Anomaly Detection Using Deep Learning

by

Mohammadhossein Reshadi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Deep learning has revolutionized many fields that process large amounts of data such as images, video, audio, speech, and text. Anomaly detection, however, is among the areas that still require major advancements. Based on the key traits of deep learning, which are the need for very little hand engineering, and the ability to effectively use large amounts of data, its applications in anomaly detection could be very beneficial. In this thesis, an anomaly detection architecture is proposed that consists of two models: a normal model, which is a time series forecaster and predicts the next expected behavior of the system under healthy conditions, and an anomaly detector that identifies any failure by comparing the expected values with the actual observations. Deep architectures such as convolutional neural networks and recurrent neural networks have been incorporated in the design of the normal model, and conventional machine learning methods, including one-class SVM, isolation forests, multi-layer perceptron, decision trees, and random forests are used for the anomaly detector. The proposed architecture has been applied to two problems: pipeline leak detection and condition monitoring and fault detection of small induction motors. The results of both applications have proved very promising and indicate the capacity for further improvements to come.

Preface

Some of the research conducted in this thesis is intended to be published. Two journal articles based on chapters 3, 4, and 5 are in the process of submission.

Acknowledgements

I would like to extend my gratitude to my supervisor, Dr. Scott Dick, whose guidance and support made this research possible.

I also wish to thank my family and friends for their endless love and support throughout all my endeavors.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning plays a major role in many aspects of the modern era: from web searches and social media to language translation [1–3]. With the advancements in deep learning, which allows for multiple processing layers to learn representations of data with multiple levels of abstraction, many domains involving image, video, audio, speech, and text processing have been revolutionized [1]. One of the fields that demand substantial improvements, however, is anomaly detection. The key assets of deep learning are the fact that it requires very little hand engineering and being able to take advantage of the availability of increased amounts of data and computational power [1]. Furthermore, studies indicate deep learning is superior to the traditional techniques when it comes to time-series analysis [4, 5] and sequential data [1], which is the nature of most anomaly detection problems. These key points are why deep learning is a great candidate for anomaly detection applications.

An anomaly could be the occurrence of any type of failure or, more generally, any slight deviation from the normal behavior. Consequently, the demand for an anomaly detection system exists in many areas. These areas include, but are not limited to, finance, sensor networks, fraud detection, medical errors, industrial plants, *etc.*. Two of the applications of interest in this study are pipeline leak detection and condition monitoring and fault detection of small induction motors.

In this research, an anomaly detection architecture is proposed that leverages the

power of deep learning. This anomaly detection system is a low-cost solution that uses the data collected by the already available equipment. The architecture comprises of two models, a normal model which is a time series forecaster and predicts the next expected behavior of the system under healthy conditions, and an anomaly detector that identifies any failure by comparing the expected values with the actual observations. It is a hybrid structure incorporating deep architectures such as convolutional neural networks and recurrent neural networks used in the design of the normal model, and conventional machine learning methods employed to build the anomaly detector.

1.1 Thesis Outline

This thesis is organized in 6 chapters as follows. Chapter 2 provides a thorough review of the architectures, methods, and techniques implemented in the study, covering all the backgrounds needed to follow the research. Chapter 3 presents a complete explanation of the proposed model and all the steps required for the design. Chapters 4 and 5 study the applications of this model for two different problems, namely, pipeline leak detection and condition monitoring and fault detection of small induction motors. Each of these two chapters provides a literature review of the problem being discussed, a description of the datasets used, the obtained results, and a conclusion which suggests possible improvements. In the end, chapter 6 yields the final conclusion of the thesis and provides directions for related future works.

1.2 Contribution

The main contribution of the thesis is the proposed architecture which is provided in chapter 3, and it's applications in pipeline leak detection and fault detection of small induction motors which are discussed in chapters 4 and 5, respectively. The Datasets used in the experiments of chapters 4 and 5 are collected and prepared

with the assistance of the following people. Jun Xiao helped with setting up the computer-generated pipeline leak detection dataset mentioned in chapter 4, while the lab-generated dataset is collected by Javier Barrios as part of his thesis [6]. The required experiments for the collection of the induction motor dataset discussed in chapter 5 are conducted by Nick Zarft, assisted by Albert Terheide. Abbas Sobhi and Kelly Keenan contributed to the preprocessing and analysis of this dataset, as well.

Chapter 2

Background

In this chapter, we will review all model architectures, methods, and techniques used though out this research. These include deep learning and architectures such as Convolutional Neural Networks (*i.e.* CNN), Recurrent Neural Networks (*i.e.* RNN), and Long Short-Term Memory (*i.e.* LSTM), as well as classical machine learning methods. Additionally, a review of the recently proposed anomaly detection approaches is provided. Finally, delay-embedding techniques are studied as they are required for time-series analyses.

2.1 Deep Learning

The main major shortcoming of traditional machine learning methods shows itself when dealing with raw natural data. To be able to train models using raw data, significant knowledge of the area used to be mandatory to carefully extract suitable features [1]. This is when deep learning comes into play. The main advantage of deep learning is the ability to automatically learn features.

Deep learning methods have the ability to learn extremely complex tasks, through several levels of representation which is acquired via simple non-linear modules. Moving deeper into the model, each representation is at a higher level of abstraction. For instance, an image represented by pixels can first be transformed into information about the location of edges, followed by the motifs based on these edges in the next

level and so on, until, entire objects are eventually detected. By eliminating the need for designing the features by a human, deep learning could be applied to many domains without extensive knowledge of the area [1].

Today, deep learning is solving crucial problems that have not been solved for years. Owing to its versatility, deep learning is applicable to a wide variety of problems, especially with the existence of huge amounts of data. Among successful applications are image recognition [7–10], speech recognition [11–13], natural language understanding [2, 3, 14, 15], and various other areas.

2.1.1 Convolutional Neural Networks (CNN)

Convolutional neural networks (CNN) are a class of deep neural networks. CNNs are able to capture the Spatial and Temporal dependencies in the input. Hence, they are most suitable for data that is in the shape of many arrays, such as 1D arrays of sequential data, 2D images, and 3D videos. CNNs are capable of using the characteristics of sequential data thanks to the existence of lots of layers, shared weights, and pooling layers [1].

Multiple studies of convolutional networks are linked back to 1989, beginning with a 1D time-delay neural networks applied for speech recognition [16, 17] as well as document reading [18]. Additionally, a few other works include object detection [19, 20], face detection [21], and handwriting recognition [22].

Convolutional networks were neglected by the machine-learning communities at the beginning, in spite of a few early successes. The 2012 ImageNet competition, however, was a starting point in their popularity where they achieved outstanding results: about half of the error gained by the other competitors [7]. The main reasons behind this success were the effective usage of GPUs, dropout which was a recent regularization method [23], ReLUs (Rectified Linear Unit), and the generation of extra training samples through manipulation of the existing ones. These days, CNNs are the best technique for many tasks [10, 24–28], and compete in performance with

humans in some cases [1].

Figure 2.1 illustrates the architecture of a common convolutional network. The initial layers in the architecture are, typically, convolutional and pooling layers. Within a convolutional layer, a group of weights known as a filter or kernel transform a window of units in the current layer to one unit in the succeeding layer of representations. Subsequently, a non-linear function, for instance a ReLU, is usually applied to the outputs. There are multiple filters in one convolutional layer, resulting in multiple *feature maps*, each of which containing units that share one filter. This operation is called a discrete convolution in mathematics [1].

The purpose of the pooling layer is to combine similar features in only one. A common pooling unit uses the maximum of a window of units as the representative. The process is shifted over the rows and columns, resulting in a dimension reduction and robustness to minor alterations and noise in the input. Several combinations of convolutional and pooling layers are then used together, as well as possible extra convolutional layers and usually one or few fully connected layers at the end [1].

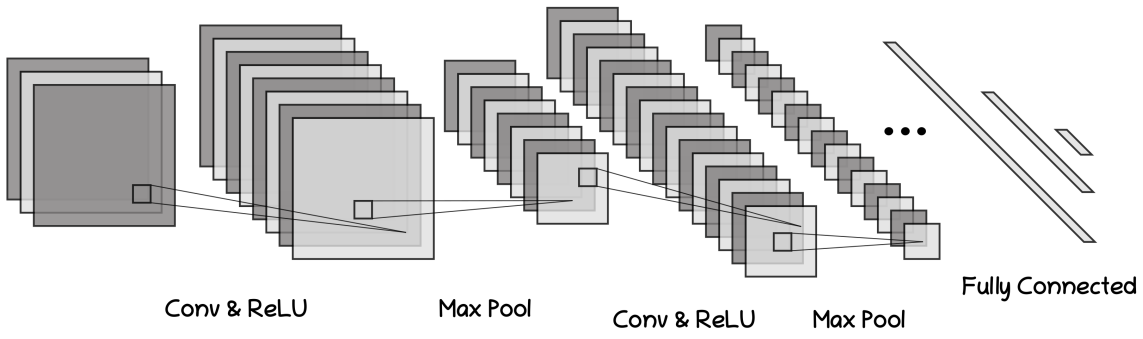

Figure 2.1: A common convolutional network: Each image represents a feature map of one of the learned features. Lower-level features perform edge detection, and, eventually, the output shows a score measured regarding each class [1].

To explain the learning process of the weights of a filter in one convolutional layer, the mathematics of the forward pass should be examined first. For a two-dimensional input I and a two-dimensional filter W, Equation (2.1) illustrates the convolution operation [29]:

$$S(i,j) = (I * W)(i,j) = \sum_m \sum_n I(m,n)W(i-m, j-n) \tag{2.1}$$

Where * is the convolution operation symbol, S is the result of the convolution, and i and j indicate the row and column in S.

The forward propagation consists of two steps. The first step is calculating the intermediate value Z, which is the result of the convolution of the input data (from the previous layer, denoted by $A^{[l-1]}$) with the tensor W (which contains the filters), and then adding bias b. The second step is to pass Z through a non-linear activation function g (*e.g.* ReLU) [30].

$$Z^{[l]}_{i,j} = \sum_m \sum_n A^{[l-1]}_{m,n} W^{[l]}_{i-m,j-n} + b^{[l]} \tag{2.2}$$

$$A^{[l]} = g^{[l]}(Z^{[l]}) \tag{2.3}$$

Where $^{[l]}$ indicates the layer and $_{i,j}$ determines the specific element of the array. The objective function (mean squared error) for the last layer's output is then given by Equation (2.4) [30]:

$$E = \frac{1}{2} \sum_p (t_p - y_p)^2 \tag{2.4}$$

Where y_p and t_p are the output and the target for the p^{th} prediction.

In order to find the update rules for the weights, we now go over the back-propagation process. The purpose of the backward pass is to calculate the derivatives $\frac{\partial E}{\partial W^{[l]}_{m',n'}}$ (which denotes the partial derivative of the objective function E with respect to the weight $W_{m',n'}$ in the layer $[l]$) and perform the updates using the gradient descent method. Using the chain rule, the gradients are calculated in Equation (2.5) [30]:

$$\frac{\partial E}{\partial W^{[l]}_{m',n'}} = \sum_i \sum_j \frac{\partial E}{\partial Z^{[l]}_{i,j}} \frac{\partial Z^{[l]}_{i,j}}{\partial W^{[l]}_{m',n'}} \tag{2.5}$$

$$= \sum_i \sum_j \delta^{[l]}_{i,j} \frac{\partial Z^{[l]}_{i,j}}{\partial W^{[l]}_{m',n'}} \tag{2.6}$$

Where $^{[l]}$ indicates the layer and m', n' determine the specific weight of that layer. Now, by substituting $Z^{[l]}_{i,j}$ we get [30]:

$$\frac{\partial Z^{[l]}_{i,j}}{\partial W^{[l]}_{m',n'}} = \frac{\partial}{\partial W^{[l]}_{m',n'}} \left(\sum_m \sum_n A^{[l-1]}_{m,n} W^{[l]}_{i-m,j-n} + b^{[l]} \right) \tag{2.7}$$

By expanding the summations in Equation (2.7) and taking the partial derivatives of the components, all of them result in zero, except for the component containing $W^{[l]}_{m',n'}$ [30]:

$$\frac{\partial Z^{[l]}_{i,j}}{\partial W^{[l]}_{m',n'}} = \frac{\partial}{\partial W^{[l]}_{m',n'}} \left(A^{[l-1]}_{m'-i,n'-j} W^{[l]}_{m',n'} \right) \tag{2.8}$$

$$= A^{[l-1]}_{m'-i,n'-j} \tag{2.9}$$

By substituting Equation (2.9) in Equation (2.6) we finally get [30]:

$$\frac{\partial E}{\partial W^{[l]}_{m',n'}} = \sum_i \sum_j \delta^{[l]}_{i,j} A^{[l-1]}_{m'-i,n'-j} \tag{2.10}$$

$$= \delta^{[l]}_{i,j} * A^{[l-1]}_{m',n'} \tag{2.11}$$

Where * denotes the convolution operation. To calculate the δs, then, we use a similar process leading to Equation (2.12), where g' denotes the derivative of the activation function g [30]:

$$\delta^{[l]}_{i,j} = \frac{\partial E}{\partial Z^{[l]}_{i,j}} = \left(\sum_m \sum_n \delta^{[l+1]}_{m,n} W^{[l+1]}_{i-m,j-n} \right) g' \left(Z^{[l]}_{i,j} \right) \tag{2.12}$$

$$= \delta^{[l+1]}_{i,j} * W^{[l+1]}_{m,n} \cdot g' \left(Z^{[l]}_{i,j} \right) \tag{2.13}$$

In this research, as we work with 1-dimensional data (time-series), 1-dimensional convolutional networks would be the suitable architecture to use. In 1-dimensional CNN, the filter only shifts along one dimension, which would be the time, when working with time-series data. As illustrated in Figure 2.2, a 1-D convolutional layer employs a 1-dimensional filter for 1-dimensional inputs. When working with multiple time-series, the height of the filter would match the number of time-series, so that the filter still shifts along one dimension only. This means the convolution process is equivalent to performing it individually on each of the time-series by shifting a different filter over each row of the input separately. That being said, the equations for 1-D CNN would be the same as the general equations discussed above, where there could be no shifting over the second dimension as the height of the filter is exactly the same as the height of the inputs.

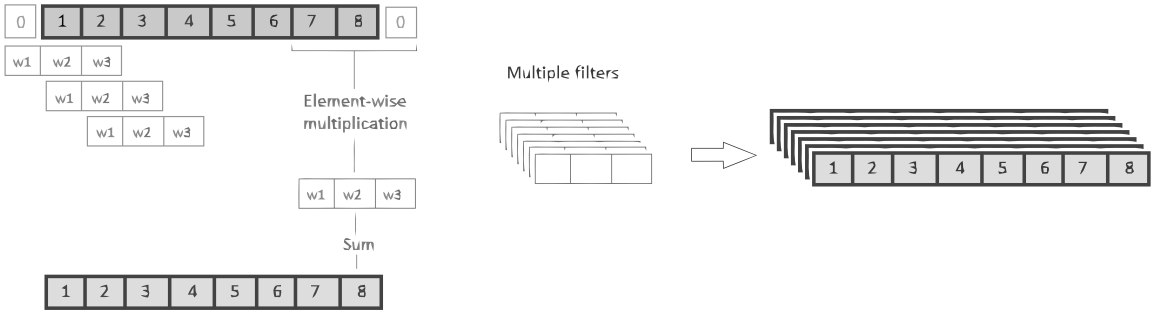

Figure 2.2: A 1-D convolutional layer: a 1-d filter shifts over the input, covering only a window at a time, performing element-wise multiplications in order to complete the convolution operation [31].

2.1.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (*i.e.* RNNs) are another class of deep architectures, designed to deal with data presented in sequences, *e.g.* speech and language. Figure 2.3 describes a basic RNN. RNNs process sequential elements one after another, and are able to keep a history of previous elements in their "state vector" [1].

As useful as RNNs are, training them used to be very difficult due to exploding and vanishing gradients [32]. Enhancements in their architecture [33, 34] and training

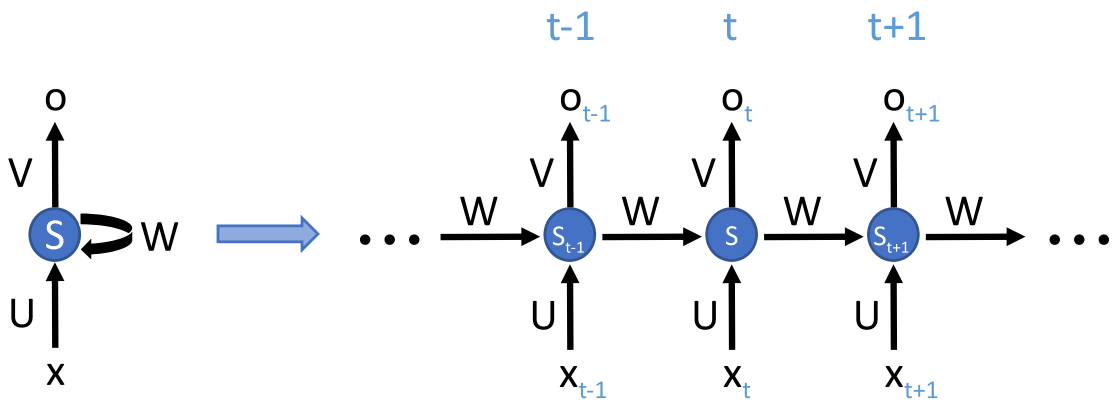

Figure 2.3: A basic RNN: If an RNN is extended in time, it could be regarded as a very deep network with the same weights (U, V, and W) being shared by all the layers [1].

methods [35, 36], however, have enabled RNNs to be extremely potent at character prediction [37], word prediction [38], and several other complex tasks [1, 3, 39–41].

LSTMs were designed to solve the exploding and vanishing gradient problems experienced when training traditional RNNs [33] and are therefore more effective, particularly with multiple layers [42] Currently, LSTM networks and other similar architectures are very successful at machine translation [1, 3, 39, 40]. Figure 2.4 illustrates a common LSTM block, which includes a cell, an input gate, an output gate, and a forget gate [42–44]. The cell is like a memory that keeps important information from previous steps to be able to pass them through later steps as well. The different gates then decide what information to keep, what gets added, and what should be removed, by considering the new inputs, as well as the previous hidden state (recurrent inputs). As the name suggests, the forget gate determines whether the information that is already in the cell needs to be kept or forgotten. The input gate decides what information in the input is important enough to get passed to the cell. After the outputs of the forget gate and the input gate are determined, the cell state will get updated with a pointwise addition of both of these outputs. Finally, the output gate decides whether the new information in the cell should get carried to the next hidden state (recurrent output) [43].

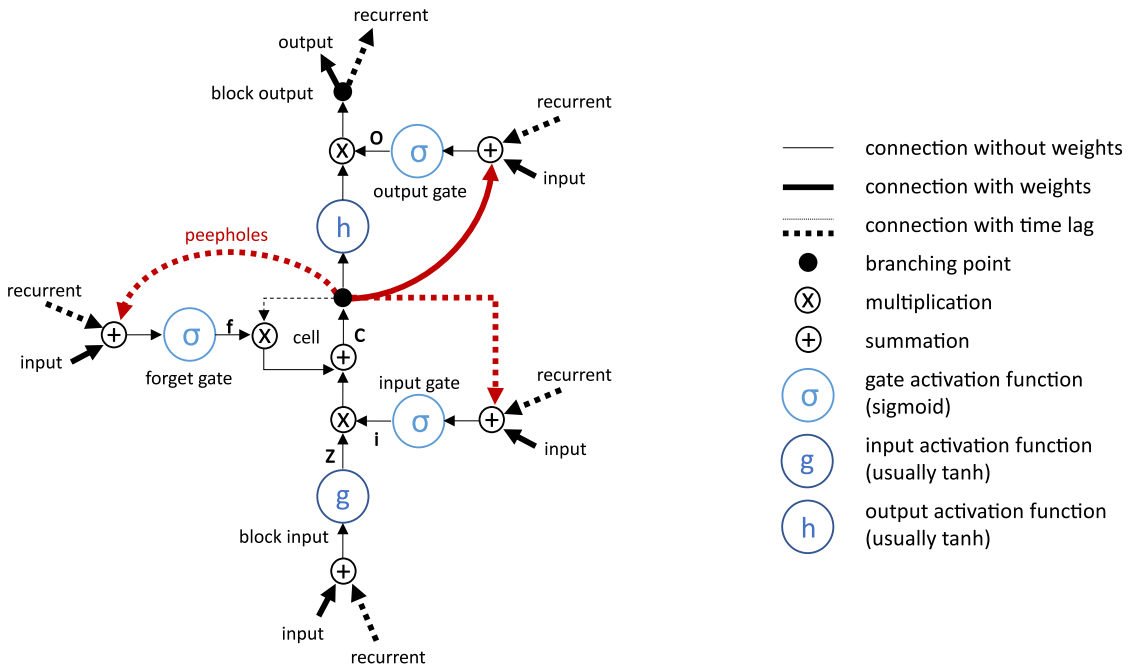

Figure 2.4: A thorough diagram of an LSTM block [43].

To explain the forward pass in an LSTM layer with N blocks and M inputs, we go through the block section by section. According to the figure, z, i, f, c, and o correspond to the activated block input, the output of the input gate, the output of the forget gate, the cell value, and the output of the output gate, respectively. With x^t being the input vector for the time t, $W_z, W_i, W_f, W_o (\in \mathbb{R}^{N \times M})$ being the weights for the inputs, $R_z, R_i, R_f, R_o (\in \mathbb{R}^{N \times N})$ being the recurrent weights, $p_i, p_f, p_o (\in \mathbb{R}^N)$ being the peephole weights, and b_z, b_i, b_f, b_o $(\in \mathbb{R}^N)$ being the biases, the vector formulas for the forward pass are as follows [43].

$$\bar{z}^t = W_z x^t + R_z y^{t-1} + b_z \tag{2.14}$$

$$z^t = g\left(\bar{z}^t\right) \qquad block\ input \tag{2.15}$$

$$\bar{i}^t = W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i \tag{2.16}$$

$$i^t = \sigma\left(\bar{i}^t\right) \qquad input\ gate \tag{2.17}$$

$$\bar{f}^t = W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f \tag{2.18}$$

$$f^t = \sigma\left(\bar{f}^t\right) \qquad forget\ gate \tag{2.19}$$

$$c^t = z^t \odot i^t + c^{t-1} \odot f^t \qquad cell \tag{2.20}$$

$$\bar{o}^t = W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o \tag{2.21}$$

$$o^t = \sigma\left(\bar{o}^t\right) \qquad output\ gate \tag{2.22}$$

$$y^t = h\left(c^t\right) \odot o^t \qquad block\ output \tag{2.23}$$

Where $\odot$ represents point-wise multiplication and σ, g and h are activation functions. The gate activation function is typically the logistic sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$), while the hyperbolic tangent is the common choice for the block input and output activation functions ($g(x) = h(x) = \tanh(x)$) [43].

With backpropagation though time, then, we can compute the δs inside the block. The presented formulas are generalized for all the layers [43]:

$$\delta y^t = \Delta^t + R_z^T \delta z^{t+1} + R_i^T \delta i^{t+1} + R_f^T \delta f^{t+1} + R_o^T \delta o^{t+1} \tag{2.24}$$

$$\delta \bar{o}^t = \delta y^t \odot h\left(c^t\right) \odot \sigma'\left(\bar{o}^t\right) \tag{2.25}$$

$$\delta c^t = \delta y^t \odot o^t \odot h'\left(c^t\right) + p_o \odot \delta \bar{o}^t + p_i \odot \delta \bar{i}^{t+1} + p_f \odot \delta \bar{f}^{t+1} + \delta c^{t+1} \odot f^{t+1} \tag{2.26}$$

$$\delta \bar{f}^t = \delta c^t \odot c^{t-1} \odot \sigma'\left(\bar{f}^t\right) \tag{2.27}$$

$$\delta \bar{i}^t = \delta c^t \odot z^t \odot \sigma'\left(\bar{i}^t\right) \tag{2.28}$$

$$\delta \bar{z}^t = \delta c^t \odot i^t \odot g'\left(\bar{z}^t\right) \tag{2.29}$$

where Δ^t would be the vector of deltas that will be transferred to the next layer

below. When applying the formulas to the last layer, it corresponds to $\frac{\partial E}{\partial y^t}$. If there is another trainable layer below, the deltas for the inputs should be calculated, as well [43]:

$$\delta x^t = W_z^T \delta \bar{z}^t + W_i^T \delta \bar{i}^t + W_f^T \delta \bar{f}^t + W_o^T \delta \bar{o}^t \tag{2.30}$$

Now we have all the information needed to compute the gradients for all the weights. In the following formulas, $\star$ is any of $\{\bar{z}, \bar{i}, \bar{f}, \bar{o}\}$, and $\langle \cdot\, , \cdot \rangle$ indicates the outer product [43].

$$\delta W_\star = \sum_{t=0}^{T} \langle \delta \star^t, x^t \rangle \tag{2.31}$$

$$\delta R_\star = \sum_{t=0}^{T-1} \langle \delta \star^{t+1}, y^t \rangle \tag{2.32}$$

$$\delta b_\star = \sum_{t=0}^{T} \delta \star^t \tag{2.33}$$

$$\delta p_i = \sum_{t=0}^{T} c^t \odot \delta \bar{i}^{t+1} \tag{2.34}$$

$$\delta p_f = \sum_{t=0}^{T} c^t \odot \delta \bar{f}^{t+1} \tag{2.35}$$

$$\delta p_o = \sum_{t=0}^{T} c^t \odot \delta \bar{o}^t \tag{2.36}$$

2.2 Anomaly Detection

Anomaly detection, also called outlier detection, is the procedure of detecting slight deviations in the data points compared to the majority of data [45]. Outliers, abnormalities, and deviants are all alternative terms for anomalies in the literature. Noise and faults in the data acquisition process could result in anomalies in the dataset. However, anomalies could, most importantly, indicate the occurrence of a new unknown event. Hawkins [46] describes an anomaly as a data instance that differs

considerably from the rest as to strike the question of whether a different system was the cause [47].

With early studies dating back to the 1960s, anomaly detection has been a popular research area for a long time [48]. Recently, deep learning has revolutionized many domains with its superb capabilities in learning from high-dimensional spatial and temporal data. A number of proposed deep anomaly detection approaches have illustrated a considerable superiority over traditional methods in a wide range of real-world problems [45, 47]. Even with the recent studies, however, there is still a noticeable lack of deep anomaly detection methods [47].

A review of conventional anomaly detection methods is provided in [49–53]. However, these studies do not cover the recently proposed deep anomaly detection techniques. In this section, we will present a summary of the different types of anomaly detection methods, focusing mainly on modern approaches. Based on the training approach and the availability of data, deep anomaly detection models could be divided into the following categories [47].

In supervised deep anomaly detection models, labeled samples of both normal and anomaly data are used for training a deep classifier. These approaches generally consist of two models, a feature extractor succeeded by a classification algorithm. Deep supervised models need a large number of training samples. Supervised approaches of deep anomaly detection are not as appealing as semi-supervised and unsupervised approaches, the main reason for which is the lack of availability of properly labeled data. Additionally, due to the class imbalance caused by the unequal number of data samples for normal and anomaly classes, the performance of deep supervised classifiers used as anomaly detectors is less than ideal. Moreover, high complexity in the dataset causes deep supervised approaches to fail to isolate the normal data from anomalies [47].

Semi-supervised methods, also referred to as one-class classification, only train on the data belonging to one class, typically the normal class. [54, 55] provide an anal-

ysis of deep semi-supervised methods for anomaly detection. These models find the boundary containing the normal data points and identify the outliers as anomalies [56, 57]. The most important outcome of this method is the need for labeled data for only one class, which can take advantage of the abundance of normal data in most cases. This could lead to noticeable a performance increase in comparison to unsupervised methods. Nonetheless, the shortcomings of semi-supervised approaches also apply to deep anomaly detection [47]. Based on the research done in [58], unless certain evidence indicates the existence of a relationship between the labels and the distribution of the unlabeled data, semi-supervised methods cannot outperform supervised methods, if not the other way around. Among the investigated architectures for semi-supervised deep anomaly detection are: convolutional neural networks [56, 59], deep convolutional recurrent neural networks [60], and generative adversarial networks [61].

The hybrid models are made up of two segments where deep learning models are usually used as feature extractors. In deep hybrid methods, the features learned by the deep models are then fed to conventional algorithms such as one-class Support Vector Machine (SVM) classifiers [47]. Some studies have achieved very promising results using various hybrid models [62, 63]. However, among its downsides is the fact that the hidden layers of the deep model use generic loss functions as opposed to anomaly detection specific functions, which could have a negative impact when the deep model acts as a feature extractor [47]. A few deep hybrid architectures employed for anomaly detection are: deep belief networks–support vector data description [62], deep autoencoder (DAE)–K nearest neighbor [64], and autoencoder–one-class support vector machine [65].

One-class neural networks integrate the representation learning capabilities of deep networks with the one-class objective function, in order to develop boundaries that isolate the normal data instances from the anomalies. Two of the main examples of such models use a hypersphere (Deep Support Vector Data Description or Deep

SVDD [66]) and a hyperplane (OC-NN [67]) as their one-class objective. The research findings indicate one-class neural networks are among the state-of-the-art methods especially for complex datasets. However, possible lengthy training times especially when working with high-dimensional data is one of their main downfalls [47].

One-class adversarial networks (OCAN) [68] is an end-to-end one-class anomaly detector that exploits the concept of bad GANs introduced in [69] to produce bad instances using normal data in the training set. Bad in this context means the generator distribution should not match the true data distribution. Hence, the data generated in bad GANs are supposed to be complementary to the training dataset as opposed to matching, on the contrary to traditional generators. A bad generator will thus help improve the generalization performance of a semi-supervised anomaly detection algorithm.

Unsupervised anomaly detection approaches rely on the intrinsic properties within the data such as distances or densities. Deep neural networks architectures are employed to identify these properties in the dataset [70]. A major advantage of unsupervised learning is its cost-effectiveness since there is no need for labeled data for the training process. Nevertheless, obtaining good performance in complex datasets is often very difficult. Unsupervised methods are also quite vulnerable to noise and faulty data and, thus, are generally outperformed by supervised or semi-supervised models. Furthermore, with autoencoders as the main unsupervised deep method for anomaly detection [71], there are critical hyper-parameters, such as dimensionality reduction, to be tuned [47]. Several deep unsupervised anomaly detection applications are: autoencoders [72–74], LSTM [75], and GAN [76].

Other notable methods and techniques used for anomaly detection are as follows. Transfer learning is a very useful approach in the absence of sufficient data and has been researched for anomaly detection [77–79]. Zero Shot Learning (ZSL) identifies classes that did not have any representative samples in the training set, using some given knowledge about the class, usually in natural language descriptions. Some

implementations of ZSL in anomaly detection have been studied [80–82]. Deep Reinforcement Learning (DRL) is another novel approach that is starting to attract attention in anomaly detection as well. Such models learn to detect new anomalies via reward signals, without the need for prior assumptions about anomalies. A few DRL based anomaly detection have been introduced [83, 84]. However, these methods demand significantly more research and investigations [47].

2.3 Time-series Delay Embedding

Machine learning has been used for time series forecasting by various authors [85–89]. The assumption in all of these studies is the deterministic nature of time series. This means the observations of the system through time follow a unique trajectory in the state space. Therefore, the next state of the system could be uniquely determined based on the knowledge of the current and previous states. Nevertheless, time-series data only provides observations of the system, without information regarding the state space. Therefore, time-series prediction requires a technique to rebuild the state space based on the data points provided in the time-series [90].

Delay embedding is a conventional method for acquiring delay vectors of a time series. Delay vectors contain delayed observations of the time series with the dimension and delay indicating the number of observations in the vector and the delay between two consecutive observations in the delay vector, respectively. Based on Takens' theorem [90, 91], on condition that the dimension is properly selected, the state space represented by the delay vectors corresponds to the original state space of the system, thus the delay vectors could be used for time-series forecasting. Equation (2.37) illustrates the general format of delay vectors in a univariate time series [90]:

$$\vec{S_n} = (S_{n-(m-1)d}, S_{n-(m-2)d}, ..., S_n) \tag{2.37}$$

where $\vec{S_n}$ is a delay vector with the n^{th} element of the time-series as its latest compo-

nent. The delay vector contains a total of m components, with delays of d in between. S_n denotes the n^{th} component of the time-series.

we will utilize two methods to determine proper values for these two parameters separately. Delay-embeddings with the same dimension (*i.e.* m) but different delays (*i.e.* d) are mathematically equivalent to each other. That said, in the real-world data with the introduction of noise, the delay parameter has a major impact on the result. In general, Smaller values of d cause higher correlations between consecutive elements of the delay vector and, thereby, causing the delay vectors to be concentrated around the diagonal of the embedding space. This could potentially result in important features becoming incomprehensible. Larger values, on the opposite side, result in the elements of the delay vector being almost independent, which will lead to unclear structures. Common approaches for establishing the delay are the first minimum of the *time-delayed mutual information* and the first zero in the *auto-correlation function* [90, 92]. The former being the more refined concept is the choice for this research.

To compute the time delayed mutual information, a histogram with a resolution of ϵ is used. p_i would, then, indicate the probability that the time-series holds a value within the i^{th} bin of the histogram, and $p_{ij}(\tau)$ denotes the probability that $s(t)$ is within bin i and $s(t+\tau)$ is within bin j. Ultimately, Equation (2.38) is used to compute the mutual information [90]:

$$I_\epsilon(\tau) = \sum_{i,j} p_{ij}(\tau) \ln p_{ij}(\tau) - 2 \sum_i p_i \ln p_i \tag{2.38}$$

False nearest neighbors is a technique that is used to find the proper dimension of the delay embedding [93, 94]. The fundamental concept is finding the number of data instances that are wrongfully neighbors, only due to an improper projection to a lower dimensionality. If points in the dataset are projected to a lower dimensional embedding, some axes among the coordinates of the points are lost. consequently, in the lower dimensional space, some points whose removed coordinates vary signifi-

cantly, might be closely located resulting in "false neighbors". Therefore, to evaluate false nearest neighbors, all pairs of nearest neighbors in the dataset when projected to an m dimensional space are considered. For each pair, the ratio of their euclidean distances in $m+1$ dimensions over m dimensions is calculated. The pair is considered a false neighbor in the event that this ratio is greater than a predetermined threshold r [90]. The fraction of false neighbors is then computed using Equation (2.39):

$$X_{fnn}(r) = \frac{\sum_{n=1}^{N-m-1} \Theta(\frac{|S_n^{(m+1)} - S_{k(n)}^{(m+1)}|}{|S_n^{(m)} - S_{k(n)}^{(m)}|} - r)\Theta(\frac{\sigma}{r} - |S_n^{(m)} - S_{k(n)}^{(m)}|)}{\sum_{n=1}^{N-m-1} \Theta(\frac{\sigma}{r} - |S_n^{(m)} - S_{k(n)}^{(m)}|)} \tag{2.39}$$

where $S_{k(n)}^{(m)}$ represents the nearest neighbor for S_n when projected to the m dimensional space, and Θ is a step function which outputs one for any input greater than or equal to zero, and zero for all negative inputs. The initial step function Θ found in the numerator would then indicate one for a false neighbor when the distance ratio is greater than r. The second step function's purpose is to ignore all pairs that are initially farther than σ/r apart, with σ being the standard deviation of the data. Since they are too far apart to be considered neighbors to begin with, and there is not sufficient space to move farther away [90].

The proper dimensionality is, then, established by considering the plots of the fraction of false nearest neighbors with respect to the number of dimensions, for multiple selections of the threshold r. The point on the dimension axis where all of the plots plateau is deemed the optimal dimension for the time-delay embedding of the time-series.

The above methods and techniques are all applied to univariate time-series. When working on a multivariate time-series, the delay vector is constructed by concatenating the delay-embeddings of each variate in different rows [95–97]. The TISEAN software package [94] includes all of the required implementations of the methods used in this research, with regards to nonlinear time-series analysis.

Chapter 3

The Proposed Model

In this chapter the proposed model is introduced. In order to build an anomaly detector, two separate models are deployed. The first one, "the normal model", is a one-step-ahead time-series forecasting algorithm. The normal model takes a sequence of values in the time-series as inputs and predicts the next state of the system. The second model, "the anomaly detector", uses the output of the normal model as the expected behavior for the system and detects anomalies by comparing the observed actual values in the time-series with the expected behavior [49, 98]. Figure 3.1 illustrates the outline of the architecture.

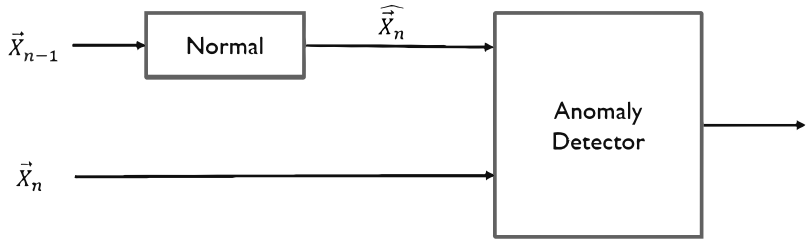

Figure 3.1: Architecture outline. $\vec{X}_{n-1}$ represents the window of previous values, $\widehat{\vec{X}}_n$ contains the forecasted values for the next time-step (n), and $\vec{X}_n$ holds the actual observations of the next time-step [31].

3.1 The Normal Model

The normal model is in charge of predicting the systems next behavior. The idea is that if the system is working properly and no fault occurs, we should be able to predict

the next state and the system is still in a healthy condition. In order for the algorithm to be able to predict the behavior, we need to have the system's data in time-series format. Each feature of the system (voltage, current, pressure, temperature, *etc.*) will be collected as an independent time-series and the next state of the system will be predicted in terms of the next value in each of these time-series. The data that the algorithm needs at each run to predict the next state is not all previous values in the time-series as this would be demanding and also unnecessary. As discussed earlier in Section 2.3, according to Takens' theorem [90, 91], given the parameters (dimension and delay) are properly selected, the state space represented by the delay vectors corresponds to the original state space of the system, thus the delay vectors could be used for time-series forecasting. Hence, we can predict the next state using just a small window of previous values in each time-series.

Taking a look back at Figure 3.1, the vector $\vec{X}_{n-1}$ represents the window of previous values and $\widehat{\vec{X}}_n$ is the predicted next state of the system. As we discussed, the concept of the normal model is to tell us what we expect from the healthy system when no anomalies occur. Therefore, this model is only trained on healthy data with no introduction of any abnormal behavior.

3.1.1 Architecture

After the delay embedding is done on the time-series data, our datasets for training the normal model become pretty straight-forward. In each sample, there are m inputs for each feature and one output. This means for a system described by n numbers of features, each sample includes n by m inputs and n targets. With the above explanations, there are numerous algorithms that could potentially be used for such a model, however, given the nature of time-series and the capabilities of deep learning, the following two architectures would be the most suitable for this model: 1. Convolutional Neural Networks (CNN) especially 1-dimensional CNN (Section 2.1.1) and 2. Recurrent Neural Networks (RNN) specially LSTMs (Section 2.1.2) [31].

Depending on the specific datasets used, the trained models would have a few layers of 1-d CNNs or alternatively LSTMs, with ReLU activation, followed by a last fully-connected (dense) layer for the outputs. Possible additions include Batch Normalization, Max Pooling, and Dropout layers which will be determined while performing hyper-parameter tuning. Figure 3.2 illustrates the overall structure of an example of the normal model.

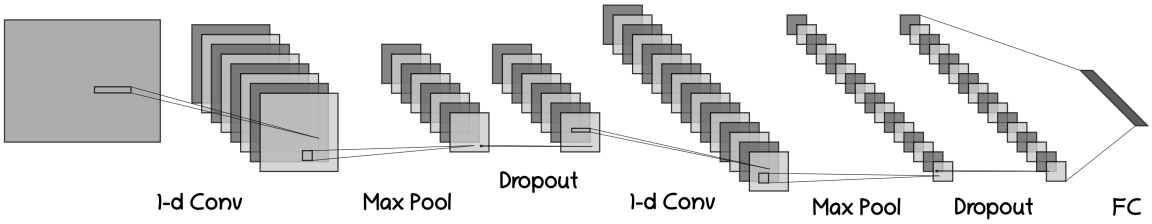

Figure 3.2: An example of the normal model.

Based on the experiments done with different datasets for different problems, both 1-dimensional convolutional neural networks and LSTMs seem very capable of predicting time-series data for the normal model. Therefore, determining the superiority of one over the other is dependent on the particular problem and dataset. Nonetheless, it is worth mentioning that when comparing models of similar size and depth, the 1-dimensional CNN based models are much more computationally efficient than LSTM based models. This is particularly important if models are trained on large amounts of data.

3.2 The anomaly Detector

With the output of the normal model giving us the expected measurements for the next time-stamp, the anomaly detector is needed to compare what happens with what is expected to happen to see if there are anomalies. To this end, the inputs of the anomaly detector include the predicted expected values in all time-series (all features) for the next time-stamp (the outputs of the normal model) as well as the actual observations for the next time-stamp. Looking back at Figure 3.1, the vectors $\widehat{\vec{X}}_n$

and $\vec{X}_n$ are the inputs of the anomaly detector which represent the predicted expected measurements and the actual observations for the next time-stamp, respectively.

3.2.1 Architecture

According to the above description of the anomaly detector, this model needs two values per each feature as its inputs. This means that if a dataset has n features (temperature, pressure, flow, *etc.*) the anomaly detector needs $2 \times n$ input values. Based on previous experiments, adding the differences of the expected values and the actual observations ($\widehat{\vec{X}}_n - \vec{X}_n$) can sometimes help the leaning process. Should we choose to add these values as well, the model will have $3 \times n$ inputs. At this point in the research, the model is only predicting the occurrence of an anomaly which requires only one output.

On the contrary to the normal model which is inherently trained only using the healthy data, this model could be trained using unhealthy data in addition to the healthy data. Whether or not to include the unhealthy data in the training process is dependent on the data restrictions as well as the types of anomalies available in the unhealthy data. The following explains how each of these factors will affect this process:

- **Data restrictions:** to understand the restrictions we'll have to talk about how the data is provided and briefly go over the data preprocessing needed. The data is available is multiple files each containing a few hours of time-series data. Each file's data is collected under specific conditions (*e.g.* type of sand, type of leak if there is any, *etc.*). Each file could either include only one type of healthy/unhealthy data or have a portion of each in the event that the anomaly (*e.g.* a leak) occurs half-way through. Because of the nature of time-series analysis and anomaly detection, in order to divide all of the data into train and test datasets, we have to perform a chronological single-split. In other words, a time-stamp along the duration of the time-series data is selected (*e.g.* 3:00:00) and

all data before this time-stamp is dedicated to the training set and everything else is for the testing set. The chronological split could be done twice to have training, validation, and test sets. Given this explanation, on condition that the data provided includes separate time-series for healthy and unhealthy data (each containing only healthy or unhealthy data) for all available conditions, regardless of what time-stamp selected, all separated datasets (train, validation, and test) include both healthy and unhealthy data. The problem arises when all files include time-series where an anomaly occurs half-way though. Therefore, where ever the time-stamp is selected, only one data set among the three train, validation, and test sets can contain both healthy and unhealthy data. As both healthy and unhealthy data are needed in the out-of-sample test set in order to have a reliable testing process, this data restriction will limit us in using only healthy data to train the anomaly detector. Figure 3.3 illustrates this issue.

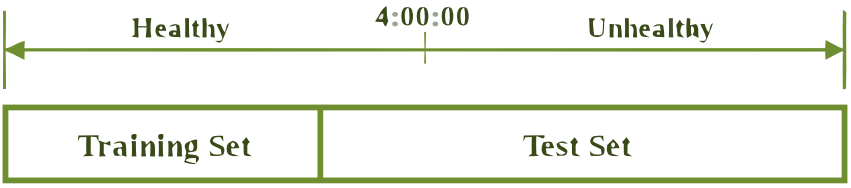

Figure 3.3: Data restrictions: the impact of an anomaly occurring half-way through the time-series data, at 4:00:00.

- **Types of anomalies available:** Many types of anomalies are usually present when applying anomaly detection systems. For instance, in the case of a leak detection system, there are different sizes of leaks and ruptures, there could be reading errors from the sensors, and *etc.*. When the anomaly detector is trained on unhealthy data it would be more inclined to detect only the types of anomalies provided to it. Hence, when the types of anomalies available or the amount of data, in general, is not sufficient, it might be a better idea not to train on unhealthy data at all. Nevertheless, Thanks to the main concept of

the proposed architecture that includes a normal model predicting the expected healthy output for the next time-stamp, the impact of this issue on the overall performance of the model is greatly minimized.

Based on the above descriptions, we have to choose whether or not we will be training the anomaly detector on both healthy and unhealthy data. If the answer is yes, this model turns into a common classifier that distinguishes between two classes of data. Examples of architectures that were explored in this scenario are Multi-layer Perceptron (MLP), Decision Tree, and Random Forest. On condition that the anomaly detector is chosen to be trained only on healthy data, the required model is a one-class classifier. One-class classifiers attempt to find the border that separates the normal data from everything else, by only training on the normal (healthy) data. Two of the most popular architecture used as one-class classifiers are Isolation Forests [99, 100] and One-class Support Vector Machines [101, 102].

3.3 Data Preprocessing

To use the data in machine learning algorithms, Several steps of data preprocessing are needed. The following explains all the steps required for the proposed model in the desired order.

3.3.1 Missing Values

The first step of data preprocessing in all projects is to look for missing or wrong values. Wrong values are particularly harder to find, that said, there's a high chance of these at the start and the end of time-series data. other ways of finding these would be looking at possible ranges for all values and also finding values that suddenly change without an anomaly occurring. There are several approaches to dealing with these values: on condition that the total number of these values is insignificant compared to the amount of data available, best practice is to simply disregard the samples

including even one missing/wrong value. Otherwise if losing all these samples would mean losing a considerable amount of data, replacing the said values with the mean or median of the same feature under the same conditions is another possibility.

3.3.2 Feature Scaling/Normalization

The next step is feature scaling/normalization. In this step, all independent features will be transformed to have a similar desired range or distribution. The reason is different features' ranges vary widely and machine learning algorithms will not perform properly without scaling. For Instance, if a feature has a very broad range of values compared to other features, this feature will dominate the decision-making process of the algorithm. Hence, scaling/normalization should be done for all features so each feature has a similar contribution to the final decision. Additionally, gradient descent will converge extremely faster when feature scaling is performed.

Now that the concept and the motivation for this process are clear, the difference between scaling and normalization must be discussed. Even though both terms are often used interchangeably, they are not the same. Scaling changes the range of the data. That is it changes the minimum, the maximum, and essentially the length of the steps in between. Normalization, however, aims to change the shape of the distribution of the data. There are several methods of scaling/normalization, the following will examine a few of the popular methods.

- **Min-Max Scaling:**

  This is a simple method which rescales the data to have the desired minimum and maximum. To rescale the data to have the desired range [a b] Equation (3.1) is utilized:

$$X' = a + \frac{(X - min(X))(b - a)}{max(X) - min(X)} \tag{3.1}$$

  where a and b are the new min and max values and X' is the scaled value.

the most popular version of min-max scaling is to scale the range between 0 and 1, which could be accomplished using Equation (3.2):

$$X' = \frac{X - min(X)}{max(X) - min(X)} \tag{3.2}$$

- **Standardization:**

  This is also a widely used method in which the data distribution is changed to have zero mean and unit variance.

$$X' = \frac{X - average(X)}{\sigma(X)} \tag{3.3}$$

  where X' is the normalized value, $average(X)$ is the mean of the feature vector X, and σ is the standard deviation.

It is worth noting that one method is not always superior that the others and which one to choose depends on the concept of the problem, the types of algorithms used, and the data available. It's always a good idea to test a few to compare. In this research, Min-Max Scaling with a range of [0 1] seemed to produce the best results so far.

3.3.3 Data Splitting

In machine learning when training and testing models it is mandatory to split the data into different datasets. The reason is fairly simple: by testing on the data used for training one can not obtain realistic results for comparing models' performances. Due to the nature of the problem which is time-series analysis, random splitting is not recommended. Instead, we perform chronological splits in which data is split at some time-stamp through the time-series, before any process that rearranges the order of the values in the series. The data is first split into a training set and an out-of-sample

test set. In order to have a validation set used for tuning the parameters, we can then repeat the splitting process on the training set and turn it into two datasets.

3.3.4 Delay Embedding

Only when we are done splitting the data can we perform delay embedding of the time-series. This way we make sure there is no data overlap between the split datasets. The concept of time-series delay embedding was thoroughly discussed earlier in Section 2.3 where we explained why this process is needed and how it is applied to time-series data. In a nutshell, after finding the proper d (*i.e.* delay) and m (*i.e.* dimensionality), a number of previous observations in the time-series are concatenated together with the next value in the time-series as the targets. Equation (3.4) demonstrates the format of the delay vectors:

$$\vec{S_n} = (S_{n-(m-1)d}, S_{n-(m-2)d}, ..., S_n) \tag{3.4}$$

It is worth emphasizing that the values m and d are very critical and proper tuning is absolutely mandatory. The TISEAN package [94] includes the implementation of mutual information and false nearest neighbors which are required to find these parameters.

Delay embedding is the last step of data preprocessing needed for the proposed model. In the end, the data sets should have samples with dimensions according to what was described as suitable for the normal model. Regarding the anomaly detector, the targets in these datasets concatenated with the outputs of the normal model would be used as the inputs, and the targets are the states of the occurrence of anomalies that are given with the time-series data.

3.4 Parameter Exploration and Evaluation

All hyper-parameters of both models are explored and tuned. Different architectures have various hyperparameters, such as the number of layers, number of units for each layer, size of filters, *etc.*, which means a comprehensive tuning process must be done. A reliable evaluation process is a necessity when it comes to hyper-parameter tuning. Two factors are important for a reliable evaluation: data set aside for validation and sufficient measures of evaluation.

Regarding the data needed for validation, an adequate approach is K-fold Cross-Validation. In this approach, the training dataset is divided into K equal portions and in K runs, the models are trained on $K-1$ folds and the remaining is used for evaluation. The averages of the measurements for all of the runs are then considered for comparison. Although suitable for some scenarios, this method has shortcomings which means we need to use other methods as well. The most important downfall is having to do the training process K times which can be very demanding for deep architectures. In addition, if the training dataset contains healthy data only, the evaluation measurements would all be limited to healthy data as well. Thus, having a separate validation dataset is essential. For the normal model, a portion of the training data is put aside for validation, and therefore there is no need for using the test set. For the anomaly detector, this depends on the data restrictions discussed earlier in Section 3.2.1. Assuming there are separate time-series for healthy and unhealthy states under all conditions, preparing another validation set including both healthy and unhealthy data for the anomaly detector is also possible. However, If all time series include the occurrence of an anomaly somewhere in the middle, only one dataset could include both healthy and unhealthy data, using the chronological split method, which would be the test set. In this case, we use a portion of the healthy data to create the validation set, as well.

As for the evaluation measurements, multiple measures are obligatory, in order

to evaluate and compare the models' performances in different aspects. The normal model, on account of the nature of a time-series forecasting algorithm that has continuous output values, is evaluated using the *Root-Mean-Squared Error (RMSE). RMSE* is one of the most frequently used measures of difference between values predicted by a model and the observed values. When evaluating the forecasts, *RMSE* is the square root of the average of the residuals. Residuals are the differences between the predicted or estimated values and the observed values or, in other words, the prediction errors. Equation (3.5) demonstrates the exact formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\widehat{y}_i - y_i)^2}{N}} \tag{3.5}$$

where N is the sample size, $\widehat{y}_i$ is the i^{th} predicted value, and y_i is the i^{th} observed value. As the formula suggests, RMSE is scale-dependent. This means that when comparing different models on the same dataset there is no problem but when it comes to comparison over different datasets if the datasets are not scaled to have the same range, the RMSE results need scaling which can simply be done via dividing by the amplitude of the models' inputs (*i.e.* the amplitude of the time-series data).

The anomaly detector needs different evaluation methods, as it is a type of classification algorithm. Additionally, the outputs of the anomaly detector would be the final outputs of the entire model which means several measurements are needed to evaluate the performance of the model in different categories. The simplest form of evaluation for a classification algorithm is accuracy. Even though it is the main evaluation measurement used for classifiers in many projects, accuracy alone would not provide a sufficient analysis of the model's performance, especially for an anomaly detector. Nonetheless, accuracy does gauge the overall capability of the model and therefore is included in the measurements. In binary classification, the accuracy is defined as the ratio of the correct predictions (both true positives and true negatives) to the total number of predicted samples. Equation (3.6) represents this measurement:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.6}$$

where TP is the number of true positives (positive cases that are correctly predicted as positive), TN is the number of true negatives (negative cases that are correctly predicted as negative), FP is the number of false positives (negative cases falsely predicted as positive), and FN is the number of false negatives (positive cases falsely predicted as negative).

The other necessary performance measurements for an anomaly detector are TPR (True Positive Rate), FPR (False Positive Rate), and FNR (False Negative Rate). These measurements essentially analyze the falsely predicted portion of all examined samples to determine whether they are mostly false positives or false negatives. A high TPR (which results in a low FNR) generally means most anomalies are predicted and the undetected cases (which are falsely predicted as negatives) are low in number. A high FPR shows that the algorithm has produced many false alarms (predicted an anomaly when there was none). This information is very beneficial for two reasons:

1. Figuring out whether the algorithm has problems with detecting the anomalies or it produces too many false alarms is useful for tuning the models' hyper-parameters. By gaining insight into what the problem is, we can understand which part of the architecture is flawed and therefore realize how to change it to get better results.

2. In most scenarios, there is a trade-off between having a low number of false alarms and a low number of undetected anomalies. This means the algorithm could be tweaked for instance to have few false alarms at the expense of more undetected anomalies. Which should be prioritized however differs case by case and depends entirely on the problem that is being addressed. In the case of a pipeline leak detection system, for example, having false alarms means the pipeline would get shut down for no reason which in turn can be very expensive

for the company. Hence, a low false alarm rate is of more importance.

According to the above explanations, evaluating the models using TPR, FPR, and FNR is deemed mandatory. Equations (3.7) to (3.9) provide details of how these measures are calculated:

$$\boldsymbol{TPR} = \frac{TP}{P} = \frac{TP}{TP + FN} = 1 - FNR \tag{3.7}$$

$$\boldsymbol{FPR} = \frac{FP}{N} = \frac{FP}{FP + TN} \tag{3.8}$$

$$\boldsymbol{FNR} = \frac{FN}{P} = \frac{FN}{TP + FN} = 1 - TPR \tag{3.9}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

All of the implementations in this research are done using an Intel® Core™ i7-7800X CPU and an NVIDIA Titan Xp GPU, running a Ubuntu 18.04.4 LTS. All codes are written in Python using libraries and frameworks such as Tensorflow [103], Keras [104], Scikit-learn [105], pandas, NumPy, and Matplotlib [106].

Chapter 4

Application in Pipeline Leak Detection

4.1 Introduction

The most economic, eco-friendly, and safest way to transport large amounts of gases, oil, and other fluid products over long distances is through pipelines [107–109]. Nevertheless, they must meet high standards in terms of efficiency, reliability, and safety. On condition that they are maintained properly, they can last without leaks for an unlimited time. Few factors that cause pipeline leak are corrosions, natural hazards, mechanical damages, and, commonly, damage from a nearby evacuation[109]. Despite the rare event of these factors occurring, they each are potent to cause significant environmental and financial costs [110].

Even though preventing leaks should be the main concern in pipelines, leak detection systems significantly decrease the impact and localize leaks as soon as they occur. This will aid enhance system reliability and productivity by minimizing inspection time, downtime, and environmental damages. Hydraulics model-based systems are the most common implementing leak detection systems. Regardless of the low observability of the pipeline that these methods provide, they depend on the ongoing calibration of the equipment and models' parameters. We use machine learning-based inferential sensing as an alternative in this study. Our solution is considerably more cost-effective and less sensitive to models' parameters.

This research focuses on inferential sensing for pipeline leak detection. Premised on the results, we are confident that for designing a comparatively low-cost solution, machine learning approaches would be quite satisfactory. We believe that the pipeline leak detection system must be categorized as an anomaly detection problem. Accordingly, we created a model of normal behavior and an anomaly detector that is able to detect any deviations from the normal behavior as anomalies. We chose to experiment with 1-dimensional Convolutional Neural Networks and Recurrent Neural Networks (specifically LSTM) for the normal model, given the observed capabilities of deep learning for designing a time-series forecaster [31]. Regarding the anomaly detector, Given that the number of the inputs for this model are not many, we focused solely on conventional ML algorithms. We concluded that the best combination on our datasets is a 1-d convolutional neural network as the normal model and an Isolation Forest for the anomaly detector.

The rest of this chapter is structured as follows: the next section provides a literature review of the leak detection systems. Next, the dataset used is described, followed by the results and discussions. Finally, a conclusion is provided in the last section.

4.2 Literature Review

A broad range of leak detection systems (LDS) are utilized in energy pipeline industries that are recognized by the American Petroleum Institute (API). These LDSs are organized in four categories: non-continuous internally based, non-continuous externally based, continuous internally based, and continuous externally based. Based on their specific approach and the different technologies used, each system has its pros and cons.

Any system in which detection is conducted in the form of physical inspection to find any leaked material falls under the category of non-continuous externally based leak detection systems. Additionally, these systems are further categorized into

sensor-based monitoring and physical inspection. The first group consists of methods such as ground-penetrating radars, satellites, smart pigs, soil sampling, tracer chemicals, and sniffer tubes [111]. The second category's systems include satellites, ground-based line surveillance, one call system/public awareness, and aerial surveillance. Public awareness system/one-call system communicates any leak information to stakeholders and addresses any leak detected and informed by laborers, operation personnel, communities, and public units [112]. As a form of manual inspections, ground-based surveillance involves workers visually inspecting the pipeline to find any kind of damage or leaks. Such manual inspections are very expensive and hazardous, and hence not at all cost-effective [113]. On the contrary, aerial surveillance, such as the use of Unmanned Aerial Vehicles (UAVs), benefits from low costs, high saftey, reliability, and usability in bad conditions. Thanks to these advantages and their availability, they are becoming more and more popular. UAVs equipped with wireless sensors could also be incorporated in Wireless Sensor Networks (WSNs). WSNs are capable of real-time monitoring which has been leveraged to increase their leak detection functionality and reliability [114]. Underground Wireless Sensor Networks (UWSN) also provide other suitable features such as concealment, easy installment, redundancy, and low power sensors [115].

In the tracer chemicals method, a tracer compound, which is a non-hazardous highly volatile gas, is distributed through pipelines, so that any leak could get identified by detecting the escaping gas. Probes located above the pipeline in the soil are used to detect the leaks [116]. Even though this approach has some benefits, such as high sensitivity and very low false alarm rates, it is highly expensive. Additionally, if used in overland pipelines, the leaked gas will disperse in the environment with little chance of detection.

"Smart Pigs" are devices that are placed in the pipeline to move along it by the flow of the liquid while recording physical data about the pipeline [117]. Smart pigs are usually equipped with specific tools, such as ultrasonic sensors, and are very

sensitive. Nonetheless, many of the pipelines, including about 30% of the pipelines in the United States, are not suitable for the utilization of smart pigs [118]. Furthermore, pigging pipelines is an extremely expensive process. Additionally, the ultrasonic pulse velocity (UPV) test that is commonly performed in this approach [119] introduces a lot of noise [120] which corrupts the information carried via the original signal [121].

The leak detection systems that use field sensors directly are categorized as continuous externally based techniques. Examples of such sensors include acoustic sensors, sensing cables, chemical analyzers, and video cameras. Considering that burying sensing cables with new pipelines requires very low additional costs, the use of these sensors such as fiber optic cables has been increasing in popularity [122]. Fiber optic sensing cables detect changes in temperature once the leaked material reaches the cable. Other types of sensing cables detect measures such as acoustic emissions caused by a leak, changes in electrical properties (capacitance, resistance, etc.), and vapor or liquid. Nevertheless, since implementing such cable-based external systems for existing pipelines are could be very expensive, their are only applied to local high-risk areas [123]. In addition, it should be emphasized that a segment of the cable could get damaged and require replacement, which means the system becomes nonfunctional [115].

Computational pipeline monitoring (CPM) is a continuous internally based leak detection system. CPM systems obtain field measurements of pipeline parameters, use a computational algorithm to estimate new values, and then indicate the status of the leak based on those values [124]. The supervisory control and acquisition (SCADA) system provides the information collected from the sensors to the CPM. Factors that determine the performance of the system include the number, type, and accuracy of the sensors, parameters of the SCADA system, and the computational algorithm. The complexity of the CPM algorithm varies significantly from basic calculations and hydraulic models, to artificial intelligence and machine learning techniques. Inferential Sensing, also called Soft Sensing, has been utilized in many different condition

monitoring problems. Soft sensing's main concept is to estimate a complex system state with the information provided by several simple sensors. Soft sensing has been used for a long time, however, with the recent introduction of artificial intelligence and machine learning, soft sensing has obtained the capability of estimating extremely complex systems [125]. A machine learning-based leak detection system uses data collected from the sensors. To create a reduced feature set, several tests are performed on the features provided by the sensors data. Subsequently, an ML model is trained on the data with the purpose of detecting leaks or failures in the pipeline [126].

At present, the incorporation of soft sensing in leak detection systems is gaining popularity owing to the wealth of information that it presents, as well as the capability of being used with the existing available equipment. Soft sensing is able to provide valuable information such as the type of the leaked material (e.g. water, crude oil, natural gas, gasoline, mineral oil, or diesel), and with the emerging methods like machine learning a lower sampling frequency is required for the sensor measurements, compared to the continuous externally based LDSs [127]. Nevertheless, artificial intelligence has a few drawbacks. There is no specific machine learning algorithm that is flawless and performs well in all cases, and each algorithm has its strengths and weaknesses [107, 128]. Moreover, soft sensing systems are prone to over-complication. Therefore, it is best that these methods are paired with a sensor-based and/or hydraulics model-based monitoring system.

4.3 Dataset

We have used two entirely different types of data: 1. computer-generated data and 2. lab generated data. The first dataset consists of computer-simulated data for an idealized pipeline and contains 4 main features: upstream and downstream flows and pressures of the pipeline. For each feature, time-series data is provided in the lengths of 8 hours. This dataset is mainly used in the first phase of the research, as it is relatively simple with few features and, thanks to the fact that it is simulated

for an idealized pipeline, it would not contain any noise or unwanted/inaccurate measurements.

After several models have been tuned using the first dataset, we have a good understanding of the concept of the problem (*i.e.* pipeline leak detection) as well as the models and how to tweak them to obtain the performance we are looking for. The models' strengths and weaknesses are clearer and we have information about what we need for the next step. This is when the second dataset, the lab generated data [6], comes into play. Including 14 features, This dataset is more likely to provide all the information needed for the anomaly detection. These features include four temperature measurements, three pressure measurements, two flow measurements, one weight measurement, 3-dimensional acceleration measurements, and the Re number. This dataset was gathered using laboratory-scale equipment that imitates the pipeline's behavior. The following describes the utilized devices and the procedure in detail.

The experimental apparatus used to create the lab generated data is illustrated in Figure 4.1. The laboratory-scale test-bed is designed and developed for a comparative assessment of leak detection methods and techniques to generate data. The test-bed needs to be a pipeline loop in order to recreate pipeline failures such as small leaks and rupture, which are then evaluated by a sensor suite.

The operation of the pipeline loop starts by the pump recirculating the fluid with its temperature and velocity being controlled. The pipeline consists of three sections, each utilizing pipes of different diameters: $\frac{3}{4}$", 1", and $1\frac{1}{2}$". This configuration optimizes the pump power by decreasing the fluid resistance by approximately 40%. The $1\frac{1}{2}$"-pipe is equipped with one flow meter and one pressure transducer. The $\frac{3}{4}$"-section of the pipeline includes two pressure transducers, one temperature transducer, and one accelerometer. This segment replicates the real-life conditions of fluid behavior. ultimately, the 1"-pipe is used for the connection to the reservoir and the centrifugal pump.

The $\frac{3}{4}$"-pipe is the section that will encompass leaks, in order to produce the

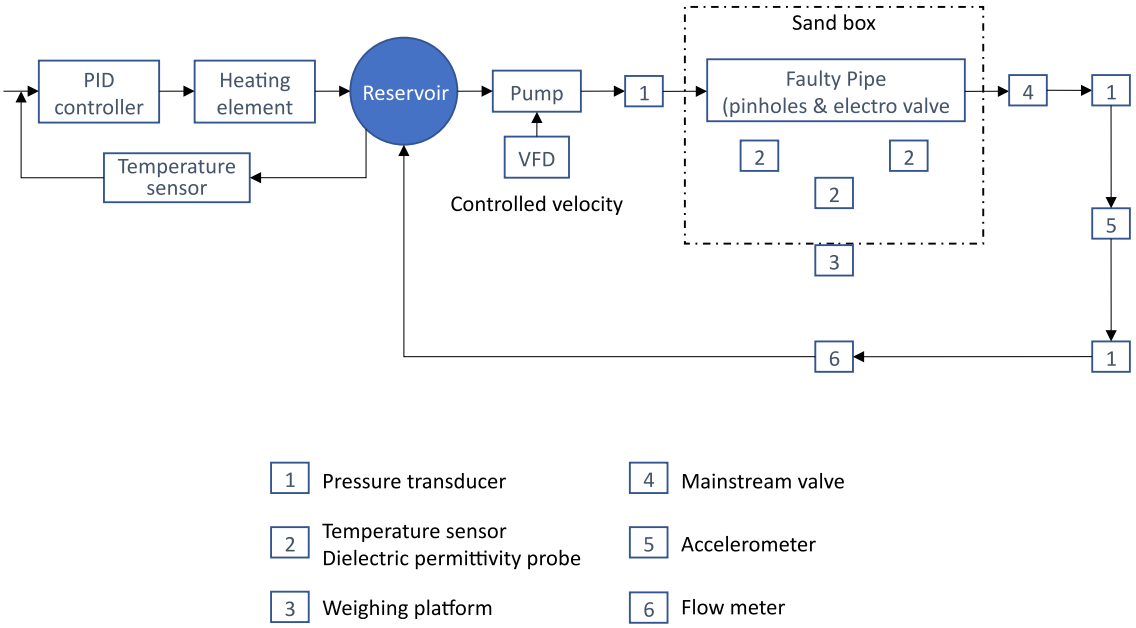

Figure 4.1: The test-bed: apparatus layout [6]

failure modes. A pinhole in the pipe functions as a small leak and the integrated electrovalve is capable of replicating a rupture. This section lies underneath the soil. As a result, the leaked fluid affects the soil's properties. The load cell measures the mass transmitted to the soil box while two pressure sensors measure the differential pressure between both ends of the leaky pipe. The soil impedance is calculated by two dielectric permittivity probes and an accelerometer measures the pipe's vibrations in 3 dimensions. Additionally, A third pressure transducer tests the transient-model-based leak detection technique. The SCADA processes the responding variables in real-time and data is acquired via LabView 2018.

The dataset generated for this research includes three leak states: 3mm leak, 2mm leak, and no leak. Every state of the leak is evaluated using two types of soil: black loam and sand, with two different levels of soil moisture: dry and saturated. Table 4.1 describes all variable ranges. For a more in-depth explanation of the apparatus and the data acquisition process see [6].

Table 4.1: Variable description.

Type of variable	**Variable name**	**Range**	**Unit**
Manipulated	Fluid temperature	$(37, 40)$	$°C$
	Fluid velocity	$(3.5, 5.0)$	m/s
	Pipe diameter	$(\frac{3}{4}, 1, 1\frac{1}{2})$	$Inch$
	Pinhole diameter	$(0, 2, 3)$	mm
Controlled	Type of fluid	Water	-
	Type of soil	Black loam, Sand	-
	Depth of pip	3	$Inch$
	Soil moisture	$(2, 47)$	%
	Depth of soil	$(5, 6)$	$Inch$
	Reynolds number	$(100000, 190000)$	-
Responding	Fluid pressure	< 3.3	psi
	Soil temperature	$(20, 40)$	$°C$
	Soil dielectric permittivity	$(4, 80)$	F/m
	Pipe vibration	< 0.04	g
	Volume of leakage	$(0.489, 2.138)$	lt/min
	Response time	$(1, 20)$	-

4.4 Results and Discussion

As was explained in Chapter 3, the proposed architecture has two models. Accordingly, separate results for each model are provided. the results of the first model, the normal model, will be discussed first. The candidate architectures for the normal model were ultimately narrowed down to 1-dimensional convolutional neural networks (1-D CNN) and long short-term memories (LSTM). The 1-D CNN model includes 2 Conv1D layers with 32 and 64 filters, strides equal to 1, and same padding. BatchNormalization and Maxpooling1D are performed after each layer and one Dropout just before the fully-connected output layer. Regarding the LSTM model, it includes 2 LSTM layers each containing 20 units and a fully-connected output layer. Tables 4.2 to 4.3 compare the performances of 1-dimensional CNN and LSTM models on the computer-generated and lab-generated datasets, respectively. The validation set error is measured in terms of Root-Mean-Square-Error ($RMSE$), as discussed before in Section 3.4.

Table 4.2: Normal model - Computer generated dataset

Model	**Training time**	**RMSE**
1-D CNN	30s	0.01
LSTM	600s	0.01

Table 4.3: Normal model - Lab generated dataset

Model	**Training time**	**RMSE**
1-D CNN	70s	0.09
LSTM	1200s	0.09

According to the numbers, both models demonstrated very similar performances, with the exception of the training time. The time required for the LSTM model is

roughly 20 times the 1-D CNN model, which can be very significant depending on the size of the dataset. The computational efficiency is a sufficient reason on its own to decide the 1-D CNN model over the LSTM model for the next step. With either of the architectures for the normal model, the results of the anomaly detector will ultimately be the same, as both models achieved the same error rates.

Moving on to the anomaly detector, the results for this model is essentially the final results of the proposed model. Therefore, all evaluations have been obtained using the out-of-sample test set. The architectures selected for this model were narrowed down to the Isolation Forest and the One-class SVM. Tables 4.4 to 4.10 illustrate the results. In each dataset, each type of leak is initially examined, and then the entire dataset is used for final evaluation. Using separate evaluations for each type of leak can be very beneficial for the development and tuning stage of the research as well. For instance, if the results on small leaks are not as good as the larger leaks, this means that the boundary of the one-class classifier has to get closer to the healthy data. This information points out which hyper-parameters to change and in which direction.

Table 4.4: Computer generated dataset - 30% leak

Model	**Accuracy**	**TPR**	**FPR**	**FNR**
Isolation Forest	0.9968	0.9962	0.0011	0.0038
One-class SVM	0.9966	0.9966	0.0036	0.0034

Table 4.5: Computer generated dataset - 5% leak

Model	**Accuracy**	**TPR**	**FPR**	**FNR**
Isolation Forest	0.9997	0.9999	0.0006	0.0001
One-class SVM	0.9999	1.0	0.0006	0.0

Table 4.6: Computer generated dataset - 2% leak

Model	Accuracy	TPR	FPR	FNR
Isolation Forest	0.9956	0.9945	0.0003	0.0055
One-class SVM	0.9955	0.9946	0.0014	0.0054

Table 4.7: Computer generated dataset - all leaks

Model	Accuracy	TPR	FPR	FNR
Isolation Forest	0.9959	0.9949	0.0006	0.0051
One-class SVM	0.9927	0.9911	0.0012	0.0089

Table 4.8: Lab generated dataset - 3mm leak

Model	Accuracy	TPR	FPR	FNR
Isolation Forest	0.9994	1.0	0.0014	0.0
One-class SVM	0.9997	1.0	0.0007	0.0

Table 4.9: Lab generated dataset - 2mm leak

Model	Accuracy	TPR	FPR	FNR
Isolation Forest	0.9995	1.0	0.0014	0.0
One-class SVM	0.9984	0.9999	0.0042	0.0001

Table 4.10: Lab generated dataset - all leak

Model	Accuracy	TPR	FPR	FNR
Isolation Forest	0.9995	1.0	0.0014	0.0
One-class SVM	0.9990	0.9999	0.0040	0.0001

The results clearly indicate both models were able to reach very high accuracy. All accuracy and TPR values are well above 0.99, and FPR and FNR measurements are far below 0.01 and almost zero. Determining the superiority of the models is simple in some cases when the differences in the evaluation measurements are very noticeable. In this case, even though the Isolation Forest seems to outperform the One-class SVM in most scenarios, it is preferred if we perform a statistical significance test to make sure.

Demšar advises using non-parametric tests when comparing such algorithms. The Wilcoxon signed ranks test is most suitable in this case, as two machine learning algorithms are being compared [129]. Defining the null hypothesis as both algorithms performing similarly with an insignificant difference, the Wilcoxon signed ranks test will reject the null hypothesis if the $p - value$ is less than a threshold, usually set to 0.05. The test achieved a $p - value$ of $1.9381e - 18$ which rejects the null hypothesis with very high confidence. Therefore, this test indicates that the Isolation Forest model's superiority is, indeed, significant. Additionally, to evaluate the effect size, Cliff's delta has been used which indicates how often one algorithm would outperform the other. The obtained d value was 0.4489, again showing that the difference is noticeable.

4.4.1 Comparison with other novel architectures

Among the novel anomaly detection approaches presented and explained in Section 2.2, we have chosen the soft-bound and one-class Deep SVDD [66], one-class neural networks (OC-NN) [67], and one-class adversarial nets (OCAN) [68] to compare the obtained results with. These were the most suitable approaches for our intended application, with regards to the main concerns in this study. The architectures were tweaked for optimal performance and trained and tested on our datasets. Tables 4.11 to 4.12 demonstrate the comparison on the computer-generated and the lab-generated datasets, respectively. According to the results, both of the proposed

models outperform all other methods on both datasets based on all of the measures.

Table 4.11: Computer generated dataset - comparison with other architectures

Model	Accuracy	TPR	FPR	FNR
Isolation Forest	0.9959	0.9949	0.0006	0.0051
One-class SVM	0.9927	0.9911	0.0012	0.0089
Soft-bound Deep SVDD [66]	0.9859	0.9885	0.0239	0.0115
One-class Deep SVDD [66]	0.9826	0.9843	0.0239	0.0157
OC-NN [67]	0.9923	0.9914	0.0047	0.0086
OCAN [68]	0.7826	0.6859	0.1208	0.3141

Table 4.12: Lab generated dataset - comparison with other architectures

Model	Accuracy	TPR	FPR	FNR
Isolation Forest	0.9995	1.0	0.0014	0.0
One-class SVM	0.9990	0.9999	0.0040	0.0001
Soft-bound Deep SVDD [66]	0.9981	1.0	0.0080	0.0
One-class Deep SVDD [66]	0.9989	1.0	0.0043	0.0
OC-NN [67]	0.9780	0.9780	0.0222	0.0220
OCAN [68]	0.9990	1.0	0.0041	0.0

4.5 Conclusion

A pipeline leak detection system is designed based on the proposed anomaly detection architecture, which includes 2 models: the normal model, a time-series forecasting algorithm using a deep architecture, and the anomaly detector, which is a one-class classifier based on conventional ML algorithms. In addition to the normal model which is trained only on healthy data by concept, the anomaly detector was trained using only the healthy data, as well. This will help prevent the model from over-fitting

to very specific types of anomalies, especially pipeline leaks which are extremely rare and there is virtually no available data collected under real leak conditions. In summary, all compared models have achieved high performance in all aspects, with the best combination being the 1-dimensional CNN followed by the Isolation Forest, owing to CNN's computational efficiency and the higher accuracy obtained by the Isolation Forest. Comparison with other novel state-of-the-art methods on our datasets also confirm the effectiveness and superiority of the proposed architecture.

Future research possibilities include enhancing the system by the addition of more capabilities such as identification of the type of failure (*e.g.* distinction between a sensor reading failure and an actual leak), leak localization, and detection of the size of the leak. Naturally, these advancements require the availability of more data, alongside the addition of possible extra models.

Chapter 5

Application in Condition Monitoring and Fault Detection of Small Induction Motors

5.1 Introduction

A crucial class of electrical machines is the induction motors which, in modern industry, carry out most of the energy transforming tasks [130]. Consequently, their maintenance is of utmost importance. Figure 5.1 illustrates an induction motor, which is mainly composed of a stator, the stationary part of the electromagnetic circuit, a rotar, the moving part of the motor that turns the shaft, bearings that support the rotor and enable it to turn, and a shaft which delivers the power and is where the load connects to the motor. Simply put, the interactions of the rotor with the stator's magnetic field moves the rotor which causes the shaft to turn.

The most frequently occurring defects in induction motors are: stator inter-turn faults, cracked rotor bars, and bearing faults [131]. Contamination, project errors, overheating, *etc.*, can all result in stator faults. Rotor misalignment and imbalance, and broken bars or end rings often provoke rotor faults. Mechanical stresses, incorrect assembling, incorrect lubrication, and misalignment could be the cause of bearing faults [132]. As Figure 5.2 states, Bearing failures, stator faults, rotor faults, and other defects account for approximately 40%, 38%, 10%, and 12%, of the total motor

failures, respectively [132, 133]. Research studies of over 40 years have confirmed that these defect types can be detected in motor sensor data before motor failure. Thereby there has always been a significant interest in utilizing the information so as to minimize motor failures.

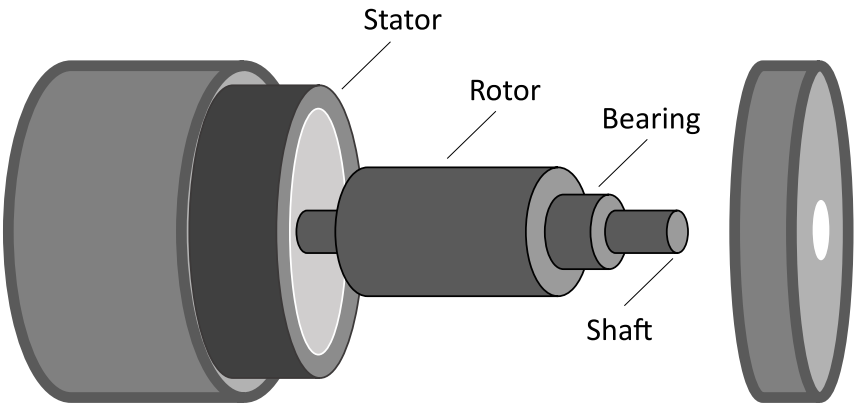

Figure 5.1: A simple diagram of an induction motor [134].

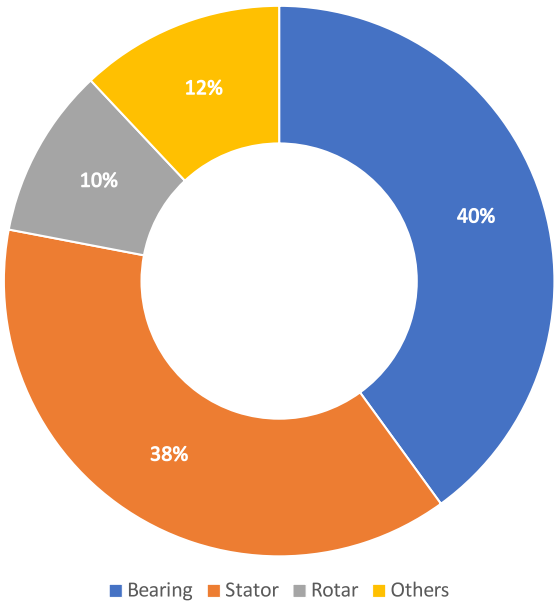

Figure 5.2: Types of faults in induction motors [133].

The use of non-destructive testing or sensed data to recognize changes within a monitored system is called *conditionMonitoring* (CM). Unlike the previous approaches that could solely flag the occurrence of a change, modern approaches can

detect the causative fault, its location, and the damage occurred so far [135]. These are substantially inferential sensing approaches (the estimation of a complex, time-variant system state using the information available from simpler sensors [136–140]). [141–144] are some examples of such approaches. Certain adaptive systems are even capable of self-repair attempts using this information [135]. Today, CM is a well-researched system which allows for predictive maintenance before a fault intensifies or causes failure, and is well suited to maintaining large and expensive systems as an economical approach [145, 146].

Productive sensing modalities for condition monitoring of electric motors consist of current and voltage monitoring, chemical analysis of motor oil, thermal/infrared sensors, axial electromagnetic flux monitoring, vibration sensors, and acoustic sensors [147]. The most recent work has emphasized stator current assessment, especially current harmonics which is an in-operation, non-invasive modality, and characteristic frequency responses for various faults have been derived from machine physics [148]. Machine learning (ML) applications in fault detection and diagnosis of complex machines are covered in [130–149]. Deep Learning approaches have also been applied to fault diagnosis in [150–153]. Nevertheless, small electrical motors (rated at 10 HP or less) have not been studied to the same extent, since they can easily be replaced and are not directly process-critical. Nonetheless, given large industrial locations usually contain several hundreds of these motors for the operation of important sub-components and protection systems for high-value systems, their failure is bound to have severe negative impacts on the plant. Examples of their usage include operating cooling, lubrication, and HVAC equipment that protects the larger systems. Certainly, the proposed CM solutions for these motors must be cost-effective, since these motors have a low replacement cost.

This research focuses on several sensing modalities for detecting small motor faults. We believe ML approaches would succeed in designing an economical solution. We treat condition monitoring as an instance of the anomaly detection problem, where

a model of "normal" behavior is constructed, and an anomaly detector detects any deviations from the expected normal behavior. Deep architectures (*e.g.* 1-dimensional convolutional neural networks and LSTM) are explored for the normal model, whereas conventional ML algorithms are preferred for the anomaly detector. Ultimately, a design combination of a one-dimensional convolutional neural network as the normal model and a random forest as the anomaly detector is proven best.

The rest of this chapter is organized as follows. The next section provides an explanation of the data collection procedures. Next, we go over the results and discussions. Finally, a conclusion is provided in the last section.

5.2 Dataset

There are no publicly accessible CM datasets for groups of electric motors as far as we are aware. Additionally, there are no data available regarding condition monitoring of small electric motors that satisfy the following conditions: 1. simultaneously record temperature, current, and voltage, 2. at sampling rates up to $10kHz$, and 3. include multiple failure modes among several instances of a common motor type. The first and second qualities provide the opportunity to examine both high-rate features such as current harmonics, along with low-rate characteristics such as power profiles. The third quality enables us to compare CM solution for both fault detection and fault identification in a constant motor design. Hence, designing an experimental apparatus and procedures to obtain a suitable dataset was a necessity for this study.

A motor manufacturer in Edmonton-area donated a group of 20 identical three-phase motors, each rated at 1 HP. A mount was then built to hold a motor with the rotor horizontal, and coupled to a dynamometer. Thermal and electrical sensors were then attached, along with data loggers. The basic experimental design was to run the motors, in an undamaged state, under full-load and no-load conditions. Ultimately, after the collection of healthy (*i.e.* undamaged) data was complete, each motor was damaged in a specific manner, and the tests were repeated. No catastrophic failure,

which results in termination of the test, occurred during the procedures [31].

The experimental apparatus used to collect the data is illustrated in Figure 5.3. The used motors were stainless steel 3-phase 1 HP motors with 6205 sized drive and fan bearings and were rated at both 208V and 480V line-to-line. In this study, 208V was selected. The results demonstrated a maximum current of 3.8 A at full-load and the full-load speed was rated at 1145 RPM.

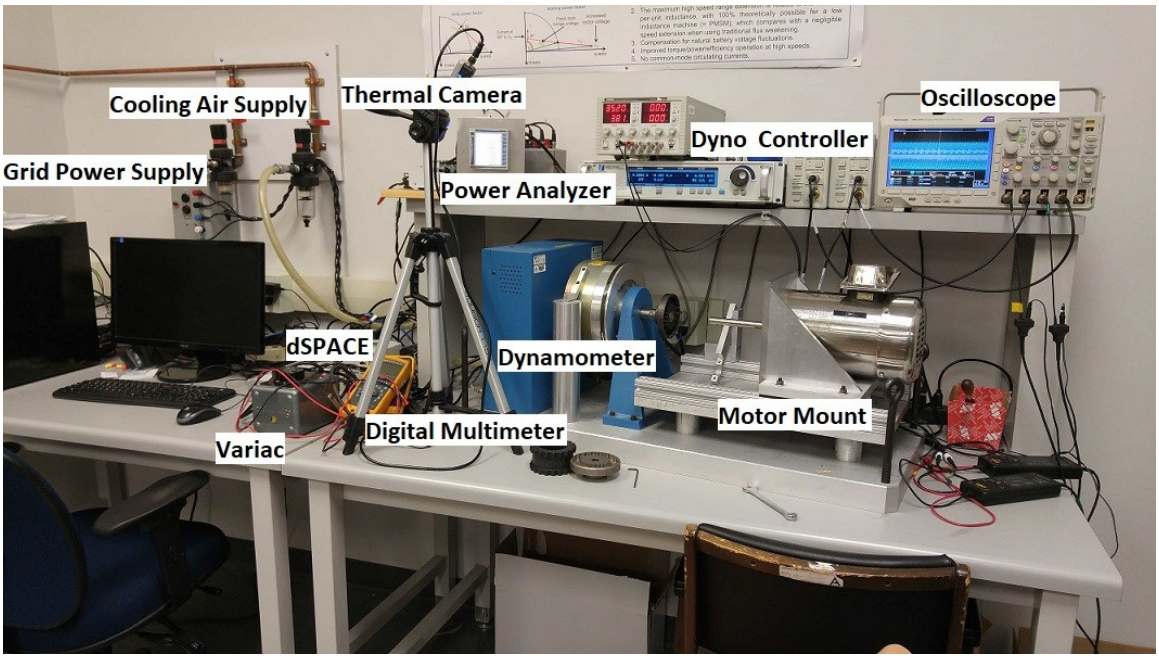

Figure 5.3: The experimental Setup [31].

To measure the current for the line to line locked-rotor tests, a digital multi-meter was used. The dSPACE tool was utilized for all the other tests. In this experiment, 6 of the 8 analog-to-digital converters (ADCs) were used (3 phase currents and 3 line to line voltages) along with one UART (Universal Asynchronous Receiver-Transmitter) input for the dynamometer, which is the device used to apply the load to the motor. The dynamometer was sampled at $2Hz$, and the ADC channels were sampled at $10kHz$. The dynamometer controller was set to generate a maximum (full load) torque of $5.624Nm$ and was used to measure the shaft's rotational speed in RPM once the motor was coupled to the dynamometer. In order to prevent the dynamometer

from overheating, the cool air supply was used. For the no-load uncoupled tests, the tachometer was used to record the shaft speed. To measure the average RMS current, RMS voltage, and real and apparent input electrical power, the power analyser is used. The Variac which is an autotransformer was used to compute the decreased impedance in the phase-to-phase locked rotor test for the stator-shorted motor. The oscilloscope used for this project is capable of sampling at $100MHz$ but was used to sample at $500kHz$ for each test except for the inrush test, where it was set to sample at $12.5MHz$. The thermal camera was connected via an Ethernet switch to a computer where the video feed was saved. A thorough video was recorded throughout each test, including the 2.5-hour warming period for each "hot" test (sensed data is collected once the motor has reached steady-state heat). Lastly, a custom mount was built out of aluminum to hold the dynamometer, rails for the mount to slide on, two mounts to bolt the motors onto, and an adjustable rotor lock ensuring that locked rotor test could be run.

Table 5.1 shows the controlled variables and their possible states in these experiments. Motor alignment is either centered (default condition) or twisted (the drive end bearings were under uneven lateral pressure). The motor temperature was either hot (motor had run under full load for 2.5 hours, reaching steady-state temperature) or cold (the motor had been stopped and allowed to cool to reach room temperature; this length of time was empirically determined using the thermal imager). Dynamometer coupling is binary, indicating whether the motor is coupled to the dynamometer or not. The applied load was either full (dynamometer set to 5.624 Nm) or no load. Rotor lock is also a binary variable indicating whether the rotor lock was applied, which means the rotor cannot rotate. Finally, each motor was either undamaged (*i.e.* healthy), or was damaged in only one way. Thus, for each motor, there are two possible values for this variable.

According to the design of the experiment, each motor experienced only one type

Table 5.1: Controlled variables - CM dataset

Variable name	**Possible values**
Motor alignment	Centered (default), Twisted
Motor temperature	Hot (steady-state), Cold (room temperature)
Dynamometer coupling	Coupled, Uncoupled
Load	Full-load ($5.624Nm$), No-load
Rotor lock	Locked, Unlocked
Motor damage	No-damage (healthy), Damaged

of damage. After performing trial runs with undamaged motors to refine observation procedures and troubleshoot the data collection systems, the "no-damage" (*i.e.* healthy) portion of the experiments was executed. Subsequently, the motors were damaged, and then, the "damaged" segment of the experiment was implemented. The types of damages induced in this experiment consist of stator short and bearing faults, which include: 1. increased wear and pitting due to foreign materials entering the bearing, 2. overheating using a torch, and 3. a 3mm hole in the outer track [31].

Ultimately, the condition monitoring dataset consists of 6 features (3-phase voltages and currents) and $130,000$ records. Additionally, the tests were combined for 3 of the motors (that were assigned to damaged-bearings treatment) together, to evaluate the algorithms' effectiveness on a population of motors, as well. In the end, the data preprocessing steps including the delay-embedding were performed. The final dataset includes $1,800,000$ samples, each containing 109 elements which include one class label indicating whether this data sample is healthy or damaged. This would be the dataset used for training and testing the algorithms.

5.3 Results and Discussion

As discussed in Chapter 3, the proposed architecture consists of two models. Hence, separate results for each model are provided. The candidate architectures for the

normal model were ultimately narrowed down to 1-dimensional convolutional neural networks (1-D CNN) based models. The 1-D CNN model includes 3 Conv1D layers with 32, 64, and 128 filters, strides equal to 1, and same padding. BatchNormalization and Maxpooling1D are performed after each layer and one Dropout just before the fully-connected output layer. Table 5.2 demonstrates the performance of the 1-dimensional CNN normal model on the condition monitoring dataset. The validation set error is measured in terms of Root-Mean-Square-Error ($RMSE$), as discussed before in Section 3.4.

Table 5.2: Normal model - Condition monitoring dataset

Model	**No. of epochs**	**RMSE**
1-D CNN	200	0.001

Figure 5.4 illustrates the training process of the normal model by showing the training and validation $RMSE$ after each epoch.

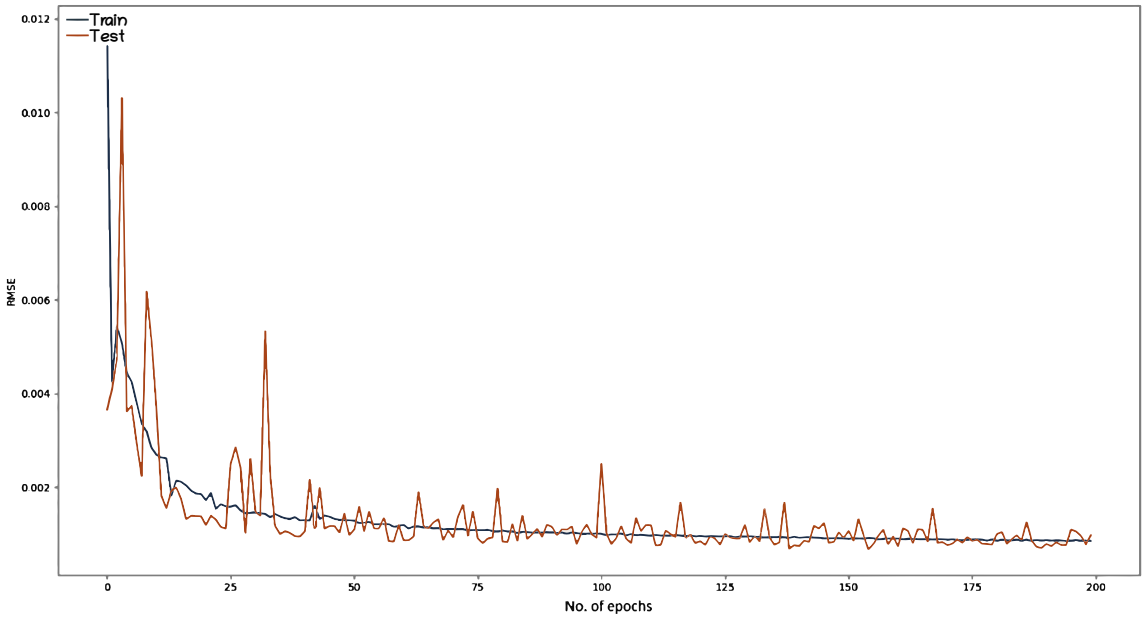

Figure 5.4: Validation and train errors of the normal model throughout the training process [31]

For the sake of comparison, Table 5.3 shows the train and test $RMSE$ of an RBFN

network trained and tested on the same dataset [31]. As expected, the 1-dimensional CNN model outperforms the RBFN model by a large margin, with the test RMSE of approximately an order of magnitude less.

Table 5.3: Normal model - RBFN [31]

No. of clusters	**Train RMSE**	**Test RMSE**
100	0.0323	0.0766

Regarding the anomaly detector, the results for this model are essentially the final results of the proposed model. Therefore, all evaluations have been obtained using the out-of-sample test set. The architectures selected for this model were ultimately narrowed down to the Multi-layer Perceptrons (MLP) [154], Decision Trees [155], and Random Forests [156]. Furthermore, the differences of the expected values and the actual observations ($\widehat{\vec{X}}_n - \vec{X}_n$ in Figure 3.1) were also added to the inputs of the anomaly detector, as this helps increase the performance in some scenarios. Thus, models have either 12 or 18 inputs. Table 5.4 illustrates the results of these models when coupled with the 1-dimensional CNN normal model.

Table 5.4: Anomaly Detector - using the 1-D CNN normal model

Model	**No. of inputs**	**TPR**	**FPR**	**Accuracy**
MLP	12	0.732	0.166	0.824
MLP	18	0.795	0.129	0.861
Decision Tree	12	0.944	0.037	0.956
Decision Tree	18	0.748	0.162	0.811
Random Forest	12	0.948	0.033	0.968
Random Forest	18	0.841	0.101	0.859

According to the results, a Random forest (incorporating 100 estimators) with the original 12 inputs is the best model in terms of accuracy, TPR, and FPR. As for the

impact of adding the differences of the inputs, that seems to only have helped the MLP. For the other two models, redundant inputs had a negative influence.

Once more, let's compare the deep learning's capabilities with the conventional models'. This time, the comparison is between the best anomaly detector using a deep architecture for the normal model with the models using the RBFN normal model. Table 5.5 shows the performance of these models [31]. According to the results, even though the Decision Tree and Random Forest achieved slightly better TPRs, their false alarm rate (FPR) is much higher, resulting in lower accuracies. In addition, having a low false alarm rate is extremely essential in anomaly detection applications in the industry, including condition monitoring. The reason why is the unnecessary and drastic costs of shutting down the system when there is no need. RBF is the best model here with an accuracy of 0.912, which is still much lower than the Random Forest using the 1-dimensional CNN normal model. Therefore, the deep model is superior in all aspects.

Table 5.5: Anomaly Detector - using the RBFN normal model [31]

Model	**TPR**	**FPR**	**Accuracy**
MLP	0.947	0.366	0.791
RBF	0.918	0.085	0.912
Decision Tree	0.967	0.207	0.881
Random Forest	0.993	0.264	0.864

5.3.1 Comparison with other novel architectures

Once again we'll be comparing our proposed model with some of the novel anomaly detection approaches presented and explained in Section 2.2. We have chosen the soft-bound and one-class Deep SVDD [66], one-class neural networks (OC-NN) [67], and one-class adversarial nets (OCAN) [68] to compare the obtained results with. These were the most suitable approaches for our intended application, with regards to the

main concerns in this study. The architectures were tweaked for optimal performance and trained and tested on our datasets. Table 5.6 demonstrates the comparison. According to the results, the proposed model using random forest outperforms all other methods based on all of the measures.

Table 5.6: comparison with other architectures

Model	**TPR**	**FPR**	**Accuracy**
Random Forest	0.948	0.033	0.968
Soft-bound Deep SVDD [66]	0.907	0.684	0.605
One-class Deep SVDD [66]	0.919	0.635	0.637
OC-NN [67]	0.861	0.697	0.581
OCAN [68]	0.273	0.260	0.506

5.4 Conclusion

A condition monitoring and leak detection system is designed based on the proposed anomaly detection architecture, which consists of 2 models: the normal model, a time-series forecasting algorithm using a deep architecture, and the anomaly detector, which is a conventional classifier. The normal model is trained only on healthy data by concept, in order to predict healthy or normal behavior. The anomaly detector is then trained on both healthy and damaged data and, thus, will distinguish the occurrence of any fault from normal behavior. In summary, using deep architectures for the normal model has given the anomaly detector an evident superiority over the conventional models, with the best combination being the 1-dimensional CNN followed by a Random Forest. Comparison with other novel state-of-the-art methods on our datasets also confirm the effectiveness and superiority of the proposed architecture.

Future research possibilities include applying the anomaly detection model to a

group of connected motors. A common layout in the industries is a group of small motors sharing a common bus. The added challenge here is that the occurrence of any anomalies and damage to one motor can easily cause further damage in the system, due to possible surges in currents and voltages. Additionally, this anomaly detector could be applied to a broad range of failure detection and prediction systems, as results in condition monitoring and pipeline leak detection are very promising so far.

Chapter 6

Conclusions, Recommendations, & Future Work

6.1 Conclusions

Machine learning and especially deep learning are dominating the modern world in many aspects and they achieve more successes every day. An area that needs more advancements, however, is anomaly detection. In this thesis, we applied both deep learning and conventional machine learning to build an anomaly detection architecture.

The proposed architecture consists of two models. The first one, the normal model, is a time-series prediction algorithm that indicates the expected healthy behavior of the system in the next time-step, without the occurrence of any anomalies. Thus, the training dataset for this model includes only healthy data. The normal model was designed using deep architectures, such as 1-dimensional convolutional neural networks (1-D CNN) and long short-term memories (LSTM). The second model, the anomaly detector, is essentially a classifier that identifies the existence of anomalies by comparing the expected healthy behavior predicted by the normal model with the actual observations. Conventional machine learning methods were used for this model.

The designed model was then applied to pipeline leak detection and condition monitoring and fault detection of small induction motors. A major difference between the

two applications was the design of the anomaly detector (the second model). Thanks to the data available in the generated condition monitoring dataset, we were able to train typical classifiers for the anomaly detector, using methods such as multi-layer perceptron (MLP), decision trees, and random forests. The pipeline leak detection dataset, however, had some restrictions. Additionally, pipeline leaks are extremely rare and there is virtually no available data collected under real leak conditions. Consequently, we chose to train the anomaly detector using healthy data only, as well, by incorporating methods such as one-class SVM and isolation forests. All in all, the hybrid deep/conventional architectures lead to excellent results in both applications.

6.2 Future Work

The anomaly detection model could certainly be improved. The addition of capabilities such as identifying the type of anomaly (as there are many things that could go wrong in a system) and localizing the problem to a small zone would be the next step. These additions most likely require extra models to be added to the architecture. Moreover, gathering real-world data is beneficial for improving the performance, as well as evaluating the models for the implementation in the industry.

Furthermore, this hybrid anomaly detector has the potential for the application to a broad range of anomaly detection and prevention problems. The promising results achieved in this research demonstrate only a fraction of the capacity.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015 (cit. on pp. 1, 4–6, 9, 10).

[2] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," *arXiv preprint arXiv:1412.2007*, 2014 (cit. on pp. 1, 5).

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112 (cit. on pp. 1, 5, 10).

[4] J. C. B. Gamboa, "Deep learning for time-series analysis," *arXiv preprint arXiv:1701.01887*, 2017 (cit. on p. 1).

[5] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019 (cit. on p. 1).

[6] J. Barrios, "Pipeline leak detection techniques and systems: Comparative assessment of pipeline leak detection methods," Master's thesis, Department of Mechanical Engineering, University of Alberta, 2019 (cit. on pp. 3, 38, 39).

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105 (cit. on p. 5).

[8] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915–1929, 2012 (cit. on p. 5).

[9] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Advances in neural information processing systems*, 2014, pp. 1799–1807 (cit. on p. 5).

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9 (cit. on p. 5).

[11] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černocký, "Strategies for training large scale neural network language models," in *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, IEEE, 2011, pp. 196–201 (cit. on p. 5).

[12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012 (cit. on p. 5).

[13] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 2013, pp. 8614–8618 (cit. on p. 5).

[14] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011 (cit. on p. 5).

[15] A. Bordes, S. Chopra, and J. Weston, "Question answering with subgraph embeddings," *arXiv preprint arXiv:1406.3676*, 2014 (cit. on p. 5).

[16] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989 (cit. on p. 5).

[17] L. Bottou, F. F. Soulié, P. Blanchet, and J.-S. Lienard, "Experiments with time delay networks and dynamic time warping for speaker independent isolated digits recognition," in *First European Conference on Speech Communication and Technology*, 1989 (cit. on p. 5).

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998 (cit. on p. 5).

[19] R. Vaillant, C. Monrocq, and Y. Le Cun, "Original approach for the localisation of objects in images," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 141, no. 4, pp. 245–250, 1994 (cit. on p. 5).

[20] S. J. Nowlan and J. C. Platt, "A convolutional neural network hand tracker," *Advances in neural information processing systems*, pp. 901–908, 1995 (cit. on p. 5).

[21] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997 (cit. on p. 5).

[22] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis.," in *Icdar*, vol. 3, 2003 (cit. on p. 5).

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014 (cit. on p. 5).

[24] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 648–656 (cit. on p. 5).

[25] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708 (cit. on p. 5).

[26] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013 (cit. on p. 5).

[27] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587 (cit. on p. 5).

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014 (cit. on p. 5).

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016 (cit. on p. 6).

[30] Jefkine, *Backpropagation in convolutional neural networks*, 2016. [Online]. Available: https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/ (cit. on pp. 7, 8).

[31] S. Sobhi, M. Reshadi, and S. Dick, "Condition monitoring and fault detection of small induction motors using machine learning algorithms," *IEEE Transactions on Systems, Man and Cybernetics: Systems*, Submitted 2019 (cit. on pp. 9, 20, 21, 34, 51, 53–56).

[32] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994 (cit. on p. 9).

[33] J. Schmidhuber and S. Hochreiter, "Long short-term memory," *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, 1997 (cit. on pp. 9, 10).

[34] S. El Hihi and Y. Bengio, "Hierarchical recurrent neural networks for long-term dependencies," in *Advances in neural information processing systems*, 1996, pp. 493–499 (cit. on p. 9).

[35] I. Sutskever, *Training recurrent neural networks*. University of Toronto Toronto, Canada, 2013 (cit. on p. 10).

[36] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, 2013, pp. 1310–1318 (cit. on p. 10).

[37] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *ICML*, 2011 (cit. on p. 10).

[38] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119 (cit. on p. 10).

[39] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014 (cit. on p. 10).

[40] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014 (cit. on p. 10).

[41] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057 (cit. on p. 10).

[42] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 2013, pp. 6645–6649 (cit. on p. 10).

[43] K. Greff, R. K. Srivastava, J. Koutnık, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016 (cit. on pp. 10–13).

[44] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999 (cit. on p. 10).

[45] G. Pang, C. Shen, L. Cao, and A. v. d. Hengel, "Deep learning for anomaly detection: A review," *arXiv preprint arXiv:2007.02500*, 2020 (cit. on pp. 13, 14).

[46] D. M. Hawkins, *Identification of outliers*. Springer, 1980, vol. 11 (cit. on p. 13).

[47] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019 (cit. on pp. 14–17).

[48] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, no. 1, pp. 1–21, 1969 (cit. on p. 14).

[49] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009 (cit. on pp. 14, 20).

[50] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: A survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015 (cit. on p. 14).

[51] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2013 (cit. on p. 14).

[52] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004 (cit. on p. 14).

[53] A. Zimek, E. Schubert, and H.-P. Kriegel, "A survey on unsupervised outlier detection in high-dimensional numerical data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 5, pp. 363–387, 2012 (cit. on p. 14).

[54] B. R. Kiran, D. M. Thomas, and R. Parakkal, "An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos," *Journal of Imaging*, vol. 4, no. 2, p. 36, 2018 (cit. on p. 14).

[55] E. Min, J. Long, Q. Liu, J. Cui, Z. Cai, and J. Ma, "Su-ids: A semi-supervised and unsupervised framework for network intrusion detection," in *International Conference on Cloud Computing and Security*, Springer, 2018, pp. 322–334 (cit. on p. 14).

[56] P. Perera and V. M. Patel, "Learning deep features for one-class classification," *IEEE Transactions on Image Processing*, vol. 28, no. 11, pp. 5450–5463, 2019 (cit. on p. 15).

[57] G. Blanchard, G. Lee, and C. Scott, "Semi-supervised novelty detection," *The Journal of Machine Learning Research*, vol. 11, pp. 2973–3009, 2010 (cit. on p. 15).

[58] T. T. Lu, "Fundamental limitations of semi-supervised learning," Master's thesis, University of Waterloo, 2009 (cit. on p. 15).

[59] E. Racah, C. Beckham, T. Maharaj, S. E. Kahou, M. Prabhat, and C. Pal, "Extremeweather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events," in *Advances in Neural Information Processing Systems*, 2017, pp. 3402–3413 (cit. on p. 15).

[60] H. Wu and S. Prasad, "Semi-supervised deep learning using pseudo labels for hyperspectral image classification," *IEEE Transactions on Image Processing*, vol. 27, no. 3, pp. 1259–1270, 2017 (cit. on p. 15).

[61] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *Asian conference on computer vision*, Springer, 2018, pp. 622–637 (cit. on p. 15).

[62] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016 (cit. on p. 15).

[63] R. Wu, B. Wang, W. Wang, and Y. Yu, "Harvesting discriminative meta objects with deep cnn features for scene classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1287–1295 (cit. on p. 15).

[64] H. Song, Z. Jiang, A. Men, and B. Yang, "A hybrid semi-supervised anomaly detection model for high-dimensional data," *Computational intelligence and neuroscience*, vol. 2017, 2017 (cit. on p. 15).

[65] J. T. Andrews, E. J. Morton, and L. D. Griffin, "Detecting anomalous data using auto-encoders," *International Journal of Machine Learning and Computing*, vol. 6, no. 1, p. 21, 2016 (cit. on p. 15).

[66] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International conference on machine learning*, 2018, pp. 4393–4402 (cit. on pp. 16, 44, 45, 56, 57).

[67] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *arXiv preprint arXiv:1802.06360*, 2018 (cit. on pp. 16, 44, 45, 56, 57).

[68] P. Zheng, S. Yuan, X. Wu, J. Li, and A. Lu, "One-class adversarial nets for fraud detection," *arXiv preprint arXiv:1803.01798*, 2018 (cit. on pp. 16, 44, 45, 56, 57).

[69] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. R. Salakhutdinov, "Good semi-supervised learning that requires a bad gan," in *Advances in neural information processing systems*, 2017, pp. 6510–6520 (cit. on p. 16).

[70] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PloS one*, vol. 11, no. 4, e0152173, 2016 (cit. on p. 16).

[71] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49 (cit. on p. 16).

[72] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *International Conference on Learning Representations*, 2018 (cit. on p. 16).

[73] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, 2014, pp. 4–11 (cit. on p. 16).

[74] D. Abati, A. Porrello, S. Calderara, and R. Cucchiara, "Latent space autoregression for novelty detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 481–490 (cit. on p. 16).

[75] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*, Presses universitaires de Louvain, vol. 89, 2015, pp. 89–94 (cit. on p. 16).

[76] W. Lawson, E. Bekele, and K. Sullivan, "Finding anomalies with generative adversarial networks for a patrolbot," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 12–13 (cit. on p. 16).

[77] J. Andrews, T. Tanay, E. J. Morton, and L. D. Griffin, "Transfer representation-learning for anomaly detection," JMLR, 2016 (cit. on p. 16).

[78] V. Vercruyssen, W. Meert, and J. Davis, "Transfer learning for time series anomaly detection," in *CEUR Workshop Proceedings*, vol. 1924, 2017, pp. 27–37 (cit. on p. 16).

[79] K. Li, N. Du, and A. Zhang, "Detecting ecg abnormalities via transductive transfer learning," in *Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, 2012, pp. 210–217 (cit. on p. 16).

[80] R. Socher, M. Ganjoo, C. D. Manning, and A. Ng, "Zero-shot learning through cross-modal transfer," in *Advances in neural information processing systems*, 2013, pp. 935–943 (cit. on p. 17).

[81] Y. Xian, B. Schiele, and Z. Akata, "Zero-shot learning-the good, the bad and the ugly," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4582–4591 (cit. on p. 17).

[82] A. Mishra, S. Krishna Reddy, A. Mittal, and H. A. Murthy, "A generative model for zero shot learning using conditional variational autoencoders," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 2188–2196 (cit. on p. 17).

[83] F. de La Bourdonnaye, C. Teuliere, T. Chateau, and J. Triesch, "Learning of binocular fixations using anomaly detection with deep reinforcement learning," in *2017 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2017, pp. 760–767 (cit. on p. 17).

[84] C. Huang, Y. Wu, Y. Zuo, K. Pei, and G. Min, "Towards experienced anomaly detector through reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018 (cit. on p. 17).

[85] R. J. Frank, N. Davey, and S. P. Hunt, "Time series prediction and neural networks," *Journal of intelligent and robotic systems*, vol. 31, no. 1-3, pp. 91–103, 2001 (cit. on p. 17).

[86] C. Wu and K.-W. Chau, "Data-driven models for monthly streamflow time series prediction," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 8, pp. 1350–1367, 2010 (cit. on p. 17).

[87] C. Wu, K. Chau, and C. Fan, "Prediction of rainfall time series using modular artificial neural networks coupled with data-preprocessing techniques," *Journal of Hydrology*, vol. 389, no. 1-2, pp. 146–167, 2010 (cit. on p. 17).

[88] F. Zhao, J. Chen, L. Guo, and X. Li, "Neuro-fuzzy based condition prediction of bearing health," *Journal of Vibration and Control*, vol. 15, no. 7, pp. 1079–1091, 2009 (cit. on p. 17).

[89] R. Jursa and K. Rohrig, "Short-term wind power forecasting using evolutionary algorithms for the automated specification of artificial intelligence models," *International Journal of Forecasting*, vol. 24, no. 4, pp. 694–709, 2008 (cit. on p. 17).

[90] H. Kantz and T. Schreiber, *Nonlinear time series analysis*. Cambridge university press, 2004, vol. 7 (cit. on pp. 17–19, 21).

[91] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical systems and turbulence, Warwick 1980*, Springer, 1981, pp. 366–381 (cit. on pp. 17, 21).

[92] O. Y. Poodeh, "Applications of complex fuzzy sets in time series prediction," PhD thesis, Department of Electrical and Computer Engineering, University of Alberta, 2017 (cit. on p. 18).

[93] M. B. Kennel, R. Brown, and H. D. Abarbanel, "Determining embedding dimension for phase-space reconstruction using a geometrical construction," *Physical review A*, vol. 45, no. 6, p. 3403, 1992 (cit. on p. 18).

[94] R. Hegger, H. Kantz, and T. Schreiber, "Practical implementation of nonlinear time series methods: The tisean package," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 9, no. 2, pp. 413–435, 1999 (cit. on pp. 18, 19, 28).

[95] L. Cao, A. Mees, and K. Judd, "Dynamics from multivariate time series," *Physica D: Nonlinear Phenomena*, vol. 121, no. 1-2, pp. 75–88, 1998 (cit. on p. 19).

[96] S. Boccaletti, D. Valladares, L. M. Pecora, H. P. Geffert, and T. Carroll, "Reconstructing embedding spaces of coupled dynamical systems from multivariate data," *Physical Review E*, vol. 65, no. 3, p. 035 204, 2002 (cit. on p. 19).

[97] L.-y. Su, "Prediction of multivariate chaotic time series with local polynomial fitting," *Computers & Mathematics with Applications*, vol. 59, no. 2, pp. 737–744, 2010 (cit. on p. 19).

[98] D. J. Hill and B. S. Minsker, "Anomaly detection in streaming environmental sensor data: A data-driven modeling approach," *Environmental Modelling & Software*, vol. 25, no. 9, pp. 1014–1022, 2010 (cit. on p. 20).

[99] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, IEEE, 2008, pp. 413–422 (cit. on p. 25).

[100] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 1, pp. 1–39, 2012 (cit. on p. 25).

[101] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001 (cit. on p. 25).

[102] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, pp. 1–27, 2011 (cit. on p. 25).

[103] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016 (cit. on p. 32).

[104] F. Chollet *et al.*, *Keras*, https://keras.io, 2015 (cit. on p. 32).

[105] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011 (cit. on p. 32).

[106] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. 1. 0. Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: https://doi.org/10.1038/s41592-019-0686-2 (cit. on p. 32).

[107] M. S. El-Abbasy, A. Senouci, T. Zayed, F. Mirahadi, and L. Parvizsedghy, "Artificial neural network models for predicting condition of offshore oil and gas pipelines," *Automation in Construction*, vol. 45, pp. 50–65, 2014 (cit. on pp. 33, 37).

[108] E. Karangwa, "Estimating the cost of pipeline transportation in canada," *Transport Canada*, Accessed on: March 23, 2020. [Online]. Available: http://ctrf.ca/wp-content/uploads/2014/07/Karangwa2008.pdf (cit. on p. 33).

[109] Interstate Natural Gas Association of America, *Safety every step of the way*, Accessed on: March 23, 2020. [Online]. Available: http://www.ingaa.org/File.aspx?id=12282 (cit. on p. 33).

[110] C. Belvederesi, M. S. Thompson, and P. E. Komers, "Statistical analysis of environmental consequences of hazardous liquid pipeline accidents," *Heliyon*, vol. 4, no. 11, e00901, 2018 (cit. on p. 33).

[111] API, RP, "1175: Pipeline leak detection-program management, errata march 2017," *American Petroleum Institute*, (cit. on p. 35).

[112] API, RP, "1162: Public awareness programs for pipeline operators (second ed.), 2015," *American Petroleum Institute*, (cit. on p. 35).

[113] A. Shukla and H. Karki, "Application of robotics in onshore oil and gas industry—a review part i," *Robotics and Autonomous Systems*, vol. 75, pp. 490–507, 2016 (cit. on p. 35).

[114] F. Karray, A. Garcia-Ortiz, M. W. Jmal, A. M. Obeid, and M. Abid, "Earnpipe: A testbed for smart water pipeline monitoring using wireless sensor network," *Procedia Computer Science*, vol. 96, pp. 285–294, 2016 (cit. on p. 35).

[115] A. M. Sadeghioon, N. Metje, D. N. Chapman, and C. J. Anthony, "Smartpipes: Smart wireless sensor networks for leak detection in water pipelines," *Journal of sensor and Actuator Networks*, vol. 3, no. 1, pp. 64–78, 2014 (cit. on pp. 35, 36).

[116] P.-S. Murvay and I. Silea, "A survey on gas leak detection and localization techniques," *Journal of Loss Prevention in the Process Industries*, vol. 25, no. 6, pp. 966–973, 2012 (cit. on p. 35).

[117] H. A. Kishawy and H. A. Gabbar, "Review of pipeline integrity management practices," *International Journal of Pressure Vessels and Piping*, vol. 87, no. 7, pp. 373–380, 2010 (cit. on p. 35).

[118] PHMSA, *Enhancement of the long-range ultrasonic method for the detection of degradation in buried, unpiggable pipelines*, Accessed on: March 7, 2018. [Online]. Available: https://primis.phmsa.dot.gov/rd/publicabstract12_3_02.htm (cit. on p. 36).

[119] S. Saechai, W. Kongprawechnon, and R. Sahamitmongkol, "Test system for defect detection in construction materials with ultrasonic waves by support vector machine and neural network," in *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems*, IEEE, 2012, pp. 1034–1039 (cit. on p. 36).

[120] J.-m. Zhao, S.-f. Yang, Y. Li, and X.-q. Wang, "Study on thickness detection of industrial pipe network by high-frequency ultrasound," in *Proceedings of the 2010 Symposium on Piezoelectricity, Acoustic Waves and Device Applications*, IEEE, 2010, pp. 517–521 (cit. on p. 36).

[121] U. Murdika, G. Elan, T. Yulianti, *et al.*, "Ultrasonic signal denoising based on wavelet haar decomposition level," in *2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, IEEE, 2016, pp. 89–94 (cit. on p. 36).

[122] C. Baldwin, "Fiber optic sensors in the oil and gas industry: Current and future applications," in *Opto-mechanical fiber optic sensors*, Elsevier, 2018, pp. 211–236 (cit. on p. 36).

[123] B. Arifin, Z. Li, S. L. Shah, G. A. Meyer, and A. Colin, "A novel data-driven leak detection and localization algorithm using the kantorovich distance," *Computers & Chemical Engineering*, vol. 108, pp. 300–313, 2018 (cit. on p. 36).

[124] API, RP, "1130: Computational pipeline monitoring for liquids (first ed.), reaffirmed april 2012," *American Petroleum Institute*, (cit. on p. 36).

[125] P. Angelov and A. Kordon, "Adaptive inferential sensors based on evolving fuzzy models," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 40, no. 2, pp. 529–539, 2009 (cit. on p. 37).

[126] S. Rashid, U. Akram, S. Qaisar, S. A. Khan, and E. Felemban, "Wireless sensor network for distributed event detection based on machine learning," in *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, IEEE, 2014, pp. 540–545 (cit. on p. 37).

[127] M. Romano, K. Woodward, and Z. Kapelan, "Statistical process control based system for approximate location of pipe bursts and leaks in water distribution systems," *Procedia Engineering*, vol. 186, pp. 236–243, 2017 (cit. on p. 37).

[128] O. E. Agwu, J. U. Akpabio, S. B. Alabi, and A. Dosunmu, "Artificial intelligence techniques and their applications in drilling fluid engineering: A review," *Journal of Petroleum Science and Engineering*, vol. 167, pp. 300–315, 2018 (cit. on p. 37).

[129] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006 (cit. on p. 44).

[130] K. Kim and A. G. Parlos, "Induction motor fault diagnosis based on neuropredictors and wavelet signal processing," *IEEE/ASME Transactions on mechatronics*, vol. 7, no. 2, pp. 201–219, 2002 (cit. on pp. 47, 49).

[131] A. Jawadekar, S. Paraskar, S. Jadhav, and G. Dhole, "Artificial neural network-based induction motor fault classifier using continuous wavelet transform," *Systems Science & Control Engineering: An Open Access Journal*, vol. 2, no. 1, pp. 684–690, 2014 (cit. on pp. 47, 49).

[132] E. L. Bonaldi, L. E. d. L. de Oliveira, J. G. B. da Silva, G. Lambert-Torresm, and L. E. B. da Silva, "Predictive maintenance by electrical signature analysis to induction motors," in *Induction Motors-Modelling and Control*, IntechOpen, 2012 (cit. on pp. 47–49).

[133] K. Guota and A. Kaut, "A review on fault diagnosis on induction motor using artificial neural networks," *International journal of science and research*, vol. 3, no. 7, pp. 680–684, 2014 (cit. on pp. 48, 49).

[134] F. Lin, C. KT, C. CC, and C. Liu, "Fault diagnosis of power components in electric vehicles," *Journal of Asian Electric Vehicles*, vol. 11, no. 2, pp. 1659–1666, 2013 (cit. on pp. 48, 49).

[135] E. P. Carden and P. Fanning, "Vibration based condition monitoring: A review," *Structural health monitoring*, vol. 3, no. 4, pp. 355–377, 2004 (cit. on p. 49).

[136] B. Joseph and C. B. Brosilow, "Inferential control of processes: Part i. steady state analysis and design," *AIChE Journal*, vol. 24, no. 3, pp. 485–492, 1978 (cit. on p. 49).

[137] C. Brosilow and M. Tong, "Inferential control of processes: Part ii. the structure and dynamics of inferential control systems," *AIChE Journal*, vol. 24, no. 3, pp. 492–500, 1978 (cit. on p. 49).

[138] B. Joseph and C. Brosilow, "Inferential control of processes: Part iii. construction of optimal and suboptimal dynamic estimators," *AIChE Journal*, vol. 24, no. 3, pp. 500–509, 1978 (cit. on p. 49).

[139] A. Jutan, J. MacGregor, and J. Wright, "Multivariable computer control of a butane hydrogenolysis reactor: Part ii. data collection, parameter estimation, and stochastic disturbance identification," *AIChE Journal*, vol. 23, no. 5, pp. 742–750, 1977 (cit. on p. 49).

[140] S. J. Qin, H. Yue, and R. Dunia, "Self-validating inferential sensors with application to air emission monitoring," *Industrial & engineering chemistry research*, vol. 36, no. 5, pp. 1675–1685, 1997 (cit. on p. 49).

[141] C. Booth and J. R. McDonald, "The use of artificial neural networks for condition monitoring of electrical power transformers," *Neurocomputing*, vol. 23, no. 1-3, pp. 97–109, 1998 (cit. on p. 49).

[142] H. Kamohara, A. Takinami, M. Takeda, M. Kano, S. Hasebe, and I. Hashimoto, "Product quality estimation and operating condition monitoring for industrial ethylene fractionator," *Journal of chemical engineering of Japan*, vol. 37, no. 3, pp. 422–428, 2004 (cit. on p. 49).

[143] R. E. Lamberson, *Apparatus and method for the remote monitoring of machine condition*, US Patent 5,845,230, Dec. 1998 (cit. on p. 49).

[144] N. Daroogheh, A. Baniamerian, N. Meskin, and K. Khorasani, "Prognosis and health monitoring of nonlinear systems using a hybrid scheme through integration of pfs and neural networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 8, pp. 1990–2004, 2016 (cit. on p. 49).

[145] P. Henriquez, J. B. Alonso, M. A. Ferrer, and C. M. Travieso, "Review of automatic fault diagnosis systems using audio and vibration signals," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 5, pp. 642–652, 2013 (cit. on p. 49).

[146] S.-j. Wu, N. Gebraeel, M. A. Lawley, and Y. Yih, "A neural network integrated decision support system for condition-based optimal predictive maintenance policy," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 37, no. 2, pp. 226–236, 2007 (cit. on p. 49).

[147] S. Nandi, H. A. Toliyat, and X. Li, "Condition monitoring and fault diagnosis of electrical motors—a review," *IEEE transactions on energy conversion*, vol. 20, no. 4, pp. 719–729, 2005 (cit. on p. 49).

[148] Y. Trachi, E. Elbouchikhi, V. Choqueuse, and M. E. H. Benbouzid, "Induction machines fault detection based on subspace spectral estimation," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 9, pp. 5641–5651, 2016 (cit. on p. 49).

[149] J. Sun, Y. Chai, C. Su, Z. Zhu, and X. Luo, "Bldc motor speed control system fault diagnosis based on lrgf neural network and adaptive lifting scheme," *Applied Soft Computing*, vol. 14, pp. 609–622, 2014 (cit. on p. 49).

[150] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, "Real-time motor fault detection by 1-d convolutional neural networks," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 7067–7075, 2016 (cit. on p. 49).

[151] K. Li and Q. Wang, "Study on signal recognition and diagnosis for spacecraft based on deep learning method," in *2015 Prognostics and System Health Management Conference (PHM)*, IEEE, 2015, pp. 1–5 (cit. on p. 49).

[152] K. K. Reddy, S. Sarkar, V. Venugopalan, and M. Giering, "Anomaly detection and fault disambiguation in large flight data: A multi-modal deep auto-encoder approach," in *Annual Conference of the Prognostics and Health Management Society*, vol. 2016, 2016 (cit. on p. 49).

[153] W. Sun, S. Shao, R. Zhao, R. Yan, X. Zhang, and X. Chen, "A sparse auto-encoder-based deep neural network approach for induction motor faults classification," *Measurement*, vol. 89, pp. 171–178, 2016 (cit. on p. 49).

[154] S. Haykin, *Neural Networks and Learning Machines, 3/E*. Pearson Education India, 2010 (cit. on p. 55).

[155] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984 (cit. on p. 55).

[156] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001 (cit. on p. 55).