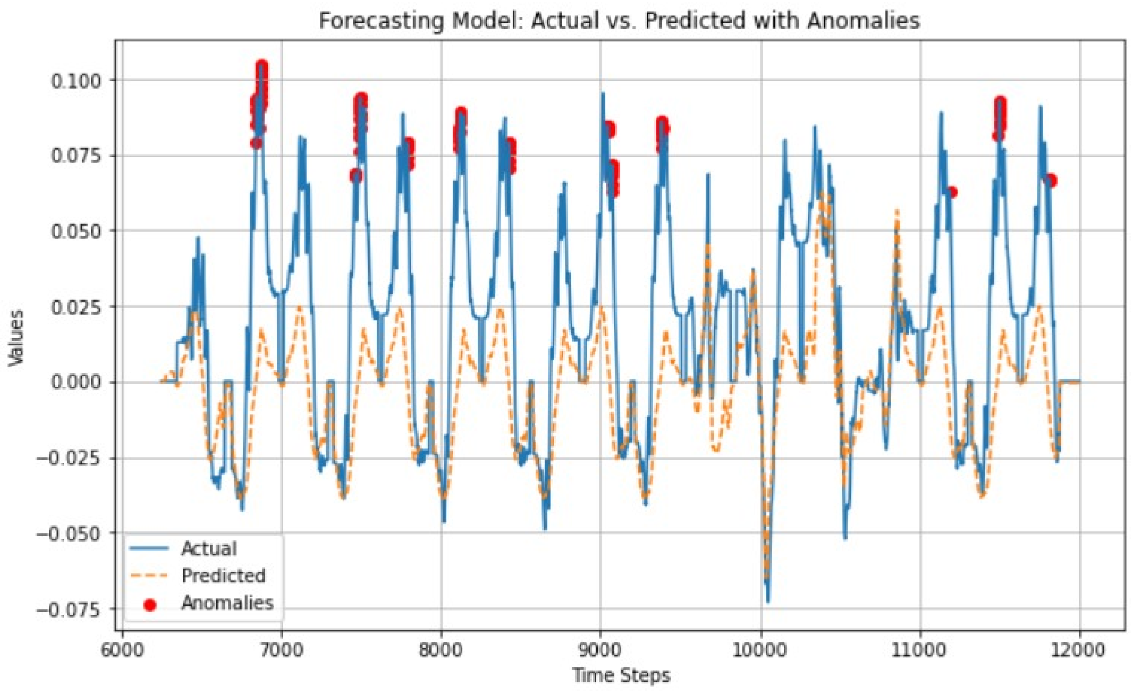


FIGURE 21. Plot of detected anomalies - CNN.

For comparison, anomaly detection scores from LSTM forecasting model is illustrated in Figure 22.

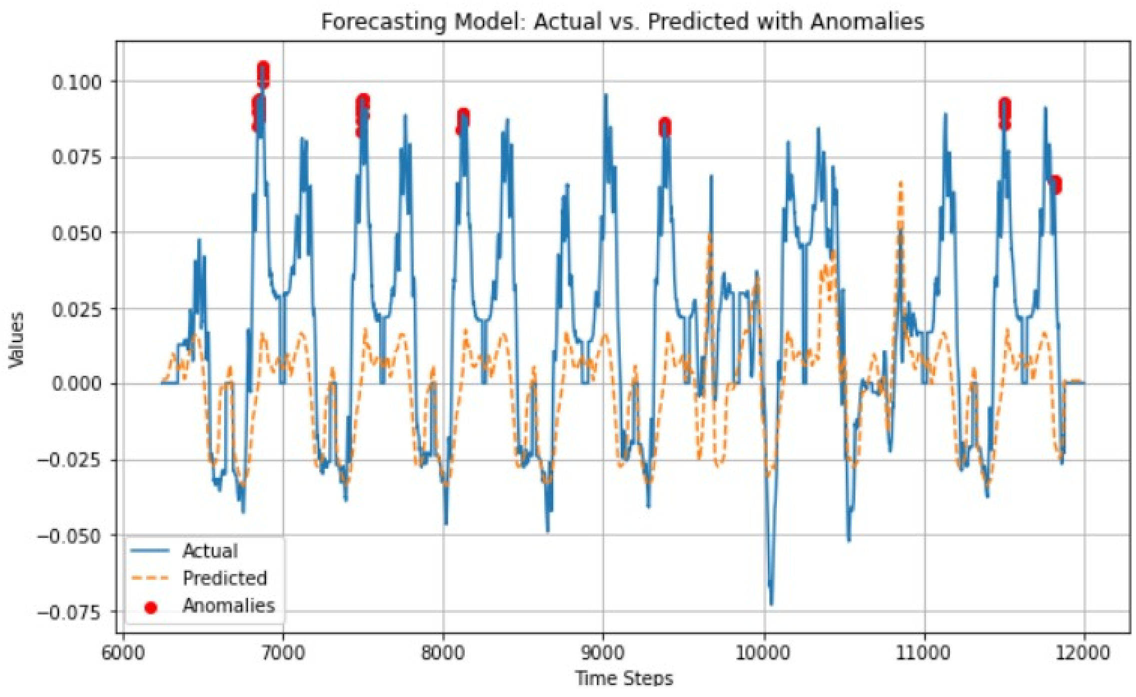


FIGURE 22. Plot of detected anomalies - LSTM.

5.5 **Model Performance Summary**

In table 2, CNN and LSTM model performance characteristics are summarized. This table provides an overview of the training time, number of parameters, and Mean Absolute Error (MAE) for both the CNN and LSTM models.

TABLE 2. Model overview

Model	**Training time (s)**	**Parameters**	**MAE**
CNN	35	29 761	0.011
LSTM	540	115 777	0.015

5.6 **Tools**

Table 3 provides a list of used libraries in the experiments. These libraries are fundamental for data analysis, machine learning, and visualization tasks in Python.

TABLE 3. Software libraries

Library	**Version**
keras	2.10.0
matplotlib	3.8.0
numpy	1.26.0
pandas	2.1.1
scikit-learn	1.3.1

6 CONCLUSIONS AND FUTURE WORK

Two deep learning models were tested in the experiments and both showed comparable results in terms of MAE when modeling the target feature. The most significant difference between the models was the training time. LSTM model took approximately 15 times longer to train than the CNN model in this specific scenario. The difference in training time may be attributed to the nature of LSTM layers, which involve more complex computations and dependencies across time steps compared to the convolutional layers used in CNNs. Additionally, the total number of parameters in the LSTM model is higher, contributing to the longer training time. The anomaly detection threshold is a subject of careful consideration. All units are individuals; meaning that data is not identical and small deviations do not mean necessarily degradation in the performance.

Regarding the research questions outlined in 1.3, the answer is affirmative to both. DL proves capable in detecting anomalies from unlabeled data and holds the potential for predictive maintenance based on the obtained results. However, several aspects need to be outlined. The experiments conducted in this thesis are simplified and from a specific driving sequence, preventing direct application of the model results to real-world operations. To bridge this gap, extensive data from diverse machines is essential for robust model training. The prediction results reveal challenges in model performance, highlighting the need for exploring new features for the model input and possibly parameter tuning in order to enhance accuracy.

From a maintenance perspective, anomaly detection alone does not provide precise guidance to service personnel on what specifically to address. Therefore, incorporating root cause analysis becomes crucial. In addition to anomaly detection, identification and isolation should be somehow included, for example as by Safavi et all.

There are architectural considerations as well in building diagnostic systems, for example regarding the optimal location for running computational loads. From the perspective of network traffic, it is logical to process data close to the source and transmit only the results or high-value data to the end user. Considering the frequent updates of DL models, there should be the flexibility to test and develop them in the cloud.

Interfacing to all subsystems must also be defined including data update rates and communication protocols. Furthermore, consideration should be given to the optimal method for gathering data from actuators and sensors. Is it more efficient for all data to flow through control systems, or should (raw) data be collected directly from communication buses using dedicated gateways?

In the future, it would be important to address the aforementioned aspects if the development of onboard diagnostic systems based on deep learning is considered.

REFERENCES

Allwright, Stephen. https://stephenallwright.com/loss-function-vs-cost-function/ - accessed 29.7.2023

Bartosz Przysucha, Tomasz Rymarczyk, Dariusz Wójcik,
Marcin Kowalski, Ryszard Białek; Management of Early Failure Detection of Production Process: The Case of the Clutch Shaft Alignment using LSTM Deep Learning Algorithm. pp189-197, 2021.

Belay, Mohammed Ayalew ; Blakseth, Sindre Stenen ; Rasheed, Adil ; Salvo Rossi, Pierluigi: Unsupervised Anomaly Detection for IoT-Based Multivariate Time Series: Existing Solutions, Performance Analysis and Future Directions, Sensors (Basel, Switzerland), 2023, Vol.23 (5), p.2844.

Bishop, Cristopher M. Pattern recognition and machine learning. Springer 2006. https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf - accessed 29.7.2023

Chalapathy, Raghavendra ; Chawla, Sanjay: Deep Learning for Anomaly Detection: A Survey. 2019. https://arxiv.org/abs/1901.03407 - Accessed 14.8.2023

Chandola, V.; Banerjee, A.; Kumar, V. (2009). "Anomaly detection: A survey"

Da Silva, Ivan Nunes, Ivan Nunes. author. Hernane Spatti, Danilo; Andrade Flauzino, Rogerio. Liboni, Luisa Helena Bartocci. Reis Alves, Silas Franco

Davari, Narjes ; Veloso, Bruno ; Ribeiro, Rita P. ; Pereira, Pedro Mota ; Gama, Joao; Predictive maintenance based on anomaly detection
using deep learning for air production unit in the
railway industry. 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA), 2021, p.1-10.

Di, Wei, Bhardwaj, Anurag; Wei, Jianing; Deep Learning Essentials, O’Reilly 2018.

Esposito, D, Introducing machine learning, 1st edition. Place of publication not identified: Published with the authorization of Microsoft Corporation by Pearson Education, 2020.

Géron, A, Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow, O'Reilly Media, 2019.

Goodfellow, I., Bengio Y., & Courville A. (2016). Deep Learning. [e-book] MIT Press, Available at: http://www.deeplearningbook.org [Accessed 19 July 2023]

He, Yangdong , ; Zhao, Jiabao: Temporal Convolutional Networks for Anomaly Detection in Time Series. Journal of physics. Conference series, 2019, Vol.1213 (4), p.42050.

Kiavash, Fathi, ; van de Venn, Hans Wernher ; Honegger, Marcel ; Predictive Maintenance: An Autoencoder Anomaly-Based
Approach for a 3 DoF Delta Robot. Sensors (Basel, Switzerland), 2021, Vol.21 (21), p.6979.

Kingma, Diederik P ; Ba, Jimmy: Adam: A Method for Stochastic Optimization, 2015. https://arxiv.org/abs/1412.6980 - Accessed 2.8.2023

Madani, Ali. Optimization process in artificial neural networks
https://blog.cyclicarx.com/optimization-in-artificial-neural-networks - accessed 1.8.2023

Munir, Mohsin ; Siddiqui, Shoaib Ahmed ; Dengel, Andreas ; Ahmed, Sheraz DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. IEEE access, 2019, Vol.7, p.1991-2005, Article 8581424.

Nielsen, M. “Neural Networks and Deep Learning”, Determination Press, 2015.

Nwankpa, Chigozie ; Ijomah, Winifred ; Gachagan, Anthony ; Marshall, Stephen “Activation Functions: Comparison of Trends in Practice and Research for Deep Learning”, 2018. https://arxiv.org/pdf/1811.03378.pdf – accessed 20.7.2023

Safavi, Saeid ; Safavi, Mohammad Amin ; Hamid, Hossein ; Fallah, Saber; Multi-Sensor Fault Detection, Identification, Isolation and
Health Forecasting for Autonomous Vehicles,Sensors (Basel, Switzerland), 2021, Vol.21 (7), p.2547

Sarker, Iqbal H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions.
SN computer science, 2021, Vol.2 (6), p.420-420, Article 420

Susto, G. A., A. Beghi, and C. DeLuca, “A predictive maintenance system
for epitaxy processes based on filtering and prediction techniques,” IEEE
Trans. Semicond. Manuf., vol. 25, pp. 638–649, 2012.

Wei, Yuanyuan ; Jang-Jaccard, Julian ; Xu, Wen ; Sabrina, Fariza ; Camtepe, Seyit ; Boulic, Mikael; LSTM-Autoencoder-Based Anomaly Detection for Indoor Air Quality Time-Series Data, IEEE sensors journal, 2023, Vol.23 (4), p.3787-3800 – accessed 21.8.2023

Yamashita, Rikiya & Mizuho Nishio & Richard Kinh Gian Do & Kaori Togashi: Convolutional neural networks: an overviewand application in radiology, Springer 2018.

Yufeng, Yu ; Zhu, Yuelong ; Li, Shijin ; Wan, Dingsheng Jiang, Jun: Time Series Outlier Detection Based on Sliding Window Prediction, 2014. https://downloads.hindawi.com/journals/mpe/2014/879736.pdf - Accessed 14.8.2023

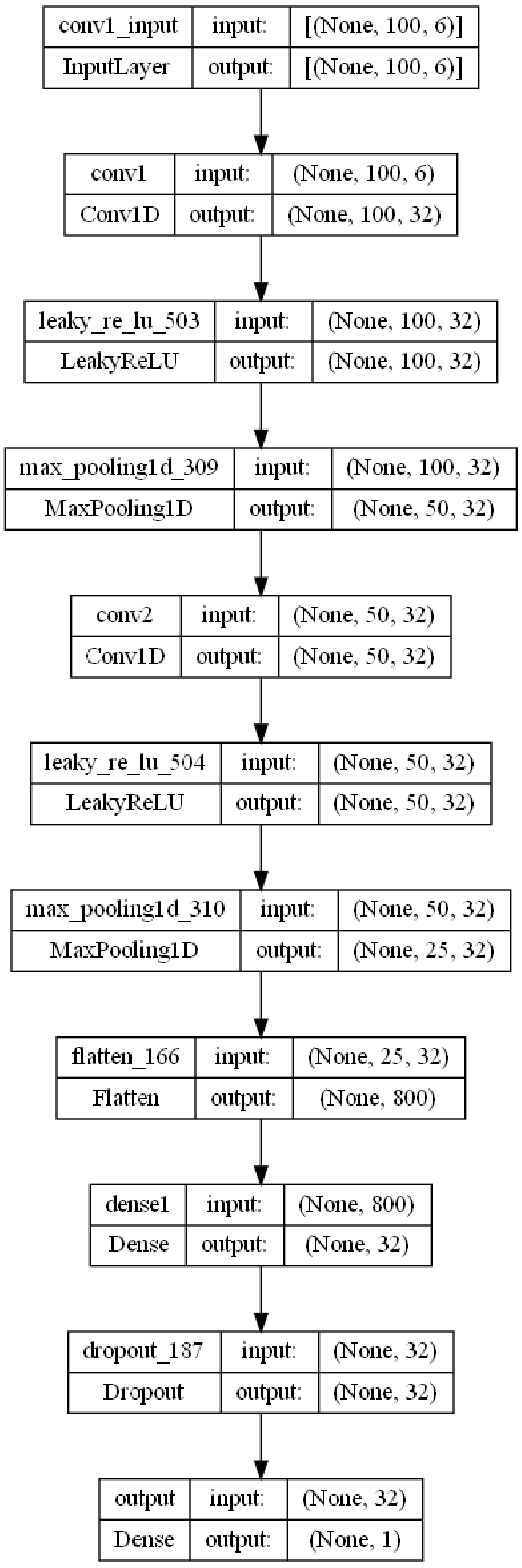
Zhang, Aston ; Lipton, Zachary C ; Li, Mu ; Smola, Alexander J. Dive into Deep Learning, 2023. https://arxiv.org/abs/2106.11342 - accessed 2.8.2023

Zhang, W., Dong Yang, and Hongchao Wang. 2019. “Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey.” IEEE Systems Journal 13 (3):2213–2227.

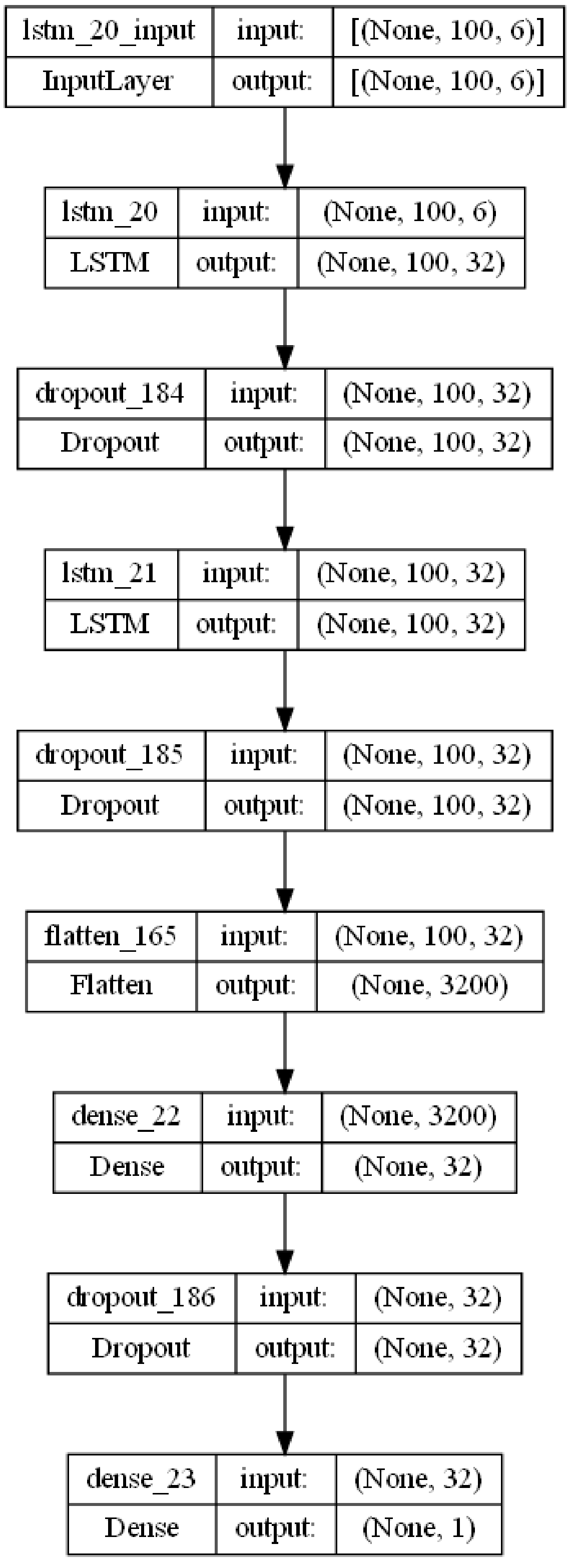
Zhang, Yuxin ; Chen, Yiqiang ; Wang, Jindong ; Pan, Zhiwen: Unsupervised Deep Anomaly Detection for Multi-Sensor Time-Series Signals, IEEE transactions on knowledge and data engineering, 2023, p.1-1

APPENDICES

Appendix 1. CNN model


conv1_input | input: | [(None, 100, 6)]
InputLayer | output: | [(None, 100, 6)]
conv1 | input: | (None, 100, 6)
Conv1D | output: | (None, 100, 32)
leaky_re_lu_503 | input: | (None, 100, 32)
LeakyReLU | output: | (None, 100, 32)
max_pooling1d_309 | input: | (None, 100, 32)
MaxPooling1D | output: | (None, 50, 32)
conv2 | input: | (None, 50, 32)
Conv1D | output: | (None, 50, 32)
leaky_re_lu_504 | input: | (None, 50, 32)
LeakyReLU | output: | (None, 50, 32)
max_pooling1d_310 | input: | (None, 50, 32)
MaxPooling1D | output: | (None, 25, 32)
flatten_166 | input: | (None, 25, 32)
Flatten | output: | (None, 800)
dense1 | input: | (None, 800)
Dense | output: | (None, 32)
dropout_187 | input: | (None, 32)
Dropout | output: | (None, 32)
output | input: | (None, 32)
Dense | output: | (None, 1)

Appendix 2. LSTM model



lstm_20_input | input: | [(None, 100, 6)]
InputLayer | output: | [(None, 100, 6)]
lstm_20 | input: | (None, 100, 6)
LSTM | output: | (None, 100, 32)
dropout_184 | input: | (None, 100, 32)
Dropout | output: | (None, 100, 32)
lstm_21 | input: | (None, 100, 32)
LSTM | output: | (None, 100, 32)
dropout_185 | input: | (None, 100, 32)
Dropout | output: | (None, 100, 32)
flatten_165 | input: | (None, 100, 32)
Flatten | output: | (None, 3200)
dense_22 | input: | (None, 3200)
Dense | output: | (None, 32)
dropout_186 | input: | (None, 32)
Dropout | output: | (None, 32)
dense_23 | input: | (None, 32)
Dense | output: | (None, 1)